OmniCRM □作指南

OmniCRM □ Omnitouch 为移动和固定线路服务提供商提供的全面 BSS/OSS 解决方案。一个完整的平台，处理从客户入职到计费、配置和支持的所有内容 - 所有这些都在一个集成系统中。

入门

对于客户服务人员

您的第一步：

1. 了解界面 - 熟悉导航和搜索
2. 创建客户 - 客户创建的逐步指南
3. 添加服务 - 配置您的第一个服务
4. 处理付款 - 处理客户付款
5. 充值服务 - 为客户账户添加信用

日常任务：

- 服务管理 - 管���客户服务
- 查看使用情况 - 检查余额和使用情况
- 修改服务 - 更改服务配置
- 生成发票 - 创建和发送发票

对于系统管理员

设置和配置：

1. 了解架构 - 系统概述
2. 配置系统 - 系统设置
3. 创建产品 - 构建您的目录
4. 编写剧本 - 自动化配置

5. [illegible] - [illegible]

[illegible]

- [illegible] - [illegible]
- **API [illegible]** - [illegible]
- [illegible] - [illegible] 2FA [illegible]

[illegible]

[illegible]

- [illegible] - [illegible]
- [illegible] - [illegible]
- [illegible] - [illegible]
- [illegible] - [illegible]
- [illegible] - [illegible]

快速任务参考

我想...	文档
创建新客户	创建客户
为客户添加服务	添加服务
查看服务使用情况和余额	服务使用情况
处理付款	处理付款
生成发票	发票管理
充值服务	充值与补充
管理库存	库存系统
创建产品	产品生命周期
编写配置剧本	Ansible 剧本
设置用户账户	RBAC
启用 2FA	双因素认证
搜索任何内容	全局搜索
了解系统	系统架构
使用 API	API 文档

文档库

必读书目

- 系统架构 - 包含图表的完整技术概述
- 产品生命周期指南 - 端到端的产品管理
- **Ansible** 剧本指南 - 掌握配置自动化

客户管理

- 客户、联系人和站点 - 数据模型和关系
- 创建客户 - 逐步指南
- 客户标签 - 使用标签进行组织
- 客户属性 - 自定义元数据
- 活动日志 - 跟踪所有更改

服务□作

- 服务管理 - 概述
- 添加服务 - 配置工作流程
- 分配计划 - 计划分配
- 修改服务 - 进行更改
- 服务使用情况和余额 - 监控使用情况
- 充值与补充 - 添加信用

计费与财务

- 计费概述 - 完整的计费指南
- 支付方式 - 管理支付方式
- 交易 - 收费和信用
- 处理付款 - 付款工作流程
- 发票管理 - 生成和管理发票

管理

- 系统配置 - 配置 OmniCRM
- 库存管理 - 资产跟踪
- API 密钥 - API 访问管理

安全与访问

- 身份验证程 - 身份验证的工作原理
- 双因素认证 - 设置 2FA
- 基于角色的访问控制 - 用户、角色和权限

集成

- Mailjet 集成 - 电子邮件服务
- API 文档 - REST API 参考

面向客户

- 自助服务门户 - 客户门户指南
- 客户关怀 - 自助服务功能
- 蜂窝广播系统 - 紧急警报

参考

- 变更日志 - 版本历史

OmniCRM 的特别之处是什么?

OmniCRM 汇集了您运行现代电信业务所需的所有工具:

自动化服务配置

忘掉手动配置 - OmniCRM 使用 **Ansible** 自动化 在几秒钟内配置服务。无论您在激活 SIM 卡、配置客户设备，还设置复杂的捆绑，系统都会自动处理。

它的功能：

- 一键配置服务
- 自动配置网络设备
- 实时创建计费账户
- 发送欢迎电子邮件和短信通知
- 如果出现故障，自动回滚

了解更多关于配置的信息 | 查看 **Ansible** 剧本指南

智能计费与支付

内置与 **CGRateS** 的集成提供实时计费和收费，而 Stripe 无缝处理支付处理。

它的功能：

- 实时使用情况跟踪和计费
- 自动生成发票
- 通过 Stripe 处理信用卡
- 预付费和后付费计费模型
- 灵活的定价和促销代码
- 税收计算和报告

探索计费功能 | 支付处理 | 发票管理

完整的客户管理

通过全面的关系模型管理客户、联系人、多个站点和服务。

它的功能：

- 跟踪客户、联系人和服务位置
- 管理每个客户的多个服务

- 存储自定义属性和元数据
- 为组织标记客户
- 完整的活动历史和审计日志
- Google 地图集成，用于站点地理编码

客户管理指南 | 创建客户 | 客户标签

库存管理

跟踪和管理您所有的物理和虚拟资产 - 从 SIM 卡到路由器再到 IP 地址块。

它的功能：

- 跟踪 SIM 卡、设备和配件
- 管理电话号码和 IP 地址块
- 在配置过程中自动分配
- 批量导入和导出功能
- 设置存储
- 自定义库存模板

库存系统文档

客户自助服务门户

通过 自助服务门户 赋予您的客户管理自己服务的能力。

客户可以做的事情：

- 查看服务和使用情况
- 检查余额和数据配额
- 在线支付发票
- 下载 PDF 格式的发票
- 更新联系信息
- 立即充值他们的服务

自助服务门户指南 | 客户关怀功能

企业安全

以安全为核心构建，具有全面的身份验证和授权功能。

安全功能：

- 基于 **JWT** 的身份验证
- 双因素认证 **(2FA)**，支持 TOTP 和备份代码
- 基于角色的访问控制 **(RBAC)**，具有细粒度权限
- 账户变更的电子邮件验证
- 通过活动日志进行完整的审计记录
- 会话管理和超时控制

身份验证指南 | **RBAC** 文档 | **2FA** 设置

强大的集成

OmniCRM 与行业领先的服务无缝集成：

- **CGRateS** - 用于电信级收费的实时计费和计费引擎
- **Stripe** - 安���的支付处理和卡片存储
- **Mailjet** - 使用模板的专业电子邮件投递
- **Google** 地图 - 地址验证和地理编码以确保准确的站点位置
- **RESTful API** - 与您现有系统集成

API 文档 | **Mailjet** 集成

灵活的产品目录

创建任何类型的产品提供 - 从简单的独立服务到包含多个组件的复杂捆绑。

支持的产品类型：

- 独立 - 单一服务（移动计划、互联网套餐）
- 捆绑 - 组合提供（互联网 + 电视 + 电话）
- 附加 - 补充服务（数据充值、国际通话）
- 促销 - 特别优惠和折扣

□□□

系统架构

查看完整架构文档

关键概念

理解产品与服务

这 OmniCRM 中最重要的概念之一：

- 产品 = 您目录中的模板或提供（例如，“无限移动计划”）
- 服务 = 针对特定客户的产品的活动实例（例如，“约翰·史密斯的无限移动计划”）

当您配置产品时，系统使用 **Ansible** 自动化 创建实际的服务。一个产品可以创建多个服务、没有服务（仅配置）或修改现有服务。

了解更多关于产品与服务的信息

配置的魔力

OmniCRM 的秘密武器□其 自动化配置系统：

剧本处理所有内容 - 从在 CGRateS 中创建账户到配置路由器再到发送欢迎电子邮件。如果出现故障，它会自动回滚。

深入了解: **Ansible** 剧本 | 配置系统

简化计费

每个服务生成 交易（收费或信用）。这些交易被分组到 发票 中并发送给客户。通过 Stripe 集成，付款可以自动处理。

计费概述 | 支付处理

客户层级

```
□户 (ABC 公司)
├── 联系人 (约翰·史密斯 - 计费, 简·多 - 技术)
├── 站点 (总部, 分支机构)
└── 服务
    ├── 互联网 - 总部
    ├── 移动计划 - 约翰·史密斯
    └── VoIP - 分支机构
```

一切都以层级方式组织，使得管理拥有多个位置和服务的客户变得容易。

客户、联系人和站点

为什么选择 **OmniCRM**?

一体化平台

您所需的一切都在一个集成系统中 - 无需 juggling 多个工具或供应商。

自动化优先

基于 Ansible 的配置意味着服务在几秒钟内部署，而不□几个小时。减少错误，节省时间，提高客户满意度。

灵活且可定制

从产品定义到配置剧本再到自定义属性 - 根据您的确切业务流程定制 OmniCRM。

专为电信而设计

专为服务提供商设计，具有 CGRateS 集成、库存管理和实时计费等功能。

企业安全

JWT 身份验证、2FA、RBAC 和完整的审计日志确保您的数据安全和合规。

基于 **API**

全面的 REST API 意味着您可以将 OmniCRM 与任何现有系统集成或构建自定义工具。

获取支持

文档资源

- 从 系统架构 开始，获取技术概述
- 查看 入门指南，获取角色特定的入职信息
- 检查 快速任务参考，获取常见操作
- 查阅术语表以了解术语

需要更多帮助?

浏览上面的完整文档库或使用 全局搜索 查找特定主题。

OmniCRM - □□□□□□□□□□ BSS/OSS

□□□□2025-12-23

双因素认证（2FA）

基于时间的一次性密码的增强安全性

OmniCRM 支持 双因素认证 **（2FA）**，使用基于时间的一次性密码（TOTP）。这通过要求用户提供他们的密码和来自身份验证应用程序的时间敏感代码，增加了一层额外的安全性。

另请参见：`RBAC <rbac>` 以获取 2FA 管理权限，`Authentication Flows <authentication_flows>` 以获取登录过程的详细信息。

目的

2FA 提供：

1. 增强安全性 — 即使密码被泄露，也能保护账户。
2. 合规性 — 满足受监管行业的安全要求。
3. 用户选择 — 对用户□可选的，可以按角色或全局强制执行。
4. 行业标准 — 使用与 Google Authenticator、Authy、Microsoft Authenticator 和其他标准应用程序兼容的 TOTP 协议。

2FA 的工作原理

当为用户启用 2FA 时：

1. 设置 — 用户在注册期间使用身份验证应用程序扫描二维码。
2. 登录 — 输入用户名/密码后，用户提供来自应用程序的 6 位数字代码。
3. 验证 — 系统验证基□□时间的代码□否与预期值匹配。
4. 访问 — 只有在验证通过两个因素后，用户才能获得访问权限。

启用 2FA

对于个人用户

用户可以为自己的账户启用 2FA:

1. 导航到 用户设置 或 个人资料
2. 选择 启用双因素认证
3. 使用身份验证应用程序扫描二维码
4. 输入验证码以确认设置
5. 将备份代码保存在安全位置

备份与恢复

备份代码

启用 2FA 时，用户会收到备份代码，如果他们的设备不可用，可以使用这些代码：

- 每个代码为一次性使用
- 安全离线存储代码

- 如果所有代码都已使用，则生成新代码

管理员重置

如果用户失去对其身份验证器的访问权限，并且所有备份代码都已用尽，则具有数据库访问权限的管理员可以通过清除数据库中的 totp_secret 字段手动重置用户的 2FA。用户可以重新启用 2FA。

API 集成

为用户启用 2FA

POST /2fa/enable/user/{user_id}

```
{
  "password": "current_password"
}
```

响应包括配置 URI（用于二维码）和备份代码。

验证 2FA 设置

POST /2fa/verify-setup/user/{user_id}

```
{
  "code": "123456"
}
```

登录时验证 2FA

POST /2fa/verify/user/{user_id}

```
{
  "code": "123456"
}
```

成功验证后返回访问令牌、刷新令牌和用户数据。

重新生成备份代码

POST /2fa/backup-codes/regenerate/user/{user_id}

需要身份验证。返回新的一组备份代码。

最佳实践

- 优先备份代码。在完成 2FA 设置之前，始终保存备份代码。
- 教育用户。提供清晰的设置和恢复说明。
- 安全重置过程。在手动重置数据库中的 2FA 之前，验证用户身份。

常见问题解答

支持哪些身份验证应用程序？任何兼容 TOTP 的应用程序（Google Authenticator、Authy、Microsoft Authenticator、1Password 等）。

如果我丢失了手机怎么办？使用备份代码或联系管理员重置 2FA。

我可以使用 **SMS** 而不是应用程序吗？目前，仅支持 TOTP 身份验证应用程序。

2FA 是必需的吗？这取决于您组织的政策。通常，2FA 对管理和支持人员角色的要求，但对客户用户可选的。系统不对客户账户强制执行 2FA（客户角色用户不会看到 2FA 注册提示）。

TOTP 代码效多长时间？代码每 30 秒刷新一次，并且有一个小的验证时间窗口（通常接受当前 30 秒窗口以及前一个/下一个窗口的代码，以容忍时钟偏差）。

API 密钥管理

API 密钥管理界面提供了一个基于 **Web** 的用户界面，用于创建、监控和管理用于程序访问 OmniCRM API 的 API 密钥。

注意

有关一般 API 认证概念和使用示例，请参见 `concepts_api`。

概述

API 密钥使 安全、长期的认证 成为可能，适用于：

- 服务器到服务器的集成
- 自动化脚本
- 第三方应用程序
- 定时任务和 cron 作业
- 外部监控系统

与 JWT 令牌（在几分钟/小时后过期）不同，API 密钥在手动撤销或到期之前始终有效。

访问 **API** 密钥管理

导航到：

或直接：

所需权限： `MANAGE_API_KEYS`（管理员角色）

API 密钥列表视图

主页面以表格格式显示所有 API 密钥：

列：

- 名称 - API 密钥的描述性标签（例如，“配置系统”，“监控工具”）

- 创建者 - 创建密钥的用户的用户名
- **API** 密钥 - 实际的密钥字符串（出于安全原因部分隐藏）
- 状态 - 活动、过期或已撤销
- 创建日期 - 密钥生成的时间
- 到期日期 - 密钥将自动过期的时间
- □作 - 编辑、删除、重新生成按钮

示例显示：

仪表板小部件

在页面顶部，显示摘要统计信息：

- 总 **API** 密钥 - 所有 API 密钥的计数（活动和非活动）
- 活动密钥 - 当前有效的密钥
- 即将到期 - 在接下来的 30 天内到期的密钥
- 过期密钥 - 超过到期日期的密钥

创建 **API** 密钥

步骤 **1**：点击“添加 **API** 密钥”

点击 API 密钥列表右上角的 **+** 添加 按钮。

步骤 **2**：填写详细信息

会弹出一个模态表单，要求输入：

> 名称：_____
> （例如，“配置系统”）
>
> 描述：_____
> （可选 - 此密钥的目的）
>
> 到期日期：[日期选择器]
> （可选 - 留空表示没有到期）

> 权限：☐ 查看客户 ☐ 创建客户 ☐ 查看服务 ☐ 创建服务 ☐ 配置 ☐ 查看库存 ☑ 管理员（所有 权限）
>
> [取消] [生成密钥]

字段指南：

名称（必填）

- 简短、描述性的标识符
- 示例：“��置系统”，“计费集成”，“监控”
- 用于审计日志并在列表中显示

描述（可选）

- 更详细的说明
- 示例：“由 Ansible 配置服务器使用”，“第三方计费同步”
- 帮助未来的管理员理解密钥的目的

到期日期（可选）

- 如果留空：密钥永不失效（不推荐）
- 如果设置：密钥在此日期后自动失效
- 推荐：出于安全考虑设置到期（90 天至 1 年）

权限

- 选择特定权限或勾选“管理员”以获得完全访问权限
- 遵循与用户帐户相同的基于角色的权限模型
- 最佳实践：分配最低必要权限

步骤 **3**：生成并复制密钥

点击 **“生成密钥”** 后，系统会显示新创建的 API 密钥：

> ⚠ 立即复制此密钥 - 以后将不再显示！
>
> sk_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0
>
> [复制到剪贴板]

> [关闭]

警告

请立即保存 **API** 密钥!

关闭此对话框后，无法再次检索完整密钥。您只会在列表视图中看到一个被遮蔽的版本（`sk_live_...XYZ`）。

如果您丢失了密钥，您必须 重新生成 它，这将使旧密钥失效，并可能破坏现有集成。

步骤 **4**: 配置的应用程序

在应用程序的请求中使用 API 密钥:

```
curl -X GET "https://yourcrm.com/crm/customers" \
  -H "X-API-KEY: sk_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0"
```

或在环境变量中:

```
export
CRM_API_KEY="sk_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0"
```

管理现有密钥

查看密钥详细信息

点击任何 API 密钥名称以查看完整详细信息:

- 完整密钥名称和描述
- 创建时间戳
- 创建者用户名
- 关联权限
- 使用统计信息（如果已实现）
- 最近的访问日志

重新生成 API 密钥

如果 API 密钥被泄露或丢失，请重新生成它：

1. 点击操作列中的 ⋮（三个点）
2. 选择“重新生成密钥”
3. 确认操作

警告

重新生成会立即使旧密钥失效。

使用旧密钥的任何应用程序将停止工作。在重新生成之前，请使用新密钥更新所有集成。

发生的事情：

- 旧密钥被撤销
- 生成具有相同权限的新密钥
- 显示新密钥（立即复制）
- 名称、描述和权限保持不变

撤销（删除）API 密钥

要永久删除 API 密钥：

1. 点击操作列中的 ⋮（三个点）
2. 选择 “删除”
3. 确认删除

发生的事情：

- 密钥立即被撤销
- 使用此密钥的所有请求返回 `401 Unauthorized`
- 密钥从数据库中删除
- 无法撤销 - 密钥无法恢复

何时撤销：

- 集成不再需要

- 密钥已被泄露
- 使用该密钥的系统已被淘汰
- 用具有不同权限的新密钥替换

编辑 API 密钥详细信息

要修改 API 密钥的详细信息：

1. 点击操作列中的 ⋮（三个点）
2. 选择“编辑”
3. 更新名称、描述、到期或权限
4. 点击“保存更改”

可编辑字段：

- 名称
- 描述
- 到期日期
- 权限

不可编辑：

- 密钥值本身（使用重新生成来更改）
- 创建日期
- 创建者用户

API 密钥状态

API 密钥可以有几种状态：

活动

- 密钥有效且可以使用
- 在到期日期内（或未设置到期）
- 未手动撤销
- 显示为绿色徽章

即将到期

- 活动但将在接下来的 30 天内到期
- 显示为橙色/警告徽章
- 考虑在到期前进行轮换

过期

- 超过到期日期
- 不再接受认证
- 显示为红色徽章
- 可以删除或延长到期

已撤销

- 手动删除/禁用
- 永久无效
- 不再显示在活动列表中

过滤和搜索

API 密钥列表支持：

搜索：

按名称、描述或部分密钥搜索：

按状态过滤：

过滤下拉菜单显示：

- 所有密钥
- 仅活动
- 即将到期（接下来的 30 天）
- 过期

排序：

点击列标题按以下内容排序：

- 名称
- 创建日期
- 到期日期
- 创建者

安全最佳实践

API 密钥生成

- 长度：密钥应至少为 32 个字符（系统强制执行此项）
- 随机性：使用密码学安全的随机数生成器生成
- 格式：通常以前缀形式（例如，`sk_live_`）进行标识

API 密钥存储

在 **CRM** 中：

- 密钥在存储前被哈希（如同密码）
- 创建时仅显示完整密钥一次
- 数据库存储哈希以供验证
- 即使管理员也无法在之后检索完整密钥

在您的应用程序中：

- 存储在环境变量中，而不代码中
- 使用秘密管理系统（AWS Secrets Manager、HashiCorp Vault）
- 永远不要将密钥提交到版本控制
- 定期轮换密钥（90-365 天）

权限管理

- 最小权限原则 - 仅授予必要的权限
- 除非绝对必要，否则避免创建管理员密钥

- 为不同的系统/目的使用单独的密钥
- 定期审查权限

监控和审计

- 通过活动日志监控 API 密钥使用情况
- ���置异常访问模式的警报
- 定期审查“最后使用”时间戳
- 删除未使用的密钥

密钥轮换

建立密钥轮换政策：

1. 创建新密钥，权限相同
2. 更新应用程序以使用新密钥
3. 监控以确保旧密钥不再使用
4. 在宽限期后撤销旧密钥

故障排除

使用 API 密钥时出现“401 Unauthorized”

- 原因：密钥无效、过期或不正确
- 解决方法：
  - 验证密钥□否正确复制（没有额外空格）
  - 检查密钥状态（活动与过期）
  - 确认密钥具有所需权限
  - 确保使用 `X-API-KEY` 头（而不□ `Authorization`）

创建后出现“未找到 API 密钥”

- 原因：密钥可能已创建但未正确存储
- 解决方法：
  - 检查 API 密钥列表□否有新条目

- 如果缺失，请创建新密钥
- 向管理员报告问题

API 密钥即将过期

- 原因：到期日期临近（30 天内）
- 解决方法：
  - 创建具有延长到期的新密钥
  - 更新应用程序以使用新密钥
  - 在迁移后撤销旧密钥

无法删除 API 密钥

- 原因：可能被保护或正在使用
- 解决方法：
  - 确保您具有管理员权限
  - 检查密钥□否被锁定/保护
  - 如果问题仍然存在，联系管理员

API 端点（用于程序管理）

API 密钥也可以通过 API 管理（需要管理员权限）：

列出 API 密钥

```
GET /crm/api-keys
Authorization: Bearer <admin-token>
```

创建 API 密钥

```
POST /crm/api-keys
Authorization: Bearer <admin-token>
Content-Type: application/json

{
  "name": "New Integration",
  "description": "Third-party billing sync",
  "expiry_date": "2026-01-10",
  "permissions": ["view_customer", "view_service"]
}
```

□□□

```
{
  "api_key_id": 123,
  "name": "New Integration",
  "api_key": "sk_live_a1b2c3d4e5f6g7h8i9j0",
  "status": "active",
  "created": "2025-01-10T10:00:00Z",
  "expiry_date": "2026-01-10T23:59:59Z"
}
```

□□ API □□

```
DELETE /crm/api-keys/{api_key_id}
Authorization: Bearer <admin-token>
```

□□ API □□

```
PATCH /crm/api-keys/{api_key_id}
Authorization: Bearer <admin-token>
Content-Type: application/json

{
  "name": "Updated Name",
  "expiry_date": "2026-12-31"
}
```

常见用例

用例 **1**：配置系统集成

为您的 Ansible 配置服务器创建一个 API 密钥：

1. 导航到 API 密钥 → 添加
2. 名称：“Ansible 配置服务器”
3. 描述：“由配置��动化使用”
4. 权限：配置、查看/创建服务、查看/更新库存
5. 到期：1 年
6. 复制密钥并添加到 Ansible `crm_config.yaml`

用例 **2**：第三方计费集成

为计费导出创建一个只读密钥：

1. 名称：“计费同步 - QuickBooks”
2. 权限：查看客户、查看交易、查看发票
3. 到期：90 天（每季度轮换）
4. 在定期导出脚本中使用

用例 **3**：监控和警报

为 Prometheus/Grafana 指标收集创建一个密钥：

1. 名称：“监控 - Grafana”
2. 权限：查看服务、查看配置
3. 到期：永不（监控需要持续访问）
4. 在 Grafana 数据源中配置

用例 **4**：客户门户 **API**

为客户自助服务门户创建一个密钥：

1. 名称：“客户门户后端”
2. 权限：查看自己的客户、查看自己的服务、创建付款
3. 到期：180 天

4. 在门户后端 API 调用中使用

相关文档

- `concepts_api` - API 认证概念和示例
- `rbac` - 基于角色的访问控制和权限
- `2fa` - 额外安全性的双因素认证

□□□□

□□□

- `regulatory_entity` = "FCC-123456"（监管标识符）

4. 运营元数据

存储运营信息：

- `preferred_contact_method` = "email"（沟通偏好）
- `account_manager` = "<john.smith@company.com>"（分配的客户经理）
- `onboarding_date` = "2025-01-15"（客户生命周期跟踪）
- `churn_risk_score` = "0.23"（预测分析）

5. 配置参数

存储特定于配置的设置：

- `radius_username_format` = "email"（自定义RADIUS格式）
- `vlan_id` = "100"（网络配置）
- `ipv6_enabled` = "true"（功能标志）

属性与标准字段

使用属性的情况：

- 数据□特定于部署的或因安装而异
- 需求经常变化
- 存储特定于集成的元数据
- 在添加数据库字段之前原型化新功能
- 数据不需要复杂的查询或连接

使用标准字段的情况：

- 数据□客户模型的核心（姓名、电子邮件、地址）
- 需要频繁搜索、过滤或报告
- 数据具有引用完整性约束
- 对于大规模查询，性能至关重要

通过UI管理属性

查看客户属性

要查看客户的属性：

1. 导航到客户的概览页面
2. 点击 属性 标签
3. 您将看到此客户的所有属性的表格，显示：
    - 属性名称（键）
    - 属性值
    - 创建日期
    - 最后修改日期

创建新属性

要为客户创建新属性：

1. 导航到客户的概览页面
2. 点击 属性 标签
3. 点击 添加属性 按钮
4. 填写必填字段：
    - 属性名称（必填）：此属性的键/名称（例如，`external_crm_id`）
    - 属性值（必填）：要存储的值（例如，`SF-12345`）
5. 点击 创建属性

命名约定：

- 使用小写字母和下划线：`external_system_id` ✓
- 避免空格：`external system id` ✗
- 保持名称描述性但简洁
- 对于相同属性类型，在客户之间使用一致的命名

编辑属性

要编辑现有属性：

1. 导航到客户的概览页面
2. 点击 属性 标签
3. 在表格中找到您要编辑的属性
4. 点击 编辑（铅笔）按钮
5. 修改属性名称或值
6. 点击 更新属性

注意

更改属性名称会创建一个新的键值对。确保这不会破坏依赖于��始属性名称的集成。

删除属性

要删除属性：

1. 导航到客户的概览页面
2. 点击 属性 标签
3. 在表格中找到您要删除的属性
4. 点击 删除（垃圾桶）按钮
5. 在弹出窗口中确认删除

警告

删除被集成、配置工作流或计费逻辑使用的属性可能会导致故障。在删除之前验证依赖关系。

属性字段参考

API集成

属性可以通过API以编程方式进行管理：

创建或更新属性

端点：`PUT /crm/attribute/`

所需权限: `create_customer_attribute`

请求体:

```
{
  "customer_id": 123,
  "attribute_name": "external_crm_id",
  "attribute_value": "SF-12345"
}
```

响应:

```
{
  "attribute_id": 456,
  "customer_id": 123,
  "attribute_name": "external_crm_id",
  "attribute_value": "SF-12345",
  "created": "2025-01-04 10:30:00",
  "last_modified": "2025-01-04 10:30:00"
}
```

更新现有属性

端点: `PATCH /crm/attribute/attribute_id/{attribute_id}`

所需权限: `update_customer_attribute`

请求体:

```
{
  "attribute_value": "SF-54321"
}
```

通过ID获取属性

端点: `GET /crm/attribute/attribute_id/{attribute_id}`

所需权限: `view_customer_attribute`

响应：


```
{
  "attribute_id": 456,
  "customer_id": 123,
  "attribute_name": "external_crm_id",
  "attribute_value": "SF-12345",
  "created": "2025-01-04 10:30:00",
  "last_modified": "2025-01-04 10:30:00"
}
```


通过客户ID获取所有属性

端点：`GET /crm/attribute/customer_id/{customer_id}`

所需权限：`view_customer_attribute`

响应：


```
[
  {
    "attribute_id": 456,
    "customer_id": 123,
    "attribute_name": "external_crm_id",
    "attribute_value": "SF-12345",
    "created": "2025-01-04 10:30:00",
    "last_modified": "2025-01-04 10:30:00"
  },
  {
    "attribute_id": 457,
    "customer_id": 123,
    "attribute_name": "billing_method",
    "attribute_value": "quarterly",
    "created": "2025-01-04 10:35:00",
    "last_modified": "2025-01-04 10:35:00"
  }
]
```

删除属性

端点: `DELETE /crm/attribute/attribute_id/{attribute_id}`

所需权限: `delete_customer_attribute`

响应:

```
{
  "result": "success"
}
```

批量属性□作

管理多个属性

要一次为客户设置多个属性（例如，在入职或集���同步期间）：

```
import requests

customer_id = 123
attributes = [
    {"attribute_name": "external_crm_id", "attribute_value": "SF-
12345"},
    {"attribute_name": "billing_method", "attribute_value":
"quarterly"},
    {"attribute_name": "support_tier", "attribute_value":
"premium"}
]

for attr in attributes:
    attr["customer_id"] = customer_id
    requests.put(
        "https://api.example.com/crm/attribute/",
        json=attr,
        headers={"Authorization": "Bearer YOUR_TOKEN"}
    )
```

按属性查询客户

虽然属性没有内置的搜索端点，但您可以使用客户搜索API通过属性过滤客户：

```
# 取所有客户，然后在应用程序代码中按属性过滤
customers = requests.get("https://api.example.com/crm/customer/").jsc

for customer in customers:
    attributes = requests.get(

f"https://api.example.com/crm/attribute/customer_id/{customer['custom
    ).json()

    # 查找具有特定属性的客户
    for attr in attributes:
        if attr['attribute_name'] == 'support_tier' and
attr['attribute_value'] == 'premium':
            print(f"Premium customer: {customer['customer_name']}")
```

注意

对于频繁的基于属性的查询，考虑添加索引数据库字段或实现专用搜索端点。

最佳实践

1. 命名约定

- 使用蛇形命名法：`external_system_id` ✓
- 描述性强：`billing_method` ✓ 与 `method` ✗
- 避免保留关键字或特殊字符
- 在您的部署指南中记录属性含义

2. 数据类型

- 属性将值存储为字符串（最大150个字符）
- 对于布尔值，使用 "true"/"false"（小写）
- 对于日期，使用ISO 8601格式："2025-01-04 10:30:00"
- 对于大型JSON数据，考虑使用专用数据库字段

3. 验证

- 在保存之前在应用程序代码中验证属性值
- 在客户之间使用一致的值格式
- 记录每个属性名称的预期值

4. 文档

- 维护属性名称和目的的注册表
- 记录哪些系统/集成依赖于特定属性
- 包含有效值的示例

5. 迁移和清理

- 定期审计未使用的属性
- 在系统迁移后删除过时的属性
- 在更改架构时版本化属性名称（例如，`api_key_v2`）

示例工作流

入职集成

在从遗留系统迁移客户时：

```
# 存储遗留系统参考以进行调试
PUT /crm/attribute/
{
  "customer_id": 123,
  "attribute_name": "legacy_crm_id",
  "attribute_value": "OLD-12345"
}

# 跟踪迁移日期
PUT /crm/attribute/
{
  "customer_id": 123,
  "attribute_name": "migrated_date",
  "attribute_value": "2025-01-04"
}
```

自定义计费规则

为特定客户覆盖默认计费周期：

```
# 设置季度计费
PUT /crm/attribute/
{
  "customer_id": 123,
  "attribute_name": "billing_cycle",
  "attribute_value": "quarterly"
}

# 然后在计费代码中，在处理之前检查属性
attributes = GET /crm/attribute/customer_id/123
billing_cycle = next(
    (a['attribute_value'] for a in attributes if
a['attribute_name'] == 'billing_cycle'),
    'monthly'  # 默认
)
```

功能标志

为特定客户启用测试功能：

```
# 启用IPv6配置
PUT /crm/attribute/
{
  "customer_id": 123,
  "attribute_name": "feature_ipv6_enabled",
  "attribute_value": "true"
}
```

权限

属性操作需要以下权限：

- `view_customer_attribute` - 查看属性
- `create_customer_attribute` - 创建新属性
- `update_customer_attribute` - 修改现有属性
- `delete_customer_attribute` - 删除属性

请参见 `rbac` 以获取基于角色的访问控制配置。

故障排除

属性未在UI中出现

- 验证属性是否已创建（检查API响应）
- 刷新页面以重新加载客户数据
- 检查用户是否具有 `view_customer_attribute` 权限

无法更新属性

- 确保您具有 `update_customer_attribute` 权限
- 验证 attribute_id 是否正确
- 检查属性是否属于指定客户

属性删除后集成失败

- 使用先前的值恢复属性

- □□□□

配置和自定义指南

本综合指南涵盖了 OmniCRM 的所有配置和自定义选项，包括后端设置、前端品牌、视觉自定义和部署最佳实践。

概述

OmniCRM 使用两个主要的配置系统：

后端配置：

- 文件：`OmniCRM-API/crm_config.yaml`
- 格式：YAML
- 要求：更改后需要重启 API
- 用于：数据库、集成、安全、供应

前端配置：

- 文件：`OmniCRM-UI/.env`
- 格式：环境变量
- 要求：更改后需要重建 UI
- 用于：品牌、功能、外部服务

后端配置（crm_config.yaml）

`crm_config.yaml` 文件包含所有后端系统设置。

位置：`/OmniCRM/OmniCRM-API/crm_config.yaml`

数据库配置

```yaml
database:
  username: omnitouch
  password: omnitouch2024
  server: localhost
```

字段：

- `username` - MySQL 数据库用户名
- `password` - MySQL 数据库密码
- `server` - 数据库服务器主机名或 IP（默认：localhost）

数据库连接：

- 数据库名称硬编码为 `omnicrm`
- 默认端口：3306（MySQL 默认）
- 连接字符串：`mysql+pymysql://username:password@server/omnicrm`

安全提示：切勿将此文件与真实凭据提交到版本控制。使用特定于环境的配置或秘密管理。

注意：这应与您的 `.env` 数据库凭据匹配。

服务类型

```yaml
service_types:
  - omnicharge
  - mobile
  - fixed
  - fixed-voice
  - hotspot
  - dongle
```

目的：定义您部署的有效服务类型。

这些在系统中用于：

- 产品分类

- 附加组件过滤（附加组件与服务类型匹配）
- 供应工作流
- 报告和分析

在此添加自定义服务类型以满足您的特定用例。

HSS 集成（家庭用户服务器）

对于具有 HSS 集成的移动网络运营商：

```
hss:
  hss_peers:
    - 'http://10.179.2.140:8080'
  apn_list: "1,2,3,4,5,6"
```

配置：

- `hss_peers` - 订阅者供应的 HSS 端点列表
- `apn_list` - 可用于供应的 APN ID 的逗号分隔列表

用于：移动网络供应和订阅者身份验证。

Mailjet 电子邮件模板配置

OmniCRM 使用 Mailjet 发送事务性电子邮件。每种电子邮件类型都有自己的模板配置：

```yaml
mailjet:
  api_key: your_mailjet_api_key
  api_secret: your_mailjet_api_secret

  # 客户欢迎电子邮件
  api_crmCommunicationCustomerWelcome:
    from_email: "support@yourcompany.com"
    from_name: "Your Company Support"
    template_id: 5977509
    subject: "Welcome to YourCompany"

  # 客户发票电子邮件
  api_crmCommunicationCustomerInvoice:
    from_email: "billing@yourcompany.com"
    from_name: "Your Company Billing"
    template_id: 6759851
    subject: "Your Invoice - "

  # 发票提醒
  api_crmCommunicationCustomerInvoiceReminder:
    from_email: "billing@yourcompany.com"
    from_name: "Your Company Billing"
    template_id: 5977570
    subject: "Invoice Payment Reminder"

  # 用户欢迎电子邮件（员工/管理员）
  api_crmCommunicationUserWelcome:
    from_email: "admin@yourcompany.com"
    from_name: "Your Company Admin"
    template_id: 6118112
    subject: "Welcome to the Team"

  # 密码重置请求
  api_crmCommunicationUserPasswordReset:
    from_email: "security@yourcompany.com"
    from_name: "Your Company Security"
    template_id: 6735666
    subject: "Password Reset Request"

  # 密码重置成功确认
  api_crmCommunicationUserPasswordResetSuccess:
    from_email: "security@yourcompany.com"
    from_name: "Your Company Security"
```

```
  template_id: 6118378
  subject: "Password Reset Successful"

# 密码更改通知
api_crmCommunicationUserPasswordChange:
  from_email: "security@yourcompany.com"
  from_name: "Your Company Security"
  template_id: 6118423
  subject: "Password Changed"

# 电子邮件验证
api_crmCommunicationEmailVerification:
  from_email: "verify@yourcompany.com"
  from_name: "Your Company Verification"
  template_id: 6267350
  subject: "Verify Your Email Address"

# 余额过期通知
api_crmCommunicationsBalanceExpired:
  from_email: "alerts@yourcompany.com"
  from_name: "Your Company Alerts"
  template_id: 7238252
  subject: "Service Balance Expired"

# 余额不足警告
api_crmCommunicationsBalanceLow:
  from_email: "alerts@yourcompany.com"
  from_name: "Your Company Alerts"
  template_id: 7238263
  subject: "Low Balance Warning"

# eSIM 交付电子邮件
# 将 template_id 设置为有效的 Mailjet 模板以使用自定义模板，
# 或将其保留为 0 以使用内置品牌 HTML 电子邮件
api_crmCommunicationEsimDelivery:
  from_email: "support@yourcompany.com"
  from_name: "Your Company Support"
  template_id: 0
  subject: "Your eSIM is Ready"
```

eSIM 交付电子邮件

当 eSIM 被供应并准备激活时发送。

```yaml
api_crmCommunicationEsimDelivery:
  from_email: "support@yourcompany.com"
  from_name: "Your Company Support"
  template_id: 0
  subject: "Your eSIM is Ready"
```

发送时间：

- eSIM 在自助注册期间被供应
- eSIM 分配给客户服务

可用模板变量（使用 **Mailjet** 模板时）：

- `{{ var:customer_name }}` - 客户的全名
- `{{ var:iccid }}` - eSIM ICCID
- `{{ var:activation_code }}` - LPA 激活代码字符串
- `{{ var:has_qr_code }}` - 布尔值，如果附加了 QR 代码图像，则为 true
- `{{ var:has_activation_code }}` - 布尔值，如果激活代码可用，则为 true

内置后备：

当 `template_id` 设置为 `0` 时，OmniCRM 发送带有 eSIM QR 代码内联的内置品牌 HTML 电子邮件。此电子邮件的品牌由 `branding` 部分控制（见下文）。

品牌（内置电子邮件模板）

当 OmniCRM 使用其内置 HTML 模板（而不是 Mailjet 模板）发送电子邮件时，`branding` 部分控制视觉外观。

```yaml
branding:
  brand_name: 'YourCompany'
  primary_dark: '#594712'
  dark_color: '#32280B'
  accent_bg: '#f5f0e0'
```

字段：

键	描述	默认	示例
`brand_name`	在电子邮件主题、问候和签名中显示的公司名称	从 `from_name` 派生	`'Norfone'`
`primary_dark`	用于标题和强调边框的主要品牌颜色	`#594712`	`'#1a73e8'`
`dark_color`	在电子邮件头部���变中使用的深色	`#32280B`	`'#0d47a1'`
`accent_bg`	信息提示框的浅背景颜色	`#f5f0e0`	`'#e3f2fd'`

当前使用者：

- eSIM 交付电子邮件（当 `api_crmCommunicationEsimDelivery.template_id` 为 `0` 时）

创建 **Mailjet** 模板：

1. 登录 Mailjet 仪表板 (<https://app.mailjet.com>)
2. 导航到 **Transactional → Templates**
3. 创建新模板或克隆现有模板
4. 注意 **Template ID**（数字值）
5. 添加与 OmniCRM 数据结构匹配的模板变量
6. 更新 `crm_config.yaml` 以包含模板 ID

可用模板变量：

每种电子邮件类型接收特定变量。常见示例：

- `{{customer_name}}` - 客户或用户名称
- `{{service_name}}` - 服务或产品名称
- `{{invoice_id}}` - 发票编号
- `{{invoice_amount}}` - 发票总金额
- `{{due_date}}` - 付款到期日
- `{{reset_link}}` - 密码重置 URL
- `{{verification_link}}` - 电子邮件验证 URL
- `{{balance}}` - 当前账户余额
- `{{expiry_date}}` - 余额或服务到期日

供应配置

```
provisioning:
  failure_list: ['admin@yourcompany.com', 'ops@yourcompany.com']
```

目的:

- `failure_list` - Ansible 供应失败时通知的电子邮件地址
- 通知包括剧本名称、错误详情和客户信息
- 允许运维团队快速响应供���问题

发票配置

```
invoice:
  template_filename: 'yourcompany_invoice_template.html'
```

目的:

指定用于 PDF 发票生成的 Jinja2 HTML 模板。

模板位置: `/OmniCRM-API/invoice_templates/`

有关创建自定义模板的详细信息，请参见下面的 发票 **PDF** 生成 部分。

CRM 基础 URL

```
crm:
  base_url: 'http://localhost:5000'
```

目的:

- 由 Ansible 剧本用于进行 API 回调
- 用于电子邮件模板中生成指向 CRM 的链接
- 应为您 API 的公共可访问 URL

示例:

- 开发: `http://localhost:5000`
- 生产: `https://api.yourcompany.com`

重要:

- 不要包含尾部斜杠
- 如果剧本在不同服务器上运行，请使用公共可访问的 URL
- 部署到生产时更新

OCS 和 CGRates 配置

```
ocs:
  ocsApi: 'http://10.179.2.142:8080/api'
  ocsTenant: 'mnc380.mcc313.3gppnetwork.org'
  cgrates: 'localhost:2080'
```

配置:

- `ocsApi` - 用于订阅者管理的 OCS API 端点
- `ocsTenant` - 多租户 OCS 部署的租户标识符
- `cgrates` - CGRates JSON-RPC API 端点（主机: 端口）

用于：实时计费、余额管理、使用跟踪。

SMSC 配置（SMS 网关）

```
smsc:
  source_msisdn: 'YourCompany'
  smsc_url: 'http://10.179.2.216/SMSc/'
  api_key: 'your_smsc_api_key'
```

目的:

- 向客户发送 SMS 通知（余额不足、服务警报、2FA 代码）
- `source_msisdn` - 显示给收件人的发件人 ID（字母数字或电话号码）
- `smsc_url` - SMSC 网关 API 端点

- `api_key` - SMSC API 的身份验证

小区广播配置

```
cbc_url: 'http://10.179.1.113:8080'
```

目的：小区广播中心（CBC）API 端点，用于紧急警报。

有关使用详细信息，请参见 `features_cell_broadcast`。

JWT 秘密密钥

```
jwt_secret: 'CHANGE_ME_ON_DEPLOYMENT'  # 由 Ansible 自动生成
```

安全性：

- 用于签名和验证身份验证令牌（JWT）
- 对身份验证安全至关重要 - 像密码一样对待
- 每个部署/环境必须具有唯一的秘密
- 切勿共享或提交到版本控制

自动秘密管理（**Ansible** 部署）：

使用 Ansible 部署时，JWT 秘密会自动管理：

1. 首次部署：Ansible 生成一个随机的 64 字符十六进制字符串（256 位熵）
2. 后续部署：Ansible 保留服务器上的现有秘密
3. 每个服务器都有自己的唯一秘密 - 会话在不同环境之间无法使用

这确保：

- Git 仓库中没有硬编码的秘密
- 每个环境在加密上□隔离的
- 秘密在���署之间持久化（会话在重新部署时不会失效）
- 重建服务器会生成新的秘密（会话会正确失效）

手动秘密生成（非 **Ansible** 部署）：

如果不使用 Ansible 部署，您必须手动生成唯一的秘密：

```
# 生成一个加密安全的随机密钥
python3 -c "import secrets; print(secrets.token_hex(32))"
# 或者
openssl rand -hex 32
```

然后更新 `crm_config.yaml`：

```
jwt_secret: 'your_generated_secret_here'
```

重要：

- 切勿在生产中使用默认占位符值 `CHANGE_ME_ON_DEPLOYMENT`
- 切勿公开共享 - 任何拥有秘密的人都可以伪造身份验证令牌
- 更改此项会使所有现有用户会话失效（用户必须重新登录）
- 每个环境（开发、暂存、生产）必须具有不同的秘密

Stripe 支付配置

```
stripe:
  secret_key: 'sk_live_xxxxxxxxxx'
  publishable_key: 'pk_live_xxxxxxxxxx'
  currency: 'aud'
  statement_descriptor_suffix: 'YOURCOMPANY'
```

配置：

- `secret_key` - Stripe 秘密 API 密钥（服务器端，保密）
- `publishable_key` - Stripe 可发布密钥（客户端，安全暴露）
- `currency` - ISO 4217 货币代码（aud、usd、gbp、eur 等）
- `statement_descriptor_suffix` - 出现在客户信用卡账单上的描述

密钥类型：

- 测试密钥：`sk_test_...` 和 `pk_test_...`（用于开发）
- 生产密钥：`sk_live_...` 和 `pk_live_...`（用于生产）

账单描述符使用：

- 在客户银行对账单上显示为 "YOURCOMPANY"
- 最多 22 个字符
- 帮助客户识别收费
- 也用于发票 PDF 文件名（例如，`YOURCOMPANY_12345.pdf`）

有关设置详细信息，请参见 `Payment Vendor Integrations <integrations_payment_vendors>`。

API 密钥和 IP 白名单

定义具有基于角色的访问和 IP 限制的 API 密钥：

```
api_keys:
  "YOUR_API_KEY_1":
    roles: ["admin"]
    ips: ["127.0.0.1", "::1"]
  "YOUR_API_KEY_2":
    roles: ["customer_service_agent_1"]
    ips: ["127.0.0.1", "::1", "10.0.1.0/24"]

# IP 白名单（独立，无需 API 密钥）
ip_whitelist:
  "10.179.2.142":
    roles: ["admin"]
```

目的：

- 允许外部系统通过 API 密钥进行身份验证
- 按 IP 地址限制访问
- 授予 API 消费者特定角色
- 对于集成（计费系统、监控、自动化）非常有用

生成 API 密钥：

```
openssl rand -base64 48
```

角色：

- `admin` - 对所有端点的完全访问
- 在 RBAC 系统中定义的自定义角色

安全最佳实践：

- 使用长且随机的 API 密钥（最少 32 个字符）
- 将 IP 限制为已知来源
- 授予最低必要角色
- 定期轮换 API 密钥
- 在日志中监控 API 密钥使用情况

安全警告：

- 仅对受信任的内部网络使用 IP 白名单
- 必须不使用本地主机 IP（127.0.0.1、::1）
- 对于外部访问，请使用 API 密钥
- 考虑防火墙规则作为额外保护

有关详细信息，请参见 `administration_api_keys` 和 `concepts_api`。

前端配置（.env）

React UI 通过 `OmniCRM-UI/.env` 中的环境变量进行配置。

位置：`/OmniCRM/OmniCRM-UI/.env`

API 密钥和外部服务

```
# Google Maps API（用于地址自动完成和地理编码）
VITE_GOOGLE_API_KEY=your_google_api_key

# 禁用 npm start 时的浏览器自动启动
BROWSER=none
```

支付供应商配置

```
# 启用的支付供应商（逗号分隔）：stripe、paypal 或两者
VITE_PAYMENT_VENDORS="stripe,paypal"

# Stripe 可发布密钥（如果启用了 Stripe，则必需）
VITE_STRIPE_PUBLISHABLE_KEY=pk_test_xxxxx

# PayPal 客户端 ID（如果启用了 PayPal，则必需）
VITE_PAYPAL_CLIENT_ID=YOUR_PAYPAL_CLIENT_ID
```

必须与后端匹配：

- `VITE_STRIPE_PUBLISHABLE_KEY` 必须与 `crm_config.yaml` 中的 `stripe.publishable_key` 匹配
- `VITE_PAYPAL_CLIENT_ID` 必须与 `crm_config.yaml` 中的 `paypal.client_id` 匹配

重要：这些 构建时 变量。更改任何 `VITE_*` 值需要重建并重启 UI。有关完整设置程序的详细信息，包括后端配置，请参见 支付系统指南 - 供应商配置。

品牌和公司信息

```
# 公司品牌
VITE_COMPANY_NAME="ShellFone"
VITE_PORTAL_NAME="ShellManager"
VITE_SELF_CARE_NAME="ShellCare"
VITE_COMPANY_TAGLINE="Phones with Shells"
```

这些值出现在整个 UI 中：

- `COMPANY_NAME` - 显示在页面标题、电子邮件和客户通信中
- `PORTAL_NAME` - 管理员/员工门户的名称（例如，“ShellManager”）
- `SELF_CARE_NAME` - 客户自助服务门户的名称（例如，“ShellCare”）
- `COMPANY_TAGLINE` - 出现在登录屏幕和营销材料中

本地化和区域设置

```
# 语言和区域
# 支持的语言: ar,ch,en,fr,gr,it,ru,sp
VITE_DEFAULT_LANGUAGE=en
VITE_LOCALE="en-GB"

# 默认位置（用于地址自动完成）
VITE_DEFAULT_LOCATION="Sydney, Australia"
VITE_DEFAULT_COUNTRY="Australia"

# 货币设置
VITE_CURRENCY_CODE="GBP"
VITE_CURRENCY_SYMBOL="£"
```

支持的语言：

- `ar` - 阿拉伯语
- `ch` - 中文
- `en` - 英语（默认）
- `fr` - 法语
- `gr` - 希腊语
- `it` - 意大利语
- `ru` - 俄语
- `sp` - 西班牙语

常见货币：

- USD - $（美元）
- GBP - £（英镑）
- EUR - €（欧元）
- AUD - $（澳大利亚元）

- CAD - $（加元）

注意：必须与 `crm_config.yaml` 中的 `stripe.currency` 匹配。

颜色主题配置

```
# 主要颜色（主要品牌颜色）
VITE_PRIMARY_COLOR=#405189

# 其他颜色选项（注释示例）
# VITE_SECONDARY_COLOR=#2bFFcf
# VITE_TERTIARY_COLOR=#1a9fbf
# VITE_SUCCESS_COLOR=#28a745
# VITE_INFO_COLOR=#17a2b8
# VITE_WARNING_COLOR=#ffc107
# VITE_DANGER_COLOR=#dc3545
```

可用颜色：

- `VITE_PRIMARY_COLOR` - 主要品牌颜色（按钮、链接、高亮）
- `VITE_SECONDARY_COLOR` - 强调颜色
- `VITE_TERTIARY_COLOR` - 额外强调
- `VITE_SUCCESS_COLOR` - 成功消息（#28a745）
- `VITE_INFO_COLOR` - 信息消息（#17a2b8）
- `VITE_WARNING_COLOR` - 警告（#ffc107）
- `VITE_DANGER_COLOR` - 错误（#dc3545）
- `VITE_LIGHT_COLOR` - 浅背景（#f8f9fa）
- `VITE_DARK_COLOR` - 深色文本（#343a40）
- `VITE_PRIMARY_DARK_COLOR` - 主要颜色的深色变体

格式：十六进制颜色代码（#RRGGBB）

主要颜色应用于：

- 导航标题
- 动作按钮
- 链接和高亮

- 活动状态
- 品牌元素

字体配置

```
# 字体系列配置
# 无衬线: Inter,Roboto,Open Sans,Lato,Quicksand,Poppins,Nunito,
Montserrat,Work Sans,Source Sans Pro,Raleway,Ubuntu,Josefin Sans,
HKGrotesk
# 衬线: Merriweather,Lora,Playfair Display
# 系统: System (本地设备字体)
# 默认: Quicksand
VITE_FONT_FAMILY=Quicksand
```

重要：所有字体都□ 本地自托管 的，位于 OmniCRM-UI 应用程序中。这意味着：

- 不加载外部字体 - 字体与应用程序一起打包
- 围墙花园兼容 - 不需要互联网访问即可使用字体
- 离线□作 - 在隔离或受限网络环境中完全功能
- 隐私 - 不会通过字体请求进行外部请求
- 性能 - 无外部依赖的更快加载
- 安全性 - 无第三方跟踪或数据泄露

可用选项：

无衬线字体：

- Inter,Roboto,Open Sans,Lato,Quicksand (默认)、Poppins,Nunito,Montserrat,Work Sans,Source Sans Pro,Raleway,Ubuntu,Josefin Sans,HKGrotesk

衬线字体：

- Merriweather,Lora,Playfair Display

系统字体：

- System - 使用本地设备字体以获得最佳性能和最小的包大小

添加自定义字体：

的，您可以添加其他字体！所有字体都存储在应用程序中。

要添加新自定义字体：

1. 将字文件添加到 `OmniCRM-UI/src/assets/fonts/your-font-name/`
   - 使用 WOFF2 格式以获得最佳压缩和浏览器支持
   - 包括多个权重（300、400、500、600、700）以确保正确渲染
   - 文件命名：`your-font-name-300.woff2`、`your-font-name-400.woff2` 等。
2. 在 `OmniCRM-UI/src/assets/scss/fonts/_google-fonts.scss` 中定义 **@font-face** 规则

```
//
// 您的自定义字体 - 描述
//

@font-face {
  font-family: 'Your Font Name';
  font-style: normal;
  font-weight: 400;
  font-display: swap;
  src: url("../../fonts/your-font-name/your-font-name-
400.woff2") format('woff2');
}

@font-face {
  font-family: 'Your Font Name';
  font-style: normal;
  font-weight: 700;
  font-display: swap;
  src: url("../../fonts/your-font-name/your-font-name-
700.woff2") format('woff2');
}
```

3. 在 **.env** 文件中设置：

```
VITE_FONT_FAMILY=Your Font Name
```

字体重量指南：

- 300 - 轻（可选，用于微妙的标题）
- 400 - 常规（必需，默认文本）
- 500 - 中等（可选，强调）
- 600 - 半粗（可选，副标题）
- 700 - 粗体（必需，标题和强文本）

Web 应用快速链接

配置最多 6 个快速访问的 Web 应用程序，这些应用程序出现在管理员仪表板中：

```
# Web 应用 1: GitHub
VITE_WEB_APP_1_NAME="GitHub"
VITE_WEB_APP_1_URL="https://github.com"
VITE_WEB_APP_1_ICON_PATH="resources/webapp_icons/github.png"

# Web 应用 2: Xero
VITE_WEB_APP_2_NAME="Xero"
VITE_WEB_APP_2_URL="https://go.xero.com/"
VITE_WEB_APP_2_ICON_PATH="resources/webapp_icons/xero.png"

# Web 应用 3-6: 其他集成
# （使用 NAME、URL 和 ICON_PATH 类似配置）
```

模式：

- `VITE_WEB_APP_N_NAME` - 显示名称
- `VITE_WEB_APP_N_URL` - 目标 URL
- `VITE_WEB_APP_N_ICON_PATH` - 图标文件路径（相对于 public/）

示例图标：GitHub、Xero、Monday.com、Gmail、MailJet、Slack

Grafana 仪表板集成

OmniCRM 与 Grafana 集成以提供分析仪表板。具有 `grafana_access` 权限的用户可以在 `/grafana` 访问 Grafana 创建自定义仪表板，这些仪表板可以嵌入到 OmniCRM 界面中。

```
# Grafana 仪表板集成
# 仪表板 ID（逗号分隔） - 从 /grafana/d/ID/name URLs 获取
VITE_GRAFANA_DASHBOARD_IDS=abc123,def456,ghi789

# 仪表板标签（逗号分隔，必须与 ID 的顺序匹配）
VITE_GRAFANA_DASHBOARD_LABELS=System Metrics,User
Analytics,Network Performance
```

字段：

- `VITE_GRAFANA_DASHBOARD_IDS` - 从 URL 路径中获取的 Grafana 仪表板 ID 的逗号分隔列表
- `VITE_GRAFANA_DASHBOARD_LABELS` - 仪表板显示名称的逗号分隔列表（必须与 ID 的顺序匹配）

如何查找仪表板 **ID**：

查看 Grafana 中的仪表板时，URL 将类似于 `/grafana/d/abc123/dashboard-name`。仪表板 ID 在 `/d/` 之后和名称之前的部分（在此示例中为 `abc123`）。

使用：

配置完成后，仪表板会自动出现在 OmniCRM 侧边栏的“仪表板”菜单下。用户可以点击查看嵌入的仪表板，同时保持 OmniCRM 导航可见。

示例：

如果您的仪表板 URL 为 `http://your-server/grafana/d/system-overview-123/system-metrics`，那么：

- 仪表板 ID：`system-overview-123`
- 仪表板标签：选择任何友好的名称，如“系统概述”

重要：

- ID 的数量必须与标签的数量匹配
- 仪表板按定义的顺序出现在侧边栏中
- 只有具有 `grafana_access` 权限的用户才能查看仪表板
- 更改需要重建前端 UI

支持和文档 URL

```
# 常见问题和支持 URL
VITE_FAQS_URL=https://support.yourcompany.com/faqs
VITE_SUPPORT_URL=https://support.yourcompany.com
```

目的：在 UI 中显示的外部支持资源链接。

社交登录

```
# 允许社交登录（yes|no）
VITE_ALLOW_SOCIAL_LOGINS=yes
```

选项：

- `yes` - 启用社交登录按钮（Google、Facebook 等）
- `no` - 禁用社交登录

注意：社交登录提供商必须单独配置。

分析和跟踪

```
# Google Analytics 4 或 Google Tag Manager（可选）
# 对于 GA4，使用格式：G-XXXXXXXXXX
# 对于 GTM，使用格式：GTM-XXXXXXX
VITE_GOOGLE_TAG_ID=G-XXXXXXXXXX
```

目的：启用 Google Analytics 4（GA4）或 Google Tag Manager（GTM）跟踪，以监控用户行为和应用程序使用情况。

支持的格式：

- **GA4** 测量 **ID**：`G-XXXXXXXXXX` - 用于 Google Analytics 4 跟踪
- **GTM** 容器 **ID**：`GTM-XXXXXXX` - 用于 Google Tag Manager
- 禁用：留空或省略以禁用跟踪

工作原理：

系统根据 ID 格式自动检测跟踪服务：

- 以 `G-` 开头的 ID 初始化 Google Analytics 4
- 以 `GTM-` 开头的 ID 初始化 Google Tag Manager
- 空值或缺失值禁用所有跟踪

功能：

- 自动页面跟踪 - 每次路由更改都被跟踪为页面视图
- 自定义事件跟踪 - 跟踪用户交互、按钮点击、表单提交
- 无需代码更改 - 只需设置环境变量并重建 UI

设置说明：

1. 获取您的跟踪 **ID**：

   □□ Google Analytics 4□

   - 登录 Google Analytics
   - 创建或选择 GA4 属性
   - 转到 **Admin → Data Streams**
   - 选择您的 Web 流
   - 复制 **Measurement ID**（格式：`G-XXXXXXXXXX`）

   □□ Google Tag Manager□

   - 登录 Google Tag Manager
   - 创建或选择一个容器
   - 复制 **Container ID**（格式：`GTM-XXXXXXX`）

2. 添加到 **.env** 文件：

   ```
   VITE_GOOGLE_TAG_ID=G-XXXXXXXXXX
   ```

3. 重建 **UI**：

```
cd OmniCRM-UI
npm run build
```

4. 验证跟踪：

   - 在浏览器中打开您的应用程序
   - 检查浏览器控制台中的初始化消息
   - 使用 Google Analytics 实时报告查看活动用户

高级用法 - 自定义事件跟踪：

集成提供了用于跟踪自定义事件的实用函数。

示例 - 跟踪按钮点击：

```
import { trackEvent } from './utils/googleAnalytics';

// 在您的组件中
const handleCheckout = () => {
  trackEvent('checkout_initiated', {
    cart_value: 99.99,
    item_count: 3,
    page: '/cart'
  });
  // 继续结账逻辑
};
```

示例 - 跟踪表单提交：

```
import { trackEvent } from './utils/googleAnalytics';

const handleFormSubmit = (formType) => {
  trackEvent('form_submission', {
    form_type: formType,
    timestamp: new Date().toISOString()
  });
};
```

示例 - 跟踪自定义页面视图：

```
import { trackPageView } from './utils/googleAnalytics';

// 手动跟踪页面视图（对模态对话框、选项卡等很有用）
trackPageView('/virtual/page-path', 'Page Title');
```

可用跟踪函数：

函数	目的	参数
`initializeGoogleTracking()`	在应用启动时初始化跟踪	无（自动调用）
`trackEvent(eventName, eventParams)`	跟踪自定义事件	事件名称（字符串）、参数（对象）
`trackPageView(path, title)`	手动跟踪页面视图	路径（字符串）、标题（字符串）

隐私和合规性：

Google Analytics 跟踪收集用户行为数据。确保遵守数据保护法规：

- **GDPR**（欧洲） - 在跟踪之前获得用户同意，提供选择退出机制
- **CCPA**（加利福尼亚） - 允许用户选择退出数据收集
- 隐私政策 - 在您的隐私政策中披露使用 Google Analytics
- **Cookie** 同意 - 如果当地法律要求，实施 cookie ���意横幅
- 数据保留 - 在 Google Analytics 设置中配置适当的数据保留期限

故障排除：

问题	可能原因	解决方案
GA 仪表板中没有数据	跟踪 ID 不正确	验证 ID 格式□否匹配 G-XXXXXXXXXX 或 GTM-XXXXXXX
页面加载时控制台错误	无效的跟踪 ID	检查 .env 文件中的拼写错误
页面视图未跟踪	UI 在更改 .env 后未重建	运行 `npm run build` 并重启 UI
事件未出现	启用广告拦截器	在禁用广告拦截器的情况下进行测试
数据延迟	GA 的正常行为	使用实时报告进行即时验证

性能考虑：

- 跟踪脚本异步加载，不会阻塞页面渲染
- 最小性能影响（< 50KB 附加负载）
- 脚本在首次加载后由浏览器缓存
- 如果 `VITE_GOOGLE_TAG_ID` 为空，则不进行跟踪

实现细节：

- 跟踪代码：`OmniCRM-UI/src/utils/googleAnalytics.js`
- 页面跟踪钩子：`OmniCRM-UI/src/utils/usePageTracking.js`
- 初始化：在 `App.js` 或 `index.js` 中自动启动

充值和充值配置

```
VITE_TOPUP_PRICE_PER_DAY=10
```

目的：设置自助服务门户中充值服务的每日价格。

字段：

- `VITE_TOPUP_PRICE_PER_DAY` - 充值服务的每日价格（数值）

示例：如果设置为 `10`，且货币为美元，则客户每天支付 10 美元的服务费用。

注意：此值必须与后端定价配置匹配。有关完整设置详细信息，请参见 `features_topup_recharge`。

品牌和视觉自定义

OmniCRM 允许您在不修改代码的情况下，用公司的徽标和启动屏幕替换默认品牌图像。

徽标文件和后备系统

徽标存储在 `/OmniCRM-UI/src/assets/images/omnitouch/` 中，并使用后备系统：

默认徽标（始终存在）：

- `DefaultLogoDark.png` - 深色主题徽标（用于浅色背景）
- `DefaultLogoLight.png` - 浅色主题徽标（用于深色背景）

自定义徽标（可选，存在时优先）：

- `logoSm.png` - 用于折叠侧边栏的小徽标（推荐：100x100px）
- `logoDark.png` - 用于标题的全尺寸深色徽标（推荐：200x50px）
- `logoLight.png` - 用于身份验证屏幕的全尺寸浅色徽标（推荐：200x50px）

徽标后备工作原理：

系统首先尝试加载自定义徽标。如果自定义徽标文件不存在，则回退到默认徽标：

```
// 来自 Header.js
const tryImport = (filename) => {
    try {
        return require(`../assets/images/omnitouch/${filename}`);
    } catch (err) {
        return null; // 回退到默认
    }
};

const userLogoSm = tryImport("logoSm.png");
const userLogoDark = tryImport("logoDark.png");
const userLogoLight = tryImport("logoLight.png");
```

徽标出现的位置：

- **logoSm.png** - 折叠侧边栏、移动导航、小标题显示
- **logoDark.png** - 主标题栏（浅色模式）、管理员仪表板标题
- **logoLight.png** - 登录/注册屏幕、深色背景、身份验证轮播

替换徽标：

1. 创建您的徽标文件：
    - 使用 PNG 格式以支持透明度
    - 确保徽标在常规和视网膜分辨率下清晰可见
2. 添加到 **OmniCRM**：

```
# 将您的徽标文件复制到 omnitouch 图像目录
cp /path/to/your/logoSm.png OmniCRM-
UI/src/assets/images/omnitouch/
cp /path/to/your/logoDark.png OmniCRM-
UI/src/assets/images/omnitouch/
cp /path/to/your/logoLight.png OmniCRM-
UI/src/assets/images/omnitouch/
```

3. 重建 **UI**：

```
cd OmniCRM-UI
npm run build
```

4. 验证更改：
    - 检查浅色模式标题（应显示 `logoDark.png`）
    - 检查深色模式标题（应显示 `logoLight.png`）
    - 检查折叠侧边栏（应显示 `logoSm.png`）
    - 检查登录屏幕（应显示 `logoLight.png`）

徽标设计最佳实践：

- 对比度 - 确保徽标在浅色和深色���景上都可见
- 简洁性 - 徽标应在小尺寸下可识别
- 格式 - 使用带透明背景的 PNG
- 视网膜 - 为高 DPI 显示提供 2 倍分辨率
- 一致性 - 在所有徽标变体中使用相同的品牌颜色

启动屏幕和身份验证背景

身份验证屏幕（登录、注册、密码重置）使用带有可自定义图像的轮播背景。

位置：`/OmniCRM-UI/src/pages/AuthenticationInner/authCarousel.js`

默认配置：

```
import logoLight from "../../assets/images/logo-light.png";

// 在身份验证屏幕上显示的徽标
<img src={logoLight} alt="" height="18" />
```

自定义身份验证屏幕：

1. 替换 `/OmniCRM-UI/src/assets/images/` 中的 `logo-light.png`
2. 将自定义背景 CSS 添加到 `.auth-one-bg` 类
3. 修改 `authCarousel.js` 中的轮播引语以匹配您的品牌声音

示例自定义：

```
/* 添加到您的自定义 CSS */
.auth-one-bg {
    background-image: url('/assets/images/custom-auth-bg.jpg');
    background-size: cover;
    background-position: center;
}
```

发票 PDF 生成

OmniCRM 使用 Jinja2 HTML 模板和 WeasyPrint PDF 渲染生成专业的 PDF 发票。

PDF 生成工作原理

1. 模板选择：
    - 模板文件名在 `crm_config.yaml` 中的 `invoice.template_filename` 下指定
    - 从 `/OmniCRM-API/invoice_templates/` 目录加载模板
2. 数据准备：
    - 从数据库获取发票数据（ID、日期、金额、状态）
    - 检索客户信息（姓名、地址）
    - 编译交易列表（发票上的所有收费/信用）
3. 模板渲染：
    - Jinja2 使用发票数据渲染 HTML 模板
    - 像 `{{ invoice_number }}`、`{{ total_amount }}` 等变量被替换
    - 渲染的 HTML 保存到 `invoice_templates/rendered/` 以供调试
4. **PDF** 生成：
    - WeasyPrint 将渲染的 HTML 转换为 PDF
    - PDF 支持 CSS 样式、图像、分页、页眉/页脚

- PDF 二进制数据在内存中生成

5. 缓存:
   - PDF 被 Base64 编码并存储在 `Invoice_PDF_Cache` 表中
   - 计算 SHA256 哈希以进行完整性验证
   - 后续请求返回缓存的 PDF（即时交付）
6. 缓存失效:
   - 当发票被修改、作废或退款时，缓存失效
   - 模板更改不会自动使现有缓存失效

发票模板结构

发票模板□带有嵌入变量和逻辑的 Jinja2 HTML 文件。

模板位置: `/OmniCRM-API/invoice_templates/yourcompany_invoice_template.html`

可用变量:

```
{
    'invoice_number': 12345,
    'date': '2025-01-04',
    'client': {
        'name': 'John Smith',
        'address': {
            'address_line_1': '123 Main St',
            'city': 'Sydney',
            'state': 'NSW',
            'zip_code': '2000',
            'country': 'Australia'
        }
    },
    'transaction_list': [
        [
            {
                'transaction_id': 1,
                'title': 'Mobile Service - Monthly Fee',
                'retail_cost': 30.00,
                'wholesale_cost': 10.00,
                'created': '2025-01-01'
            },
            {
                'transaction_id': 2,
                'title': 'Data Addon - 5GB',
                'retail_cost': 15.00,
                'wholesale_cost': 5.00,
                'created': '2025-01-15'
            }
        ]
    ],
    'total_amount': 45.00,
    'due_date': '2025-01-31',
    'start_date': '2025-01-01',
    'end_date': '2025-01-31',
    'paid': False,
    'void': False
}
```

□□□□□□□□

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Invoice {{ invoice_number }}</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 40px;
        }
        .header {
            text-align: center;
            margin-bottom: 30px;
        }
        .invoice-details {
            margin-bottom: 20px;
        }
        table {
            width: 100%;
            border-collapse: collapse;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 12px;
            text-align: left;
        }
        th {
            background-color: #405189;
            color: white;
        }
        .total {
            text-align: right;
            font-size: 18px;
            font-weight: bold;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="header">
        <h1>INVOICE</h1>
    </div>
```

```
    <div class="invoice-details">
        <p><strong>Invoice Number:</strong> {{ invoice_number }}
</p>
        <p><strong>Date:</strong> {{ date }}</p>
        <p><strong>Due Date:</strong> {{ due_date }}</p>
        <p><strong>Billing Period:</strong> {{ start_date }} to {{
end_date }}</p>
    </div>

    <div class="customer-details">
        <h3>Bill To:</h3>
        <p>{{ client.name }}</p>
        <p>{{ client.address.address_line_1 }}</p>
        <p>{{ client.address.city }}, {{ client.address.state }}
{{ client.address.zip_code }}</p>
        <p>{{ client.address.country }}</p>
    </div>

    <table>
        <thead>
            <tr>
                <th>Description</th>
                <th>Date</th>
                <th>Amount</th>
            </tr>
        </thead>
        <tbody>
            {% for transaction in transaction_list[0] %}
            <tr>
                <td>{{ transaction.title }}</td>
                <td>{{ transaction.created }}</td>
                <td>${{ "%.2f"|format(transaction.retail_cost) }}
</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>

    <div class="total">
        <p>Total Amount Due: ${{ "%.2f"|format(total_amount) }}
</p>
    </div>

    {% if paid %}
```

```
    <div style="text-align: center; color: green; font-weight:
bold;">
        PAID
    </div>
    {% endif %}

    {% if void %}
    <div style="text-align: center; color: red; font-weight:
bold;">
        VOID
    </div>
    {% endif %}
</body>
</html>
```

模板最佳实践：

- 使用绝对路径的图像 - `file:///absolute/path/to/image.png`
- 内联 **CSS** - WeasyPrint 不可靠地加载外部样式表
- 使用示例数据进行测试 - 使用 `invoice_templates/rendered/` 检查 HTML
- 分页 - 使用 `<div style="page-break-after: always;"></div>` 进行多页发票
- 页眉和页脚 - 使用 `@page` CSS 规则进行重复元素
- 货币格式化 - 使用 Jinja2 过滤器：`{{ "%.2f"|format(amount) }}`

创建自定义发票模板

1. 复制示例模板：

   ```
   cd /OmniCRM/OmniCRM-API/invoice_templates
   cp norfone_invoice_template.html
   yourcompany_invoice_template.html
   ```

2. 编辑模板：

   - 替换公司名称、徽标、联系信息
   - 调整样式（颜色、字体、布局）以匹配品牌
   - 根据需要添加或删除部分（税务细目、付款说明等）

3. 更新配置：

   编辑 `crm_config.yaml`：

   ```yaml
   invoice:
     template_filename: 'yourcompany_invoice_template.html'
   ```

4. 测试发票生成：

   - 在 CRM 中创建测试发票
   - 下载 PDF 并验证格式
   - 检查 `invoice_templates/rendered/{invoice_id}.html` 以进行调试

5. 使旧缓存失效（如有需要）：

   如果您更改了模板并希望重新生成现有发票：

   ```sql
   -- 清除所有缓存的 PDF（强制重新生成）
   DELETE FROM Invoice_PDF_Cache;
   ```

PDF 缓存系统

为了提高性能，OmniCRM 缓存生成的 PDF：

缓存行为：

- 首次请求 - 生成 PDF、缓存并返回
- 后续请求 - 立即返回缓存的 PDF（无需重新生成）
- 缓存失效 - 当发票被修改、作废或退款时失效
- 缓存清理 - 旧缓存在 30 天不活动后自动清除

缓存存储：

- Base64 编码的 PDF 存储在 `Invoice_PDF_Cache` 表中
- SHA256 内容哈希用于完整性验证
- 包括文件名、创建时间戳、最后访问时间戳

手动缓存管理:

```
# 在 OmniCRM API 或 Python shell 中
from services.invoice_service import cleanup_old_pdf_cache,
invalidate_invoice_cache
from utils.db_helpers import get_db_session

session = get_db_session()

# 清理超过 30 天的缓存
result = cleanup_old_pdf_cache(session, days_old=30)
print(result)  # {'status': 'success', 'deleted_count': 15}

# 使特定发票缓存失效
invalidate_invoice_cache(session, invoice_id='12345')
```

API 端点:

生成/下载发票 PDF:

GET /invoice/pdf/{invoice_id}

响应: PDF 文件下载，文件名来自 Stripe 账单描述符

缓存头:

- 首次请求：响应较慢（生成时间）
- 缓存请求：即时响应
- 缓存命中/未命中对用户透明

应用配置更改

后端 (crm_config.yaml)

1. 编辑 `OmniCRM-API/crm_config.yaml`
2. 保存更改
3. 重启 API 服务:

```
cd OmniCRM-API
sudo systemctl restart omnicrm-api
# 或
./restart_api.sh
```

更改在重启后立即生效。

前端 (.env)

1. 编辑 `OmniCRM-UI/.env`
2. 保存更改
3. 重建 UI:

```
cd OmniCRM-UI
npm run build
```

4. 重启 UI 服务或 Web 服务器

开发模式:

在使用 `npm start` 进行开发时，重启开发服务器以应用更改。

配置最佳实践

安全性

- 切勿提交秘密 - 使用 `.gitignore` 忽略包含凭据的配置文件
- 使用强密码 - 至少 16 个字符，包含大小写、数字和符号
- 定期轮换凭据 - 特别□用于生产部署
- 限制数据库访问 - 使用 IP 白名单和防火墙规则
- 使用特定于环境的配置 - 将开发/暂存/生产配置分开
- 限制 **API** 密钥权限 - 分配最低必要角色
- 谨慎使用 **IP** 白名单 - 优先使用 API 密钥以提高安全性

维护

- 记录更改 - 保持配置修改的变更日志
- 备份配置 - 在重大更改之前存储副本
- 在暂存环境中测试 - 在生产部署之前验证配置更改
- 版本控制 - 在 git 中跟踪配置模板（不包含秘密）

性能

- 使用本地数据库 - 避免远程数据库以提高性能
- 配置缓存 - 如果可用，启用 OCS 缓存
- 优化 **Grafana** - 限制嵌入仪表板的数量

品牌

- 匹配颜色 - 确保 UI 颜色与徽标相辅相成
- 测试对比度 - 验证文本在彩色背景上的可读性
- 移动测试 - 检查移动设备上的品牌
- 徽标位置 - 在不同上下文中使用适当的徽标大小

故障排除

常见问题

更改未应用

- 原因：服务未重启或 UI 未重建
- 修复：在配置更改后重启 API/UI 服务

YAML 语法错误

- 原因：无效的 YAML 格式（缩进、引号等）
- 修复：在线验证 YAML 或使用 `yamllint crm_config.yaml`

数据库连接失败

- 原因：凭据错误或服务器不可达
- 修复：验证数据库是否正在运行，凭据是否正确

Stripe 付款未工作

- 原因：后端和前端之间的密钥不匹配
- 修复：确保 `publishable_key` 在两个文件中匹配

电子邮件未发送

- 原因：无效的 Mailjet 凭据或模板 ID
- 修复：验证 Mailjet API 密钥/秘密，检查模板 ID 是否存在

PDF 生成失败：

- 检查 WeasyPrint 是否已安装：`pip install weasyprint`
- 验证模板文件名是否与 `crm_config.yaml` 匹配
- 检查 `invoice_templates/rendered/` 以获取 HTML 渲染错误
- 查看 API 日志以获取 Jinja2 模板错误

图像未出现在 **PDF** 中：

- 使用绝对文件路径：`file:///full/path/to/image.png`
- 确保图像文件存在并可读
- 检查图像格式（PNG 和 JPEG 最佳）
- 验证图像路径中不包含特殊字符

样式问题：

- 内联所有 CSS（不支持外部样式表）
- 避免复杂的 CSS 特性（flexbox、grid 可能无法正确渲染）
- 首先使用简单布局进行测试，然后逐步增加复杂性
- 尽可能使用表格进行布局而不是 div

缓存未失效：

- 验证在发票修改时是否调用了 `invalidate_invoice_cache()`

- 检查交易更新□否触发缓存失效
- 如有需要，手动从 `Invoice_PDF_Cache` 表中删除

配置清单

在部署 OmniCRM 时使用此清单：

后端配置

- [ ] 将 `.env.example` 复制到 `.env`
- [ ] 设置强数据库密码
- [ ] 配置 CGRates 凭据
- [ ] 更新 `crm_config.yaml` 以包含您的设置：
  - [ ] 数据库连接
  - [ ] 服务类型
  - [ ] Mailjet API 密钥和模板 ID
  - [ ] 供应失败通知电子邮件
  - [ ] 发票模板文件名
  - [ ] CRM 基础 URL（公共可访问）
  - [ ] OCS/CGRates 端点
  - [ ] SMSC 配置
  - [ ] 生成新的 JWT 秘密密钥
  - [ ] Stripe 密钥（生产，不□测试）
  - [ ] API 密钥和 IP 白名单

前端配置

- [ ] 将 `OmniCRM-UI/.env.example` 复制到 `OmniCRM-UI/.env`
- [ ] 设置 Google Maps API 密钥
- [ ] 设置 Stripe 可发布密钥
- [ ] 更新公司品牌：
  - [ ] 公司名称
  - [ ] 门户名称

- [ ] 自助服务名称
- [ ] 公司标语

- [ ] 配置本地化：
  - [ ] 默认语言
  - [ ] 区域
  - [ ] 默[illegible]位置和国家
  - [ ] 货币代码和符号
- [ ] 设置主要品牌颜色
- [ ] 配置 Web 应用集成（可选）
- [ ] 添加支持和常见问题 URL（可选）
- [ ] 设置 Google Analytics/Tag Manager 跟踪 ID（可选）

品牌资产

- [ ] 创建徽标文件（logoSm.png、logoDark.png、logoLight.png）
- [ ] 将徽标上传到 `OmniCRM-UI/src/assets/images/omnitouch/`
- [ ] 创建自定义发票模板 HTML
- [ ] 将发票模板上传到 `OmniCRM-API/invoice_templates/`
- [ ] 更新 `crm_config.yaml` 以包含发票模板文件名
- [ ] 测试发票 PDF 生成
- [ ] 重建 UI：`npm run build`

安全性

- [ ] 更改所有默认密码
- [ ] 生成唯一的 JWT 秘密
- [ ] 使用生产 Stripe 密钥（而不□测试密钥）
- [ ] 轮换 Mailjet API 密钥
- [ ] 启用防火墙规则
- [ ] 配置 API 访问的 IP 白名单
- [ ] 设置 SSL/TLS 证书
- [ ] 为所有端点启用 HTTPS
- [ ] 审查 CORS 设置
- [ ] 实施速率限制

- [ ] 配置备份和恢复程序

测试

- [ ] 测试客户注册流程
- [ ] 测试服务供应的端到端
- [ ] 验证电子邮件通知□否正确发送
- [ ] 测试发票生成和 PDF 下载
- [ ] 验证付款处理（Stripe）
- [ ] 检查用户身份验证和 2FA
- [ ] 测试冒充和审计日志
- [ ] 验证 OCS 的使用数据同步
- [ ] 测试 ActionPlan 创建和续订
- [ ] 确认库存分配正常工作

部署

- [ ] 启动数据库（MySQL）
- [ ] 启动 CGRates
- [ ] 启动 OmniCRM API
- [ ] 启动 OmniCRM UI
- [ ] 配置反向代理（nginx，traefik）
- [ ] 设置监控（Grafana，Prometheus）
- [ ] 配置日志聚合
- [ ] 设置自动备份
- [ ] 培训员工使用系统

相关文档

- `RBAC 和用户管理 </rbac>`
- `产品和服务 </concepts_products_and_services>`
- `Ansible 供应 </concepts_ansible>`

- 库存管理 `</administration_inventory>`
- □户发票 `</payments_invoices>`
- 双因素身份验证 `</2fa>`
- □户关怀和冒充 `</customer_care>`
- 支付供应商集成 `<integrations_payment_vendors>`
- `administration_api_keys` - API 密钥管理
- `integrations_mailjet` - 电子邮件集成
- `concepts_api` - API 身份验证

OmniCRM中的库存概述

OmniCRM中的库存系统旨在管理和跟踪网络运营商和客户使用的物理和虚拟物品。

这意味着我们可以跟踪各种物品，例如调制解调器、电话号码、IP地址块，甚至GPON ONT或固定无线CPE等物理硬件。

另请参见：`Customer Attributes <administration_attributes>` 用于存储自定义元数据，以及 `Customer Tags <administration_tags>` 用于视觉分类。

为了支持具有固定网络覆盖范围的客户，库存系统还可以跟踪某项服务所经过的家庭，使运营商能够远程进行服务资格验证；对于那些运营固定无线网络的公司，我们可以跟踪在现场部署的CPE及其位置。

注意

在配置过程中，库存项目通过 `inventory_items_list` 字段与产品关联。有关库存如何与产品配置集成的完整流程，包括库存选择器UI和Ansible剧本集成，请参见 `Complete Product Lifecycle Guide - Inventory Requirements <guide_product_lifecycle>`。

目的

OmniCRM库存有几个关键目的：

1. 配置服务：当客户注册服务时，可能需要分配调制解调器、SIM卡或电话号码等物品。库存系统跟踪这些物品并将其与客户关联。
2. 库存管理：对于物理库存，例如硬件或其他设备，库存帮助运营商保持对可用物品的可见性，了解其存储位置以及已分配或出售给客户的物品。
3. 客户分配：该系统允许将物品分配给客户，无论用于服务（例如，将调制解调器分配给客户的互联网账户）还直接销售。
4. 服务资格/网络覆盖范围：通过存储网络覆盖范围的信息，例如每个经过GPON服务的家庭或每个已部署的固定无线CPE，使员工能够远程进行服务资格验证，并查看特定区域否有网络覆盖。

示例库存生命周期

为了说明库存系统的工作原理，请考虑以下示例

SIM卡示例

从Omnitouch订购了一批1000个SIM卡。首先为SIM卡创建一个库存模板（如果尚不存在），并将订购的SIM卡加载到状态为运输中的库存中。

一旦收到SIM卡，它们将标记为库存中，并可以分配给不同的零售店，库存项目的位置将更新以反映每个SIM卡所在的零售店 - 这对于跟踪每个���店的库存水平以及确保每个商店有足够的库存以满足客户需求非常有用。

当客户在店内注册移动服务时，将SIM卡分配给客户，状态更改为已分配。然后，SIM卡库存项目被分配给客户，状态更新为使用中。

如果客户取消服务或服务处于休眠状态，SIM卡将标记为退役。

GPON经过的家庭示例

对于GPON网络，库存系统可以跟踪每个经过特定服务的家庭。

当新区域建设完成时，可以将每个经过的地址添加到库存中。

这使运营商能够查看哪些家庭经过特定服务，哪些家庭尚未经过。

当客户注册服务时，OmniCRM可以自动对客户的地址进行服务资格验证，以查看该地址□否在经过的家庭库存中，以及可以提供哪些服务。

库存模板

InventoryTemplate功能允许创建任意数量的具有预定义字段的物品类型。这些模板充当蓝图，定义不同物品的基本特征，例如：

- 调制解调器，具有MAC地址（`itemtext1`）和序列号（`itemtext2`）。
- 经过的家庭，具有位置和状态（例如，经过或未经过）。
- 电话号码，具有主要号码（`itemtext1`）和地理位置��`itemtext2`）。
- 虚拟资源，如IP地址块，通过模板映射的标识符。

每个库存模板定义最多20个可自定义文本字段（`itemtext1`到`itemtext20`），以及相应的标签（`itemtext1_label`到`itemtext20_label`），描述每个字段所代表的内容。例如，调制解调器模板可能将`itemtext1_label`设置为“MAC地址”，将`itemtext2_label`设置为“序列号”。

运营商可以使用**InventoryTemplates**自定义每种物品类型的字段。这些模板允许以结构化的方式对物品进行分类和管理，确保物品跟踪的一致性。

链接到产品：

库存模板名称在产品定义中通过`inventory_items_list`字段引用。当配置产品时，系统显示一个库存选择器，仅显示与所需模板类型匹配的物品。

示例：一个产品具有`inventory_items_list: "['SIM Card', 'Mobile Number']"`，需要存在两个名称完全相同的库存模板“SIM卡”和“移动号码”。模板名称区分大小写。

有关库存模板如何与产品配置连接的完整详细信息，请参见`Product Lifecycle - Inventory Requirements <guide_product_lifecycle>`。

通过**UI**创建库存模板

要创建新的库存模板：

1. 从主菜单导航到 ���存 → 模板
2. 点击 添加模板 按钮
3. 填写所需字段：

   基本信息：

   - 图标（可选）：用于视觉识别的图标类名称（例如，`fa-solid fa-sim-card`）
   - 物品（必填）：模板名称（必须与产品的`inventory_items_list`中使用的名称完全匹配）

   成本信息（必填）：

   - 批发成本（必填）：您购买或配置此物品类型的成本

- □□□□

常见模板示例

SIM卡模板：

- 物品：“SIM卡”
- 批发成本：2.50
- 零售成本：10.00
- 物品文本1标签：“ICCID”
- 物品文本2标签：“IMSI”
- 物品文本3标签：“SIM类型”（物理/eSIM）

移动号码模板：

- ���品：“移动号码”
- 批发成本：1.00
- 零售成本：0.00
- 物品文本1标签：“电话号码”
- 物品文本2标签：“地理区域”
- 物品文本3标签：“号码类型”（移动/固定电话）

固定无线**CPE**模板：

- 物品：“固定无线CPE”
- 批发成本：250.00
- 零售成本：450.00
- 物品文本1标签：“MAC地址”
- 物品文本2标签：“序列号”
- 物品文本3标签：“固件版本”
- 物品文本4标签：“制造商”
- 物品文本5标签：“型号”

GPON ONT模板：

- 物品：“GPON ONT”
- 批发成本：45.00
- 零售成本：0.00（包含在服务中）

- 物品文本1标签：“序列号”
- 物品文本2标签：“MAC地址”
- 物品文本3标签：“PON位置”
- 物品文本4标签：“型号”

创建和管理库存项目

一旦定义了**InventoryTemplate**，可以创建单个库存项目。每个库存项目代表特定物品类型的具体实例（例如，特定的调制解调器或电话号码），可以：

- 分配给客户：物品与客户关联以进行服务配置（例如，为互联网连接分配硬件）。
- 跟踪库存：运营商可以监控可用库存，例如未售出或未分配的物品。
- 出售或退役：一旦出售，物品将标记相关时间戳（例如，`sold_date`），并且不再被视为可用库存。

通过该系统，OmniCRM促进了高效的库存管理，帮助将资源分配给客户，并提供每个项目状态和历史的详细可见性。

服务可以与特定的库存项目链接，从而轻松跟踪哪些项目与哪些客户或服务相关联。

一旦库存项目分配给客户，Ansible剧本可以更新项目的状态和历史，以反映分配。这确保运营商准确记录哪些项目正在使用，哪些项目可供分配，以及知道哪个客户在使用哪个项目。

我们可以在库存选项卡的客户资料页面中查看分配给客户的项目。

□□□

库存项目字段

每个库存项目包含组织成几个类别的全面信息：

基本项目信息

- **inventory_id** - 库存项目的唯一标识符（自动生成）
- **item** - 物品类型（与库存模板名称匹配，例如，“SIM卡”、“调制解调器”、“电话号码”）
- **inventory_template_id** - 链接到定义此物品类型的库存模板
- **customer_id** - 如果分配给客户，则为客户的ID（可为空）
- **service_id** - 如果链接到特定服务，则为服务ID（可为空）
- **item_location** - 物品的物理或逻辑位置：
  - 对于物理物品：建筑物、仓库、货架位置、商店名称等。
  - 对于虚拟物品：地理位置、IP块位置、号码范围区域等。
- **item_state** - 库存项目的当前状态（枚举值）：
  - 新 - 全新、未使用的物品
  - 已使用 - 以前使用过但功能正常
  - 内部使用 - 分配给内部测试或员工使用
  - 已分配 - 当前分配给客户或服务
  - 损坏 - 无功能，需要修理或处置
  - 停用 - 暂时不可用
  - 丢失 - 找不到物品
  - 被盗 - 物品被盗

可自定义项目字段（来自模板）

库存系统支持多达20个可自定义文本字段，其含义由库存模板定义：

- **itemtext1** - 第一个可自定义字段（必填，标签由模板的`itemtext1_label`定义）
  - 示例：对于调制解调器，可能□“MAC地址”
  - 示例：对于SIM卡，可能□“ICCID”
  - 示例：对于电话号码，可能□“电话号码”
- **itemtext2**到**itemtext20** - 额外的可自定义字段（可选，标签由模板定义）
  - 示例：调制解调器的itemtext2可能□“序列号”
  - 示例：SIM卡的itemtext2可能□“IMSI”

    - 示例：调制解调器的itemtext3可能“固件版本”

每个库存模板定义哪些字段被使用以及它们通过相应的标签字段（`itemtext1_label`、`itemtext2_label`等）表示的内容。

成本信息

- **wholesale_cost** - 您购买/配置此物品的成本（浮动）
- **retail_cost** - 如果单独出售给客户的价格（浮动）
- **sold_date** - 物品出售或分配给客户的时间戳

物理地址（用于网络设备和站点）

用于跟踪物理部署位置，特别固定网络设备（CPE、ONT、调制解调器）或经过的家庭：

- **address_line_1** - 街道地址、建筑物编号、单元编号
- **address_line_2** - 额外的地址信息（套房、公寓、楼层）
- **city** - 城市或城镇
- **state** - 州、省或地区
- **zip_code** - 邮政编码/邮政区号
- **country** - 国家名称

地理位置（从**Web UI**自动填充）

通过Web UI使用地址自动完成创建库存项目时，这些字段会自动填充：

- **google_maps_place_id** - 地址的Google地图地点ID
- **plus_code** - Google地图加号代码（放位置代码），用于精确位置
- **latitude** - 地理纬度（十进制度）
- **longitude** - 地理经度（十进制度）

这些字段使得：

- 在地图视图上绘制库存位置
- 进行服务资格的接近计算
- 网络规划的覆盖分析
- 现场技术人员调度的路线优化

设备管理和访问**URL**

`management_url`字段提供快速访问设备接口和配置URL:

- **management_url** - 库存项目的访问URL
  - 网络设备: Web界面URL（例如，`https://192.168.1.1`用于路由器、交换机、ONT、CPE）
  - **eSIM**配置: eSIM激活的LPA（本地配置助手）地址（例如，`LPA:1$smdp.example.com$ACTIVATION-CODE-HERE`）
  - 其他用例: 任何需要通过移动设备轻松访问的URL

  二维码生成

  当查看具有`management_url`的库存项目时，系统会自动生成一个可扫描的二维码:

  - 库存项目详细视图: 128x128的二维码与URL并排显示
  - 服务库存表: 为分配的项目显示64x64的二维码
  - 格式: 二维码和可点击的超链接一起显示

  常见用例

  - 网络技术人员: 扫描二维码以访问设备管理界面，而无需输入IP地址
  - **eSIM**激活: 客户扫描CRM中的二维码以安装eSIM配置
  - 客户自助服务: 提供轻松访问设备配置或客户门户的方式
- **management_username** - 设备访问的管理员用户名
- **management_password** - 设备访问的管理员密码（加密存储）

配置管理

对于具有配置文件的设备:

- **config_content** - 完整的配置文件内容（以文本形式存储）
  - 适用于备份、版本控制和灾难恢复
  - 可以存储路由器配置、交换机配置、CPE配置等。
- **config_file_path** - 如果单独存储，外部配置文件的路径
  - 作为在数据库中存储完整配置的替代方案
  - 网络共享、版本控制库或配置管理系统的路径

备注和元数据

- **inventory_notes** - 关于库存项目的自由格式备注
  - 安装备注
  - 维护历史
  - 特殊处理要求或特性
  - 供应商信息
  - 保修详情
- **created** - 库存项目在系统中创建的时间戳（自动设置）
- **last_modified** - 库存项目最后更新的时间戳（自动更新）

字段使用示例

示例**1**：移动**SIM**卡

```
{
  "inventory_id": 1001,
  "item": "SIM Card",
  "inventory_template_id": 5,
  "itemtext1": "89612345678901234567",
  "itemtext2": "310120123456789",
  "itemtext3": "Physical",
  "item_location": "Warehouse A, Shelf 3",
  "item_state": "Assigned",
  "customer_id": 456,
  "service_id": 789,
  "wholesale_cost": 2.50,
  "retail_cost": 10.00,
  "sold_date": "2025-01-15T10:30:00Z",
  "inventory_notes": "Activated on 2025-01-15"
}
```

示例**2**：移动**eSIM**配置

```
{
  "inventory_id": 1002,
  "item": "eSIM",
  "inventory_template_id": 6,
  "itemtext1": "89612345678901234 57",
  "itemtext2": "310120123456790",
  "itemtext3": "eSIM",
  "item_location": "Virtual Inventory",
  "item_state": "Assigned",
  "customer_id": 457,
  "service_id": 790,
  "management_url": "LPA:1$smdp.example.com$ACTIVATION-CODE-
ABC123XYZ",
  "wholesale_cost": 0.00,
  "retail_cost": 0.00,
  "sold_date": "2025-01-16T14:20:00Z",
  "inventory_notes": "eSIM profile ready for activation"
}
```


查看此eSIM库存项目时，UI显示一个包含LPA地址的二维码。客户使用移动设备扫描此二维码以安装eSIM配置。

示例**3**：客户驻地设备（**CPE**） **-** 固定无线

```
{
  "inventory_id": 2001,
  "item": "Fixed Wireless CPE",
  "inventory_template_id": 10,
  "itemtext1": "AA:BB:CC:DD:EE:FF",
  "itemtext2": "FW2024-12345",
  "itemtext3": "v2.4.1",
  "itemtext4": "Ubiquiti",
  "itemtext5": "LiteBeam AC Gen2",
  "item_location": "Customer Site",
  "item_state": "Assigned",
  "customer_id": 789,
  "service_id": 1234,
  "address_line_1": "123 Main Street",
  "address_line_2": "Apt 4B",
  "city": "Sydney",
  "state": "NSW",
  "zip_code": "2000",
  "country": "Australia",
  "latitude": "-33.8688",
  "longitude": "151.2093",
  "management_url": "https://192.168.100.1",
  "management_username": "admin",
  "management_password": "encrypted_password_here",
  "config_file_path": "/configs/cpe/fw2024-12345.conf",
  "inventory_notes": "Installed 2025-01-10. Customer reports
excellent signal. Pointing: Azimuth 45°, Elevation 15°"
}
```


□□4□GPON ONT□□□□□

```
{
  "inventory_id": 3001,
  "item": "GPON ONT",
  "inventory_template_id": 15,
  "itemtext1": "ALCL12345678",
  "itemtext2": "AA:BB:CC:DD:EE:FF",
  "itemtext3": "OLT-1, PON 3, ONT 42",
  "itemtext4": "Nokia G-010G-A",
  "item_location": "Customer Premises",
  "item_state": "Assigned",
  "customer_id": 321,
  "service_id": 654,
  "address_line_1": "456 Fiber Lane",
  "city": "Melbourne",
  "state": "VIC",
  "zip_code": "3000",
  "country": "Australia",
  "google_maps_place_id": "ChIJ1234567890",
  "plus_code": "4RRH+2C Melbourne VIC",
  "latitude": "-37.8136",
  "longitude": "144.9631",
  "management_url": "https://192.168.1.1",
  "management_username": "admin",
  "config_content": "# ONT Configuration\nwlan-ssid:
HomeNetwork\nwlan-password: encrypted...",
  "wholesale_cost": 45.00,
  "retail_cost": 0.00,
  "inventory_notes": "Provisioned 2025-01-20. Optical power:
-22dBm"
}
```


注意

当查看具有`management_url`的库存项目（如示例2、3和4）时，UI会自动显示：

- 一个包含URL或LPA地址的可扫描二维码
- 一个可点击的超链接（对于Web URL）

用例：

- **eSIM**激活（示例2）：客户扫描二维码以在其设备上安装eSIM配置

- 网络设备访问（示例3和4）：技术人员扫描以访问设备管理界面，而无需手动输入IP地址

示例**5**：电话号码（虚拟库存）

```
{
  "inventory_id": 4001,
  "item": "Phone Number",
  "inventory_template_id": 20,
  "itemtext1": "+61412345678",
  "itemtext2": "Melbourne",
  "itemtext3": "Mobile",
  "item_location": "Australia - VIC",
  "item_state": "Assigned",
  "customer_id": 555,
  "service_id": 888,
  "wholesale_cost": 1.00,
  "retail_cost": 0.00,
  "inventory_notes": "Ported from Telstra on 2025-01-05"
}
```

库存项目状态解释

`item_state`字段跟踪库存项目的生命周期：

- 新 → 已分配 - 正常流向客户配置
- 已分配 → 已使用 - 服务停用后，物品可以重新使用
- 新 → 内部使用 - 分配用于测试、演示或员工使用
- 已分配 → 损坏 - 物品故障，需要RMA或处置
- 任何状态 → 丢失 - 找不到物品（触发审计）
- 任何状态 → 被盗 - 物品被盗（触发安全报告）
- 损坏/已使用 → 新 - 经过翻新或修理后

按状态过滤和搜索库存允许运营商：

- 跟踪可用库存（新物品）
- 确定分配给客户的物品（已分配）
- 查找可供重新使用的物品（已使用）
- 监控设备问题（损坏、停用）
- 审计缺失物品（丢失、被盗）

□□□□

□□□

- ◆◆◆签文本
- 激活日期（标签可见的时间）
- 失效日期（标签将被隐藏的时间）
- 链接（如果已配置）

创建新标签

要为客户创建新标签：

1. 导航到客户的概览页面
2. 点击 标签 选项卡
3. 点击 添加标签 按钮
4. 填写必填字段：
   - 标签文本（必填）：将在标签上显示的文本
   - 标签颜色（必填）：使用颜色选择器选择颜色或输入十六进制代码
   - 标签链接（可选）：点击标签时将打开的 URL
   - 激活日期（必填）：标签应开始显示的日期
   - 失效日期（必填）：标签应停止显示的日期（默认为 2099-01-01）
5. 在预览部分预览您的标签
6. 点击 创建标签

编辑标签

要编辑现有标签：

1. 导航到客户的概览页面
2. 点击 标签 选项卡
3. 在列表中找到您要编辑的标签
4. 点击 编辑（铅笔）按钮
5. 根据需要修改字段
6. 点击 更新标签

删除标签

要删除标签：

1. 导航到客户的概览页面
2. 点击 标签 选项卡
3. 在列表中找到您要删除的标签
4. 点击 删除（垃圾桶）按钮
5. 在弹出窗口中确认删除

标签字段参考

API 集成

标签也可以通过 API 以编程方式进行管理：

创建标签：

```
PUT /crm/tag/
{
  "tag_text": "VIP Customer",
  "tag_hex_color": "FFD700",
  "tag_link": "https://example.com/vip",
  "tag_active_date": "2025-01-01 00:00:00",
  "tag_deactivate_date": "2099-12-31 23:59:59",
  "customer_id": 12
}
```

更新标签:

```
PATCH /crm/tag/tag_id/{tag_id}
{
  "tag_text": "Updated Tag Text",
  "tag_hex_color": "FF0000"
}
```

按客户获取标签:

```
GET /crm/tag/customer_id/{customer_id}
```

删除标签:

```
DELETE /crm/tag/tag_id/{tag_id}
```

OmniCRM 系统架构

本文档提供了 OmniCRM 系统架构的概述，包括组件关系和数据流。

高级系统概述

OmniCRM 是一个综合的 BSS/OSS 平台，集成了多个关键组件，以为电信提供商提供完整的服务管理。

核心组件

1. 前端应用

OmniCRM Web UI

- React 单页应用
- 客户管理、服务配置、计费的员工界面
- 实时配置状态更新
- 基于角色的访问控制

自助服务门户

- 面向客户的门户
- 服务管理和使用跟踪
- 发票查看和支付
- 与员工 UI 共享代码库，不同视图

2. API 层

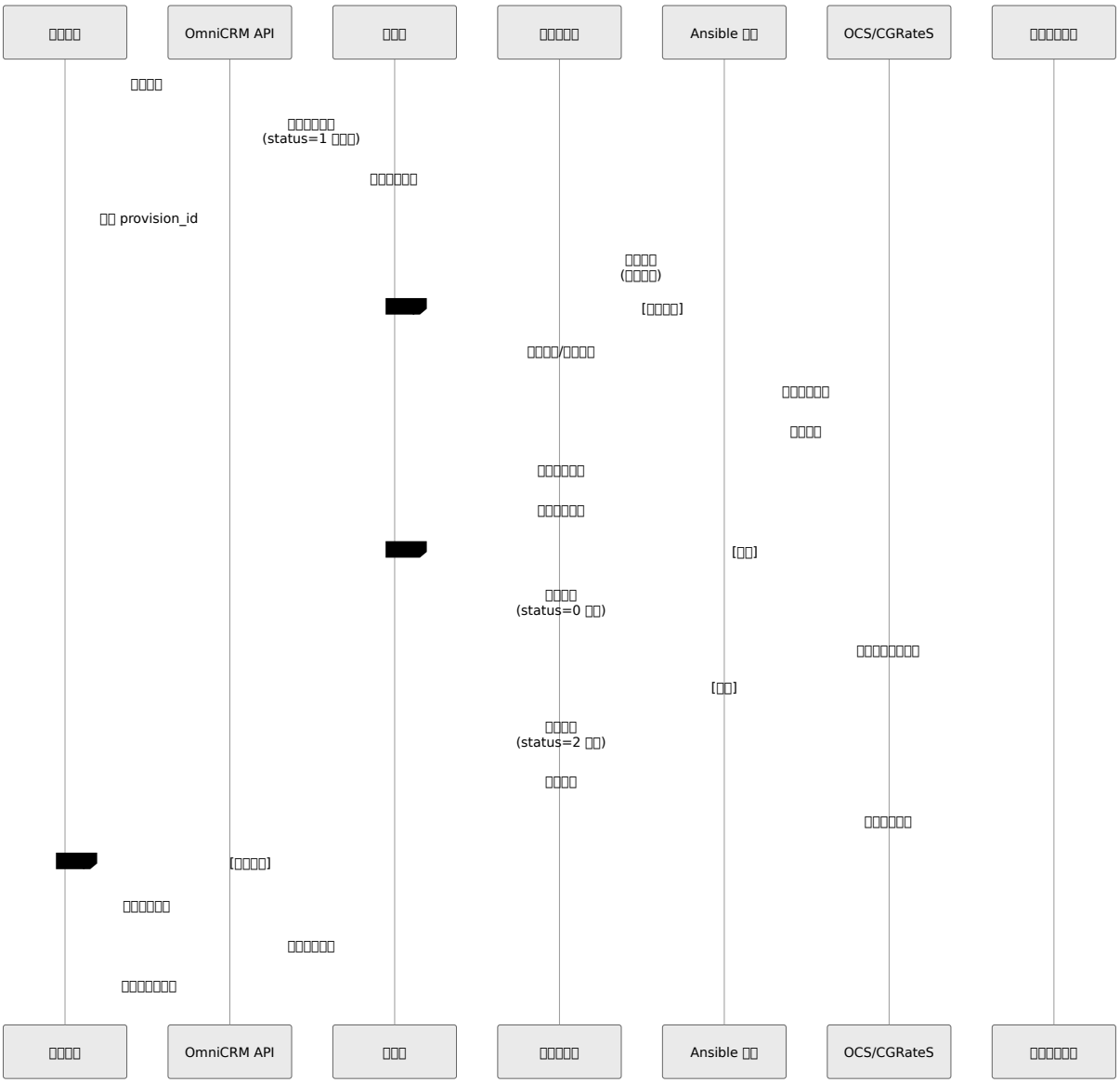
OmniCRM API (Flask/Python)

- 所有操作的 RESTful API
- OpenAPI/Swagger 文档
- 基于 JWT 的认证
- 速率限制和缓存
- WebSocket 支持实时更新

关键 **API** 路由**:**

- `/crm/customer/*` - 客户管理
- `/crm/service/*` - 服务操作
- `/crm/product/*` - 产品目录
- `/crm/provision/*` - 配置操作
- `/crm/transaction/*` - 计费交易
- `/crm/invoice/*` - 发票管理

3. ���□□□

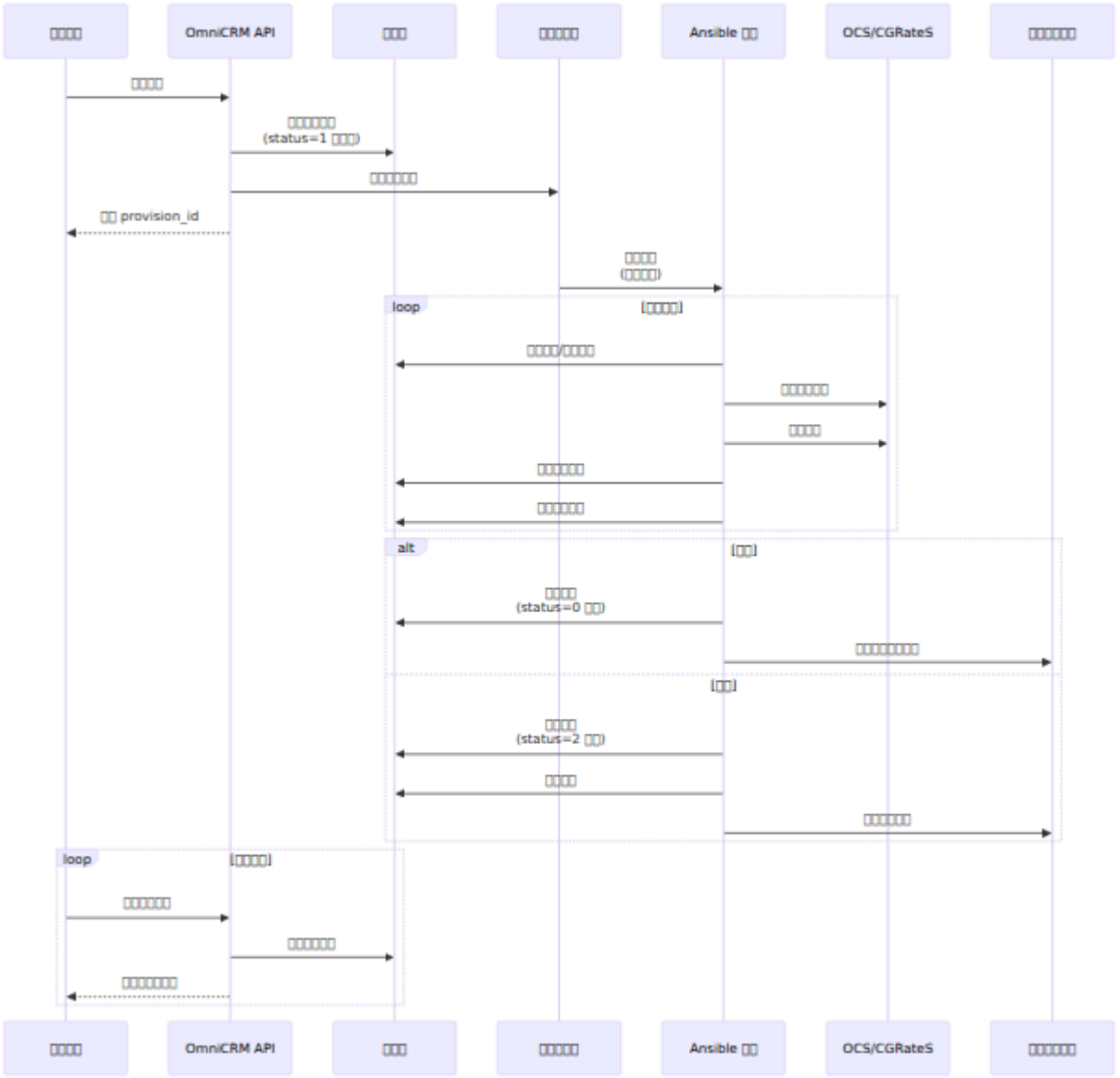


4. □□□□□

OCS/CGRateS □□

- □□□□□□□□
- □□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□
- □□□□□□□□□

□□□□□□:


OmniCRM API
Ansible
OCS/CGRateS
loop
alt
loop
OmniCRM API
Ansible
OCS/CGRateS

5. □□□□

OmniCRM □□□□

字段	类型	描述
customer_id	整数 (PK)	唯一标识符
customer_name	字符串	公司或个人名称
customer_account_type	枚举	'个人' 或 '企业'
customer_status	枚举	'开放', '关闭', '暂停', '归档'
customer_payment_type	字符串	'预付' 或 '后付'
customer_enabled	布尔	账户□否活跃
tax_identifier	字符串	VAT/GST 号码
contract_start_date	日期时间	合同开始
contract_end_date	日期时间	合同结束

联系人 - ���客户相关联的人员

字段	类型	描述
contact_id	整数 (PK)	唯一标识符
customer_id	整数 (FK)	父客户
contact_firstname	字符串	名字
contact_lastname	字符串	姓氏
contact_email	字符串	电子邮件地址
contact_phone	字符串	电话号码
contact_types	字符串	'主要', '计费', '技术'

站点 - 实际服务交付地点

字段	类型	描述
site_id	整数 (PK)	唯一标识符
customer_id	整数 (FK)	父客户
site_name	字符串	位置名称
address_line_1	字符串	街道地址
city, state, zip_code	字符串	位置详情
latitude, longitude	浮点	GPS 坐标
google_maps_place_id	字符串	Google Maps 参考
plus_code	字符串	开放位置代码

服务与产品模型

服务 - 活动服务实例

字段	类型	描述
service_id	整数 (PK)	唯一标识符
customer_id	整数 (FK)	父客户
product_id	整数 (FK)	产品模板
site_id	整数 (FK)	服务位置
service_name	字符串	显示名称
service_uuid	字符串	计费系统标识符
service_status	枚举	当前状态
service_billed	布尔	生成交易
wholesale_cost	浮点	供应商成本
retail_cost	浮点	客户价格
bundled_parent	整数 (FK)	捆绑服务的父服务

产品 - 服务提供模板

字段	类型	描述
product_id	整数 (PK)	唯一标识符
product_name	字符串	显示名称
product_slug	字符串	URL 友好名称
category	枚举	'独立', '捆绑', '附加', '促销'
provisioning_play	字符串	Ansible 剧本名称
provisioning_json_vars	JSON	剧本变量
inventory_items_list	字符串	所需库存
retail_cost	浮点	月价格
retail_setup_cost	浮点	一次性费用
enabled	布尔	可供销售

计费模型

交易 - 单个收费/信用

字段	类型	描述
transaction_id	整数 (PK)	唯一标识符
customer_id	整数 (FK)	父客户
invoice_id	整数 (FK)	父发票（可选）
service_id	整数 (FK)	相关服务
title	字符串	交易描述
retail_cost	浮点	金额
tax_percentage	浮点	税率
tax_amount	浮点	计算的税
void	布尔	取消的交易

发票 - 分组交易以进行计费

字段	类型	描述
invoice_id	整数 (PK)	唯一标识符
customer_id	整数 (FK)	父客户
paid	布尔	收到付款
void	布尔	取消的发票
payment_reference	字符串	Stripe 交易 ID
start_date, end_date	日期	计费周期
due_date	日期	付款截止日期
retail_cost	浮点	总金额

库存模型

库存 - 实体和虚拟资产

字段	类型	描述
inventory_id	整数 (PK)	唯一标识符
customer_id	整数 (FK)	指定客户
service_id	整数 (FK)	关联服务
inventory_template_id	整数 (FK)	项目类型模板
item	字符串	项目类型（SIM 卡、路由器等）
item_state	枚举	'新', '已分配', '已使用' 等
itemtext1-20	字符串	灵活字段
management_url	字符串	设备管理 URL
config_content	文本	配置文件

库存模板 - 定义库存项目结构

字段	类型	描述
inventory_template_id	整数 (PK)	唯一标识符
item	字符串	模板名称
itemtext1_label	字符串	itemtext1 字段的标签
itemtext2_label	字符串	itemtext2 字段的标签

配置模型

配置 - 配置任务跟踪

字段	��型	描述
provision_id	整数 (PK)	唯一标识符
product_id	整数 (FK)	正在配置的产品
customer_id	整数 (FK)	目标客户
service_id	整数 (FK)	创建/修改的服务
provisioning_play	字符串	Ansible 剧本名称
provisioning_status	整数	0=成功, 1=运行中, 2=失败

配置事件 - 单个配置步骤

字段	类型	描述
provision_event_id	整数 (PK)	唯一标识符
provision_id	整数 (FK)	父配置任务
event_name	字符串	任务名称
event_number	整数	序列号
provisioning_status	整数	0=成功, 1=运行中, 2=失败
provisioning_result_json	JSON	完整任务输出

用户与安全模型

用户 - 用户账户

字段	类型	描述
id	整数 (PK)	唯一标识符
username	字符串	登录用户名
email	字符串	电子邮件地址
email_verified	布尔	电子邮件已确认
is_2fa_enabled	布尔	启用双因素认证
totp_secret	字符串	TOTP 秘钥

角色 - 用户角色

字段	类型	描述
id	整数 (PK)	唯一标识符
name	字符串	角色名称
description	字符串	角色描述

权限 - 细粒度权限

字段	类型	描述
id	整数 (PK)	唯一标识符
name	字符串	权限名称（资源.动作）
description	字符串	权限描述

关系：

- 用户有多个角色（多对多）
- 角色有多个权限（多对多）
- 用户可以链接到一个联系人（用于客户门户访问）

集成点

Stripe 支付网关

- 令牌化支付方式
- 符合 PCI 的卡存储
- 自动发票支付
- 退款处理
- 卡过期通知

Mailjet 电子邮件服务

- 事务性电子邮件（发票、欢迎、通知）
- 联系人同步
- 模板管理
- 投递跟踪

Google Maps

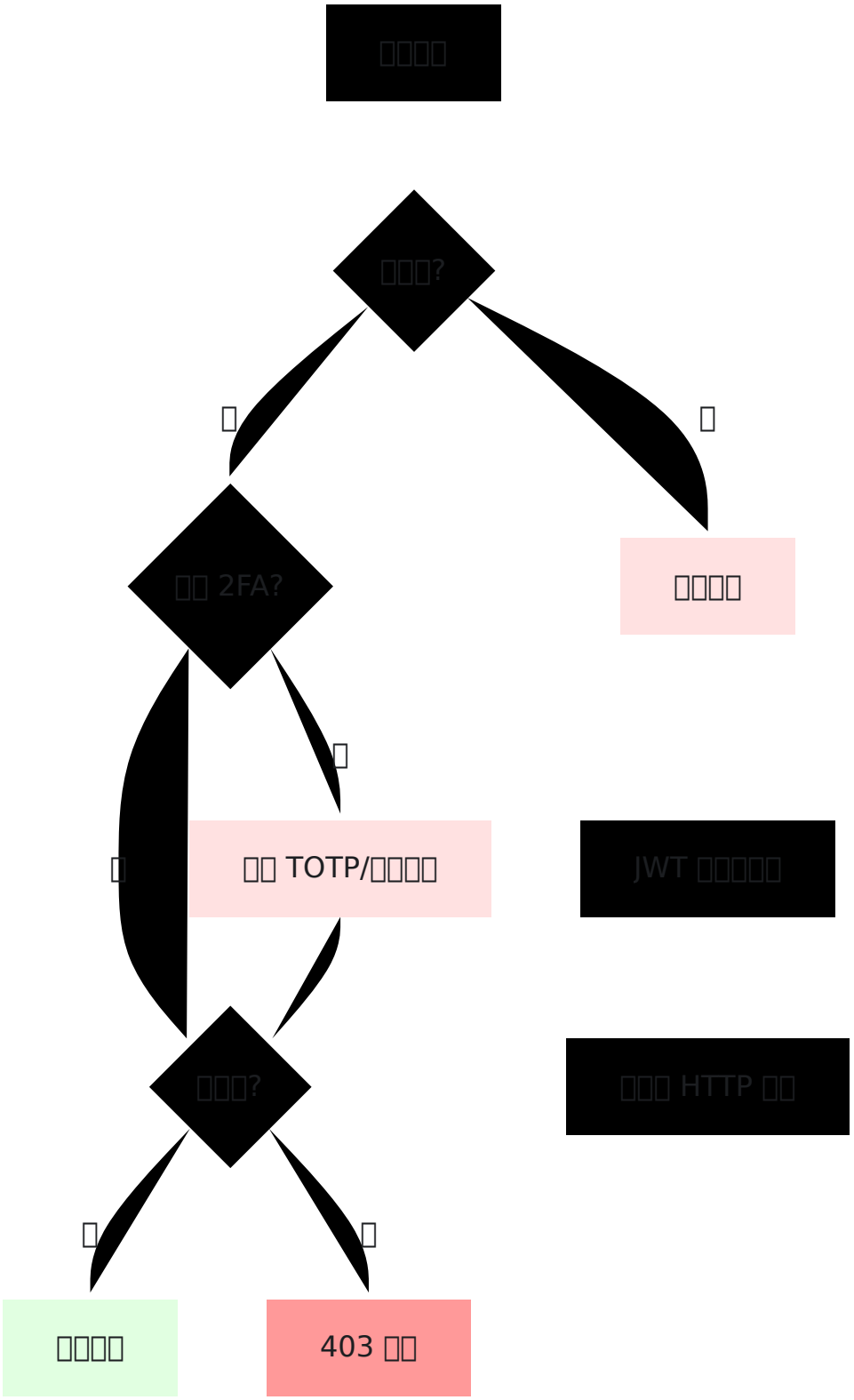
- 地址自动补全
- 地理编码和反向地理编码
- Plus Code 生成
- 站点位置映射

OCS/CGRateS

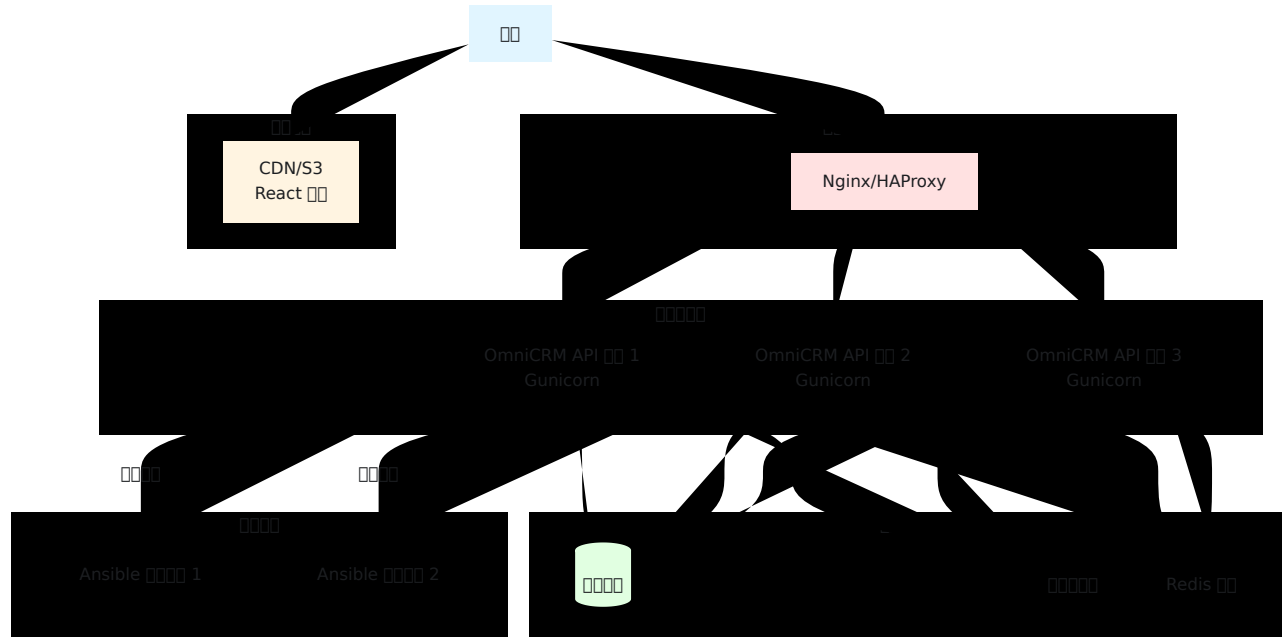
- 账户配置
- 实时评级
- 余额管理
- CDR 处理

- □□□□□□□□□□

□□□□


□□□□
□□□?
□
□
□□ 2FA?
□□□□
□
□□ TOTP/□□□□
JWT □□□□□
□□□?
□□□ HTTP □□
□
□
□□□□
403 □□

□□□□:



- 基于 JWT 的认证
- 双因素认证（TOTP）
- 基于角色的访问控制（RBAC）
- 基于权限的资源访问
- 使用 Redis 进行会话管理
- 密码哈希（bcrypt）
- 电子邮件验证
- 通过活动日志进行审计记录

部署架构

推荐的生产设置：

技术栈

后端：

- Python 3.x
- Flask 网络框架
- SQLAlchemy ORM
- Alembic 迁移

- Ansible 用于自动化

前端：

- React
- 状态管理
- React Router
- Axios 用于 API 调用

数据库：

- 关系数据库（主要数据存储）
- Redis（缓存和会话）

外部服务：

- CGRateS（计费/评级）
- Stripe（支付）
- Mailjet（电子邮件）
- Google Maps（地理编码）

可扩展性考虑

水平扩展：

- 无状态 API 设计允许多个实例
- 负载均衡器分配请求
- Redis 用于共享会话状态

数据库扩展：

- 读取副本用于报告查询
- 连接池
- 查询优化和索引

大规模配置：

- 后台任务处理
- 多个 Ansible 工作进程
- 任务队列管理
- 失败配置的重试逻辑

监控与可观察性

OmniCRM 提供全面的基于 Prometheus 的指标，用于监控系统的各个方面。有关完整细节，请参见 监控与指标。

关键指标：

- API 响应时间和请求速率
- 配置成功/失败率和任务持续时间
- 数据库查询性能和连接池使用
- 外部集成健康（OCS、Stripe、Mailjet）
- 后台任务执行和错误

日志记录：

- 应���日志（Flask）
- 配置日志（Ansible 输出）
- 活动日志（审计追踪）
- 错误跟踪和警报

指标端点：所有指标都在 `/crm/metrics` 以 Prometheus 格式公开。有关抓取配置、警报规则和仪表板示例，请参见 监控与指标。

相关文档

- Ansible 剧本 - 配置自动化
- 配置系统 - 工作流细节
- 产品和服务 - 产品架构
- API 文档 - API 参考
- RBAC - 安全和权限

认证流程和管理员控制

OmniCRM 提供全面的认证功能，包括登录、双因素认证 (2FA)、密码管理和用于管理用户安全的管理员控制。此指南专注于最终用户和管理员的 UI 工作流程。

另请参见：`Self-Care Portal <self_care_portal>` 用于客户登录和门户访问，`RBAC <rbac>` 用于员工权限。

概述

OmniCRM 认证包括：

- 电子邮件/密码登录 - 标准凭据基础的认证
- 双因素认证 **(2FA)** - 可选的基于 TOTP 的第二因素
- 记住我 - 延长会话时间最多 30 天
- 密码重置 - 通过电子邮件自助服务密码恢复
- 管理员控制 - 用于重置 2FA 和密码的管理工具
- 社交登录 - 可选的 Google、Apple、Facebook 集成（如果启用）
- 基于角色的导航 - 根据用户角色自动路由

登录流程

登录页面□所有用户（员工和客户）的入口点。

标准登录

登录过程：

1. 输入 电子邮件地址（员工或客户电子邮件地址）
2. 输入 密码
3. 可选：勾选 **“**记住我 **30** 天**”** 以延长会话
4. 点击 **“**登录**”**

接下来发生的事情：

- 没有 **2FA**：用户立即登录，根据角色导航：
  - 客户 → 自助服务门户 (`/self-care`)
  - 员工/管理员 → 客户仪表板 (`/customers`)
  - **CBC** 模式 → 单元广播界面 (`/create-cell-broadcast`)
- 启用 **2FA**：重定向到 2FA 验证屏幕

记住我功能：

启用时，会话持续 **30** 天，而不在浏览器关闭时过期。使用安全的 HTTP-only cookies。

显示/隐藏密码：

点击 眼睛图标 **(**) 切换密码可见性。

使用 **2FA** 登录

如果用户启用了 2FA，在输入电子邮件/密码后，2FA 挑战屏幕会出现：

使用身份验证器代码：

1. 打开身份验证器应用程序（Google Authenticator、Authy 等）
2. 找到 OmniCRM 条目
3. 输入 6 位数字代码
4. 输入所有 6 位数字时，代码会自动提交
5. 如果有效，用户登录并导航到相应的仪表板

使用恢复代码：

如果身份验证器应用程序不可用：

1. 点击 **“**恢复代码**”** 选项卡
2. 输入您保存的备份代码之一（例如，`3fa5b9c2`）
3. 点击 **“**验证**”**
4. 代码被消耗（只能使用一次）

取消：

点击 **“**取消**”** 返回登录页面。

社交登录（可选）

如果启用（`VITE_ALLOW_SOCIAL_LOGINS=yes`），社交登录按钮会出现：

> [使用 Google 登录] [● 使用 Apple 登录] [使用 Facebook 登录]

点击任何按钮通过该提供商进行身份验证。当前显示“即将推出”消息（社交登录实现中）。

忘记密码链接

点击 **“忘记密码？”** 链接以启动密码重置流程。

双因素认证（2FA）设置

用户可以启用 2FA 以增强帐户安全性。2FA 使用 TOTP（基于时间的一次性密码），与标准身份验证器应用程序兼容。

访问 **2FA** 设置

从用户个人资料或设置中：

客户注意事项：

客户角色用户不会看到 2FA 提示。2FA 通常仅对员工和管理用户要求。

第一步：确认密码

> 当前密码
>
> [取消] [继续]

输入您的当前密码以继续。这在启用 2FA 之前验证您的身份。

第二步：扫描二维码

> [取消] [确认]

设置说明：

1. 下载身份验证器应用程序（如果您没有）：
    - iOS: Apple App Store → “Google Authenticator”
    - Android: Google Play → “Google Authenticator”
    - 替代��案: Authy、Microsoft Authenticator、1Password
2. 扫描二维码：
    - 打开身份验证器应用程序
    - 点击“+”或“添加帐户”
    - 选择“扫描二维码”
    - 将相机对准屏幕上的二维码
    - 应用程序添加“OmniCRM”条目，带有 6 位数字代码
3. 保存备份代码：
    - 重要：记录或复制这 8 个代码
    - 存放在安全位置（密码管理器、安全箱等）
    - 每个代码仅限一次使用
    - 如果您失去对身份验证器应用程序的访问，将使用
    - 点击“复制代码”将所有代码复制到剪贴板
4. 验证设置：
    - 输入身份验证器应用程序中的当前 6 位数字代码
    - 点击“确认”
    - 如果有效，2FA 现在已启用

第三步：**2FA** 已启用

成功消息出现：

从现在开始，登录需要密码和 2FA 代码。

密码重置流程（自助服务）

忘记密码的用户可以通过电子邮件重置密码。

第一步：请求重置链接

从登录页面，点击“忘记密码？”

1. 输入 电子邮件地址
2. 点击“发送重置链接”

发生的事情：

- 系统检查电邮件否存在于数据库中
- 如果找到，通过 Mailjet 发送密码重置电子邮件
- 电子邮件包含时间限制的重置链接（通常 1 小时过期）
- 成功消息出现：“重置说明已发送到您的电子邮件”

如果未找到电子邮件：

错误消息：“未找到该电子邮件地址的帐户”

第二步：检查电子邮件

用户收到主题类似的电子邮件：

> 嗨 [姓名]，

> 您请求重置 OmniCRM 帐户的密码。
>
> 点击下面的链接重置您的密码：<https://yourcompany.com/reset-password/abc123token456>
>
> 此链接在 1 小时内过期。
>
> 如果您没有请求此操作，请忽略此电子邮件。

点击重置链接以继续。

第三步：设置新密码

重置链接打开密码创建页面：

> （锁图标）
>
> 密码
>
> 确认密码
>
> [重置密码]

1. 输入 新密码
2. 在 确认密码 字段中重新输入
3. 点击“重置密码”

密码要求：

- 最小长度（通常 8 个字符以上）
- 密码必须匹配

成功：

- 成功消息：“密码已成功重置”
- 自动重定向到登录页面
- 用户现在可以使用新密码登录

过期/无效令牌：

如果重置链接过期或无效：

> [请求新的重置链接]

用户管理的管理员控制

具有适当权限的管理员可以从用户管理界面管理用户认证设置。

访问用户管理

显示所有用户的表格及操作按钮。

> 姓名 电子邮件 电话 操作 John Smith <john@example.com> +44 123... ✏ □ □ □ Jane Doe <jane@example.com> +44 456... ✏ □ □ Bob Wilson <bob@example.com> +44 789... ✏ □ □ □ ✉

□作图标：

- ✏ 编辑 - 修改用户详细信息、角色、权限
- □ 删除 - 移除用户帐户
- □ 重置密码 - 生成临时密码
- □ 重置 **2FA** - 禁用用户的 2FA（仅在启用 2FA 时显示）
- ✉ 发送欢迎电子邮件 - 重新发送欢迎电子邮件（仅在用户从未登录时显示）

管理员：重置用户密码

当用户忘记密码且管理员需要帮助时：

第一步：点击重置密码图标（□）

确认模态框出现：

> 您确定要重置以下用户的密码吗：
>
> 用户：John Smith (<john@example.com>)
>
> 将生成并显示一个临时密码。用户必须在下次登录时更改此密码。

> [取消] [重置密码]

第二步：确认重置

点击 **“重置密码”**。系统生成安全的临时密码。

第三步：显��临时密码

> John Smith 的临时密码：
>
> [复制密码]
>
> ⚠ 重要： • 通过安全渠道将此密码发送给用户 • 不要通过电子邮件或不安全的消息发送 • 用户将在下次登录时被迫更改密码
>
> [关闭]

管理员作：

- 复制临时密码
- 给用户打电话或通过安全方式沟通
- 口头提供临时密码
- 指示用户登录并更改密码

用户体验：

当用户使用临时密码登录时：

1. 登录成功
2. 立即重定向到“更改密码”屏幕
3. 必须设置新密码才能访问系统
4. 不能跳过密码更改

管理员：重置用户 **2FA**

当用户失去对身份验证器应用程序和所有备份代码的访问时：

第一步：点击重置 **2FA** 图标 **( )**

仅对当前启用 2FA 的用户显示。

确认模态框出现：

第二步：确认重置

点击 **“重置 2FA”**

第三步：确认

成功消息：

> John Smith 现在可以仅使用密码登录。他们可以从用户设置中重新启用 2FA。

用户体验：

- 用户现在可以仅使用密码登录（不需要 2FA 代码）
- 2FA 盾牌图标 (□) 从管理员表中的用户行中消失
- 用户可以自愿从���设置中重新启用 2FA

重要安全提示：

在重置 2FA 之前，管理员应：

1. 通过替代方式验证用户身份：
    - 政府身份证明验证
    - 安全问题
    - 最近交易验证

- 亲自验证（如适用）

2. 在客户备注中记录重置
3. 通知用户在恢复访问后重新启用 2FA

管理员：发送欢迎电子邮件

对于尚未收到或丢失欢迎电子邮件的用户：

可用时：

纸飞机图标 (✉) 仅对 从未登录 的用户显示 (`login_count = 0`)。

点击发送欢迎电子邮件图标 **(✉)**

> 向以下用户发送欢迎电子邮件：
>
> 用户: Bob Wilson (<bob@example.com>)
>
> 电子邮件将包括： • 欢迎消息 • 登录说明 • 设置初始密码的链接（如适用） • 支持联系信息
>
> [取消] [发送电子邮件]

点击 **“发送电子邮件”**

成功消息：

通过 **Mailjet** 发送的电子邮件：

使用模板：`api_crmCommunicationUserWelcome`

管理员：编辑用户

点击 编辑图标 **(✏)** 修改用户详细信息：

> 名字
>
> 姓氏
>
> 电子邮件

> 电话号码
>
> 角色 ☑ admin ☐ customer_service_agent_1 ☐ customer
>
> [取消] [保存更改]

可编辑字段：

- 姓名、电子邮件、电话
- 角色 - 分配/移除角色（影响权限）
- 活跃/非活跃状态

管理员：删除用户

点击 删除图标（）移除用户：

> 您确定要删除：
>
> 用户：John Smith (<john@example.com>)
>
> ⚠ 警告：此操作无法撤销。
>
> 这将永久删除： • 用户帐户和凭据 • 2FA 设置 • 会话历史
>
> 客户数据和交易将不会被删除。
>
> [取消] [删除用户]

点击“删除用户”以确认。

成功消息：

最佳实践

对于最终用户

登录安全：

- 使用强大、独特的密码

- 仅在个人设备上启用“记住我”
- 在共享计算机上始终注销
- 启用 2FA 以增加安全性

2FA 管理:

- 启用 2FA 后立即保存备份代码
- 将代码存储在密码管理器或安全位置
- 测试备份代码以确保其有效
- 如果使用多个备份代码，请重新生成备份代码
- 如果您同时丢失身份验证器和备份代码，请联系管理员

密码管理:

- 使用密码管理器生成和存储密码
- 永远不要通过电子邮件或消息共享密码
- 如果怀疑被泄露，请更改密码
- 为 OmniCRM 使用独特的密码（不要重复使用密码）

对于管理员

用户安全管理:

- 在重置 2FA 或密码��前验证用户身份
- 永远不要通过电子邮件发送临时密码
- 在用户备注中记录所有安全重置
- 鼓励员工启用 2FA
- 监控异常登录模式

密码重置:

- 仅通过电话或亲自沟通临时密码
- 生成强大的临时密码（系统会自动执行此操作）
- 确保用户在第一次登录时更改密码
- 不要不必要地重置密码 - 尽可能使用电子邮件重置流程

2FA 重置:

- 将 2FA 重置视为高安全性操作
- 在重置之前通过多个渠道验证身份
- 记录重置原因
- 鼓励用户在恢复访问后立即重新启用 2FA
- 考虑要求所有管理用户启用 2FA

用户管理：

- 定期审查用户列表以查找非活跃帐户
- 移除已离开组织的用户
- 确保适当的角色分配
- 监控从未登录的用户
- 每季度审核用户权限

故障排除

“无效的电子邮件或密码**”**错误

- 原因：凭据不正确
- 修复：
  - 验证电子邮件地址□否正确
  - 检查大写锁定□否关闭
  - 如果忘记密码，请尝试密码重置
  - 如果帐户被锁定，请联系管理员

2FA 代码未被接受

- 原因：时间同步问题或代码不正确
- 修复：
  - 确保设备时间正确（设置 → 日期和时间 → 自动）
  - 等待代码刷新（代码每 30 秒更改）
  - 尝试下一个出现的代码
  - 如果身份验证器无法工作，请使用备份代码
  - 如果其他方法都失败，请联系管理员重置 2FA

“记住我”无效

- 原因：Cookies 被禁用或清除
- 修复：
  - 在浏览器设置中启用 Cookies
  - 关闭浏览器时不要清除 Cookies
  - 为 OmniCRM 域禁用隐私扩展
  - 尝试不同的浏览器

未收到密码重置电子邮件

- 原因：电子邮件未发送、垃圾邮件过滤器或错误的电子邮件
- 修复：
  - 检查垃圾邮件/垃圾邮件文件夹
  - 验证电子邮件地址□否正确
  - 等待 5-10 分钟（电子邮件投递可能会延迟）
  - 检查 Mailjet 集成□否正常工作（管理员）
  - 联系管理员进行手动密码重置

密码重置链接过期

- 原因：令牌过期（通常为 1 小时）
- 修复：
  - 请求新的密码重置
  - 检查电子邮件并及时点击链接
  - 如果出现重复问题，请联系管理员

无法启用 **2FA**（密码不正确）

- 原因：当前密码输入不正确
- 修复：
  - 验证当前密码
  - 如果不确定，请先重置密码
  - 联系管理员以获取帮助

丢失身份验证器应用程序和备份代码

- 原因：手机丢失/重置，备份代码未保存
- 修复：
  - 立即联系管理员
  - 管理员将验证身份并重置 2FA
  - 重置后仅使用密码登录
  - 重新启用 2FA，并这次保存备份代码

管理员：“重置 **2FA** 失败”错误

- 原因：权限不足
- 修复：
  - 确保您具有管理员角色
  - 检查 API 权限
  - 联系系统管理员

管理员：未生成临时密码

- 原因： API 错误或权限问题
- 修复：
  - 刷新页面并重试
  - 验证管理员权限
  - 检查 API 日志以查找错误
  - 确保数据库可访问

安全考虑

会话管理：

- 会话在不活动期间过期
- “记住我”将会话延长至 30 天
- 会话存储为 HTTP-only cookies（JavaScript 无法访问）
- 安全标志确保 cookies 仅通过 HTTPS 发送

密码安全：

- 密码使用行业标准算法进行哈希处理

- 从不存储明文密码
- 临时密码在首次使用后自动过期
- 失败的登录尝试被跟踪（潜在的速率限制）

2FA 安全:

- TOTP 密钥在数据库中加密
- 尽可能在客户端生成二维码
- 备份代码在存储前进行哈希处理
- 每个备份代码仅限一次使用

管理员作:

- 2FA 重置记录在活动日志中
- 密码重置创建审计跟踪
- 管理操作需要适当的角色权限
- 安全事件记录 IP 地址

相关文档

- `2fa` - 详细的 2FA API 参考（以 API 为中心）
- `rbac` - 基于角色的访问控制和权限
- `administration_configuration` - Mailjet 电子邮件配置用于密码重置
- `integrations_mailjet` - 电子邮件模板配置
- `customer_care` - 客户自助服务门户

□□□□

0fT52ZvoZBE

客户、联系人、站点与服务

我们有一个简单的 客户 模型，在这个客户下，可以有多个 联系人 和多个 站点、服务 等。

一个 客户 □与我们有关系的公司或个人，我们会向其发送发票/账单。

一个 联系人 □与客户合作的人，对于个人来说，可能与客户本人□同一个人，但我们可能有家庭成员或其他联系人，每个联系人都有一个类型，例如账单联系人、技术联系人等，这会影响我们如何处理该联系人。

一个 站点 □我们提供服务的实际位置，可以□家庭、办公室或其他地点。这使我们能够为单个客户拥有多个站点，例如，一个拥有多个办公室的客户，并了解哪些服务与哪个站点相关联。

一个 服务 □我们向客户收费的项目，可以□家庭互联网服务、移动服务，甚至□抽象服务，如租赁子网或为机架提供计量电力。每个服务都与一个客户和一个站点相关联，并可以有多个相关费用。

客户还有一个 活动日志 `<csa_activity_log>`，记录所有的变更，`标签 <administration_tags>`，`属性 <administration_attributes>` 用于存储自定义元数据，`库存项目 <administration_inventory>` 和财务信息，如`交易 <payments_transaction>`，`发票 <payments_invoices>` 和 `支付方式 <basics_payment>`。

一旦我们创建了客户，我们可以`为该□户添加服务 <csa_add_service>`，这□我们向他们收费的项目。

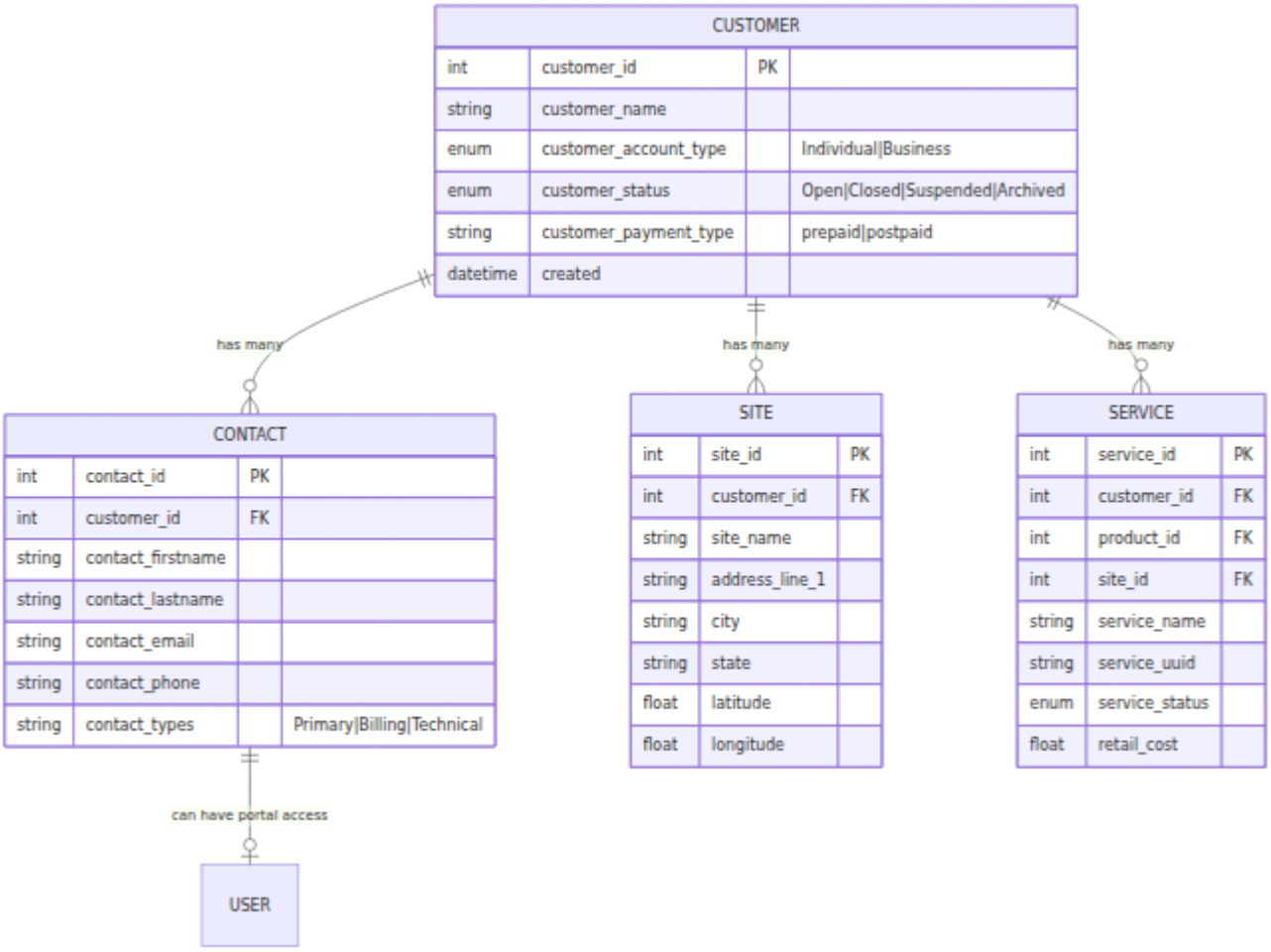
有关创建客户的信息，请参见 创建客户。

数据模型概述

OmniCRM 使用围绕客户及其服务组织的关系数据模型。该模型在下面分为几个专注的部分。

客户核心关系

客户□中心实体，具有相关的联系人、站点和服务。

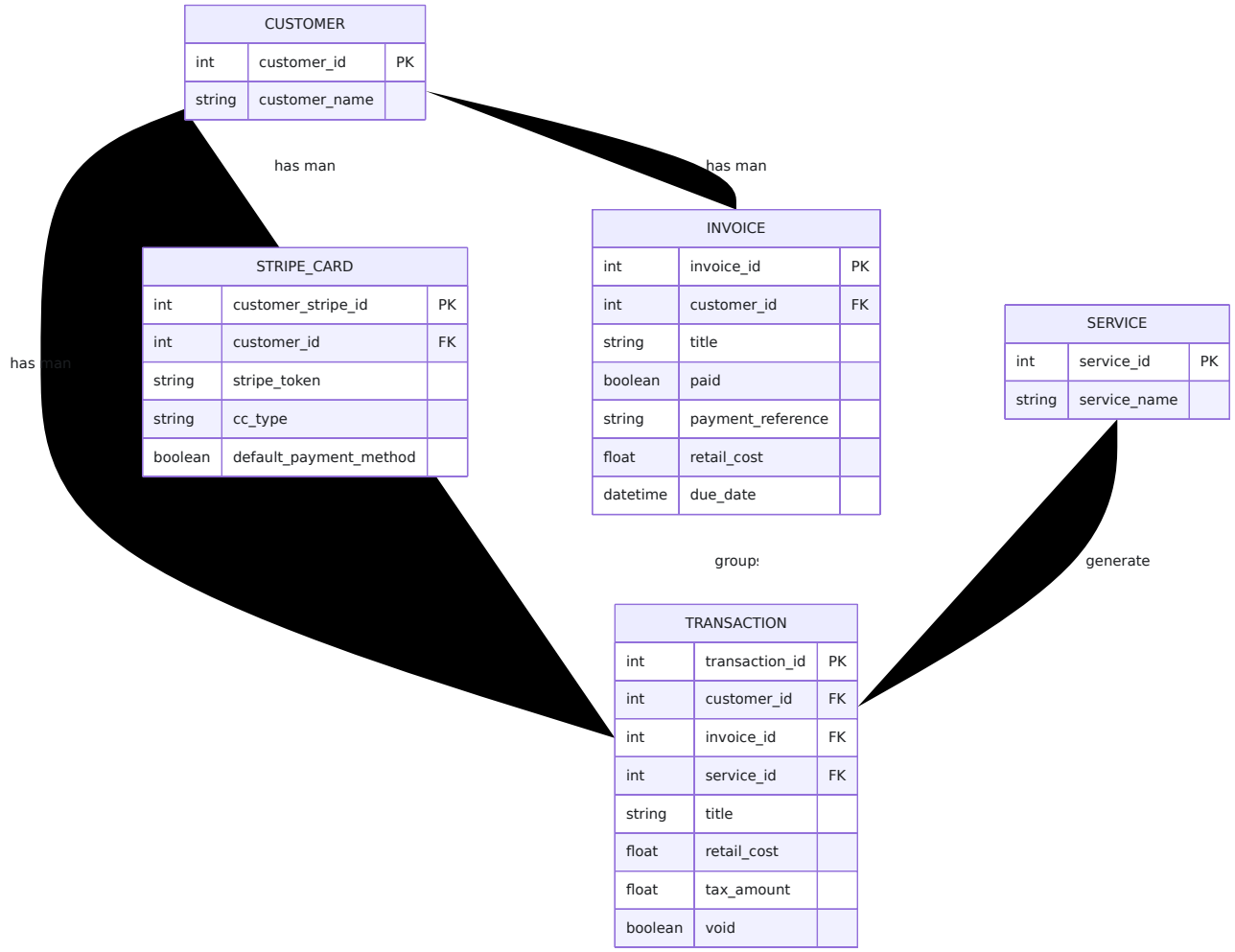


关键点：

- 一个客户可以有多个联系人（账单、技术等）
- 一个客户可以有多个站点（分支办公室、位置）
- 服务交付到站点
- 联系人可以通过链接的用户帐户拥有门户访问权限

账单与财务数据

交易和发票跟踪所有财务活动。

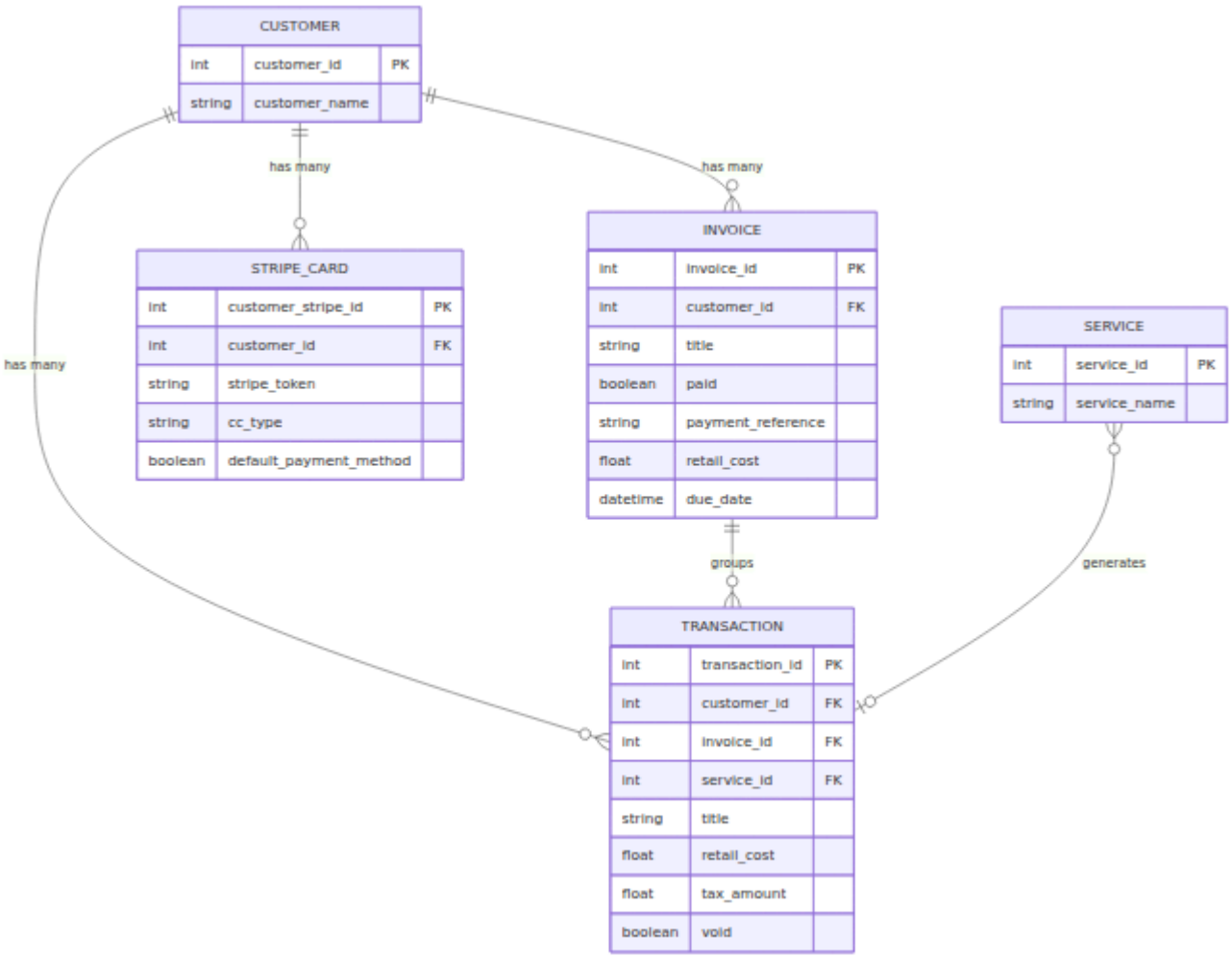


关键点：

- 交易可以□独立的或分组到发票中
- 服务会自动生成每月交易
- Stripe 卡片被令牌化并安全存储
- 发票将多个交易分组以进行计费

产品与配置

产品定义服务提供；配置创建实际服务。

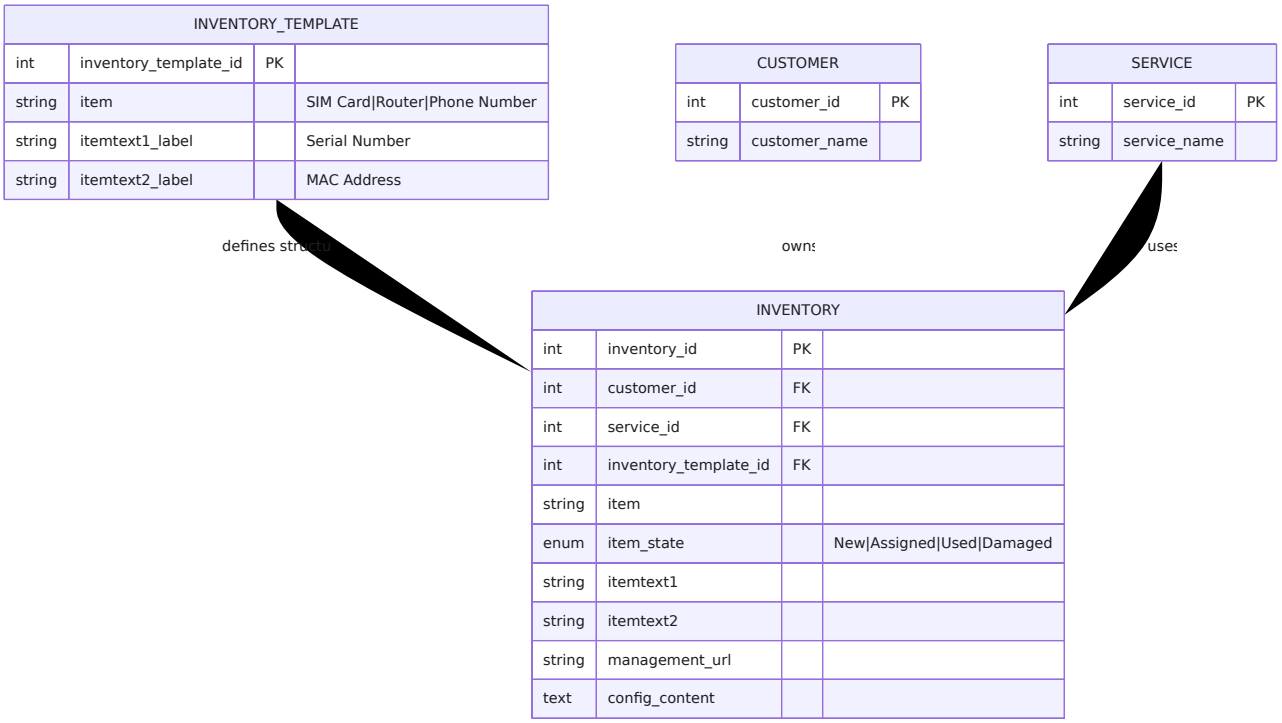


关键点：

- 产品□模板；服务□活动实例
- 配置通过 Ansible 剧本创建或修改服务
- 每个配置作业有多个事件用于跟踪进度
- 一个产品配置可以创建多个服务（捆绑）

库存系统

跟踪分配给客户的物理和虚拟资产。

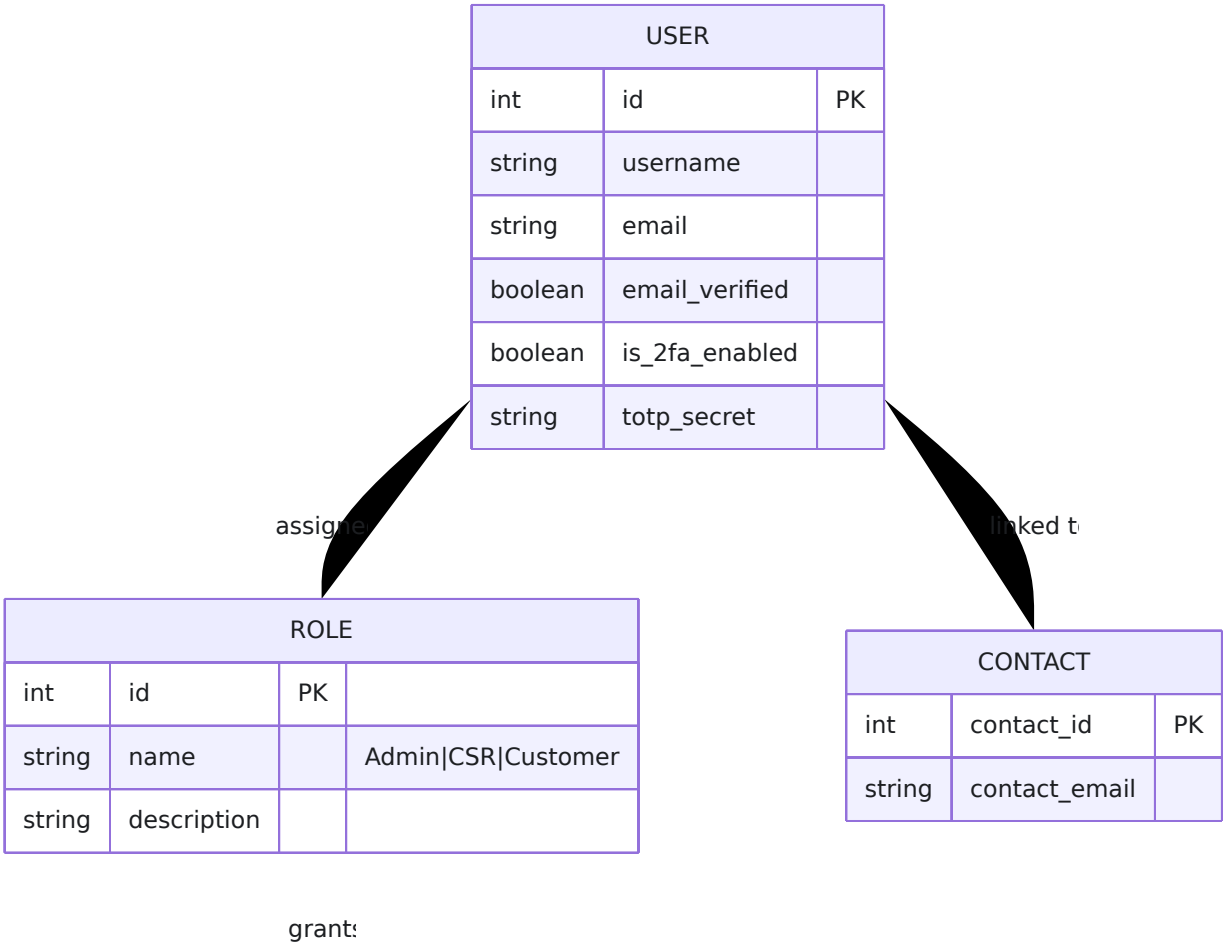


关键点：

- 库存模板定义每种物品类型的结构（字段）
- 灵活的 itemtext1-20 字段适应不同的库存类型
- 物品可以分配给客户并链接到服务
- 存储设备配置和管理凭据

用户与安全

具有基于角色的访问控制的用户帐户。



关键点：

- 用户可以有多个角色；角色有多个权限
- 每个用户的双因素认证（2FA）□可选的
- 员工用户□独立的；客户用户链接到联系人
- 细粒度权限控制对资源的访问

客户列表

客户列表提供一个可搜索、可过滤的所有客户的表格。

功能：

- 搜索 - 按名称或 ID 过滤客户
- 批量□作 - 选择多个客户进行批处理操作
- 分页 - 浏览大型客户列表
- 快速□作 - 直接从列表查看或删除客户

客户 - 详细信息

客户对象本身不包含太多信息，仅包含一个名称和对联系人和站点的引用。

您的特定部署可能包含额外字段或自定义，但基本客户对象非常简单。

在概述页面上还有一个图表显示客户的每用户平均收入（ARPU），这□总收入除以服务数量，并与系统中所有客户的平均 ARPU 进行比较。

客户状态选项根据您业务的具体需求量身定制，但通常包括活动、非活动、待处理等选项，每个选项中都有不同的规则，控制客户在该状态下在系统中的���为。

只有在客户没有活动服务、未支付发票或未开票交易的情况下，才能删除客户。如果客户有这些，您需要关闭活动服务并确保支付完成，然后才能删除客户，这将归档客户及所有相关数据，日后如有需要可以恢复。

站点 - 详细信息

站点提供服务的实际位置，可以与多个服务相关联。

它们主要用于商业客户，其中单个客户可能有多个站点，例如，一个拥有多个办公室的客户。

拥有多个站点使我们能够跟踪哪些服务与哪个站点相关联，例如，如果客户为新办公室订购新服务，我们需要确保将正确的服务交付到正确的位置。这使我们能够按站点跟踪服务，并在需要时单独计费。

Google Maps 集成和地理编码

每个站点都与 **Google Maps** 集成，以确保准确的地址数据和地理位置。用户界面会自动进行地理编码地址并生成位置信息。

地址地理编码的工作原理：

添加或编辑站点时，界面提供两种设置位置的方法：

1. 地址搜索（推荐）
    - 使用表单顶部的搜索栏
    - 输入地址，Google Maps 自动完成建议匹配项
    - 从下拉列表中选择正确的地
    - 系统会自动填充：
        - 站点名称 - 来自 Google Maps 的地点名称
        - 地址行 **1** - 街道号码和名称

- 地址行 **2** - 子房产（单元/套房号码）
- 城市 - 居住地
- 州/地区 - 行政区域
- 邮政编码 - 邮政编码
- 国家 - 国家名称
- 纬度和经度 - 精确坐标
- 加号代码 - 11 字符的开放位置代码（例如，`8C3MFJV8+2F`）
- **Google Maps** 地点 **ID** - 唯一地点标识符

2. 可拖动地图标记（手动）
    - 将图钉拖动到确切位置
    - 系统执行 反向地理编码 从坐标获取地址
    - 基于图钉位置自动填充相同字段
    - 对于农村地区或地址不准确时非常有用

加号代码生成：

加号代码□通过使用开放位置代码库从纬度/经度自动生成的。加号代码□一个短代码（11 个字符），表示世界上任何地方的精确位置。

示例：

> 自动填充字段：✓ 站点名称：“123 主街”✓ 地址行 1：“123 主街”✓ 城市：“伦敦”✓ 州：“大伦敦”✓ 国家：“英国”✓ 邮政编码：“SW1A 1AA”✓ 纬度：51.5074 ✓ 经度：-0.1278 ✓ 加号代码：“9C3XGPHC+3Q”✓ Google 地点 ID：“ChIJdd4hrwug2EcRmSrV3Vo6llI”

验证要求：

在保存站点之前，系统会验证：

- 纬度和经度 必须设置（通过搜索或拖动标记）
- 国家 必须填充（如果未设置，则默认为 `VITE_DEFAULT_COUNTRY`）
- 加号代码 必须为 12 个字符（11 + 1 用于填充）

如果验证失败，您将看到错误：

视觉反馈：

界面会显示实时反馈：

或

位置数据的使用

地理编码的位置信息（纬度、经度、加号代码）在 OmniCRM 中用于：

1. 服务交付和安装

- 现场技术人员 - 访问加号代码以导航到确切站点位置
- 安装调度 - 根据地理位置分配技术人员
- 设备部署 - 确保正确的设备交付到正确的站点

2. 故障通知

- 地理围栏警报 - 如果特定区域发生网络故障，按纬度/经度半径查询站点
- 定向通信 - 仅向受影响的客户发送故障通知，通过 `Mailjet <integrations_mailjet>`
- 状态页面 - 显示受影响站点的故障地图

示例：

> 查询: SELECT * FROM Customer_Site
> WHERE distance(latitude, longitude, 51.5074, -0.1278) < 5
>
> 结果: 47 个受影响站点 操作: 向 47 个客户发送故障通知

3. 报告和分析

- 地理收入 - 按城市、州、地区的收入
- 服务密度地图 - 服务位置的热图
- 扩展规划 - 确定服务不足的区域

4. 多站点商业客户

- 站点管理 - 跟踪哪些服务在何处
- 单独计费 - 如有需要按站点开具发票
- 服务分配 - 在配置期间将服务链接到特定站点

农村和偏远站点

对于位于农村地区、街道地址可能不存在或不准确的客户：

1. 使用地图拖动
    - 放大到大致区域
    - 将图钉拖动到确切的物业/建筑
    - 系统为该精确位置生成加号代码
2. 加号代码共享
    - 与客户共享加号代码（例如，`8C3MFJV8+2F`）
    - 客户可以在 Google Maps 中输入此代码以查看确切位置
    - 现场技术人员使用加号代码进行导航
3. 地址备注
    - 使用“地址备注”字段提供额外方向
    - 示例：“在红色谷仓左转，过牛栏 500 米”
    - 备注对安装团队可见

提示

如果地址不准确，您可以在地图上拖动图钉到正确的位置。系统将反向地理编码该位置并自动填充所有地址字段。

联系人 - 详细信息

联系[illegible]与客户相关联的人。他们可以[illegible]账单联系人、技术联系人或其他类型，每个联系人都有一个类型，影响我们如何处理该联系人。

我们可以为单个客户拥有多个联系人，例如，一个拥有多个账单联系人的客户，或一个拥有多个技术联系人的客户。

一个好的例子[illegible]拥有管理服务提供商的公司，处理技术方面的事务，以及一个单独的账单联系人，处理财务方面的事务，或者一个家庭，每个成员都有自己的联系人，但并非所有人都被授权进行更改。

同样，我们可能只想向技术联系人发送故障通知，或仅向账单联系人发送发票，联系人类型允许我们控制这一点。

联系人类型的具体逻辑取决于您的业务，但基本思想[illegible]每个联系人都有一个类型，影响我们如何处理他们，并且与客户相关联的每个人都[illegible]一个联系人。

联系人与 `Mailjet <integrations_mailjet>` 集成同步，使我们能够根据联系人类型、位置、每月支出或购买的服务发送定向电子邮件活动，并管理用于事务通信的所有电子邮件模板。

导航

OmniCRM 从头开始设计，以响应式和直观为目标。

本指南将帮助您浏览系统并找到所需的信息。

登录后，屏幕左侧的导航栏将显示系统的主要部分，以及每个部分内的子部分。

您可以在 WebApp 栏中找到组织中最常用的所有 Web 应用程序的链接（这可以根据您组织的需求进行定制）。

您未处理的任何警报将在屏幕右上角可见，您可以单击警报以查看更多信息。

□□□

付款、发票和交易

客户交易

系统中任何需要花费金钱的内容都被记录为客户的交易。

每笔交易都有批发成本和零售成本的货币金额，以及交易的描述。

交易可以由系统自动生成，例如，当服务被提供时，会为设置费用创建一笔交易，当服务被计费时，会为零售成本创建一笔交易。

交易也可以手动创建，例如，如果客户获得了信用额度，则会为信用额度创建一笔交易，或者收取安装费用时，会为安装费用创建一笔交易。

交易被组合在一起形成发票 `<payments_invoices>`，并发送给客户进行付款。

访问交易

交易可以在系统级别或按客户查看：

按客户查看：

1. 导航到 客户 → 【选择客户】
2. 点击 计费 标签
3. 在第一个卡片中查看交易列表

系统范围查看：

1. 导航到 计费 → 交易（从主菜单）
2. 查看所有客户的所有交易

交易统计小部件

在交易页面顶部，四个统计���片显示财务摘要：

小部件描述：

- 总交易数 - 所有交易的零售成本总和（所有时间）
- 总未开票交易 - 尚未包含在发票中的交易总和
- 本月总交易数 - 本日历月创建的交易总和
- 上月总交易数 - 上日历月创建的交易总和

值格式化：

- 超过1,000的值：显示为“k”后缀（例如，$1.5k）
- 超过1,000,000的值：显示为“M”后缀（例如，$2.3M）
- 超过1,000,000,000的值：显示为“B”后缀（例如，$1.1B）

交易列表

交易表显示所有交易，包含以下列：

列描述：

- **ID** - 唯一交易ID
- 日期 - 交易创建日期
- 标题 - 简短交易名称
- 描述 - 交易的详细描述
- 金额 - 零售成本（收费为正，信用为负）
- 发票 - 如果交易已开票，则为发票ID（可点击链接）
- 状态 - 如果已开票则为勾选标记，未开票则为破折号

每行□作：

每行都有一个操作菜单（⋮），选项包括：

- 查看详情 - 打开交易详情模态框
- 下载发票**PDF** - 下载PDF（仅在已开票时）
- 作废交易 - 将交易标记为作废（仅在未开票时）

交易类型

交易分为两大类：

借记交易（收费）

正金额，增加客户应付余额：

- 服务设置费用 - 提供服务时的一次性收费
- 月度服务费用 - 服务的定期收费
- 安装费用 - 现场技术人员访问的收费
- 设备费用 - 调制解调器、路由器、SIM卡的收费
- 逾期付款费用 - 逾期发票的罚款

- 手动收费 - 员工添加的自定义收费

信用交易（付款/退款）

负金额，减少客户应付余额：

- 现金付款 - 客户以现金支付
- 卡付款 - 客户以信用卡/借记卡支付
- 银行转账付款 - 客户通过银行转账支付
- 账户信用 - 善意信用，补偿
- 退款 - 退还给客户的钱
- 折扣 - 促销或忠诚度折扣

手动添加交易

点击 "+ 添加交易" 打开添加交易模态框。

借记交易（收费）：

信用交易（付款/退款）：

字段描述：

- 交易类型 - 选择借记（收费）或信用（付款/退款）
- 信用类型 - 如果选择信用，选择付款方式（现金、卡、银行转账）
- 标题 - 交易的简短名称（必填）
- 描述 - 详细说明（可选）
- 零售成本 - 客户支付的金额（必填，正数）
- 批发成本 - 您的成本（可选，用于利润跟踪）
- 税率 - 应用于此交易的税率（可选，默认为产品税或0%）
- 服务 - 将交易链接到特定服务（可选）
- 站点 - 将交易链接到特定站点（可选）
- 交易日期 - 交易日期（默认为今天）

验证：

- 标题和零售成本□必填项
- 零售成本必须□正数
- 如果选择了信用类型，必须选择一种信用类型

发生的事情：

1. 交易在数据库中创建
2. 出现在客户的交易列表中

3. 包含在“未开票交易”计数中
4. 可用于下次发票生成
5. 创建活动日志条目

搜索和过滤交易

搜索

使用搜索栏查找交易。搜索范围包括：

- 交易ID
- 标题
- 描述
- 发票ID

过滤器

应用过滤器以缩小交易列表：

可用过滤器：

- 作废状态 - 所有、作废、未作废
- 发票状态 - 所有、已开票、未开票

过滤□作：

- 应用过滤器 - 将所选过滤器应用于列表
- 重置过滤器 - 清除所有过滤器并显示所有交易

排序

单击任何列标题进行排序：

- **ID** - 按交易ID排序（最新/最旧）
- 日期 - 按交易日期排序
- 标题 - 按字母顺序排序

- 金额 - 按零售成本排序（最高/最低）
- 发票 - 按发票ID排序

再次单击以反转排序方向（升序 ↔ 降序）。

作废交易

错误添加的交易可以被 作废（标记为已删除）。

要求：

- 交易必须未开票
- 一旦开票，交易不能被作废（必须退款）

如何作废：

1. 在列表中找到交易
2. 点击操作菜单（⋮）
3. 选择 "作废交易"
4. 在模态框中确认

发生的事情：

- 交易标记为 `void = true`
- 不再出现在默认交易列表中

- 从发票生成中排除
- 可以通过过滤“作废”交易查看
- 从“未开票交易”总数中扣除

注意：作废与退款不同。作废意味着“这笔交易本不该存在。”退款意味着“撤销一笔有效交易。”

交易的税

交易可以包括税，税□根据产品的税配置自动计算的，或者每笔交易手动指定。

税行为：

- 借记交易（收费） - 税适用于收费，基于：
  - 产品税率 - 如果交易链接到产品，则自动应用产品的税率
  - ��动覆盖 - 员工在创建交易时可以覆盖税率
  - 税额 - 计算为：`retail_cost × (tax_percentage / 100)`
  - 显示格式 - 在交易列表中显示为：`$10.00 (10%)`
- 信用交易（付款/退款） - 信用不适用税
  - 信用交易的税率字段隐藏
  - 所有付款和退款的税自动设置为0%
  - 信用在没有税务影响的情况下减少客户的未偿余额

税计算示例：

- 产品：移动计划，税率10%，$50.00零售成本
- 自动税计算：$50.00 × 0.10 = $5.00
- 显示：$5.00 (10%)

零税（**NIL/**免税）：

- 通过将税率设置为0，可以使产品免税
- 如果未指定，税默认为0%
- 免税交易在税列中显示“-”

交易详情视图

单击交易以查看完整详情：

已开票与未开票交易

未开票交易：

- 尚未包含在任何发票中
- 可用于下次发票生成
- 可以作废
- 计入“未开票交易”总数

- 状态显示破折号（-）

已开票交易：

- 包含在发票中
- 不能作废（如有需要，必须退款）
- 发票ID可点击（链接到发票详情）
- 状态显示勾选标记（✓）
- 不能修改

发票生成：

当您为客户生成发票时：

1. 系统查找该客��的所有未开票交易
2. 可选择按日期范围过滤
3. 交易包含在新发票中
4. 交易 `invoice_id` 字段填充
5. 交易现在标记为“已开票”

请参见 `payments_invoices` 以获取发票生成的详细信息。

常见工作流程

工作流程1：因服务中断手动信用

1. 客户来电：“服务中断了2天”
2. 员工决定给予£10的信用
3. 导航到客户 计费 标签
4. 点击 "+ 添加交易"
5. 选择 信用 交易类型
6. 选择 现金付款 信用类型
7. 输入标题：“服务中断信用”
8. 输入描述：“因1月8-9日的中断补偿”
9. 输入零售成本：10.00

10. 从下拉菜单中选择受影响的服务
11. 点击 "添加交易"
12. 交易以-£10.00的金额出现
13. 将在下次发票中作为信用包含

工作流程2：手动安装费用

1. 现场技术人员安装服务
2. 员工需要收取£75的安装费用
3. 导航到客户 计费 标签
4. 点击 "+ 添加交易"
5. 选择 借记 交易类型
6. 输入标题：“安装费用”
7. 输入描述：“现场技术人员进行光纤安装”
8. 输入零售成本：75.00
9. 输入批发成本：45.00（可选，用于利润跟踪）
10. 选择已安装的服务
11. 选择安装地点
12. 点击 "添加交���"
13. 交易出现在未开票列表中
14. 将在下次发票中包含

工作流程3：作废重复交易

1. 员工注意到重复交易
2. 验证交易尚未开票
3. 点击重复交易的操作菜单（⋮）
4. 选择 "作废交易"
5. 在模态框中确认
6. 交易从列表中移除
7. 未开票总数相应减少

工作流程4：查找发票的交易

1. 需要生成月度发票
2. 点击 发票过滤器：“未开票”
3. 点击 应用过滤器
4. 查看所有未开票交易
5. 从小部件中注意总金额
6. 导航到生成发票
7. 选择日期范围（例如，1月1日至31日）
8. 范围内的交易包含在发票中

故障排除

无法作废交易

- 原因：交易已开票
- 解决方案：交易□发票历史的一部分。如果需要退款，请改为创建信用交易。

重复交易出现

- 原因：服务多次收费或提供错误
- 解决方案：如果未开票，请作废重复交易。如果已开票，请发放信用。

交易未出现在列表中

- 原因：应用过滤器或交易已作废
- 解决方案：点击“重置过滤器”以显示所有交易。要查看作废交易，请按“作废：作废”过滤。

未开票总数与预期不符

- 原因：一些��易已开票，或作废交易被排除
- 解决方案：应用过滤器“发票：未开票”以仅查看未开票。单独检查作废交易。

无法添加交易（客户字段禁用）

- 原因：查看客户特定的交易页面
- 解决方案：客户已预先选择。如果需要为不同客户添加交易，请转到系统范围的交易页面。

相关文档

- `payments_invoices` - 发票生成和管理
- `payments_process` - 处理发票的付款
- `basics_payment` - 付款方式概述
- `csa_activity_log` - 在活动日志中查看交易历史

客户发票

交易 `</payments_transaction>` 被组合在一起形成发票，并发送给客户进行付款。

发票有开始和结束日期，表示发票覆盖的期间，以及到期日，表示发票到期付款的日期。

发票可以由系统自动生成，例如，当服务被计费时，会为零售成本创建发票，或者可以手动创建，例如，如果客户请求发票副本，或者如果客户因一次性收费而被计费。

客户发票完全使用 `Mailjet <integrations_mailjet>` 模板，可以自定义以包含公司徽标、地址和付款详情，并可以通过电子邮件发送给客户，或下载为PDF。

自定义发票模板

OmniCRM使用HTML模板和Jinja2模板生成发票。您可以完全自定义发票设计、品牌、颜色和布局。

发票模板位置

发票模板存储在 `OmniCRM-API/invoice_templates/`

默认模板：

- `norfone_invoice_template.html` - 示例发票模板
- `cifi_invoice_template.html` - 替代模板示例

配置：

活动发票模板在 `OmniCRM-API/crm_config.yaml` 中指定：

```
invoice:
  template_filename: 'norfone_invoice_template.html'
```

可用模板变量

发票模板可以访问以下Jinja2变量：

发票信息：

- `{{ invoice_number }}` - 唯一发票ID（例如，`INV-2025-001234`）
- `{{ date }}` - 发票发布日期（ISO格式：`2025-01-10T12:00:00`）
- `{{ due_date }}` - 付款到期日（例如，`2025-02-10`）
- `{{ start_date }}` - 计费周期开始日��
- `{{ end_date }}` - 计费周期结束日期
- `{{ total_amount }}` - 税前发票总金额（数字）
- `{{ total_tax }}` - 从所有交易计算的总税额（数字）

客户信息：

- `{{ client.name }}` - 客户的全名或公司名称

- `{{ client.address.address_line_1 }}` - 地址行1
- `{{ client.address.address_line_2 }}` - 地址行2
- `{{ client.address.city }}` - 城市
- `{{ client.address.state }}` - 州/省
- `{{ client.address.zip_code }}` - 邮政编码
- `{{ client.address.country }}` - 国家

交易行项目：

使用以下方式循环遍历交易：

```
{% for sub_transaction in transactions %}
    <tr>
        <td>{{ sub_transaction.transaction_id }}</td>
        <td>{{ sub_transaction.created.split("T")[0] }}</td>
        <td>{{ sub_transaction.title }}</td>
        <td>{{ sub_transaction.description }}</td>
        <td>${{ "%.2f"|format(sub_transaction.retail_cost) }}</td>
    </tr>
{% endfor %}
```

交易字段：

- `sub_transaction.transaction_id` - 交易ID
- `sub_transaction.created` - 交易日期/时间
- `sub_transaction.title` - 交易标题
- `sub_transaction.description` - 详细描述
- `sub_transaction.retail_cost` - 行项目金额
- `sub_transaction.tax_percentage` - 应用的税率（例如，10表示10%）
- `sub_transaction.tax_amount` - 以美元计算的税额

在模板中显示税：

```
<td>
    {% if sub_transaction.tax_amount and
sub_transaction.tax_amount > 0 %}
        ${{ "%.2f"|format(sub_transaction.tax_amount) }} ({{
sub_transaction.tax_percentage }}%)
    {% else %}
        -
    {% endif %}
</td>
```

创建自定义发票模板

步骤**1**：复制现有模板

```
cd OmniCRM-API/invoice_templates/
cp norfone_invoice_template.html
your_company_invoice_template.html
```

步骤**2**：自定义**HTML/CSS**

编辑 `your_company_invoice_template.html` 以匹配您的品牌：

关键自定义区域：

1. 公司徽标和品牌

```
<!-- 用您的徽标URL替换 -->


<!-- 更新公司名称 -->
<h1>您的公司名称</h1>
```

2. 配色方案

```
<style>
    /* 主要品牌颜色 */
    .navbar {
        background: linear-gradient(to bottom right, #your-
color-1, #your-color-2);
    }

    /* 表头 */
    .table thead th {
        background-color: #your-brand-color !important;
        color: white !important;
    }

    /* 按钮和链接 */
    .btn-primary {
        background-color: #your-brand-color;
    }
</style>
```

3. 公司信息页脚

```
<footer>
    <p>您的公司名称</p>
    <p>123商业街，城市，国家</p>
    <p>电话: +1-555-123-4567 | 电子邮件: billing@yourcompany.com</p>
    <p>ABN/税号: 12345678900</p>
</footer>
```

4. 付款说明

```
<div class="payment-info">
    <h3>付款方式</h3>
    <p><strong>在线: </strong> 在 https://yourcompany.com/pay 付款
</p>
    <p><strong>银行转账: </strong></p>
    <ul>
        <li>账户名称: 您的公司有限公司</li>
        <li>BSB: 123-456</li>
        <li>账户号码: 987654321</li>
        <li>参考: {{ invoice_number }}</li>
    </ul>
</div>
```

5. 条款和条件

```
<div class="terms">
    <h4>付款条款</h4>
    <p>发票日期后30天内到期付款。</p>
    <p>逾期付款费用: 逾期余额每月2%。</p>
    <p>有关账单查询: billing@yourcompany.com</p>
</div>
```

步骤**3**: 更新配置

编辑 `OmniCRM-API/crm_config.yaml`:

```
invoice:
  template_filename: 'your_company_invoice_template.html'
```

步骤**4**: 重启**API**

```
cd OmniCRM-API
sudo systemctl restart omnicrm-api
```

步骤**5**: 测试发票生成

1. 导航到有交易的客户
2. 生成测试发票

3. 下载PDF以验证格式
4. 将发票通过电子邮件发送给自己以测试电子邮件发送

高级自定义

条件内容：

使用Jinja2条件语句显示/隐藏内容：

```
{% if total_amount > 1000 %}
    <div class="high-value-notice">
        <p><strong>注意：</strong> 大额余额 - 可根据请求提供付款计划。</p>
    </div>
{% endif %}

{% if client.address.country == "Australia" %}
    <p>包含GST：${{ "%.2f"|format(total_amount * 0.10) }}</p>
{% endif %}
```

多语言支持：

创建特定语言的模板：

```
invoice_template_en.html
invoice_template_es.html
invoice_template_fr.html
```

根据客户的语言偏好进行配置。

自定义计算：

```
<!-- 显示小计和税务明细 -->
<tr>
    <td colspan="4" class="text-right"><strong>小计: </strong></td>
    <td>${{ "%.2f"|format(total_amount) }}</td>
</tr>
<tr>
    <td colspan="4" class="text-right"><strong>税: </strong></td>
    <td>${{ "%.2f"|format(total_tax) }}</td>
</tr>
<tr>
    <td colspan="4" class="text-right"><strong>总计: </strong></td>
    <td>${{ "%.2f"|format(total_amount + total_tax) }}</td>
</tr>
```

注意：`total_tax` 变量通过将发票中所有交易的 `tax_amount` 相加自动计算的。每笔交易的税根据其 `tax_percentage` 字段计算的，默认为产品的 `tax_percentage` 或0%（如果未指定）。

付款的二维码：

生成用于移动支付的二维码：

```
<div class="qr-payment">
    
    <p>用您的手机扫描以立即付款</p>
</div>
```

PDF样式最佳实践

OmniCRM使用 **WeasyPrint** 将HTML转换为PDF。遵循以下指南：

支持的**CSS**：

- 大多数CSS 2.1属性
- 有限的CSS3（flexbox，一些变换）
- 通过 `@font-face` 使用网络字体

不支持：

- JavaScript
- CSS Grid（使用表格代替）
- 复杂动画
- 一些现代CSS属性

页面大小和边距：

```
@page {
    size: A4;
    margin: 1cm;
}

body {
    font-family: Arial, sans-serif;
    font-size: 10pt;
}
```

打印特定样式：

```
@media print {
    .no-print {
        display: none;
    }

    .page-break {
        page-break-after: always;
    }
}
```

表格布局：

```
.table {
    table-layout: fixed;
    width: 100%;
}

.table th, .table td {
    word-wrap: break-word;
    padding: 4px;
}
```

字体嵌入：

对于自定义字体��使用网络安全字体或嵌入：

```
@font-face {
    font-family: 'YourFont';
    src: url('https://yourcompany.com/fonts/yourfont.woff2')
format('woff2');
}

body {
    font-family: 'YourFont', Arial, sans-serif;
}
```

测试发票模板

测试清单：

1. 视觉检查：
    - 徽标正确显示
    - 颜色符合品牌指南
    - 文本可读（不太小）
    - 表格对齐正确
    - 所有部分存在
2. 数据准确性：
    - 客户详细信息正确
    - 交易金额正确相加

- 日期格式正确
- 所有变量正确替换

3. **PDF**质量:
   - 文件大小合理（<5MB）
   - 图像清晰
   - 无文本截断或溢出
   - 页面在适当位置换行
4. 多页发票:
   - 每页重复页眉
   - 显示页码
   - 长交易列表正确分页
5. 电子邮件发送:
   - PDF附加到电子邮件
   - 文件大小低于Mailjet限制（15MB）
   - 在Gmail、Outlook、Apple Mail中呈现

测试命令（手动生成）:

您可以通过API测试发票生成:

```
curl -X GET "http://localhost:5000/crm/invoice/{invoice_id}/pdf" \
     -H "Authorization: Bearer YOUR_TOKEN" \
     --output test_invoice.pdf
```

常见模板问题

变量未替换:

- 原因：变量名称拼写错误或缺少数据
- 解决方案：检查拼写□否完全正确（区分大小写），验证数据库中□否存在数据

PDF样式破损:

- 原因：不支持的CSS属性
- 解决方案：使用CSS 2.1属性，测试与WeasyPrint兼容的CSS

图像未显示：

- 原因：相对URL或阻止外部资源
- 解决方案：使用绝对HTTPS URL，确保图像可以公开访问

表格溢出页面：

- 原因：固定列宽过宽
- 解决方案：使用百分比宽度，`table-layout: fixed`

字体未呈现：

- 原因：字体未嵌入或不可用
- 解决方案：使用网络安全字体（Arial、Times New Roman等）或正确嵌入自定义字体

PDF生成失败：

- 原因： HTML语法错误或WeasyPrint崩溃
- 解决方案：验证HTML，检查WeasyPrint日志，简化复杂布局

发票**PDF**缓存

为了提高性能并减少冗余的PDF生成，OmniCRM包含发票PDF缓存系统。当首次生成发票PDF时，它会在数据库中缓存以供后续请求使用。

PDF缓存工作原理：

1. 首次请求 - 当请求发票PDF（下载或电子邮件）时，系统：
    - 从发票模板生成PDF
    - 将PDF编码为Base64
    - 计算PDF内容的SHA256哈希
    - 在 `Invoice_PDF_Cache` 表中存储��
        - 发票ID引用
        - PDF数据（Base64编码）
        - 文件名
        - 内容哈希（用于完整性验证）
        - 创建时间戳

2. 后续请求 - 当再次请求相同发票时：
    - 系统根据invoice_id检查缓存PDF
    - 如果缓存存在且有效，则立即返回缓存PDF
    - 更新 `last_accessed` 时间戳以跟踪缓存使用情况
3. 缓存失效 - 当以下情况发生时，缓存的PDF将失效：
    - 发票被修改（添加/删除交易，详情更改）
    - 发票模板被更新
    - 手动触发缓存清除

好处：

- 性能 - 对于重复请求，立即提供PDF（无再生成延迟）
- 一致性 - 所有发票的下载均为相同PDF（除非发票被修改）
- 服务器负载 - 减少PDF生成的CPU使用
- 用户体验 - 初次生成时显示加载指示器，后续请求立即完成

缓存管理：

发票PDF缓存由系统自动管理。可以根据以下情况定期清除旧的或未使用的缓存条目：

- 年龄（例如，删除超过90天的缓存条目）
- 访问模式（删除30天未访问的条目）
- 存储限制（如有需要，实施缓存大小限制）

API行为：

通过API或UI下载发票时：

- 首次请求：在PDF生成时显示加载指示器，然后缓存
- 后续请求：立即缓存下载
- 缓存命中/未命中对用户透明的

重要：当您更新发票模板时，请清除缓存以确保新发票使用更新的设计：

```
-- 清除所有缓存的发票PDF（在MySQL中运行）
DELETE FROM Invoice_PDF_Cache;
```

或更新 `crm_config.yaml` 以在模板更改时自动使缓存失效。

访问发票

发票可以在系统级别或按客户查看：

按客户查看：

1. 导航到 客户 → 【选择客户】
2. 点击 计费 标签
3. 在第三个卡片中查看发票列表

系统范围查看：

1. 导航到 计费 → 发票（从主菜单）
2. 查看所有客户的所有发票

发票统计小部件

在发票页面顶部，四个统计卡片显示财务摘要。

小部件描述：

- 总发票数 - 所有发票的零售成本总和（所有时间）和已发送发票的计数

- 未付款发票 - 尚未付款的发票总和和未付款发票的计数
- 本月发票 - 本日历月创建的发票总和及计数
- 上月发票 - 上日历月创建的发票总和及计数

值格式化：

- 超过1,000的值：显示为“k”后缀（例如，$1.5k）
- 超过1,000,000的值：显示为“M”后缀（例如，$2.3M）
- 超过1,000,000,000的值：显示为“B”后缀（例如，$1.1B）

趋势指示器：

- “本月”和“上月”的小部件显示百分比变化
- 绿色箭头向上：与前一时期相比增加
- 红色箭头向下：与前一时期相比减少
- 灰色箭头向右：没有变化

发票列表

发票表显示所有发票，包含以下列：

列描述：

- **ID** - 唯一发票ID
- 标题 - 发票标题/描述
- 期间 - 计费期间（开始日期 - 结束日期）或“一次性”发票的“N/A”
- 到期日 - 付款到期日
- 创建 - 发票创建日期
- 金额 - 发票总金额（零售成本）
- 状态 - 已付款、未付款或已退款
- 操作 - 可用操作（根据状态而异）

操作图标：

- ⬇（下载）- 下载发票PDF
- 🗑（删除）- 作废发票（仅在未付款时）
- 💳（付款）- 在线支付发票（仅在未付款时）
- ✉（电子邮件）- 将发票电子邮件发送给客户
- ↩（退款）- 退款Stripe付款（仅适用于已付款的Stripe发票）

生成发票

点击 "+ 生成形式发票" 创建新发票。

字段描述：

- □□□□ - □□□□

查看发票详情

单击表中任何发票行以查看完整发票详情，包括所有交易、总计和可用操作。

发票详情模态框：

- 发票信息 - 显示发票ID、标题、日期、付款状态和作废状态
- 交易列表 - 显示发票中包含的所有交易，包含：
  - 交易日期
  - 标题和描述
  - 零售成本
  - 税额和百分比（格式为`$10.00 (10%)`）
  - 免税交易在税列中显示“-”
- 总计摘要 - 实时计算显示：
  - 交易计数
  - 小计（所有零售成本之和）
  - 税（所有税额之和）
  - 发票总计（小计 + 税）
- 作按钮 - 与表中相同的可用操作：
  - 下载**PDF** - 下载发票PDF（始终可用）
  - 发送电子邮件 - 将发票电子邮件发送给客户（未作废的发票）
  - 支付发票 - 处理付款（未付款、未作废的发票）
  - 退款 - 退款Stripe付款（仅适用于已付款的Stripe发票）

- 删除 - 作废发票（未付款、未作废的发票）

下载发票PDF

单击表中的 下载图标（⬇）或在发票详情模态框中单击 "下载PDF" 按钮以下载发票为PDF。

下载过程：

1. 单击发票旁的下载图标
2. 生成期间显示加载旋转图标（仅首次）
3. 浏览器提示保存文件：`Invoice_01234.pdf`
4. PDF打开或保存到下载文件夹

PDF缓存行为：

- 首次下载 - 从模板生成PDF，缓存到数据库（可能需要2-3秒）
- 后续下载 - 从缓存立即下载
- 缓存失效 - 如果发票被修改或模板更新，则清除缓存

故障排除下载问题：

- 旋转图标永远不停止 - 检查浏览器控制台，API可能已关闭
- **PDF**为空或损坏 - 检查发票模板□否存在语法错误
- 下载失败 - 检查弹出窗口阻止程序设置，尝试不同的浏览器

支付发票

单击 付款图标 □ 在线支付发票。

支付过程：

1. 单击未付款发票上的付款图标
2. 支付模��框打开，显示发票详情
3. 选择付款方式：
    - **Stripe**交易 - 收取保存的信用卡（对所有用户可用）
    - 现金 - 手动现金付款（仅限员工）
    - 退款 - 将退款作为付款（仅限员工）
    - **POS**交易 - 销售终端（仅限员工）
    - 银行转账 - 手动银行转账（仅限员工）
4. 如果选择Stripe：
    - 从保存的付款方式中选择卡
    - 默认卡预选
    - 单击选择不同的卡
5. 如果选择其他方式：
    - 输入参考号码（可选）
6. 单击 "支付发票" 进行处理
7. 系统处理付款：
    - **Stripe** - 通过Stripe API收取卡费
    - 其他方式 - 为付款金额创建负交易
8. 发票状态更改为“已付款”
9. 显示成功通知

自助服务与员工付款：

:doc:`自助服务门户 <self_care_portal>`（客户）：

- 仅提供Stripe付款
- 必须有保存的付款方式
- 如果没有付款方式，则显示警告
- 提供添加付款方式的链接

员工门户（管理员）：

- 所有付款方式可用
- 可以手动标记发票为已付款（现金、POS、银行转账）
- 可以输入参考号码以便跟踪

付款方式警告：

如果客户没有保存的付款方式，则会显示警告，提示他们在支付��票之前添加付款方式。

电子邮件发票

单击 电子邮件图标（✉）将发票发送给客户。

发生的事情：

1. 单击发票旁的电子邮件图标

2. 系统从缓存中检索发票PDF（如果未缓存则生成）
3. 通过 `Mailjet <integrations_mailjet>` 发送电子邮件，使用 `api_crmCommunicationCustomerInvoice` 模板
4. 电子邮件包括：
    - 发票PDF作为附件
    - 客户姓名
    - 发票号码和到期日
    - 应付总额
    - 在线支付发票的链接
    - 查看/下载发票的链接
5. 成功通知：“发票电子邮件成功发送”

电子邮件收件人：

电子邮件发送给所有类型为“计费”的客户联系人，或者如果没有计费联系人，则发送给主要联系人。

电子邮件模板变量：

- `{{ var:customer_name }}` - 客户的全名
- `{{ var:invoice_number }}` - 发票ID
- `{{ var:invoice_date }}` - 发票发布日期
- `{{ var:due_date }}` - 付款到期日
- `{{ var:total_amount }}` - 应付总额
- `{{ var:invoice_url }}` - 查看/下载PDF的链接
- `{{ var:pay_url }}` - 在线支付发票的链接

故障排除电子邮件问题：

- 电子邮件未发送 - 检查 `crm_config.yaml` 中的Mailjet API凭据
- 客户未收到 - 验证客户联系电子邮件地址
- **PDF**未附加 - 检查PDF生成□否成功（先尝试下载）

作废发票

单击 删除图标（□）作废发票。

要求：

- 发票必须 未付款
- 已付款的发票不能作废（必须退款）

如何作废：

1. 在列表中找到未付款的发票
2. 单击删除图标（）
3. 在模态框中确认：

发生的事情：

- 发票标记为 `void = true`
- 所有交易与发票解除链接（`invoice_id` 设置为null）
- 交易再次变为“未开票”
- 交易可以包含在新发票中
- 发票在列表中显示“作废：”前缀
- 发票操作被禁用（无法下载、支付或发送电子邮件）
- 可以通过过滤“作废”发票查看

重要说明：

- 作废与退款不同
- 作废 =“这张发票本不该存在”（计费错误，重复）

- 退款 = “撤销一张有效的已付款发票”（将钱退还给客户）

退款发票

单击 退款图标 () 退款已付款的发票。

要求：

- 发票必须 已付款
- 发票必须通过 **Stripe** 付款
- 发票必须有有效的 `payment_reference`（Stripe付款意图ID）
- 仅限员工用户可用（不适用于自助服务）

如何退款：

1. 找到已付款的Stripe发票
2. 单击退款图标 ()
3. 退款确认模态框打开：

4. 单击 "确认退款"
5. 系统处理Stripe退款：
    - 调用Stripe API退款付款
    - 在Stripe中创建退款交易
    - 更新发票以包含 `refund_reference`
6. 发票状态更改为“已退款”

7. 显示成功通知

退款后的发生情况：

- 发票仍保留在系统中（未作废）
- 状态显示“已退款”
- 交易仍与发票关联
- 客户收到退款至原付款方式（3-7个工作日）
- Stripe仪表板显示退款交易

退款限制：

- 不能退款通过现金、POS或银行转账支付的发票（需要手动撤销）
- 不能部分退款（仅限全额发票金额）
- 不能退款两次

搜索和过滤发票

搜索

使用搜索栏查找发票。搜索范围包括：

- 发票ID
- 发票标题
- 客户姓名（仅在系统范围查看中）

过滤器

应用过滤器以缩小发票列表：

可用过滤器：

- 作废状态 - 所有、作废、未作废
- 付款状态 - 所有、已付款、尚未付款

过滤□作：

- 应用过滤器 - 将所选过滤��应用于列表
- 重置过滤器 - 清除所有过滤器并显示所有发票

排序

单击任何列标题进行排序：

- **ID** - 按发票ID排序（最新/最旧）
- 标题 - 按字母顺序排序
- 到期日 - 按到期日排序
- 创建 - 按创建日期排序
- 金额 - 按零售成本排序（最高/最低）
- 状态 - 按付款状态排序（已付款优先或未付款优先）

再次单击以反转排序方向（升序 ↔ 降序）。

分页

通过页面控件在大型发票列表中导航，显示当前页、总页数和每页项目选择器（10、25、50或100个项目）。

常见发票工作流程

工作流程1：带交易预览的月度计费

1. 月末到来（例如，1月31日）
2. 导航到 计费 → 发票
3. 点击 "+ 生成形式发票"
4. 选择客户（或如果计费多个客户则按客户操作）
5. 设置日期：
   - 开始日期：2025-01-01
   - 结束日期：2025-01-31
   - 到期日期：2025-02-15（从现在起15天）
   - 标题：“2025年1月服务”（可选）
6. 交易预览 部分显示1月的所有未开票交易

7. 审查预览：
    - 默认包含所有交易
    - 检查总计：小计、税和发票总计
    - 验证所有费用□否正确
8. 点击 "生成发票"（��钮显示交易计数，例如，“生成发票（15）”）
9. 发票创建，包含所有选定交易
10. 点击发票行以查看详情并验证
11. 点击详情模态框中的 "发送电子邮件" 按钮或表中的电子邮件图标
12. 客户收到带PDF和付款链接的发票电子邮件

工作流程2：选择性交易开票

1. 客户有多个服务（移动+互联网）和杂项费用
2. 希望为每项服务开具单独的发票
3. 生成第一张发票（移动服务）：
    - 点击 "+ 生成形式发票"
    - 标题：“移动服务 - 2025年1月”
    - 开始/结束：1月1日至31日
    - 到期日期：2月15日
    - 在交易预览中，排除所有非移动交易：
        - 点击互联网交易旁的 X 按钮
        - 点击杂项费用旁的 X 按钮
        - 仅剩下移动服务交易
    - 验证总计仅反映移动服务
    - 点击 "生成发票"（显示移动交易的计数）
4. 生成第二张发票（互联网服务）：
    - 再次点击 "+ 生成形式发票"
    - 标题：“互联网服务 - 2025年1月”
    - 开始/结束：1月1日至31日（相同期间）
    - 在交易预览中：
        - 移动交易已开票（不再出现）
        - 使用 X 按钮排除杂项费用
        - 仅剩下互联网服务交��
    - 点击 "生成发票"

5. 生成第三张发票（额外费用）：
    - 再次点击 "+ 生成形式发票"
    - 标题：“额外费用 - 2025年1月”
    - 仅未开票的杂项费用出现在预览中
    - 点击 "全选" 以包含所有
    - 点击 "生成发票"
6. 将所有三张发票通过电子邮件发送给客户

工作流程3：排除争议或待处理交易

1. 计费周期结束
2. 导航到客户 计费 标签
3. 点击 "+ 生成形式发票"
4. 设置计费期间日期
5. 交易预览显示20笔交易
6. 客户对一项收费提出争议，另一项正在调查中
7. 在交易预览中：
    - 找到争议交易（例如，“数据超限收费”）
    - 点击 X 按钮将其排除
    - 找到待处理交易（例如，“安装费用”）
    - 点击 X 按钮将其排除
    - 交易计数更新：“选择了18笔交易”
    - 总计自动重新计算
8. 审查更新后的总计（排除争议金额）
9. 点击 "生成发票（18）"
10. 仅生成包含已批准交易的发票
11. 争议/待处理交易保持未开票状态，待下一个计费周期处理

工作流程4：快速发票审查和调整

1. 员工生成月度发票
2. 交易预览显示意外的高总额
3. 审查预览中的每笔交易：
    - 注意到同一服务的重复收费

- 点击 **X** 排除重复项
- 注意到不应计费的测试交易
- 点击 **X** 排除测试交易

4. 总计实时更新
5. 验证新总额与预期金额相符
6. 点击 **"**生成发票**"**，包含更正后的交易
7. 如果需要，返回并作废/删除排除的交易
8. 自信地将发票通过电子邮件发送给客户

工作流程**5**：一次性安装发票

1. 现场技术人员完成安装
2. 员工手动添加安装交易
3. 导航到客户 计费 标签
4. 点击 **"+** 生成形式发票**"**
5. 设置**���**期:
   - 开始日期：今天
   - 结束日期：今天
   - 到期日期：今天 + 7天
   - 标题:“安装服务”
6. 交易预览仅显示今天的交易
7. 验证安装费用出现
8. 使用 **X** 按钮排除任何定期费用（如果存在）
9. 点击 **"**生成发票**"**
10. 立即发送电子邮件给客户
11. 客户通过Stripe在线付款

工作流程**6**：在客户联系之前查看发票

1. 客户来电询问账单问题
2. 员工导航到客户的发票列表
3. 单击发票行 打开发票详情模态框
4. 审查发票信息:
   - 发票ID、日期、状态

    - 包含的所有交易及描述
    - 每笔交易的税务明细
    - 小计、税和总金额
5. 用确切的细节回答客户的问题
6. 如果客户请求PDF，单击模态框中的 "下载**PDF**" 按钮
7. 如果客户请求重新发送电子邮件，单击 "发送电子邮件" 按钮
8. 完成后关闭模态框

工作流程**7**：更正计费错误

1. 客户报告错误收费
2. 员工单击发票行查看详情
3. 在发票详情模态框中审查交易列表
4. 确定错误交易
5. 发票未付款，因此可以作废
6. 单击模态框底部的 "删除" 按钮
7. 确认作废
8. 交易再次变为未开票
9. 员工修改或从交易列表中删除错误交易
10. 生成包含更正交易的新发票：
    - 如果需要，使用交易预览排除更正的交易
    - 仅包含有效费用
11. 将更正后的发票通过电子邮件发送给客户

工作流程**8**：处理多笔付款

1. 客户带现金支付多张发票
2. 导航到客户 计费 标签
3. 查看未付款发票
4. 单击第一张发票行查看详情
5. 验证金额和交易
6. 单击模态框底部的 "支付发票" 按钮
7. 选择 "现金" 付款方式
8. 输入参考：“现金支付2025-01-15”

9. 单击 "支付发票"
10. 模态框关闭，发票标记为“已付款”
11. 对剩余发票重复此操作
12. 所有发票现在标记为“已付款”

工作流程9：处理退款请求

1. 客户请求退款以补偿多付
2. 员工验证发票□通过Stripe支付的
3. 导航到发票列表
4. 单击发票行查看详情
5. 验证付款信息和金额
6. 单击模态框底部的 "退款" 按钮（仅对Stripe发票可见）
7. 确认退款
8. 系统处理Stripe退款
9. 发票状态更改为“已退款”
10. 客户在3-7个工作日内收到退款
11. 员工跟进客户以确认收到

故障排除

无法生成发票 - 找不到交易

- 原因：指定日期范围内没有未开票交易
- 解决方案：检查交易列表，验证交易存在且未开票。调整日期范围或移除过滤器。

发票**PDF**生成失败

- 原因：模板语法错误、WeasyPrint崩溃或缺少客户数据
- 解决方案：检查发票模板HTML□否存在错误，验证客户地址字段□否已填充，查看API日志。

Stripe错误导致付款失败

- 原因：卡被拒绝、资金不足、卡过期或Stripe API问题
- 解决方案：尝试不同的付款方式，验证卡□否有效，检查Stripe仪表板以获取拒绝原因。

无法作废发票

- 原因：发票已付款
- 解决方案：已付款的发票不能作废。如果需要退款，请对Stripe发票使用退款功能或手动创建信用交易。

发票电子邮件未发送

- 原因： Mailjet API凭据无效、客户没有计费联系人或电子邮件模板缺失
- 解决方案：验证 `crm_config.yaml` 中的Mailjet配置，检查客户联系人，验证发票电子邮件模板□否存在。

退款按钮未出现

- 原因：发票通过现金/POS/银行转账支付（不□Stripe），或发票未付款
- 解决方案：退款按钮仅在Stripe付款时出现。对于其他付款方式，请创建手动信用交易。

下载**PDF**显示旧模板设计

- 原因： PDF在模板更新之前缓存
- 解决方案：清除发票PDF缓存： `DELETE FROM Invoice_PDF_Cache WHERE invoice_id = X;`

客户无法支付发票（没有付款方式）

- 原因：自助服务门户中没有保存的付款方式
- 解决方案：客户必须在 付款方式 页面添加信用卡，然后才能支付发票。

同一期间生成多张发票

- 原因：员工生成发票两次，或日期范围重叠
- 解决方案：作废重复发票。调整日期范围以防止重叠。使用交易预览确保唯一的交易集。

交易预览未显示任何交易

- 原因：日期范围内的所有交易已开票或不存在交易
- 解决方案：验证日期范围□否正确。检查交易列表以确认交易存在。过滤发票以查看哪些发票包含交易。

无法排除交易生成发票

- 原因：交易已开票或浏览器问题
- 解决方案：验证交易在预览中显示并带有勾选标记。刷新页面并重试。如果问题仍然存在，请清除浏览器缓存。

发票总额与预期金额不符

- 原因：包含意外交易、未计算税或排除的交易仍被计算
- 解决方案：仔细审查交易预览。检查每笔交易的零售成本和税。验证排除的交易□否显示为灰色。检查□□成发票按钮上的交易计数徽章。

生成发票按钮被禁用

- 原因：没有选择任何交易或无效的日期范围
- 解决方案：确保至少包含一笔交易（未排除）。验证开始日期在结束日期之前。检查到期日期□否已设置。

发票详情模态框未打开

- 原因：JavaScript错误或页面未完全加载
- 解决方案：刷新页面。检查浏览器控制台□否有错误。尝试不同的浏览器。验证互联网连接。

交易税未在发票详情中显示

- 原因：交易税率为0%或税额为null
- 解决方案：验证交易□否设置了税率。检查交易创建时□否计算了税额。如有需要，更新交易。

发票详情模态框中缺少□作按钮

- 原因：发票已作废或用户权限不足
- 解决方案：作废的发票仅显示下载PDF按钮。验证发票状态。检查用户角色和权限。

相关文档

- `integrations_mailjet` - 发票电子邮件发送和模板
- `administration_configuration` - 发票模板配置
- `payments_transaction` - 创建出现在发票上的交易
- `payments_process` - 处理发票付款
- `payment_system_guide` - 付款API参考和供应商配置

付款方式管理

OmniCRM的付款方式系统允许客户和员工安全地管理付款卡，使用 多供应商支付处理（Stripe、PayPal等）。付款方式使服务的自动计费、一次性付款和定期收费成为可能，而无需在OmniCRM中存储敏感的卡数据。

另请参见：付款系统指南 `<payment_system_guide>`，计费概述 `<billing_overview>`，付款处理 `<payments_process>`，发票 `<payments_invoices>`。

概述

付款方式系统提供：

- 安全的卡存储 - 由支付供应商（Stripe、PayPal）进行令牌化，永远不会存储在OmniCRM中
- 多供应商支持 - 支持Stripe和PayPal付款方式
- 多张卡 - 客户可以存储多种付款方式
- 默认选择 - 指定首选付款方式以进行自动收费
- 到期跟踪 - 监控和更新即将到期的卡
- 自助服务 - 客户可以通过自助服务门户 `<self_care_portal>` 管理自己的卡
- 员工管理 - 支持员工可以代表客户添加/删除卡

支持的付款方式：

- 信用卡（Visa、Mastercard、American Express、Discover）
- 借记卡
- 预付卡（如果卡网络支持）

不在**OmniCRM**中存储：

卡详细信息由支付供应商令牌化并安全存储。OmniCRM仅存储：

- 支付供应商（stripe、paypal）
- 卡品牌（Visa、Mastercard等）
- 后4位数字
- 到期月/年
- 持卡人姓名/昵称
- 供应商特定的付款方式令牌

访问付款方式

从客户页面：

1. 导航到 客户 → 【选择客户】
2. 点击 计费 标签
3. 滚动到 付款方式 部分

或直接：

从即将到期的卡仪表板：

查看所有即将到期的卡客户：

这显示了系统范围内在接下来的60天内到期的卡的列表。

付款方式列表

付款方式表显示客户的所有存储卡：

列描述：

- 昵称 - 卡的友好名称（例如，“个人卡”，“工作Visa”）
- 发行者 - 卡品牌和后4位数字
- 到期 - 到期月/年（MM/YYYY格式）

- 添加 - 卡添加到帐户的日期
- 默认 - 勾选标记表示自动收费的默认付款方式

每张卡的□作:

每行都有一个操作菜单（⋮），选项包括:

- 设为默认 - 将其设为默认付款方式
- 删除 - 从帐户中删除卡

添加付款方式

单击 "添加付款方式" 打开安全付款模态框。

步骤1：输入卡详细信息

安全付款表单出现（由Stripe Elements或PayPal SDK提供支持）:

必填字段:

- 卡信息 - 卡号、��期、CVC（由Stripe验证）
- 持卡人姓名 - 卡上的姓名
- 国家/地区 - 账单国家

可选字段：

- 卡昵称 - 友好的标签以区分卡片

安全性：

- 卡详细信息直接输入到供应商托管的安全iframe中（Stripe Elements / PayPal SDK）
- OmniCRM从未看到或存储完整的卡号
- PCI DSS合规性由支付供应商处理
- 实时验证防止无效卡号

步骤2：提交并令牌化

当您单击"添加付款方式"时：

1. 客户端验证：
   - 支付供应商验证卡号格式
   - 检查到期日期□否在未来
   - 验证CVC格式
2. 令牌化：
   - 卡详细信息直接发送到支付供应商（而不□OmniCRM）
   - 供应商创建一个安全令牌（例如，`pm_1A2B3C4D`用于Stripe）
   - 令牌返回给OmniCRM
3. 服务器处理：
   - OmniCRM将令牌保存到客户记录中，并附上供应商标识符
   - 存储后4位数字、品牌、到期和供应商名称以供显示
   - 从未有完整的卡号接触OmniCRM服务器

步骤3：确认

成功消息出现：

> 您的Visa卡（后四位为1234）已添加到您的帐户。

新卡出现在付款方式表中。

自动默认选择：

- 如果这□客户的第一张卡，则自动设为默认
- 如果客户已经有卡，则新卡作为非默认添加
- 客户可以在添加后更改默认设置

设置默认付款方式

默认付款方式用于：

- 自动定期服务收费
- 发票付款
- 充值和补充
- 一次性交易（除非另有说明）

更改默认设置：

1. 在付款方式表中找到要设为默认的卡
2. 单击卡旁的 □作菜单（ ⋮ ）
3. 选择 "设为默认"
4. 确认消息出现

   Visa卡（后四位为5678）现在□您的默认付款方式。

勾选标记移动到新选择的卡上。

视觉指示：

默认卡在默认列中显示，通常带有绿色勾选标记徽章。

删除付款方式

删除过期、丢失或不再需要的卡。

步骤1：启动删除

1. 在付款方式表中找到要删除的卡
2. 单击 □作菜单（⋮）
3. 选择 "删除"

步骤2：确认删除

确认模态框出现：

> 您确定要删除此付款方式吗？
>
> 卡：Visa（后四位为1234）到期：12/2026
>
> ⚠ 警告：如果这□您唯一的付款方式，您将需要添加新的付款方式以继续使用需要自动计费的服务。
>
> [取消] [删除付款方式]

单击 "删除付款方式" 以确认。

步骤3：删除完成

成功消息：

卡从表中移除并从Stripe中删���。

重要限制：

- 如果存在其他卡，则无法删除默认卡 - 首先将其他卡设为默认
- 如果删除最后一张卡，则会发出警告 - 需要付款的服务可能会暂停
- 无撤销 - 删除□永久的；客户必须重新添加卡（如有需要）

管理即将到期的卡

OmniCRM跟踪卡到期日期，并提供工具主动更新即将到期的卡。

即将到期的卡仪表板

导航到 计费 → 即将到期的卡 查看系统范围的列表：

> 客户 卡到期 距离采取行动的天数 John Smith Visa **1234 02/2025 12天 更新 **Acme Corp MC5678 03/2025 45天 更新 **Jane Doe Amex**9012 01/2025 已过期 更新

过滤器：

- 到期范围 - 接下来的30/60/90天或已过期
- 客户类型 - 个人与企业
- 服务类型 - 按需要付款方式的服务过滤

作：

- 更新 - 打开客户的付款方式页面以添加新卡
- 通知 - 向客户发送电子邮件提醒（如果已配置Mailjet）

到期通知

如果已配置Mailjet，将自动发送电子邮件：

- 到期前**60**天 - 第一次提醒

- 到期前**30**天 - 第二次提醒
- 到期前**7**天 - 最后警告
- 到期时 - 卡已过期通知

客户可以单击电子邮件中的链接，通过自助服务门户更新其付款方式。

电子邮件模板变量：

Mailjet模板接收：

- 客户姓名
- 卡品牌和后4位数字
- 到期日期
- 自助服务付款方式页面的链接

请参见 `integrations_mailjet` 以获取电子邮件模板配置。

更新即将到期的卡

推荐工作流程：

1. 客户收到到期通知电子邮件
2. 客户登录自助服务门户
3. 导航到 计费 → 付款方式
4. 点击 "添加付款方式"
5. 输入新卡详细信息（相同卡的更新到期，或替换卡）
6. 将新卡设为默认
7. 删除旧/过期卡

员工工作流程：

如果客户拨打支持电话：

1. 员工打开客户帐户
2. 导航到 计费 → 付款方式
3. 代表客户添加新卡（客户通过电话提供详细信息）
4. 将新卡设为默认

5. 删除过期卡
6. 与客户确认

警告

绝不要要求客户通过电子邮件或短信发送卡详细信息。始终使用：

- 安全的自助服务门户进行自助服务
- 电话与员工直接输入系统
- 在零售地点亲自进行

当卡过期时发生的事情

当付款卡达到到期日期且未更新时：

立即影响：

1. 自动付款失败
    - 支付供应商拒绝使用过期卡的交易
    - 月度服务续订未能处理
    - 自动充值失败
    - 发票自动付款失败
2. 客户通知
    - 系统尝试收费
    - 发送付款失败通知
    - 发送“更新付款方式”电子邮件，附带自助服务门户链接
3. 服务状态更改
    - 后付费服务 - 可能会暂时继续，余额未结清
    - 预付费服务 - 当余额耗尽时暂停服务
    - 自动续订服务 - 续订失败，服务可能过期

后续行动：

第**1-3**天（宽限期）：

- 服务正常继续
- 客户收到第一次付款失败通知

- 系统尝试重试（根据配置）

第**4-7**天：

- 第二次付款尝试（如果配置）
- 发送警告电子邮件
- 客户服务可能联系客户

第**8-14**天：

- 由于未付款，服务可能会暂停
- 暂停状态阻止使用，但保留帐户
- 客户可以通过更新付款方式和支付未结余额恢复服务

第**15**天及以后：

- 由于未付款，服务可能会终止
- 库存（SIM卡、设备）标记为退回
- 发送最终通知
- 帐户转交给催收（如适用）

防止服务中断：

为避免服务中断：

- 在到期前30天更新卡
- 添加多种付款方式以备份
- 启用付款失败警报
- 每周监控即将到期的卡仪表板

到期后恢复服务：

如果由于卡过期而暂停服务：

1. 添加新的有效付款方式
2. 将其设为默认
3. 支付未结余额（如有）
4. 联系支持以重新激活服务

5. 服务在几分钟到几小时内恢复

付款方式安全

令牌化

OmniCRM使用供应商令牌化以确保安全:

1. 客户输入卡 → 直接发送到支付供应商服务器
2. 供应商验证并令牌化 → 创建唯一令牌
3. 令牌存储在**OmniCRM**中 → 从未存储完整的卡号
4. 付款处理 → 令牌发送给供应商，供应商收费

OmniCRM存储的内容:

```
{
  "vendor": "stripe",
  "vendor_payment_method_id": "pm_1A2B3C4D5E6F",
  "payment_type": "card",
  "brand": "visa",
  "last4": "1234",
  "exp_month": 12,
  "exp_year": 2026,
  "name": "John Smith",
  "nickname": "个人卡",
  "is_default": true
}
```

OmniCRM不存储的内容:

- 完整的卡号
- CVV/CVC代码
- 磁条数据
- PIN号码

PCI合规性

通过使用供应商托管的付款表单：

- 减少PCI范围 - 卡数据从未接触OmniCRM服务器
- 供应商托管字段 - 卡输入发生在供应商的安全iframe中
- 不存储卡 - 使用令牌代替原始卡数据
- 安全传输 - 所有通信通过HTTPS/TLS进行

请参见 付款系统指南 `<payment_system_guide>` 以获取支付供应商安全详细信息。

常见工作流程

工作流程1：客户添加第一张付款方式

场景：新客户注册服务

1. 客户创建帐户
2. 选择服务计划
3. 在结账时提示添加付款方式
4. 在Stripe模态框中输入卡详细信息
5. 卡令牌化并保存
6. 自动设为默认
7. 服务提供
8. 处理第一次收费

工作流程2：

账单选项卡概述

账单选项卡提供了客户所有财务信息的统一视图，将支付方式、交易和发票结合到一个单一界面中，以实现高效的账单管理。

相关文档：`支付方式 <payment_methods>`，`交易 <payments_transaction>`，`发票 <payments_invoices>`，`支付处理 <payments_process>`，`钱包与□户交易分类账 <billing_wallet_ledger>`。

访问账单选项卡

按客户查看：

1. 导航到 客户 → 【选择客户】
2. 点击 账单 选项卡
3. 查看所有三个部分：支付方式、交易和发票

系统范围视图：

系统范围的账单数据可以单独访问：

- 账单 → 交易 - 所有客户的所有交易
- 账单 → 发票 - 所有客户的所有发票

自助服务门户：

访问 `自助服务门户 <self_care_portal>` 的客户看到相同的账单选项卡结构：

- 查看和管理他们的支付方式
- 查看交易历史
- 在线查看和支付发票

账单选项卡结构

账单选项卡分为三个主要部分，以卡片形式显示：

部分 1：支付方式

目的：管理客户支付服务的方式

主要特点：

- 查看所有保存的信用卡
- 设置默认支付方式
- 添加新支付方式（通过 Stripe）
- 删除过期或未使用的卡

文档：`basics_payment`

部分 2：交易

目的：跟踪客户的所有费用和信用

主要特点：

- 查看交易统计（总计、未开票、本月、上月）
- 列出所有交易，并按作废/开票状态过滤
- 添加手动交易（费用或信用）
- 作废不正确的交易
- 查看哪些交易已开票与未开票

文档：`payments_transaction`

部分 3：发票

目的：将交易分组为客户支付的账单

主要特点：

- 查看发票统计（总计、未支付、本月、上月）
- 列出所有发票，并按已支付/作废状态过滤
- 从未开票交易生成新发票
- 下载发票 PDF
- 将发票通过电子邮件发送给客户
- 在线支付发票（Stripe 或手动支付方式）

- 作废或退款发票

文档：`payments_invoices`

部分之间的数据流

理解三个部分之间的数据流对于有效的账单管理至关重要。

流程图

交易 → 发票 关系

1. 交易创建：

当服务被提供或添加手动费用时：

- 在 交易 部分创建交易
- 交易状态：未开票
- 交易的 `invoice_id` 字段为 `null`

示例：

2. 发票生成：

当工作人员生成发票时：

- 在 发票 部分创建发票
- 在日期范围内的所有未开票交易分组到发票中
- 交易的 `invoice_id` 字段被填充
- 交易状态更改为：已开票

示例：

3. 交易统计更新：

- 未开票交易 总数减少
- 总发票 统计增加
- 未支付发票 总数增加

发票 → 支付 关系

1. 发票支付：

当客户支付发票时：

- 使用保存的 支付方式（Stripe 卡）处理支付
- 或选择手动支付方式（现金、POS、银行转账）
- 发票状态更改为：已支付

2. 创建支付交易：

对于手动支付（非 Stripe）：

- 自动创建负交易
- 交易标题：“发票 #1234 的支付”
- 交易金额： -$45.00（负数，给客户信用）
- 交易的 `invoice_id` 字段：链接到已支付的发票

示例：

3. 统计更新：

- 未支付票 总数减少
- 本月总发票 不变（发票已经存在）

支付方式 → 发票 关系

Stripe 支付流程：

1. 客户在 支付方式 中添加信用卡
2. 卡通过 Stripe 进行令牌化，安全存储
3. 支付发票时，客户选择保存的卡
4. Stripe 收取卡费
5. 发票标记为已支付
6. `payment_reference` 字段填充 Stripe 支付意图 ID

手动支付流程：

1. 客户通过现金/POS/银行转账支付（不需要支付方式）
2. 工作人员在支付发票模态中选择支付方式
3. 工作人员输入参考号码（可选）
4. 为支付金额创建负交易
5. 发票标记为已支付

完整的账单工作流程

这些工作流程演示了三个部分如何协同工作以完成常见任务。

工作流程 1：新客户设置和第一张发票

目标：为新客户设置账单并收取第一笔付款

1. 添加支付方式：
    - 导航到客户 → 账单 选项卡
    - 支付方式 部分 → 点击 "添加支付方式"
    - 客户通过 Stripe 添加信用卡

- 卡保存为默认支付方式

2. 验证交易：
   - 交易 部分显示未开票交易：
     - 服务设置费：$50.00
     - 第一个月服务：$45.00
     - 未开票总计：$95.00
3. 生成发票：
   - 发票 部分 → 点击 "生成形式发票"
   - 设置日期范围以包括设置和第一个月
   - 点击 "生成发票"
   - 发票 #INV-2025-001234 创建，金额为 $95.00
4. 交易更新：
   - 两个交易现在显示：发票 #INV-2025-001234
   - 未开票交易 总计现在为 $0.00
5. 发送发票电子邮件：
   - 点击发票旁边的电子邮件图标
   - 客户收到包含 PDF 和支付链接的发票电子邮件
6. 客户在线支付：
   - 客户点击电子邮件中的支付链接
   - 被重定向到自助服务门户
   - 点击 "支付发票" 按钮
   - 选择默认支付方式
   - 点击 "支付发票"
   - Stripe 收取卡费
7. 发票更新：
   - 发票状态更改为 "已支付"
   - 未支付发票 总数减少 $95.00

结果：客户完全设置了支付方式，第一张发票已支付。

工作流程 2：每月定期账单

目标：在月底向所有客户收取每月服务费用

1. 服务自动收费：

- 月底到来（1 月 31 日）
- 账单系统自动为所有定期服务创建交易
- 交易 部分显示新的未开票交易

2. 查看未开票交易：
   - 导航到 交易 部分
   - 过滤：发票状态：未开□□□
   - 查看所有准备开票的交易列表
   - 验证金额和描述□否正确
3. 生成发票：
   - 导航到 账单 → 发票（系统范围）
   - 对每个客户（或使用批处理）：
     - 点击 "生成形式发票"
     - 选择客户
     - 开始日期：2025-01-01
     - 结束日期：2025-01-31
     - 到期日期：2025-02-15
     - 点击 "生成发票"
4. 交易更新：
   - 所有交易现在链接到发票
   - 未开票交易 总数重置为 $0.00
5. 发送所有发票电子邮件：
   - 对每个发票，点击电子邮件图标
   - 所有客户收到每月发票
6. 客户支付：
   - 使用保存的支付方式的客户通过自助服务在线支付
   - 工作人员为亲自支付的客户处理现金/POS 支付
   - 未支付发票 总数随着收到的付款而减少

结果：所有客户在 1 月份开票，发票已发送，付款已处理。

工作流程 3：处理服务问题信用

目标：因服务中断而给予客户信用，应用于未支付的发票

1. 客户报告问题：

- 服务停机 2 天
- 客户应获得 $10 信用

2. 添加信用交易：
   - 导航到客户 → 账单 选项卡 → 交易 部分
   - 点击 "+ 添加交易"
   - 交易类型：信用
   - 信用类型：现金支付（或适当类型）
   - 标题：“服务中断信用”
   - 描述：“1 月 8-9 日的 2 天停机补偿”
   - 零售成本：10.00
   - 点击 "添加交易"
3. 交易创建：
   - 交易以金额 **-$10.00** 出现在列表中
   - 交易状态：未开票
   - 未开票交易 总数现在包括 -$10.00
4. 应用于发票：
   - 如果客户已经有未支付的发票：
     - 发票保持未支付，金额不变
     - 信用将在下次发票生成时应用
   - 如果生成新发票：
     - 发票 部分 → 点击 "生成形式发票"
     - 包括包含信用交易的日期范围
     - 生成的发票应用了信用：
5. 客户支付：
   - 客户支付减少的金额：$35.00
   - 发票标记为已支付

结果：客户因停机获得信用，信用应用于下次发票，收取较低的付款。

工作流程 4：支付方式过期 - 更新并重试

目标：客户的卡过期，导致支付失败 - 更新卡并重试支付

1. 支付失败通知：
   - 客户尝试支付发票

- Stripe 返回错误：“卡过期”
- 支付失败，发票保持未支付

2. 更新支付方式：
   - 客户导航到 账单 选项卡
   - 支付方式 部分 → 点击 "添加支付方式"
   - 输入新卡信息（更新的到期日期）
   - 新卡保存
3. 设置为默认：
   - 客户点击新卡上的 "设置为默认"
   - 旧卡自动删除（如果需要）
4. 重试支付：
   - 导航到 发票 部分
   - 找到未支付的发票
   - 点击 "支付" 图标
   - 支付模态打开，新的默认卡预先选择
   - 点击 "支付发票"
   - Stripe 成功收取新卡费用
5. 发票更新：
   - 发票状态更改为 "已支付"
   - `payment_reference` 字段填充新的 Stripe 支付意图 ID

结果：客户更新了支付方式，发票成功用新卡支付。

工作流程 5：作废不正确的发票并重新开票

目标：工作人员生成了错误交易的发票 - 作废并正确重新生成

1. 发现错误：
   - 发票 #INV-2025-001234 生成了错误的日期范围
   - 包含了错误月份的交易
   - 发票未支付
2. 作废发票：
   - 导航到 账单 选项卡 → 发票 部分
   - 找到不正确的发票
   - 点击删除图标 (🗑)

- 确认作废
- 发票已作废

3. 交易释放：
   - 导航到 交易 部分
   - 所有来自作废发票的交易现在显示为：未开票
   - 未开票交易 总数增加
   - 交易可用于新发票
4. 生成正确的发票：
   - 发票 部分 → 点击 "生成形式发票"
   - 设置正确的日期范围
   - 如有需要，应用过滤器（例如，“移动”用于仅移动的发票）
   - 点击 "生成发票"
   - 创建了包含正确交易的新发票
5. 验证并发送电子邮件：
   - 审查新发票详情
   - 验证包含的交易□否正确
   - 点击电子邮件图标发送给客户

结果：不正确的发票已作废，交易已正确重新开票，客户收到更正的发票。

工作流程 6：处理多张发票的现金支付

目标：客户用一笔现金支付多张未支付的发票

1. 客户带着现金到达：
   - 客户带来 $300 现金以支付未结发票
   - 导航到客户 → 账单 选项卡
2. 查看未支付的发票：
   - 发票 部分 → 过滤：已支付：尚未支付
   - 查看未支付的发票：
3. 支付第一张发票：
   - 点击发票 #1234 上的支付图标
   - 支付模态打开
   - 选择 "现金" 支付方式
   - 输入参考：“现金支付 2025-02-10 - 收据 #001”

    - 点击 "支付发票"
    - 发票 #1234 标记为 "已支付"
4. 支付其余发票：
    - 对发票 #1235 重复该过程：
        - 参考："现金支付 2025-02-10 - 收据 #001"
    - 对发票 #1236 重复：
        - 参考："现金支付 2025-02-10 - 收据 #001"
5. 验证交易：
    - 导航到 交易 部分
    - 创建三条新的支付交易：
    - 所有交易链接到各自的发票
6. 更新统计：
    - 发票 部分 → 未支付发票 总数减少 $300.00
    - 所有发票现在显示 "已支付" 状态

结果：客户用现金支付了所有未结发票，支付交易记录了收据参考。

最佳实践

对于工作人员用户

交易管理：

- 立即添加手动交易（不要延迟）
- 使用描述性标题和描述以确保清晰
- 在适用时将交易链接到服务和站点
- 在开票前作废不正确的交易

发票生成：

- 在一致的时间间隔生成发票（例如，每月的第 1 天）
- 小心使用日期范围以避免重叠或间隙
- 在需要时使用过滤器创建特定服务的发票
- 在生成后立即发送发票电子邮件
- 在发送给客户之前审查发票 PDF

支付处理：

- 在尝试收费之前验证支付方式有效
- 对于手动支付（现金、POS、银行转账），始终输入参考号码
- 在收到付款后立即将发票标记为已支付
- 仅通过 Stripe 退款（为其他支付方式创建手动信用）

数据卫生：

- 定期审查未开票交易
- 调查作废交易以了解账单错误
- 监控未支付发票并跟进客户
- 保持支付方式最新（删除过期卡）

对于客户（自助服务门户）

支付方式：

- 保持至少一种有效的支付方式在档案中
- 在卡过期之前更新支付方式
- 将您首选的卡设置为默认

发票支付：

- 在到期日前支付发票以避免滞纳金
- 在支付之前查看发票详情和交易
- 下载发票 PDF 以备记录
- 如果发票看起来不正确，请立即联系支持

交易审查：

- 定期查看交易历史
- 立即报告任何意外收费
- 了解哪些交易已开票与未开票

对于管理员

系统配置：

- 配置 Mailjet 电子邮件模板以专业方式发送发票
- 自定义发票 PDF 模板以匹配品牌
- 设置支付供应商集成（Stripe、PayPal）以安全处理支付
- 配置支付条款和到期日期

监控和报告：

- 使用统计小部件监控账单健康
- 跟踪 未开票交易 总数 - 应在账单周期后减少
- 监控 未支付发票 总数 - 跟进逾期付款
- 审查 本月 与 上月 的统计数据以识别趋势

自动化：

- 通过产品配置自动化定期服务收费
- 设置定期账单的自动发票生成（如果可用）
- 配置逾期发票的电子邮件提醒

常见问题及解决方案

问题：客户无法支付发票

症状：

- 客户点击支付按钮但没有反应
- 错误消息：“未找到支付方式”

诊断：

1. 导航到客户 → 账单 选项卡 → 支付方式 部分
2. 检查客户□否有任何保存的支付方式
3. 检查保存的卡□否过期

解决方案：

- 客户必须在支付发票之前添加有效的支付方式
- 指导客户前往 支付方式 页面以添加信用卡
- 验证卡□否被接受（Visa、Mastercard、Amex 等）
- 在添加卡后重试支付

问题：发票生成了错误的交易

症状：

- 发票包含错误期间的交易
- 发票缺少预期的交易
- 发票总额不正确

诊断：

1. 在 发票 部分打开发票
2. 审查发票中包含的交易
3. 检查交易日期与发票日期范围
4. 检查生成时□否应用了过滤器

解决方案：

- 如果发票未支付：作废发票，验证交易未开票，使用正确的日期范围重新生成
- 如果发票已支付：不能作废 - 为错误金额创建信用交易，生成更正的发票
- 预防：在生成发票之前始终审查 交易 部分以验证将包含正确的交易

问题：未开票交易总数未减少

症状：

- 未开票交易 小部件显示高金额
- 交易列表显示许多未开票交易
- 已生成每月发票

诊断：

1. 按 发票状态：未开票 过滤交易
2. 查看未开票交易列表
3. 检查交易日期 - 可能□上次发票生成后最近的收费
4. 检查□否有一些交易被作废（不应计入未开票总数）

解决方案：

- 预期行为：在上次发票生成后创建的交易在下一个账单周期之前保持未开票
- 如果旧交易未开票：为这些交易生成发票并使用适当的日期范围
- 如果作废的交易被计入：系统应自动排除作废的交易 - 如果没有，请报告错误

问题：支付方式不断被拒绝

症状：

- Stripe 支付失败，显示“卡被拒绝”
- 客户报告卡应该可以使用
- 多次重试失败

诊断：

1. 检查 Stripe 仪表板以获取拒绝原因
2. 常见拒绝原因：
    - 资金不足
    - 卡过期
    - 卡被报告为丢失/被盗
    - 银行欺诈预防
    - 国际卡被阻止

解决方案：

- 请客户联系他们的银行以授权支���
- 尝试不同的支付方式（不同的卡）
- 对于工作人员：将其作为手动支付（现金、POS）处理并标记发票为已支付
- 验证账单地址与卡的账单地址□否匹配

快捷键和提示

导航：

- 快速访问账单选项卡：客户页面 → **B** 键（如果启用了键盘快捷键）

过滤：

- 交易：在生成发票之前点击 "未开票" 过滤器，以查看将包含的内容
- 发票：点击 "尚未支付" 过滤器以查看所有需要跟进的未支付发票

批量□作：

- 选择多个发票（复选框）以一次性删除/作废多个（仅限工作人员）
- 使用搜索栏快速按 ID 查找特定交易或发票

快速□作：

- 点击交易/发票标题以在模态中查看完整详情
- 右键点击操作菜单（⋮）以快速访问下载/电子邮件/支付选项

相关文档

- `basics_payment` - 支付方式和多供应商支付集成
- `payment_system_guide` - 支付 API 参考和供应商配置
- `payments_transaction` - 交易详细信息
- `payments_invoices` - 发票详细信息
- `integrations_mailjet` - 电子邮件发票发送
- `csa_activity_log` - 查看账单活动历史
- 监控与指标 - OCS/CGRateS 账单系统指标和监控

CGRateS □作和充值行为

本指南解释了 CGRateS 操作在 OmniCRM 中的工作原理，特别关注余额管理、充值行为以及不同操作类型如何影响附加产品。

概述

在 OmniCRM 的在线计费系统 (CGRateS) 中，□作 □在客户账户上添加、修改或删除余额的机制。

当您提供附加或充值产品时，实际上□在执行一个 CGRateS 操作，该操作会操纵账户的余额。（您也可以采用其他方法，例如通过 Playbooks 手动向账户添加余额 - 这只□我们用来保持整洁的常见模式）

关键概念

□作 - 在账户上执行的一组操作（添加余额、扣除余额、记录 CDR 等）

余额 - 一种资源（数据、语音、短信、货币）的数量，具有到期时间和权重

余额 ID - 余额类型的唯一标识符（例如，“数据包”、“语音分钟”）

权重 - 余额消耗的优先级（先消耗权重较高的）

到期时间 - 余额到期的时间（绝对日期或相对时间，如“+5 天”）

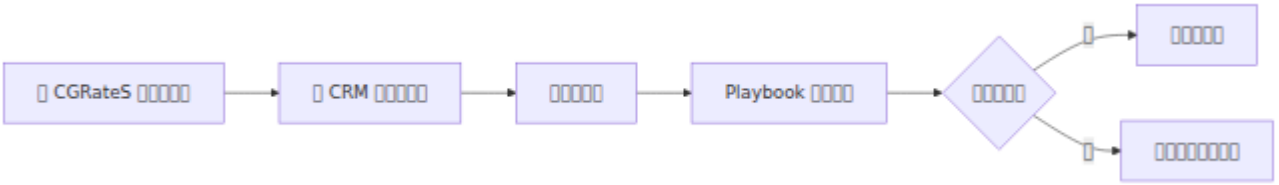
阻止器 - 一种特殊的余额标志，当余额达到零时阻止所有使用，即使存在其他余额（见 余额阻止器）

关键：□作必须先定义

在您可以使用 `ExecuteAction` 在 **Playbook** 中执行□作之前，该操作必须已经在 CGRateS 中定义。这□一个常常被忽视的关键前提。

□作何时定义

操作通常在 初始系统配置 或 产品设置 期间定义，而不□在提供服务时。它们通常通过 Python 脚本创建，同时配置 CRM 和 OCS。



□作如何链接到产品

操作通过命名约定与产品链接：

- **CGRateS** □作：`ActionsId = "Action_50gb-data-pack"`
- **CRM** 产品：`product_slug = "50gb-data-pack"`
- 在 **Playbook** 中：`cgr_action_name = "Action_" + product_slug`

当 Playbook 运行时，它从 product_slug 构建操作名称并调用 `ExecuteAction`。如果该操作在 CGRateS 中不存在，则提供失败。

在哪里定义□作

操作应在您的产品配置脚本中定义，通常在：

1. 初始设置期间 - 在首次配置系统时
2. 创建新产品时 - 在创建产品之前定义操作
3. 通过配置脚本 - 配置 OCS 和 CRM 的 Python 脚本

示例：在创建产品之前定义□作

```
import cgrateshttpapi

OCS_Obj = cgrateshttpapi.CGRateS("ocs.example.com", "2080")
tenant = "your_tenant"

# 步骤 1: 首先在 CGRateS 中定义操作
Action_50GB_Data_Pack = {
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_50gb-data-pack",
        "Tenant": tenant,
        "Actions": [
            {
                "Identifier": "*topup",
                "BalanceType": "*data",
                "Units": 50 * 1024 * 1024 * 1024,
                "ExpiryTime": "+720h",
                "Weight": 10
            }
        ]
    }]
}

result = OCS_Obj.SendData(Action_50GB_Data_Pack)
assert result['error'] is None or result['error'] == "EXISTS"

# 步骤 2: 现在在 CRM 中创建产品
# (product_slug = "50gb-data-pack" 将链接到 Action_50gb-data-pack)
```

如果□作不存在会发生什么：

```
# 在 Playbook 中
- name: 执行操作
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body:
      {
        "method": "APIerSv1.ExecuteAction",
        "params": [{
          "ActionsId": "Action_50gb-data-pack"
        }]
      }
  register: response

# 如果操作未定义的结果:
# response.json.error = "SERVER_ERROR: 未找到操作"
# 提供失败，客户未□得余额
```

有关定义操作和将其链接到产品的完整详细信息，请参见 定义产品。

余额独立性

默认情况下，附加产品创建 独立余额，这些余额彼此独立工作。这意味着：

- 您可以同时拥有多个活动附加产品
- 每个附加产品维护自己的余额和到期时间
- 余额根据权重和到期规则进行消耗

示例：多个独立附加产品

```
# 客户拥有这些活动余额：
余额 1:
  ID: "Data_5GB_5days_uuid_abc123"
  类型: *data
  值: 5368709120  # 5GB (字节)
  到期: 2024-12-29
  权重: 10

余额 2:
  ID: "Data_10GB_30days_uuid_def456"
  类型: *data
  值: 10737418240  # 10GB (字节)
  到期: 2025-01-24
  权重: 10

# 两个余额独立共存
# 系统根据权重消耗；如果权重相等，则不保证顺序
```

当余额具有相同的权重并匹配相同的目的地时，CGRateS 不保证基于到期的消耗顺序 - 顺序取决于余额的存储和检索方式。使用不同的权重来控制消耗顺序。

作类型： *topup 与 *topup_reset

操作类型定了新余额如何与相同 ID 的现有余额交互。

*topup - 加法行为

`*topup` 操作 添加到现有余额，并 延长到期时间。

行为：

- 查找具有匹配 ID 的现有余额
- 将新值添加到现有值
- 更新到期为新的到期时间
- 保留现有余额（滚存）

示例：

```
# 初始状态：
余额：
  ID: "Data_Package__5368709120"
  值：1073741824  # 剩余 1GB
  到期：2024-12-24（剩余 1 天）

# 运行 *topup 操作：
操作：
  标识符："*topup"
  余额：
    ID: "Data_Package__5368709120"  # 相同 ID - 触发滚存
    值：5368709120  # 5GB
    ExpiryTime: "+5d"

# *topup 后的结果：
余额：
  ID: "Data_Package__5368709120"
  值：6442450944  # 6GB（1GB + 5GB 滚存）
  OriginalValue: 5368709120  # 仍然显示原始 5GB
  Value_hr: "6 GB"
  OriginalValue_hr: "5 GB"
  Remaining_hr: "6 GB of 5 GB (1 GB rolled over)"
  到期：2024-12-29（从现在起 5 天）
```

用例：

- 忠诚奖励（向现有套餐添加额外数据）
- 补偿信用（向客户余额添加）
- 滚存数据包
- 宽限期延长

*topup_reset - 重置行为

`*topup_reset` 操作 替换现有余额，具有相同的余额 ID。

行为：

- 查找具有匹配 ID 的现有余额

- 丢弃旧值（��滚存）
- 将余额设置为新值
- 更新到期为新的到期时间

示例：

```
# 初始状态：
余额：
  ID: "Data_Package__5368709120"
  值: 1073741824  # 剩余 1GB
  到期: 2024-12-24（剩余 1 天）

# 运行 *topup_reset 操作：
操作：
  标识符: "*topup_reset"
  余额：
    ID: "Data_Package__5368709120"  # 相同 ID - 触发重置
    值: 5368709120  # 5GB
    ExpiryTime: "+5d"

# *topup_reset 后的结果：
余额：
  ID: "Data_Package__5368709120"
  值: 5368709120  # 5GB（旧的 1GB 被丢弃）
  OriginalValue: 5368709120
  Value_hr: "5 GB"
  Remaining_hr: "5 GB of 5 GB"
  PercentUsed: 0
  到期: 2024-12-29（从现在起 5 天）
```

用例：

- 每月定期套餐（每月重置为全额）
- 固定大小的充值（始终获得确切金额）
- 计划变更（用新计划余额替换旧计划）
- 防止滥用（无法堆叠无限附加产品）

使用余额 ID 控制余额行为

余额 ID 对于确定余额□独立的还□相互交互至关重要。

余额 ID 命名约定和可读视图

OmniCRM 使用特定的余额 ID 命名约定，编码描述性名称和原始大小。这允许 API 自动生成用于 Web UI 的可读字段。

余额 ID 模式：

```
{描述性名称}___{原始大小（基本单位）}
```

示例余额 ID：

```
# 数据余额：100GB
"AU_Data_Domestic__107374182400"
# 分解为：
# - 描述部分："AU_Data_Domestic"（它是什么 - 类型/目的地）
# - 分隔符："__"（双下划线）
# - 原始大小："107374182400"（100GB 以字节为单位）
# - UI 显示："AU Data Domestic - 100 GB"

# 语音余额：3000分钟
"AU_Voice_Domestic__1800000000000000"
# - 描述部分："AU_Voice_Domestic"（而不是 "AU_Voice_Domestic_3000min"）
# - 原始大小："1800000000000000"（3000分钟以纳秒为单位）
# - UI 显示："AU Voice Domestic - 3000 min"

# 短信余额：3000条消息
"AU_SMS_Domestic__3000"
# - 描述部分："AU_SMS_Domestic"
# - 原始大小："3000"（计数）
# - UI 显示："AU SMS Domestic - 3000 msgs"
```

重要：不要在描述部分中包含大小信息 - 这□多余的，因为大小在 __ 后编码，API 会自动将其转换为可读格式。

API 如何创建可读视图：

当 OmniCRM API 从 CGRateS 检索余额数据时，它会自动解析余额 ID 并生成 _hr（人类可读）字段：

```
{
  "BalanceMap": {
    "*data": [
      {
        "ID": "AU_Data_Domestic__107374182400",
        "Value": 53687091200,
        "ExpiryTime": "2025-01-25T23:59:59Z",
        "Weight": 1200,

        // 自动生成的人类可读字段:
        "ID_hr": "AU Data Domestic",
        "OriginalValue": 107374182400,
        "OriginalValue_hr": "100 GB",
        "Value_hr": "50 GB",
        "Remaining_hr": "50 GB of 100 GB",
        "PercentUsed": 50,
        "ExpiryTime_hr": "2025年1月25日 (22天) "
      }
    ]
  }
}
```

API 处理逻辑:

1. 解析余额 **ID**:

```
balance_id = "AU_Data_Domestic__107374182400"
parts = balance_id.split("__")

descriptive_name = parts[0]  # "AU_Data_Domestic"
original_size = int(parts[1]) if len(parts) > 1 else None  # 107374182400
```

2. 生成可读的描述性名称:

```
# 用空格替换下划线
id_hr = descriptive_name.replace("_", " ")  # "AU Data Domestic"
```

3. 将原始大小转换为人类单位：

```
# 对于数据余额（字节）
if balance_type == "*data":
    original_value_hr = convert_bytes_to_gb(original_size)  # "100 GB"

# 对于语音余额（纳秒）
elif balance_type == "*voice":
    original_value_hr = convert_ns_to_minutes(original_size)  # "3000 min"

# 对于短信余额（计数）
elif balance_type == "*sms":
    original_value_hr = f"{original_size} msgs"  # "3000 msgs"
```

4. 计算使用百分比：

```
if original_size and original_size > 0:
    percent_used = ((original_size - current_value) / original_size) * 100
```

5. 格式化剩余显示：

```
remaining_hr = f"{current_value_hr} of {original_value_hr}"
# "50 GB of 100 GB"
```

Web UI 显示：

前端使用这些 `_hr` 字段来显示用户友好的余额信息：

```
// 而不是显示原始值:
// ID: "AU_Data_Domestic__107374182400"
// 值: 53687091200

// 显示人类可读:
<BalanceCard>
  <Title>{balance.ID_hr}</Title>           {/* "AU Data Domestic" */}
  <Progress value={balance.PercentUsed}>   {/* 50% */}
    {balance.Remaining_hr}                 {/* "50 GB of 100 GB" */}
  </Progress>
  <Expiry>{balance.ExpiryTime_hr}</Expiry> {/* "2025年1月25日（22天）" */}
</BalanceCard>
```

为什么这很重要:

1. 原始大小跟踪 - 即使余额部分消耗，UI 也可以显示 "50 GB of 100 GB"，而不仅仅□ "50 GB 剩余"
2. 进度可视化 - 百分比计算使准确的进度条成为可能
3. 一致的命名 - 从余额 ID 中提取的描述性名称确保后端和前端之间的一致性
4. 滚存显示 - 当使用 `*topup`（滚存）时，如果客户有 70 GB 剩余并充值 100 GB:
   - 余额 ID 保持不变: `"AU_Data_Domestic__107374182400"`（原始 100 GB）
   - 当前值变为: 170 GB
   - UI 显示: "170 GB (70 GB rolled over + 100 GB new)"

最佳实践 - 创建余额 ID:

始终在 `__` 后包含原始大小，以便正确的 UI 显示。不要在描述名称中重复大小信息:

```
# 好 - 描述名称 + 基本单位中的大小
Action_Data_100GB = {
    "Actions": [
        {
            "BalanceId": f"AU_Data_Domestic__{100 * 1024 * 1024 *
1024}",
            "Units": 100 * 1024 * 1024 * 1024
        }
    ]
}

# 坏 - 描述名称中冗余大小
Action_Data_100GB = {
    "Actions": [
        {
            "BalanceId": f"AU_Data_Domestic_100GB__{100 * 1024 *
1024 * 1024}",  # 冗余!
            "Units": 100 * 1024 * 1024 * 1024
        }
    ]
}

# 坏 - 没有大小信息（UI 无法计算百分比）
Action_Data_100GB = {
    "Actions": [
        {
            "BalanceId": "AU_Data_Domestic",  # 缺少 __size
            "Units": 100 * 1024 * 1024 * 1024
        }
    ]
}
```

特殊情况 - 货币余额：

货币余额通常不包括原始大小，因为它们可以充值到任何金额：

```
# 没有大小编码的货币余额
{
    "BalanceId": "PAYG_Monetary_Balance",
    "BalanceType": "*monetary",
    "Units": 5000  # $50.00
}

# UI 仅显示当前余额，而不显示百分比
# "余额: $50.00"
```

策略 1：唯一 ID（独立余额）

使用唯一 ID（例如，带有 UUID）创建完全独立的余额，这些余额永远不会相互交互。

示例实现：

```yaml
- name: 生成唯一余额标识符
  set_fact:
    uuid: "{{ 99999999 | random | to_uuid }}"
    balance_id: "Data_5days__5368709120_{{ uuid[0:8] }}"

- name: 使用 *topup 添加独立余额
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Balance": {
            "ID": "{{ balance_id }}",
            "Value": 5368709120,
            "ExpiryTime": "+5d",
            "Weight": 10
          }
        }]
      }
```

结果：每个附加产品即使客户多次购买相同的附加产品，也会创建一个新的、独立的余额。

```
# 客户购买 "5 天数据附加产品" 三次:
余额 1:
  ID: "Data_5days__5368709120_a1b2c3d4"
  值: 5368709120
  Value_hr: "5 GB"
  OriginalValue_hr: "5 GB"
  Remaining_hr: "5 GB of 5 GB"
  到期: 2024-12-29

余额 2:
  ID: "Data_5days__5368709120_e5f6g7h8"
  值: 5368709120
  Value_hr: "5 GB"
  Remaining_hr: "5 GB of 5 GB"
  到期: 2024-12-30

余额 3:
  ID: "Data_5days__5368709120_i9j0k1l2"
  值: 5368709120
  Value_hr: "5 GB"
  Remaining_hr: "5 GB of 5 GB"
  到期: 2024-12-31

# 总可用: 15GB, 分为三个独立余额
# 每个在 UI 中显示为 "Data 5days - 5 GB of 5 GB"
```

策略 2: 共享 ID 与 *topup (滚存)

使用相同的余额 ID 和 `*topup` 操作，允许余额滚存和到期延长。

示例实现:

```
- name: 设置固定余额 ID
  set_fact:
    balance_id: "Data_5days__5368709120"

- name: 使用 *topup 添加余额（滚存）
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Balance": {
            "ID": "Data_5days__5368709120",
            "Value": 5368709120,
            "ExpiryTime": "+5d",
            "Weight": 10
          }
        }]
      }
```

结果：后续购买将添加到现有余额并延长到期。

```
# 第 1 天：客户购买 "5 天数据附加产品"：
余额：
  ID: "Data_5days__5368709120"
  值: 5368709120
  Value_hr: "5 GB"
  Remaining_hr: "5 GB of 5 GB"
  PercentUsed: 0
  到期: 2024-12-29

# 第 3 天：客户使用 1GB，然后再次购买相同的附加产品：
余额：
  ID: "Data_5days__5368709120"
  值: 9663676416  # 4GB 剩余 + 5GB 新增
  Value_hr: "9 GB"
  OriginalValue_hr: "5 GB"
  Remaining_hr: "9 GB (4 GB rolled over + 5 GB new)"
  PercentUsed: -80  # 负值表示滚存
  到期: 2024-12-27（从今天起 +5 天）
```

策略 3：共享 ID 与 *topup_reset（固定金额）

使用相同的余额 ID 和 `*topup_reset` 操作，始终重置为固定金额。

示例实现：

```yaml
- name: 设置固定余额 ID
  set_fact:
    balance_id: "Monthly_Plan__32212254720"

- name: 使用 *topup_reset 添加余额（无滚存）
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.SetActions",
        "params": [{
          "ActionsId": "Action_Reset_Monthly_Plan",
          "Actions": [{
            "Identifier": "*topup_reset",
            "BalanceType": "*data",
            "Units": 32212254720,  # 30GB
            "ExpiryTime": "*monthly",
            "DestinationIds": "*any",
            "BalanceId": "Monthly_Plan__32212254720",
            "Weight": 10
          }]
        }]
      }
```

结果：每个月，余额重置为确切的 30GB，无论使用了多少。

```
# 第 1 个月，第 1 天:
余额:
  ID: "Monthly_Plan__32212254720"
  值: 32212254720
  Value_hr: "30 GB"
  Remaining_hr: "30 GB of 30 GB"
  PercentUsed: 0
  到期: 2024-12-31

# 第 1 个月，第 25 天: 客户使用了 28GB
余额:
  ID: "Monthly_Plan__32212254720"
  值: 2147483648
  Value_hr: "2 GB"
  Remaining_hr: "2 GB of 30 GB"
  PercentUsed: 93
  到期: 2024-12-31

# 第 2 个月，第 1 天: ActionPlan 运行 *topup_reset
余额:
  ID: "Monthly_Plan__32212254720"
  值: 32212254720  # 重置为全额，未使用的 2GB 丢失
  Value_hr: "30 GB"
  Remaining_hr: "30 GB of 30 GB"
  PercentUsed: 0
  到期: 2025-01-31
```

余额阻止器

余额阻止器 CGRateS 的一个强大功能，允许您 阻止或限制使用，即使存在余额。当余额达到零时，阻止器余额停止消费，阻止进一步使用，无论否存在其他可用余额。

阻止器的工作原理

当余额具 `Blocker: true` 时:

1. 当阻止器有值时 → 允许使用，直到阻止器金额
2. 当阻止器达到零时 → 所有使用停止，即使存在其他余额
3. 返回错误 → `INSUFFICIENT_CREDIT_BALANCE_BLOCKER` 阻止会话

关键特征：

- 阻止器余额即使在值为零时也会被 检查（与正常余额不同，正常余额会被跳过）
- 当遇到具有剩余使用请求的阻止器时，CGRateS 停止处理 并返回错误
- 阻止器适用于所有余额类型：`*voice`、`*data`、`*sms`、`*monetary`

阻止器的用例

1. 账户暂停

当账户暂停时阻止所有使用（例如，付款失败）：

```
Action_Suspend_Account = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_suspend-account",
        "Overwrite": True,
        "Tenant": tenant,
        "Actions": [
            # 添加零值阻止器以防止所有使用
            {
                "Identifier": "*topup",
                "BalanceId": "Suspension_Blocker",
                "BalanceType": "*monetary",
                "DestinationIDs": "*any",
                "Units": 0,  # 零值
                "BalanceWeight": 9999,  # 最高优先级 - 首先检查
                "Blocker": True,  # 阻止所有使用
                "Weight": 10
            }
        ]
    }]
}

result = OCS_Obj.SendData(Action_Suspend_Account)
```

结果：所有通话/数据/SMS 被阻止，无论其他余额如何。

2. 支出限制

限制最大支出以防止账单冲击：

```
Action_Monthly_Plan_With_Cap = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_monthly-with-cap",
        "Overwrite": True,
        "Tenant": tenant,
        "Actions": [
            # 10GB 包含数据
            {
                "Identifier": "*topup_reset",
                "BalanceId": f"Included_Data__{10 * 1024 * 1024 *
1024}",
                "BalanceType": "*data",
                "DestinationIDs": "Dest_PLMN_OnNet",
                "Units": 10 * 1024 * 1024 * 1024,
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,  # 首先消耗
                "Weight": 95
            },
            # $50 超支限制（阻止器）
            {
                "Identifier": "*topup_reset",
                "BalanceId": "Overage_Cap",
                "BalanceType": "*monetary",
                "DestinationIDs": "*any",
                "Units": 5000,  # $50.00 最大超支
                "ExpiryTime": "*month",
                "BalanceWeight": 1000,  # 在包含后消耗
                "Blocker": True,  # 当 $50 花费达到时停止
                "Weight": 90
            }
        ]
    }]
}
```

流程：

1. 客户使用 10GB 包含 → 免费（来自 Included_Data_10GB）
2. 客户使用额外的 5GB → 从 Overage_Cap 收费（按 PAYG 费率）

3. 当 Overage_Cap 达到 $0 → 所有使用被阻止（达到支出限制）

3. 限时免费试用

为试用账户提供有限的免费使用：

```
Action_Trial_Account = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_trial-100-minutes",
        "Overwrite": True,
        "Tenant": tenant,
        "Actions": [
            # 100 免费分钟（阻止器 - 耗尽时停止）
            {
                "Identifier": "*topup",
                "BalanceId": f"Trial_Voice__{100 * 60 *
1000000000}",
                "BalanceType": "*voice",
                "DestinationIDs": "Dest_Domestic_All",
                "Units": 100 * 60 * 1000000000,  # 100分钟
                "ExpiryTime": "+720h",  # 30天
                "BalanceWeight": 1200,
                "Blocker": True,  # 用完 100 分钟后不再使用
                "Weight": 10
            }
        ]
    }]
}
```

结果：客户获得 100 分钟免费。之后，所有通话被阻止（没有自动收费）。

4. 特定目的地阻止

在允许其他目的地的同时阻止特定目的地：

```
Action_Block_Premium_Numbers = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_block-premium",
        "Overwrite": True,
        "Tenant": tenant,
        "Actions": [
            # 允许常规使用
            {
                "Identifier": "*topup_reset",
                "BalanceId": "Regular_Usage",
                "BalanceType": "*monetary",
                "DestinationIDs": "*any",
                "Units": 10000,  # $100
                "ExpiryTime": "*month",
                "BalanceWeight": 1000,
                "Weight": 20
            },
            # 阻止高级号码（0900 等）
            {
                "Identifier": "*topup",
                "BalanceId": "Premium_Blocker",
                "BalanceType": "*monetary",
                "DestinationIDs": "Dest_Domestic_Premium",
                "Units": 0,  # 零值
                "BalanceWeight": 2000,  # 更高权重 - 首先检查高级
                "Blocker": True,
                "Weight": 10
            }
        ]
    }]
}
```

流程:

- 国内通话 → 使用 Regular_Usage 余额
- 高级号码通话 → 匹配 Premium_Blocker（权重 2000 > 1000） → 被阻止

阻止器与禁用余额的区别

不要将 `Blocker` 与 `Disabled` 混淆：

特征	阻止器	禁用
目的	当余额耗尽时停止使用	暂时暂停余额
当值 **> 0**	余额正常可用	余额被跳过/忽略
当值 **= 0**	阻止所有进一步使用	余额被跳过（尝试下一个余额）
用例	支出限制、上限、试用限制	暂时暂停特定余额

```
# 阻止器：允许 10GB，然后阻止所有使用
{
    "BalanceId": f"Data_Cap__{10 * 1024 * 1024 * 1024}",
    "Units": 10 * 1024 * 1024 * 1024,
    "Blocker": True  # 在 10GB 时停止使用
}

# 禁用：完全忽略此余额（暂时暂停）
{
    "BalanceId": f"Bonus_Data__{5 * 1024 * 1024 * 1024}",
    "Units": 5 * 1024 * 1024 * 1024,
    "Disabled": True  # 此余额将不会被使用
}
```

实际示例：混合计划与安全上限

结合单元余额、货币溢出和阻止器上限：

```
Action_Safe_Hybrid_Plan = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_safe-hybrid-plan",
        "Overwrite": True,
        "Tenant": tenant,
        "Actions": [
            {
                "Identifier": "*reset_account",
                "Weight": 700
            },
            # 500 □□□□□□
            {
                "Identifier": "*topup_reset",
                "BalanceId": f"Domestic_Voice__{500 * 60 *
1000000000}",
                "BalanceType": "*voice",
                "DestinationIDs": "Dest_Domestic_All",
                "Units": 500 * 60 * 1000000000,
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 95
            },
            # $20 □□□□
            {
                "Identifier": "*topup_reset",
                "BalanceId": "Overage_Allowance",
                "BalanceType": "*monetary",
                "DestinationIDs": "*any",
                "Units": 2000,  # $20.00
                "ExpiryTime": "*month",
                "BalanceWeight": 1000,
                "Weight": 90
            },
            # $50 □□□□□□□□
            {
                "Identifier": "*topup_reset",
                "BalanceId": "Hard_Spending_Cap",
                "BalanceType": "*monetary",
                "DestinationIDs": "*any",
                "Units": 5000,  # $50.00 □□□□
                "ExpiryTime": "*month",
```

```
                "BalanceWeight": 500,   # 低于超支 - 最后使用
                "Blocker": True,   # 达到上限时停止
                "Weight": 85
            }
        ]
    }]
}
```

客户旅程：

1. **0-500**分钟：使用 Domestic_Voice_500min（免费）
2. **500-700**分钟：使用 Overage_Allowance，每分钟 $0.10 = $20（200分钟）
3. **700-1200**分钟：使用 Hard_Spending_Cap，每分钟 $0.10 = $50（500分钟）
4. 在**1200**分钟时：Hard_Spending_Cap 耗尽 → 所有使用被阻止

客户获得总共 1200 分钟，超支最大为 $50。

最佳实践与阻止器

1. 为阻止器使用高权重

   ```
   "BalanceWeight": 9999   # 确保阻止器首先被检查
   ```

2. 零值阻止器以实现立即阻止

   ```
   "Units": 0,   # 立即阻止
   "Blocker": True
   ```

3. 在阻止器耗尽之前通知客户

   - 使用 ActionTriggers 在 80%、90%、100% 阻止器使用时发送通知
   - 在阻止发生之前给客户增加上限的选项

4. 在取消暂停时移除阻止器

```
# 使用 *remove_balance 删除阻??器
{
    "Identifier": "*remove_balance",
    "BalanceId": "Suspension_Blocker"
}
```

5. 测试阻止器行为

   - 验证阻止器返回 `INSUFFICIENT_CREDIT_BALANCE_BLOCKER` 错误
   - 确认 CDR 显示成本 = -1.0 当被阻止时
   - 测试在阻止器耗尽后其他余额不被使用

故障排除阻止器

问题：尽管阻止器为零，但使用未被阻止

可能原因：

1. 阻止器权重太低（其他余额首先检查）
2. DestinationIDs 不匹配使用目的地
3. 阻止器字段未设置为 `True`

解决方案：

```
# 验证阻止器配置
OCS_Obj.SendData({
    'method': 'ApierV2.GetAccount',
    'params': [{"Tenant": tenant, "Account": "service_uuid"}]
})

# 检查：
# - Blocker: true
# - BalanceWeight 是最高的（例如，9999）
# - DestinationIDs 包括使用目的地
# - 值 = 0
```

问题：意外阻止使用

可能原因：不小心创建了低值的阻止器余额

解决方案：检查所有余额□否有 `Blocker: true`，并验证其值□否适合您的用例。

余额消耗规则

规则 1：目的地精度优先

具有 更高目的地精度（更具体的目的地匹配）的余额首先被消耗。这□由前缀匹配长度决定的。

```
# 客户拨打 +44-20-1234-5678（伦敦，英国）

余额 1:
  DestinationIDs: "Dest_UK_London"   # 前缀："4420"（精度：4）
  值：100分钟
  权重：10

余额 2:
  DestinationIDs: "Dest_UK_All"      # 前缀："44"（精度：2）
  值：200分钟
  权重：10

# 余额 1 首先被消耗（精度 4 > 精度 2）
# 更具体的目的地匹配胜出
```

用例：城市特定或区域特定的余额优先于国家范围的余额。

规则 2：权重优先（相同精度）

当目的地精度相等时，具有 更高权重 的余额首先被消耗。

```
余额 1:
  ID: "Premium_Data"
  值: 5GB
  权重: 20

余额 2:
  ID: "Standard_Data"
  值: 10GB
  权重: 10

# 余额 1 首先被消耗（权重 20 > 权重 10）
# 即使余额 2 的数据更多
```

用例：优先余额（奖励数据在常规数据之前消耗）。

规则 3：最旧优先

当权重匹配时，最旧的余额首先被使用。

```
余额 1:
  ID: "Data_Package_A"
  值: 5GB
  到期: 2024-12-25
  权重: 10

余额 2:
  ID: "Data_Package_B"
  值: 10GB
  到期: 2025-01-15
  权重: 10
```

要控制消耗顺序，请使用不同的权重：

```yaml
# 正确的方法：使用权重来优先考虑即将到期的余额
余额 1:
  ID: "Data_Package_A"
  值: 5GB
  ��期: 2024-12-25
  权重: 11  # 更高的权重 = 首先消耗

余额 2:
  ID: "Data_Package_B"
  值: 10GB
  到期: 2025-01-15
  权重: 10  # 较低的权重 = 第二消耗

# 余额 1 首先被消耗（权重 11 > 权重 10）
```

最佳实践：如果您希望确保即将到期的余额首先被消耗，请在创建时为其分配更高的权重。

实际示例

示例 1：简单数据附加产品（独立）

场景：客户可以多次购买“5GB 5天”附加产品，每次创建一个单独的余额。

实现：

```yaml
- name: 生成 UUID 用于唯一余额
  set_fact:
    uuid: "{{ 99999999 | random | to_uuid }}"

- name: 添加独立的 5GB 余额
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Categories": "*any",
          "Balance": {
            "ID": "Data_5GB_{{ uuid[0:8] }}",
            "Value": 5368709120,
            "ExpiryTime": "+120h",  # 5天
            "Weight": 10
          }
        }]
      }
```

客户体验:

- 在 12 月 24 日购买附加产品 → 获得 5GB，2024 年 12 月 29 日到期
- 在 12 月 25 日购买附加产品 → 获得 5GB，2024 年 12 月 30 日到期
- 两个余额共存；消耗顺序取决于余额权重（两个权重均为 10，因此顺序不保证）

示例 2：滚存数据包

场景：“每月 50GB 计划”，客户在提前充值时未使用的数据滚存。

实现:

```
- name: 添加滚存数据余额
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Balance": {
            "ID": "Rollover_Monthly_50GB",
            "Value": 53687091200,  # 50GB
            "ExpiryTime": "+720h",  # 30天
            "Weight": 10
          }
        }]
      }
```

□作类型：使用默认的 `*topup` 行为（启用滚存）

客户体验：

- 第 1 天：获得 50GB，30 天后到期
- 第 20 天：使用 30GB，剩余 20GB
- 第 20 天：再次充值 → 获得 70GB（20GB + 50GB），到期延长至第 20 天起 +30 天

示例 3：固定每月计划（无滚存□□□

场景：“无限 100GB 每月”计划，每月重置为确切的 100GB，无滚存。

实现：

```
- name: □□□□□□□□□
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.SetActions",
        "params": [{
          "ActionsId": "Action_Monthly_100GB_Reset",
          "Overwrite": true,
          "Actions": [{
            "Identifier": "*topup_reset",
            "BalanceType": "*data",
            "Units": 107374182400,  # 100GB
            "ExpiryTime": "*monthly",
            "BalanceId": "Monthly_Plan__107374182400",
            "Weight": 10
          }]
        }]
      }

- name: □□□□ ActionPlan
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.SetActionPlan",
        "params": [{
          "Id": "ActionPlan_Monthly_100GB",
          "ActionPlan": [{
            "ActionsId": "Action_Monthly_100GB_Reset",
            "Time": "*monthly",
            "Weight": 10
          }],
          "Overwrite": true,
          "ReloadScheduler": true
        }]
      }

- name: □ ActionPlan □□□□□
```

```
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV2.SetAccount",
        "params": [{
          "Account": "{{ service_uuid }}",
          "ActionPlanIds": ["ActionPlan_Monthly_100GB"],
          "ReloadScheduler": true
        }]
      }
```

客户体验：

- 第 1 个月，第 1 天：获得 100GB，使用 95GB，剩余 5GB
- 第 2 个月：余额重置为 100GB（未使用的 5GB 丢失）
- 第 2 个月：使用 20GB，剩余 80GB
- 第 3 个月：余额重置为 100GB（未使用的 80GB 丢失）

示例 4：多层余额与权重优先

场景：客户拥有“奖励数据”（高优先级）和“常规数据”（低优先级）。奖励数据优先消耗。

实现：

```
# 添加高权重的奖励数据
- name: 添加奖励数据余额
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Balance": {
            "ID": "Bonus_Data",
            "Value": 5368709120,  # 5GB
            "ExpiryTime": "+240h",  # 10天
            "Weight": 20
          }
        }]
      }

# 添加常规数据，权重正常
- name: 添加常规数据余额
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Balance": {
            "ID": "Regular_Data",
            "Value": 53687091200,  # 50GB
            "ExpiryTime": "+720h",  # 30天
            "Weight": 10  # 正常优先级
          }
        }]
      }
```

客户体验：

- 拥有 5GB 奖励数据（权重 20）+ 50GB 常规数据（权重 10）
- 首先消耗所有 5GB 奖励数据
- 然后从 50GB 常规数据池中消耗
- 常规数据保留更长时间

常见□作标识符

CGRateS 支持多种操作标识符以进行不同操作：

余额□作

***topup** - 添加到现有余额（滚存）
***topup_reset** - 将余额重置为新值（无滚存）
***debit** - 从余额中扣除
***debit_reset** - 将余额设置为负值
***reset_account** - 删除所有余额

设计附加产品

在 OmniCRM 中设计附加产品时，请考虑以下问题：

问题 1：余额应该堆叠吗？

是（独立） → 使用唯一余额 ID（带 UUID）
否（替换） → 使用固定余额 ID 和 `*topup_reset`
是（滚存） → 使用固定余额 ID 和 `*topup`

问题 2：未使用的余额会发生什么？

滚存 → 使用 `*topup` 操作
丢失 → 使用 `*topup_reset` 操作
独立池 → 使用唯一余额 ID

问题 3：余额应该如何消耗？

最旧优先 → 使用不同的权重（为较旧/即将到期的余额分配更高的权重）

优先级高 → 不同的权重（更高的权重 = 更高的优先级）

特定顺序 → 使用权重：30（优先），20（奖励），10（常规）

随机/无偏好 → 使用相同的权重（消耗顺序不保证）

问题 4：到期策略是什么？

固定持续时间 → 使用相对到期（`+720h` 为 30 天）

月底 → 在 ActionPlan 中使用 `*monthly`

永不到期 → 使用 `*unlimited` 或非常长的持续时间

CGRateS 作结构在 Playbooks 中

以下创建操作的完整结构：

```
- name: 创建 CGRateS 操作
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.SetActions",
        "params": [{
          "ActionsId": "Action_Name_Here",
          "Overwrite": true,  # 如果存在则替换
          "Actions": [
            {
              "Identifier": "*topup",  # 或 *topup_reset
              "BalanceType": "*data",  # *data、*voice、*sms、
*monetary
              "Units": 5368709120,  # 添加的金额
              "ExpiryTime": "+120h",  # +Xh、*unlimited、*monthly
              "DestinationIds": "*any",  # 通常为 *any
              "BalanceId": "Balance_Name",  # 唯一或共享
              "Weight": 10,  # 优先级（更高 = 首先消耗）
              "Blocker": false,  # 如果为 true，则防止负余额
              "Disabled": false,  # 如果为 true，则余额被禁用
              "SharedGroups": ""  # 共享余额组（可选）
            },
            {
              "Identifier": "*cdrlog",  # 将此操作记录为 CDR
              "BalanceType": "*generic",
              "ExtraParameters": "
{\"Category\":\"^activation\",\"Destination\":\"Addon Name\"}"
            }
          ]
        }]
      }
```

字段描述

ActionsId - 此操作集的唯一标识符

Identifier - 操作类型（`*topup`、`*topup_reset`、`*cdrlog` 等）

BalanceType - 余额类型：

- `*data` - 数据余额（字节）
- `*voice` - 语音余额（秒）
- `*sms` - 短信余额（计数）
- `*monetary` - 货币余额（货币单位）
- `*generic` - 通用余额

Units - 添加/扣除的金额（基本单位：数据为字节，语音为秒）

ExpiryTime - 余额到期时间：

- `+Xh` - 相对（例如，`+720h` = 30 天）
- `*unlimited` - 永不到期
- `*monthly` - 月底
- `2024-12-31T23:59:59Z` - 绝对时间戳

BalanceId - 此余额的标识符（共享 ID = 交互，唯一 ID = 独立）

Weight - 优先级（更高的数字 = 更高的优先级，首先消耗）

Blocker - 如果为 true，则防止账户变为负数

Disabled - 如果为 true，则余额存在但无法使用

通过 Python 定义操作（初始设置）

操作通常在初始系统配置期间使用 Python 脚本通过 `cgrateshttpapi` 库定义。这些示例展示了如何使用 `OCS_Obj.SendData()` 定义操作。

先决条件

```
import cgrateshttpapi
import time

OCS_Obj = cgrateshttpapi.CGRateS("ocs.example.com", "2080")
tenant = "your_tenant_name"
tpid = str(tenant) + "_" + str(int(time.time()))
```

定义目的地

在创建□作之前，您必须定义目的地，以指定余额可以在哪里使用。

目的地 有两种类型：

- 地理目的地 - 语音/SMS 的号码前缀（例如，`Dest_International_UK`）
- **PLMN** 目的地 - 数据的网络代码（例如，`Dest_PLMN_OnNet`、`Dest_PLMN_US_Verizon`）

关键规则：

- 语音/**SMS** 余额 → 使用地理目的地（拨打的号码 TO）
- 数据余额 → 使用 PLMN 目的地（客户连接的网络 FROM）

有关完整的目的地配置，包括：

- 地理目的地（国内、国际、免费电话、高级）
- PLMN 目的地（本网、漫游网络、区域）
- PLMN 格式规则和最佳实践
- 故障排除目的地问题

请参见：**CGRateS** 目的地配置

单位计算

理解单位转换对于正确定义余额至关重要：

```
# 数据余额（以字节为单位）
1_GB = 1 * 1024 * 1024 * 1024  # 1073741824 字节
100_GB = 100 * 1024 * 1024 * 1024

# 语音余额（以纳秒为单位）
1_minute = 60 * 1000000000  # 60 亿纳秒
3000_minutes = 3000 * 60 * 1000000000

# 短信余额（以计数为单位）
3000_sms = 3000
```

示例 1：多余额每月计划（Python）

一个综合的每月计划，包含数据、语音、短信和漫游余额，每月重置。

```
Action_AU_Premium_Plan_1 = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_au-premium-plan-1",
        "Overwrite": True,
        "Tenant": str(tenant),
        "Actions": [
            # 首先重置账户以清除旧余额
            {
                "Identifier": "*reset_account",
                "Weight": 700
            },
            # 添加 100GB 数据余额
            # 重要：数据余额使用 PLMN 目的地（客户连接的网络）
            # 而不是地理目的地。使用您的本网 PLMN。
            {
                "Identifier": "*topup_reset",
                "BalanceId": "AU_Data_Domestic__" + str(100 * 1024
* 1024 * 1024),
                "BalanceType": "*data",
                "DestinationIDs": "Dest_PLMN_OnNet",  # 您的本网 PLMN
（mcc505.mnc057）
                "Units": 100 * 1024 * 1024 * 1024,
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 90
            },
            # 添加 3000 分钟语音余额
            {
                "Identifier": "*topup_reset",
                "BalanceId": "AU_Voice_Domestic__" + str(3000 * 60
* 1000000000),
                "BalanceType": "*voice",
                "DestinationIDs":
"Dest_AU_Mobile;Dest_AU_Fixed;Dest_AU_TollFree;",
                "Units": 3000 * 60 * 1000000000,
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 89
            },
            # 添加 3000 短信余额
            {
```

```
                "Identifier": "*topup_reset",
                "BalanceId": "AU_SMS_Domestic__" + str(3000),
                "BalanceType": "*sms",
                "DestinationIDs": "Dest_AU_Mobile;",
                "Units": 3000,
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 88
            },
            # 添加 6GB 漫游数据
            {
                "Identifier": "*topup_reset",
                "BalanceId": "AU_Roaming_Data__" + str(6 * 1024 *
1024 * 1024),
                "BalanceType": "*data",
                "DestinationIDs": "Dest_Roaming_All",
                "Units": 6 * 1024 * 1024 * 1024,
                "ExpiryTime": "*month",
                "BalanceWeight": 1100,
                "Weight": 87
            },
            # 将此操作记录为 CDR
            {
                "Identifier": "*cdrlog",
                "BalanceType": "*generic",
                "ExtraParameters": "
{\"Category\":\"^activation\",\"Destination\":\"AU Premium Plan
1\"}"
            }
        ]
    }]
}

# 将操作定义发送到 CGRateS
result = OCS_Obj.SendData(Action_AU_Premium_Plan_1)
assert result['error'] is None or result['error'] == "EXISTS"
print("创建操作: Action_au-premium-plan-1")
```

关键点：

- 使用 `*reset_account` 首先清除旧余额
- 使用 `*topup_reset` 进行固定的每月津贴（无滚存）

- BalanceWeight 决定消耗顺序（国内 1200 > 漫游 1100）
- Weight 决定操作集内的执行顺序
- 包含 `*cdrlog` 用于跟踪激活

示例 2：简单数据附加产品（Python）

一个简单的 20GB 数据附加产品，滚存被禁用。

```
Action_AU_Data_Addon_20GB = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_au-data-addon-20gb",
        "Overwrite": True,
        "Tenant": str(tenant),
        "Actions": [
            # 首先重置账户
            {
                "Identifier": "*reset_account",
                "Weight": 700
            },
            # 添加 20GB 数据
            # 数据余额使用 PLMN 目的地（客户连接的网络）
            {
                "Identifier": "*topup_reset",
                "BalanceId": "AU_Data_Domestic__" + str(20 * 1024
* 1024 * 1024),
                "BalanceType": "*data",
                "DestinationIDs": "Dest_PLMN_OnNet",  # 您的本网 PLMN
（mcc505.mnc057）
                "Units": 20 * 1024 * 1024 * 1024,
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 90
            }
        ]
    }]
}

result = OCS_Obj.SendData(Action_AU_Data_Addon_20GB)
assert result['error'] is None or result['error'] == "EXISTS"
print("创建操作: Action_au-data-addon-20gb")
```

示例 3：国际语音附加产品（Python）

一个用于国际通话分钟的附加产品。

```
Action_AU_International_Voice_100min = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_au-international-voice-100min",
        "Overwrite": True,
        "Tenant": str(tenant),
        "Actions": [
            {
                "Identifier": "*reset_account",
                "Weight": 700
            },
            # 添加 100 分钟国际通话
            {
                "Identifier": "*topup_reset",
                "BalanceId": "AU_Voice_International__" + str(100
* 60 * 1000000000),
                "BalanceType": "*voice",
                "DestinationIDs": "Dest_International_All",
                "Units": 100 * 60 * 1000000000,
                "ExpiryTime": "*month",
                "BalanceWeight": 1000,
                "Weight": 90
            }
        ]
    }]
}

result = OCS_Obj.SendData(Action_AU_International_Voice_100min)
assert result['error'] is None or result['error'] == "EXISTS"
print("创建操作: Action_au-international-voice-100min")
```

注意：语音/SMS 使用地理目的地（拨打的号码），而数据使用 PLMN 目的地（客户连接的网络）。请参见 定义产品 以获取目的地配置。

Python 作字段参考

作定义字段：

- **ActionsId**（必需） - 此操作集的唯一标识符（必须与 CRM 中的 product_slug 约定匹配）
- **Overwrite** - 如果为 True，则替换具有相同 ID 的现有操作
- **Tenant** - CGRateS 租户名称

- **Actions** - 要执行的单个操作的数组

单个□作字段：

- **Identifier** - 操作类型（`*topup`、`*topup_reset`、`*cdrlog` 等）
- **BalanceId** - 此余额的唯一标识符（必须在滚存时匹配）
- **BalanceType** - 余额类型（`*data`、`*voice`、`*sms`、`*monetary`）
- **DestinationIDs** - 控制余额可以使用的地方：
  - 对于 语音/**SMS**：使用地理目的地（例如，`"Dest_AU_Mobile"`、`"Dest_International_UK"`）
  - 对于 数据：使用 PLMN 目的地（例如，`"Dest_PLMN_OnNet"`、`"Dest_PLMN_US_Verizon"`）
- **Units** - 添加的金额（字节用于数据，秒用于语音）
- **ExpiryTime** - 余额到期时间（`*month`、`+720h`、`2024-12-31` 等）
- **BalanceWeight** - 消耗优先级（更高 = 首先消耗）
- **Weight** - 操作集内的执行顺序（更高 = 首先执行）
- **Blocker**（可选） - 如果为 `True`，则防止账户变为负数
- **Disabled**（可选） - 如果为 `True`，则此余额被忽略/跳过

执行□作

一旦创建了操作，就可以在账户上执行它：

```
- name: 在账户上执行操作
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "APIerSv1.ExecuteAction",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "ActionsId": "Action_Name_Here"
        }]
      }
```

这将在指定账户上执行操作中定义的所有操作。

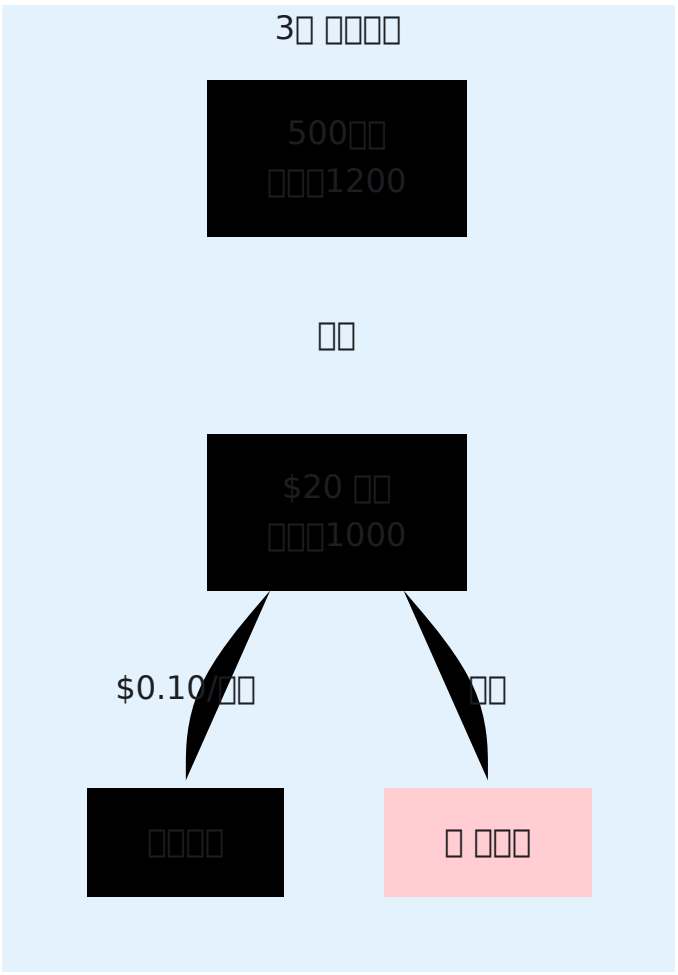
余额管理方法

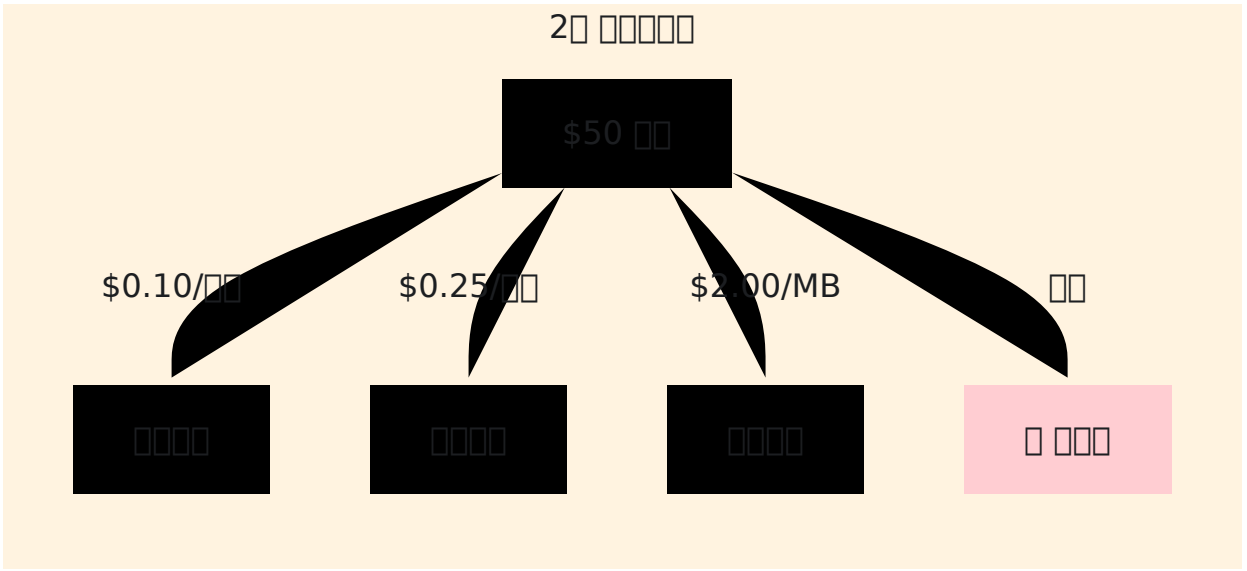
在 CGRateS 中，主要有三种管理余额的方法，每种方法在客户体验、账单可预测性和操作复杂性方面都有不同的权衡。

比较：单元制 **VS** 货币制 **VS** 混合

下表总结了三种余额方法之间的权衡：

□□	□□□	□□□□**PAYG**□	□□
□□□□□□	□ □□□□□□	□ □□□□	⚠ □□□□□□
□□□□□□	□ □□□□□□□□	□ □□□□□□□/□□	□ □□ + □□□□
□□□□	□ □□□	□ □□□□	□ □□□□
□□□□□□	□ □□□□□□	□ □□□□□□□	⚠ □□□□□□□□
□□□□□	⚠ □□□□□□□□	□ □□□□□□□□	□ □□□□□□
□□□□	⚠ □□□ ARPU	□ □ ARPU □□□□□□	□ □□□ ARPU
□□□□□	□ □□□□□□□	□ □□□□□□□	□ □□□□□□□□
□□□□□□	□□□□□□□	□□□□	□□□□□


3단 퍼널링
500명의
리드1200
전환
$20 구매
리드1000
$0.10/리드
전환
리드풀
더 필요


2단 퍼널링
$50 구매
$0.10/리드
$0.25/리드
$2.00/MB
전환
리드풀
리드풀
리드풀
더 필요

1단 퍼널링
500명의
리드
100명의
리드
5GB의
리드

方法 1：单元制余额

概念：提供特定数量用于特定目的地。每个余额都有固定的金额（分钟、GB、短信计数），与特定目的地相关联。当耗尽时，使用被阻止，除非有货币后备。

示例：国内计划与多个余额

```
Action_Domestic_Plan = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_domestic-plan",
        "Overwrite": True,
        "Tenant": tenant,
        "Actions": [
            {
                "Identifier": "*reset_account",
                "Weight": 700
            },
            # 500分钟国内通话
            {
                "Identifier": "*topup_reset",
                "BalanceId": f"Domestic_Voice__{500 * 60 *
1000000000}",
                "BalanceType": "*voice",
                "DestinationIDs": "Dest_Domestic_All",  # 仅限国内
                "Units": 500 * 60 * 1000000000,  # 500分钟（以纳秒为单位）
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 90
            },
            # 1000条国内短信
            {
                "Identifier": "*topup_reset",
                "BalanceId": "Domestic_SMS__1000",
                "BalanceType": "*sms",
                "DestinationIDs": "Dest_Domestic_All",
                "Units": 1000,
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 89
            },
            # 10GB 国内数据
            {
                "Identifier": "*topup_reset",
                "BalanceId": f"Domestic_Data__{10 * 1024 * 1024 *
1024}",
                "BalanceType": "*data",
                "DestinationIDs": "Dest_PLMN_OnNet",  # 您的本网 PLMN
                "Units": 10 * 1024 * 1024 * 1024,
```

```
                "ExpiryTime": "*month",
                "BalanceWeight": 1200,
                "Weight": 88
            },
            {
                "Identifier": "*cdrlog",
                "BalanceType": "*generic",
                "ExtraParameters": "
{\"Category\":\"^activation\",\"Destination\":\"Domestic Plan\"}",
                "Weight": 80
            }
        ]
    }]
}

result = OCS_Obj.SendData(Action_Domestic_Plan)
assert result['error'] is None or result['error'] == "EXISTS"
```

工作原理:

- 国内通话（1-555-1234） → 使用 500 分钟余额
- 国际通话（44-20-xxx） → 没有可用余额，被阻止或使用货币余额（如果有）
- 国内短信 → 使用 1000 短信余额
- 家庭数据使用 → 使用 10GB 余额
- 漫游数据 → 没有可用余额（需要单独的漫游数据余额）

优点:

- 客户可预测的费用
- 没有账单冲击
- 清晰的限制

缺点:

- 灵活性较差 - 不能在包含的目的地之外使用服务
- 需要多个余额以适应不同的用例
- 客户可能会感到受限

方法 2：货币制（PAYG）

概念：提供货币信用，按目的地特定费率收费。单一货币余额用于所有使用类型。CGRateS 查找每个目的地的费率并从信用中扣除费用。

注意： PAYG 需要为每个目的地定义费率配置文件，以设置每单位的美元金额。请参见 PAYG/货币余额的费率配置文件 以获取完整配置。

示例：**$50 PAYG** 信用

```
Action_PAYG_Credit = {
    "id": "0",
    "method": "ApierV1.SetActions",
    "params": [{
        "ActionsId": "Action_payg-50-credit",
        "Overwrite": True,
        "Tenant": tenant,
        "Actions": [
            # $50 货币余额
            {
                "Identifier": "*topup",
                "BalanceId": "PAYG_Monetary_Balance",
                "BalanceType": "*monetary",
                "DestinationIDs": "*any",  # 适用于任何目的地
                "Units": 5000,  # $50.00 (以美分为单位)
                "ExpiryTime": "+2160h",  # 90天
                "BalanceWeight": 1000,  # 低于单元制余额
                "Weight": 90
            },
            {
                "Identifier": "*cdrlog",
                "BalanceType": "*generic",
                "ExtraParameters": "
{\"Category\":\"^activation\",\"Destination\":\"$50 PAYG
Credit\"}",
                "Weight": 80
            }
        ]
    }]
}

result = OCS_Obj.SendData(Action_PAYG_Credit)
```

PAYG 的工作原理：

场景 **1**：国内通话，**10**分钟

- 费率：$0.10/分钟
- 收费：10 × $0.10 = $1.00
- 剩余：$49.00

场景 **2**：拨打英国电话，**5**分钟

- 费率：$0.25/分钟 + $0.05 连接费
- 收费：（5 × $0.25） + $0.05 = $1.30
- 剩余：$47.70

场景 **3**：在 **Verizon** 漫游，使用 **100MB** 数据

- 费率

CGRateS 目的地配置

本指南解释了如何在 CGRateS 中为 OmniCRM 配置目的地。目的地定义了余额可以使用的地方 - 无论□拨打特定号码（地理）还□在特定网络上使用数据/语音（PLMN）。

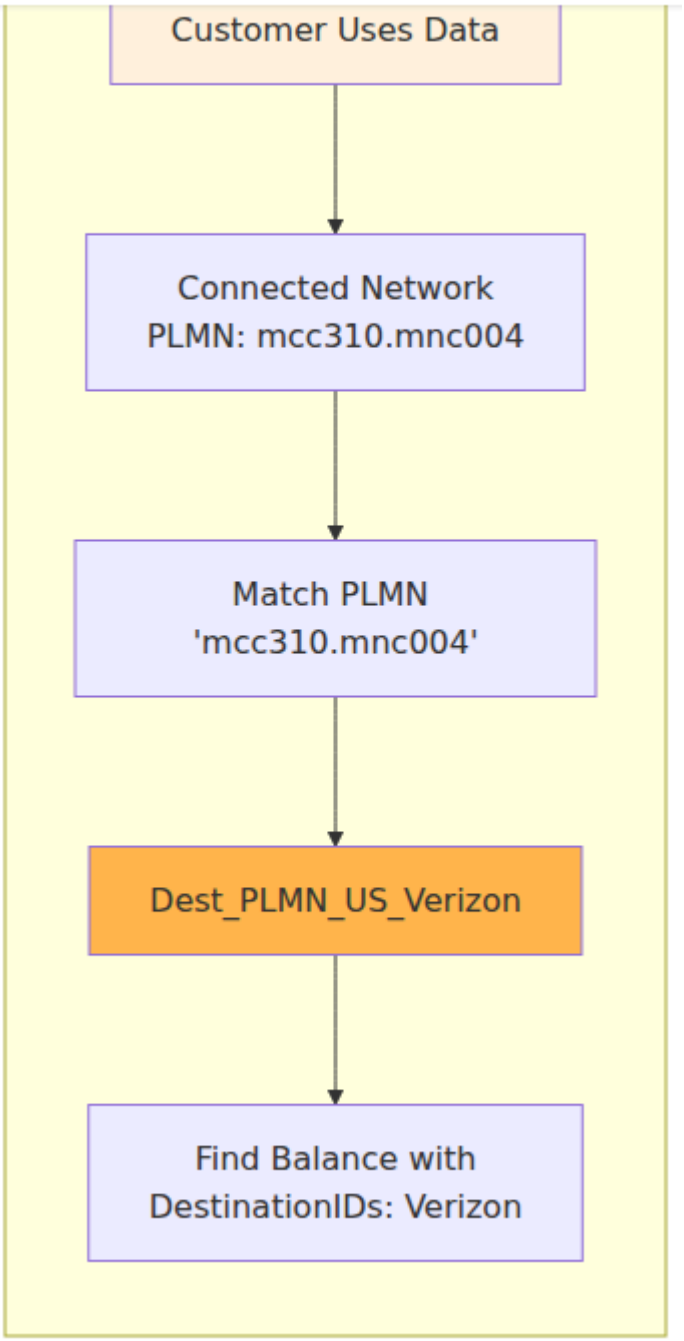
概述

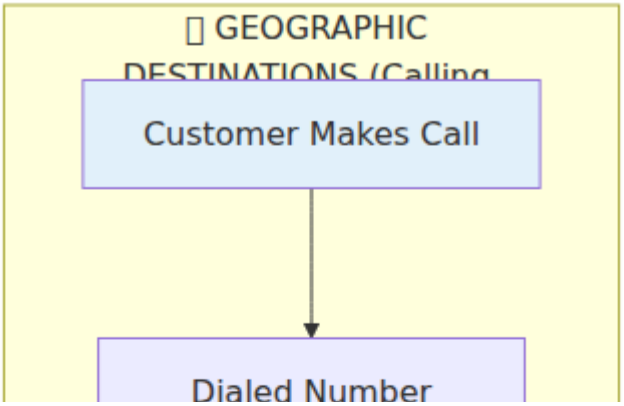
CGRateS 使用 目的地 来控制客户可以在哪里使用他们的余额：

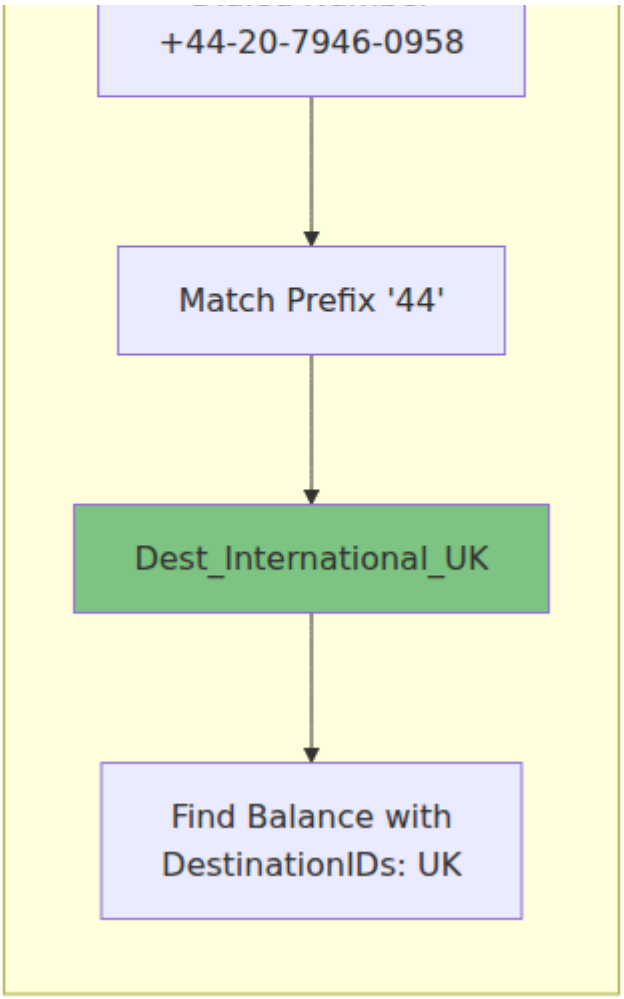
- 地理目的地 - 拨打特定位置的电话/SMS 的号码前缀（例如，英国号码、美国免费电话）
- **PLMN** 目的地 - 用于特定网络的数据使用和漫游的网络代码（例如，Verizon、Vodafone UK）

当客户使用服务时，CGRateS 匹配：

- 语音/**SMS**：拨打的号码前缀 → 地理目的地
- 数据：网络 PLMN 代码 → PLMN 目的地


Customer Uses Data
Connected Network
PLMN: mcc310.mnc004
Match PLMN
'mcc310.mnc004'
Dest_PLMN_US_Verizon
Find Balance with
DestinationIDs: Verizon


GEOGRAPHIC
Customer Makes Call
Dialed Number



关键规则

- 语音/**SMS** 余额 → 使用地理目的地（拨打的号码）
- 数据余额 → 使用 PLMN 目的地（客户连接的网络）

前提条件

```
import cgrateshttpapi
import time

OCS_Obj = cgrateshttpapi.CGRateS("ocs.example.com", "2080")
tenant = "your_tenant_name"
tpid = str(tenant) + "_" + str(int(time.time()))
```

第 1 部分：地理目的地（拨打到地方）

地理目的地定义了拨打特定位置的语音电话和 SMS 的号码前缀。

□□□□□

```
#
# ============================================================================
# DOMESTIC DESTINATIONS (US/Canada - NANP)
#
# ============================================================================

# US/Canada Mobile & Fixed (share area codes)
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_Domestic_All",
        "Prefixes": ["1"]  # NANP (US/Canada)
    }]
})

# US Toll-Free
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_Domestic_TollFree",
        "Prefixes": ["1800", "1888", "1877", "1866", "1855",
"1844", "1833"]
    }]
})

# US Premium Rate
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_Domestic_Premium",
        "Prefixes": ["1900"]
    }]
})
```

□□□□□

```
#
# ============================================================================
# INTERNATIONAL DESTINATIONS (By Country)
#
# ============================================================================

# United Kingdom
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_International_UK",
        "Prefixes": ["44"]
    }]
})

# China
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_International_China",
        "Prefixes": ["86"]
    }]
})

# Australia - Mobile
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_AU_Mobile",
        "Prefixes": ["614"]  # Australian mobiles (04xx dialed as
614xx)
    }]
})

# Australia - Fixed Line
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
```

```
        "TPid": tpid,
        "ID": "Dest_AU_Fixed",
        "Prefixes": [
            "612",  # NSW (Sydney)
            "613",  # VIC (Melbourne)
            "617",  # TAS (Hobart)
            "618"   # SA, WA, NT (Adelaide, Perth, Darwin)
        ]
    }]
})

# Australia - Toll-Free
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_AU_TollFree",
        "Prefixes": [
            "611800",  # Toll-free
            "611300"   # Local rate
        ]
    }]
})

# Australia - Premium Rate
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_AU_Premium",
        "Prefixes": ["6119"]  # Premium rate services
    }]
})

# Europe (grouped)
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_International_Europe",
        "Prefixes": [
            "33",  # France
            "49",  # Germany
            "39",  # Italy
```

```
            "34",  # Spain
            "31",  # Netherlands
            "32",  # Belgium
            "41"   # Switzerland
        ]
    }]
})

# International - All (catch-all, excludes domestic)
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_International_All",
        "Prefixes": ["2", "3", "4", "5", "6", "7", "8", "9"]  # All except "1" (NANP)
    }]
})
```

用例：

- 国内计划：“无限拨打美国/加拿大号码” → `Dest_Domestic_All`
- 英国拨打附加包：“100分钟拨打英国” → `Dest_International_UK`
- 澳大利亚计划：“3000分钟拨打澳大利亚手机和固定电话” → `Dest_AU_Mobile`, `Dest_AU_Fixed`, `Dest_AU_TollFree`
- 国际套餐：“50分钟拨打任何国际号码” → `Dest_International_All`

第 2 部分：PLMN 目的地（使用来自地方 - 漫游/数据使用）

PLMN 目的地定义了数据使用和漫游场景的网络代码。使用格式 `mccXXX.mncYYY`，其中：

- **MCC** = 移动国家代码（3 位数字）
- **MNC** = 移动网络代码（2-3 位数字）

PLMN 格式规则

- 特定网络：`"mcc310.mnc004"` → 仅 Verizon 网络 004
- 国家内所有网络：`"mcc310"` → 任何美国网络

- 示例：`mcc310.mnc004`，`mcc234.mnc015`，`mcc505.mnc057`

本网（您的本地网络）

重要：定义您特定的网络，您的 SIM 卡已在该网络上配置。

```
# On-Net (Your Home Network)
# This is YOUR operator's PLMN - the network where YOUR customers' SIMs are active
# All domestic on-net data balances should use this destination

OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_OnNet",
        "Prefixes": ["mcc505.mnc057"]  # Example: MCC 505, MNC 57 if this is your home network
    }]
})
# Replace mcc505.mnc057 with YOUR actual operator's PLMN code
# All domestic on-net data balances should use the prefixes set out here
# Otherwise they're considered roaming
```

□□□□ PLMN □□□

```
#
# ====================================================================
# US ROAMING PLMN DESTINATIONS
#
# ====================================================================

# Verizon Wireless
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_US_Verizon",
        "Prefixes": [
            "mcc310.mnc004", "mcc310.mnc010", "mcc310.mnc012",
"mcc310.mnc013",
            "mcc311.mnc480", "mcc311.mnc481", "mcc311.mnc482",
"mcc311.mnc483"
        ]
    }]
})

# All US PLMNs (catch-all)
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_US_All",
        "Prefixes": ["mcc310", "mcc311", "mcc312", "mcc313",
"mcc316"]
    }]
})
# Note: This matches ANY network with these MCCs (310, 311, etc.)
# Customer on mcc310.mnc004 (Verizon), mcc310.mnc410 (AT&T),
mcc311.mnc580 (US Cellular)
# would ALL match this destination
```

英国漫游 **PLMN** 目的地

```
#
# ============================================================================
# UK ROAMING PLMN DESTINATIONS
#
# ============================================================================

# Vodafone UK
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_UK_Vodafone",
        "Prefixes": ["mcc234.mnc015"]
    }]
})

# EE (Everything Everywhere)
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_UK_EE",
        "Prefixes": ["mcc234.mnc030", "mcc234.mnc033"]
    }]
})

# All UK PLMNs
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_UK_All",
        "Prefixes": ["mcc234"]
    }]
})
```

漫游区（多国组）

非常适合区域漫游套餐，如“欧洲漫游”或“亚太漫游”。

```
#
================================================================
# ROAMING ZONES (Multi-Country Groups)
#
================================================================

# Zone 1: North America
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_Zone_NorthAmerica",
        "Prefixes": [
            "mcc310", "mcc311", "mcc312", "mcc313", "mcc316",  #
USA
            "mcc302", "mcc334"  # Canada
        ]
    }]
})
# Using MCC-only prefixes provides coverage on ANY network in these
countries
# Great for "North America roaming" packages that work on all
carriers

# Zone 2: Europe
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_Zone_Europe",
        "Prefixes": [
            "mcc234",  # United Kingdom
            "mcc208",  # France
            "mcc262",  # Germany
            "mcc222",  # Italy
            "mcc214",  # Spain
            "mcc228"   # Switzerland
        ]
    }]
})

# Zone 3: Asia Pacific
OCS_Obj.SendData({
```

```
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_Zone_AsiaPacific",
        "Prefixes": [
            "mcc505",  # Australia
            "mcc460",  # China
            "mcc454",  # Hong Kong
            "mcc440",  # Japan
            "mcc520",  # Thailand
            "mcc525",  # Singapore
            "mcc530"   # New Zealand
        ]
    }]
})

# Catch-all for any roaming (use with caution)
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_Roaming_All",
        "Prefixes": ["mcc"]  # Matches ANY PLMN (very broad)
    }]
})
```

用例:

- 特定网络漫游:“仅在 Verizon 上使用 5GB” → `Dest_PLMN_US_Verizon`
- 全国漫游:“在美国（任何网络）使用 10GB” → `Dest_PLMN_US_All`
- 区域漫游:“欧洲漫游 15GB” → `Dest_PLMN_Zone_Europe`
- 全球漫游:“在任何地方使用 1GB” → `Dest_Roaming_All`

何时使用每种方法

目的地类型	用例	示例	余额类型
地理	拨打/短信到特定国家	“100分钟拨打英国号码”	语音，SMS
本网 **PLMN**	在您的网络上使用国内数据	“100GB 本网数据”	数据
特定 **PLMN**	在首选网络上使用高级漫游	“仅在 Verizon 上使用 5GB”	数据（和漫游语音/SMS）
全国 **PLMN**	在国家内任何网络漫游	“在美国（任何网络）使用 5GB”	数据（和漫游语音/SMS）
区域 **PLMN**	区域漫游套餐	“10GB 欧洲漫游”	数据（和漫游语音/SMS）

动态目的地更新和服务访问

目的地更新如何工作

关键概念： CGRateS 中的目的地 通过 **ID** 引用，而不复制。当您更新目的地的前缀或 PLMN 代码并 将 **TarifPlan** 加载到运行时 时，所有引用该目的地的现有余额会立即受到影响。

这意味着：

- □ □□□□□□□□□□□□□□□□□□□□□□□□
- □ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- ⚠ 更改在加载 **TarifPlan** 时生效 - 您可以在加载之前多次修改目的地，更改仅在 `LoadTariffPlanFromStorDb` 后生效

示例 **1**：扩展覆盖范围

您有一个用于欧洲漫游的目的地：

```
# Initial configuration - Only 3 countries
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_Zone_Europe",
        "Prefixes": [
            "mcc262",  # Germany
            "mcc208",  # France
            "mcc222"   # Italy
        ]
    }]
})
```

具有余额的客户 引用 `Dest_PLMN_Zone_Europe` 可以在德国、法国和意大利漫游。

稍后，您添加西班牙和英国：

```
# Updated configuration - Now 5 countries
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_Zone_Europe",
        "Prefixes": [
            "mcc262",  # Germany
            "mcc208",  # France
            "mcc222",  # Italy
            "mcc214",  # Spain (NEW)
            "mcc234"   # UK (NEW)
        ]
    }]
})

# Load the tariff plan to make changes active
OCS_Obj.SendData({
    'method': 'ApierV1.LoadTariffPlanFromStorDb',
    'params': [{
        "TPid": tpid,
        "DryRun": False,
        "Validate": True
    }]
})

# Reload the destination cache
OCS_Obj.SendData({
    'method': 'CacheSv1.ReloadCache',
    'params': [{
        "DestinationIDs": ["Dest_PLMN_Zone_Europe"]
    }]
})
```

结果：加载资费计划后，所有现有客户的余额 `Dest_PLMN_Zone_Europe` 现在可以在西班牙和英国漫游 - 无需单独更新余额。

示例 2：限制覆盖范围

从目的地中删除一个国家 会在加载资费计划���阻止访问：

```
# Remove Germany from Europe zone
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_Zone_Europe",
        "Prefixes": [
            "mcc208",  # France
            "mcc222",  # Italy
            "mcc214",  # Spain
            "mcc234"   # UK
            # Germany (mcc262) REMOVED
        ]
    }]
})

# Load the tariff plan to make changes active
OCS_Obj.SendData({
    'method': 'ApierV1.LoadTariffPlanFromStorDb',
    'params': [{
        "TPid": tpid,
        "DryRun": False,
        "Validate": True
    }]
})

# Reload cache
OCS_Obj.SendData({
    'method': 'CacheSv1.ReloadCache',
    'params': [{
        "DestinationIDs": ["Dest_PLMN_Zone_Europe"]
    }]
})
```

结果：加载资费计划后，试图在德国漫游的客户将被 阻止，即使他们仍然有余额用于 `Dest_PLMN_Zone_Europe`。余额存在，但不再匹配 PLMN 目的地。

示例 3：语音/SMS 地理目的地

同样的原则适用于语音和 SMS 的地理目的地：

```
# Add Canada to "Domestic" calling
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_Domestic_All",
        "Prefixes": [
            "1",  # Already includes USA (and previously Canada)
            # No change needed - "1" prefix already covers both
        ]
    }]
})

# Or create separate North America destination
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_North_America",
        "Prefixes": [
            "1"  # USA + Canada
        ]
    }]
})
```

□□□ □□□□ `DestinationIDs: "Dest_Domestic_All"` □□□□

```
# Customer has balance:
{
    "BalanceId": "Europe_Data_5GB",
    "BalanceType": "*data",
    "DestinationIDs": "Dest_PLMN_Zone_Europe",  # Only France,
Germany, Italy
    "Units": 5 * 1024 * 1024 * 1024
}

# Customer roams on network in Spain (mcc214.mnc001)
# Spain is NOT in Dest_PLMN_Zone_Europe
```

结果：

- □ □□□□ □□□□□□□□□□□□□□
- 余额保持未使用
- 客户收到“信用不足”或类似错误

修复：将西班牙���加到 `Dest_PLMN_Zone_Europe` 或确保客户有带有 `DestinationIDs: "*any"` 的货币余额作为后备。

场景 **2**：没有通配符后备

```
# Customer has two balances:
{
    "BalanceId": "UK_Voice_100min",
    "BalanceType": "*voice",
    "DestinationIDs": "Dest_International_UK",  # Only prefix "44"
    "Units": 100 * 60 * 1000000000
}
# NO monetary balance with "*any" destination

# Customer calls Australia (+61)
```

结果：

- □ □□ □□□□□□□□□□□□□□“61”）
- 英国分钟余额未使用（仅匹配“44”）
- 没有货币后备来覆盖呼叫

修复：添加带有 `DestinationIDs: "*any"` 的货币余额或将特定澳大利亚目的地添加到他们的余额中。

场景 **3**：通配符后备有效

```
# Customer has:
# Balance 1: UK minutes
{
    "BalanceId": "UK_Voice_100min",
    "BalanceType": "*voice",
    "DestinationIDs": "Dest_International_UK",
    "Units": 100 * 60 * 1000000000,
    "BalanceWeight": 1200
}

# Balance 2: Monetary with wildcard
{
    "BalanceId": "PAYG_Credit",
    "BalanceType": "*monetary",
    "DestinationIDs": "*any",  # Matches ANYTHING
    "Units": 5000,  # $50
    "BalanceWeight": 1000  # Lower priority
}

# Customer calls Australia (+61)
```

结果：

- □ □□ □□□
- 英国分钟余额未使用（不匹配“61”）
- PAYG 信用余额使用（匹配“*any”）
- 按 PAYG 费率收费拨打澳大利亚

测试和模拟

目的地更改可以在非生产环境中进行测试和模拟，然后再应用于生产。这使您能够：

- 验证新的 PLMN 代码□否正确
- 测试覆盖范围扩展/限制的影响
- 验证现有余额在更新目的地时的行为□否符合预期

- 确保后备机制正常工作

建议：在将目的地更新应用于生产之前，始终在具有测试账户的暂存环境中进行测试。

关键要点

1. 目的地是动态的 - 更改会影响所有引用它们的余额，一旦加载资费计划
2. 加载 **TarifPlan** 以激活 - 使用 `LoadTariffPlanFromStorDb` 使目的地更改在运行时生效
3. 没有目的地匹配 = 没有服务 - 如果客户的使用不匹配任何目的地，他们将被阻止
4. 通配符（`*any`）非常强大 - 用于货币后备以防止意外阻止
5. 需要重新加载缓存 - 加载资费计划后始终重新加载目的地缓存
6. 在生产之前测试 - 一旦加载，目的地更改会影响所有客户

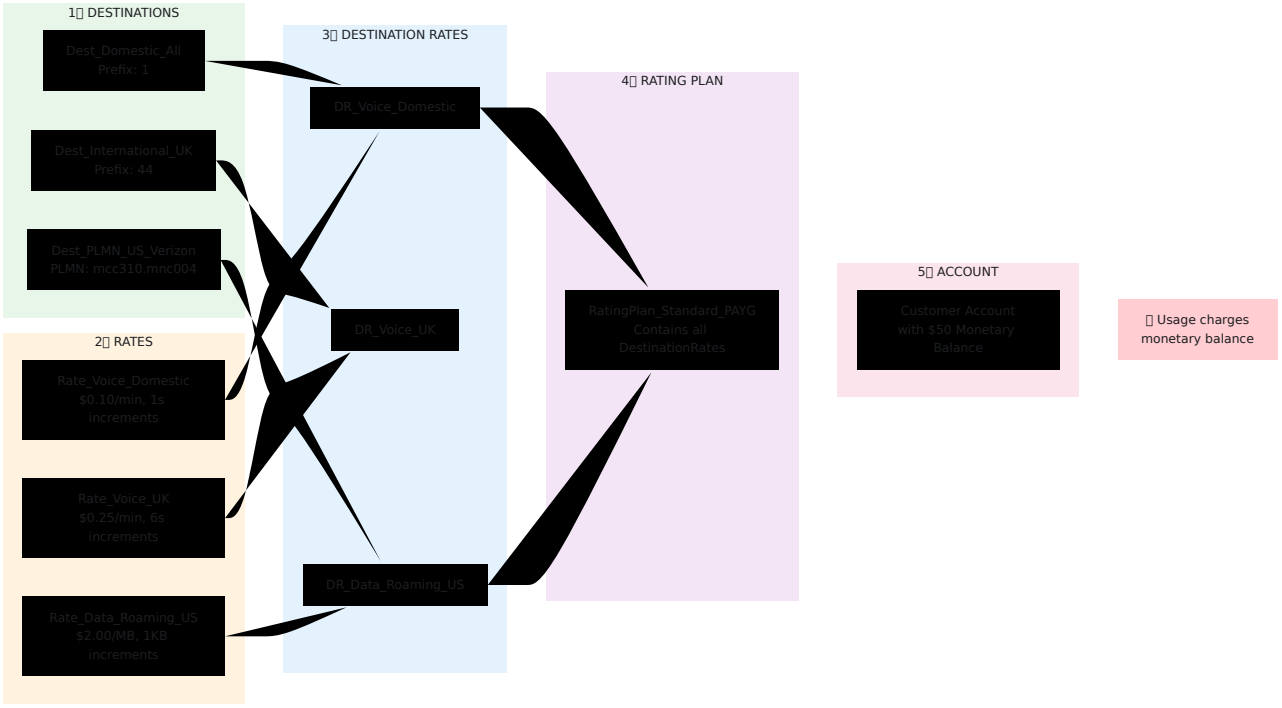
PAYG/货币余额的费率配置文件

在使用货币余额（PAYG - 按需付费）时，您需要定义费率配置文件，指定不同目的地每单位的收费标准。

关于 **CGRateS** 灵活性的注意事项： CGRateS 非常灵活，可以根据许多因素应用不同的费率和计划，包括：

- 客户拨打的来源（PLMN - 在这里涵盖）
- 客户拨打的目的地（地理目的地 - 在这里涵盖）
- 一天中的时间（高峰/非高峰费率）
- 客户等级（VIP、标准等）
- 服务质量水平
- 以及许多更复杂的计费场景

本指南专注于基于 PLMN 的漫游和基于地理目的地的计费的 **CRM** 级配置，涵盖最常见的用例。高级计费场景（时间段、客户等级、基于质量的路由）在 CGRateS 中可能的，但超出了此 CRM 重点文档的范围。



费率配置文件如何工作

定义费率配置文件

费率配置文件定义了不同使用类型的每单位费用。以下⃣语音、SMS 和数据的示例：

```
#
# ============================================================================
# VOICE RATES (Per Minute)
#
# ============================================================================

# Domestic voice: $0.10/minute
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPRate',
    'params': [{
        "TPid": tpid,
        "ID": "Rate_Voice_Domestic",
        "RateSlots": [{
            "ConnectFee": 0,
            "Rate": 0.10,
            "RateUnit": "60s",         # 1 minute
            "RateIncrement": "1s",     # Per-second billing
            "GroupIntervalStart": "0s"
        }]
    }]
})

# UK voice: $0.25/minute
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPRate',
    'params': [{
        "TPid": tpid,
        "ID": "Rate_Voice_UK",
        "RateSlots": [{
            "ConnectFee": 0.05,        # $0.05 connection fee
            "Rate": 0.25,
            "RateUnit": "60s",
            "RateIncrement": "6s",     # 6-second blocks
            "GroupIntervalStart": "0s"
        }]
    }]
})

#
# ============================================================================
# SMS RATES (Per Message)
#
# ============================================================================
```

```
# Domestic SMS: $0.05/message
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPRate',
    'params': [{
        "TPid": tpid,
        "ID": "Rate_SMS_Domestic",
        "RateSlots": [{
            "ConnectFee": 0,
            "Rate": 0.05,
            "RateUnit": "1",          # 1 SMS
            "RateIncrement": "1",     # Per message
            "GroupIntervalStart": "0s"
        }]
    }]
})

#
# ============================================================================
# DATA RATES (Per MB) - For Roaming
#
# ============================================================================

# Roaming on US Premium Networks (Verizon, AT&T): $2.00/MB
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPRate',
    'params': [{
        "TPid": tpid,
        "ID": "Rate_Data_Roaming_US_Premium",
        "RateSlots": [{
            "ConnectFee": 0,
            "Rate": 2.00,
            "RateUnit": "1048576",    # 1 MB in bytes
            "RateIncrement": "1024",  # Per KB
            "GroupIntervalStart": "0s"
        }]
    }]
})
```

□□□□□□

- **ConnectFee** - □□□□□□□□□□
- **Rate** - □□ RateUnit □□□

- **RateUnit** - 计费单位大小（60s、1MB、1 消息）
- **RateIncrement** - 最小可收费单位
- **GroupIntervalStart** - 此费率层级开始的时间

将目的地链接到费率

定义费率后，将它们链接到目的地，使用 DestinationRates：

```
# Link Domestic destination to Domestic voice rate
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestinationRate',
    'params': [{
        "TPid": tpid,
        "ID": "DR_Voice_Domestic",
        "DestinationRates": [{
            "DestinationId": "Dest_Domestic_All",
            "RateId": "Rate_Voice_Domestic",
            "RoundingMethod": "*up",
            "RoundingDecimals": 4
        }]
    }]
})

# Link UK destination to UK voice rate
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestinationRate',
    'params': [{
        "TPid": tpid,
        "ID": "DR_Voice_UK",
        "DestinationRates": [{
            "DestinationId": "Dest_International_UK",
            "RateId": "Rate_Voice_UK",
            "RoundingMethod": "*up",
            "RoundingDecimals": 4
        }]
    }]
})

# Link Verizon PLMN to roaming data rate
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestinationRate',
    'params': [{
        "TPid": tpid,
        "ID": "DR_Data_Roaming_US_Premium",
        "DestinationRates": [{
            "DestinationId": "Dest_PLMN_US_Verizon",
            "RateId": "Rate_Data_Roaming_US_Premium",
            "RoundingMethod": "*up",
            "RoundingDecimals": 4
        }]
```

```
    }]
})
```

创建计费计划

将所有 DestinationRates 组合到一个计费计划中:

```
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPRatingPlan',
    'params': [{
        "TPid": tpid,
        "ID": "RatingPlan_Standard_PAYG",
        "RatingPlanBindings": [
            # Voice rates (higher weight = more specific)
            {"DestinationRatesId": "DR_Voice_UK", "TimingId":
"*any", "Weight": 40},
            {"DestinationRatesId": "DR_Voice_Domestic",
"TimingId": "*any", "Weight": 20},

            # SMS rates
            {"DestinationRatesId": "DR_SMS_Domestic", "TimingId":
"*any", "Weight": 20},

            # Data roaming rates
            {"DestinationRatesId": "DR_Data_Roaming_US_Premium",
"TimingId": "*any", "Weight": 50}
        ]
    }]
})
```

加载资费计划

最后，将资费计划加载到 CGRateS 中:

```
# Load tariff plan
OCS_Obj.SendData({
    'method': 'ApierV1.LoadTariffPlanFromStorDb',
    'params': [{
        "TPid": tpid,
        "DryRun": False,
        "Validate": True
    }]
})

# Reload cache
OCS_Obj.SendData({
    'method': 'CacheSv1.ReloadCache',
    'params': [{
        "Tenant": tenant,
        "DestinationIDs": ["*all"]
    }]
})
```

另请参阅：

- 定义产品 - 创建产品的完整工作流程
- CGRateS 操作和充值行为 - 如何定义操作和余额管理方法

最佳实践

1. 对本网进行具体定义

```
# Good - specific home network
"DestinationIDs": "Dest_PLMN_OnNet",  # mcc505.mnc057

# Bad - too broad, matches competitors
"DestinationIDs": "Dest_PLMN_AU_All",  # mcc505 (all AU operators)
```

2. 使用有意义的目的地名称

```
# Good - self-documenting
Dest_PLMN_US_Verizon
Dest_International_UK
Dest_PLMN_Zone_Europe

# Bad - unclear
Dest_Data_1
Dest_Voice_Package
```

3. 逻辑上分组相关目的地

```
# Good - logical grouping
Dest_Domestic_All
Dest_Domestic_TollFree
Dest_Domestic_Premium

# Bad - too granular
Dest_Area_Code_212
Dest_Area_Code_213
Dest_Area_Code_214
```

4. 记录 PLMN 来源

定义 PLMN 目的地时，添加注释以指示 MCC/MNC 代码的来源：

```
# Verizon Wireless
# Source: https://www.mcc-mnc.com/ (verified 2024-12)
OCS_Obj.SendData({
    'method': 'ApierV2.SetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_US_Verizon",
        "Prefixes": [
            "mcc310.mnc004",  # Verizon - Nationwide
            "mcc310.mnc010",  # Verizon - East
            ...
        ]
    }]
})
```

故障排除

问题：余额不匹配目的地

症状：客户有余额，但无法使用于特定目的地

检查：

```
# 1. 验证目的地是否存在
OCS_Obj.SendData({
    'method': 'ApierV2.GetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_US_Verizon"
    }]
})

# 2. 检查余额 DestinationIDs 字段
# 确保它包含目的地或 "*any"

# 3. 对于 PLMN: 验证客户是否实际在该网上
# 检查 Diameter CCR 消息中的 Visited-PLMN-Id AVP
```

问题：数据使用计入漫游

症状：国内数据使用消耗漫游余额而不□本地余额

原因：客户的 PLMN 未在 `Dest_PLMN_OnNet` 中定义

解决方案：

```
# 检查客户在哪个 PLMN（来自 Diameter CCR）
# 验证余额是否包括该 PLMN：

OCS_Obj.SendData({
    'method': 'ApierV2.GetTPDestination',
    'params': [{
        "TPid": tpid,
        "ID": "Dest_PLMN_OnNet"
    }]
})
# 确保前缀包括客户的实际网络 PLMN
# 示例：["mcc505.mnc057"] 必须与客户的 SIM 网络匹配
```

问题：国际电话无法拨打

症状：客户有“国际分钟”，但拨打失败

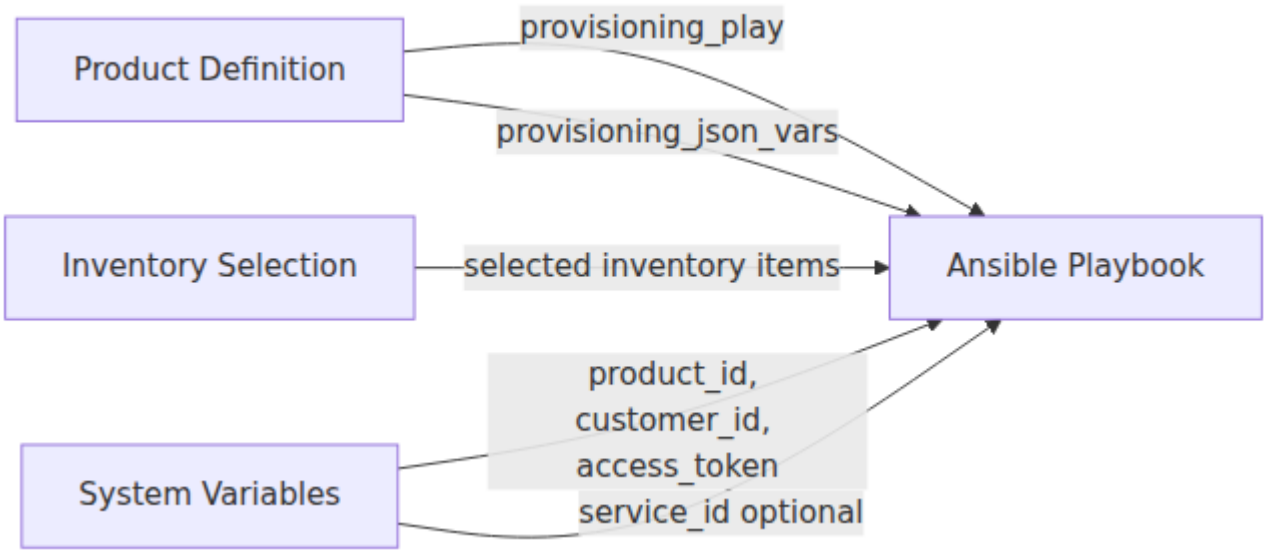
检查：

1. 目的地前缀：拨打的号码□否与目的地前缀匹配？
   - 拨打 +44-20-xxxx → 需要目的地前缀为“44”
2. 余额 **DestinationIDs**：余额□否包括该目的地？
   - 检查余额□否包含 `"DestinationIDs": "Dest_International_UK"` 或 `"Dest_International_All"`
3. 余额未过期：检查 `ExpiryTime`
4. 余额有单位：检查 `Units > 0`

相关文档

- CGRateS 操作和充值行为 - 如何在操作、余额方法和计费策略中使用目的地

- 定义产品 - 完整的产品创建工作流程



Ansible Playbooks: 详细指南

OmniCRM 产品通过 **Ansible** 进行配置，允许根据每个产品及其相关库存的特定要求进行自动化服务管理。

另请参见: `SIM Card Provisioning <concepts_sim_provisioning>` 以获取基于 Ansible 的移动服务配置的完整示例，包括物理 SIM 卡和 eSIM。

Playbooks 和产品如何协同工作

关键概念： Playbooks 实际上在 OmniCRM 中创建服务的工具。当您将 playbook 分配给产品时，您正在定义 发生什么 当该产品被配置 - 但这对不同产品可能意味着不同的事情。

产品触发 Playbooks

当产品在 OmniCRM 中被配置时:

1. 产品定义指定要运行的 playbook（通过 `provisioning_play` 字段）
2. 产品将变量传递给 playbook（通过 `provisioning_json_vars` 和库存选择）
3. playbook 执行并执行其编程的操作
4. playbook 确定创建什么（如果有的话）

Playbooks 可以做什么

单个配置 playbook 可以:

创建多个服务

一个捆绑产品的 playbook 可能创建:

- 一个主要的互联网服务记录
- 一个 IPTV 附加服务记录
- 一个 VoIP 服务记录
- 所有这些都通过一个产品配置操作完成

不创建服务

有些 playbooks 根本不创建服务记录:

- 一个仅配置 CPE 设备的 playbook
- 一个将配置发送到网络设备的 playbook
- 一个更新外部系统的 playbook

创建一个服务

最常见的模式:

- 为客户创建一个服务记录
- 将库存链接到该服务
- 为该服务设置计费

修改现有服务

充值和附加 playbooks:

- 不创建新服务
- 更新现有服务记录（添加数据、延长到期等）
- 向现有计费账户添加余额

执行没有服务记录的□作

一些 playbooks 纯粹□操作性的:

- 重置账户余额
- 在客户之间交换库存项目

- 生成报告或配置

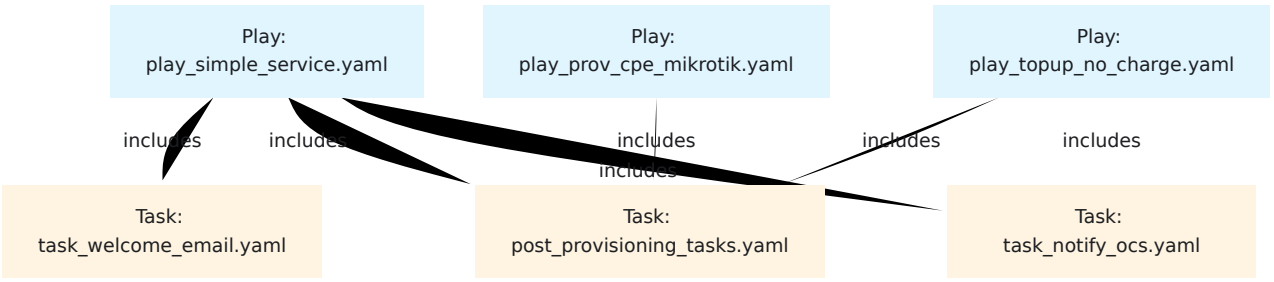
示例：不同的 **Playbook** 行为

```
# 产品 1：移动 SIM 服务（创建 1 个服务）
# provisioning_play: play_simple_service
- 在 CRM 中创建服务记录
- 在 OCS 中创建计费账户
- 分配 SIM 卡和电话号码库存
- 发送欢迎电子邮件

# 产品 2：互联网捆绑包（创建 3 个服务）
# provisioning_play: play_bundle_internet_tv_voice
- 创建互联网服务记录
- 创建 IPTV 服务记录
- 创建 VoIP 服务记录
- 将所有服务链接到同一□户
- 为捆绑包创建单一计费账户

# 产品 3：数据充值（创建 0 个服务）
# provisioning_play: play_topup_no_charge
- 根据 service_id 查找现有服务
- 向现有 OCS 账户添加数据余额
- 更新服务到期日期
- 不创建新服务

# 产品 4：CPE 配置（创建 0 个服务）
# provisioning_play: play_prov_cpe_mikrotik
- 生成路由器配置
- 用配置更新库存记录
- 将配置通过电子邮件发送给支持团队
- 不创建服务（仅设备设置）
```

关键点：**playbook** 定义行为，产品只是触发器。

Plays 与 Tasks

理解 Plays 和 Tasks 之间的区别对于使用 OmniCRM playbooks 至关重要。



Play (Playbook)

一个完整的配置工作流，协调多个任务以实现业务目标。 Plays 存储在 `OmniCRM-API/Provisioners/plays/` 中的顶级 playbooks，并在产品定义中引用。

示例：

- `play_simple_service.yaml` - 配置基本服务
- `play_topup_no_charge.yaml` - 对服务应用免费充值
- `play_prov_cpe_mikrotik.yaml` - 配置客户场所设备

Task (可重用组件)

一组自包含的、可重用的操作，可以被多个 plays 包含。 Tasks 以 `task_` 为前缀，并位于同一目录中。

示例：

- `task_welcome_email.yaml` - 向客户发送欢迎电子邮件
- `task_activate_olt.yaml` - 激活 OLT 设备
- `task_notify_ocs.yaml` - 向计费系统发送通知

它们之间的关系：

```yaml
# play_simple_service.yaml (A Play)
- name: Simple Provisioning Play
  hosts: localhost
  tasks:
    - name: Main provisioning block
      block:
        - name: Create service
          uri: ...

        - name: Configure billing
          uri: ...

        # Include reusable task
        - include_tasks: task_welcome_email.yaml

        # Include post-provisioning tasks
        - include_tasks: post_provisioning_tasks.yaml
```

Playbook 结构和组成

所有 OmniCRM playbooks 遵循一致的结构。理解这一结构对于创建和维护 playbooks 至关重要。

基本结构

每个 playbook 都以这些标准头开始:

```yaml
- name: Descriptive Name of the Playbook
  hosts: localhost            # 始终为 localhost
  gather_facts: no            # 为了性能禁用
  become: False               # 不提升权限

  tasks:
    - name: Main block
      block:
        # 配置任务在这里

      rescue:
        # 回滚/清理任务在这里
```

头部解释

name

在配置日志和 UI 中显示的描述性名称。这在配置记录中显示为 `playbook_description`。

hosts: localhost

所有 OmniCRM playbooks 都在 localhost 上运行，因为它们通过 API 与远程系统交互，而不通过 SSH。

gather_facts: no

Ansible 的事实收集被禁用，因为：

- 我们不需要系统信息
- 它增加了不必要的开销
- 如果在调试输出中显示，可能会导致浏览器崩溃

become: False

由于我们正在进行 API 调用，而不修改系统文件，因此不需要提升权限。

配置加载

每个 playbook 必须加载中央配置文件：

```
tasks:
  - name: Include vars of crm_config
    ansible.builtin.include_vars:
      file: "../../crm_config.yaml"
      name: crm_config
```

这使得配置可用，如 `crm_config.ocs.cgrates`、`crm_config.crm.base_url` 等。

`crm_config.yaml` 通常包含：

```
ocs:
  cgrates: "10.0.1.100:2080"
  ocsTenant: "default_tenant"
crm:
  base_url: "https://crm.example.com"
```

变量访问模式

变量可以来自多个来源:

来自产品定义:

```
- name: Access product_id passed by OmniCRM
  debug:
    msg: "Provisioning product {{ product_id }}"
```

来自库存选择:

```
- name: Get inventory ID for SIM Card
  set_fact:
    sim_card_id: "{{ hostvars[inventory_hostname]['SIM Card'] |
int }}"
  when: "'SIM Card' in hostvars[inventory_hostname]"
```

来自 **API** 响应:

```
- name: Get Product information from CRM API
  uri:
    url: "http://localhost:5000/crm/product/product_id/{{
product_id }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: api_response_product

- name: Use the product name
  debug:
    msg: "Product name is {{
api_response_product.json.product_name }}"
```

常见 **Playbook** 模式

服务配置模式

这□创建新服务的最常见模式。

```yaml
- name: Service Provisioning Playbook
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: Main block
      block:

      # 1. 加载配置
      - name: Include vars of crm_config
        ansible.builtin.include_vars:
          file: "../../crm_config.yaml"
          name: crm_config

      # 2. 取产品信息
      - name: Get Product information from CRM API
        uri:
          url: "http://localhost:5000/crm/product/product_id/{{
product_id }}"
          method: GET
          headers:
            Authorization: "Bearer {{ access_token }}"
          return_content: yes
          validate_certs: no
        register: api_response_product

      # 3. 取客户信息
      - name: Get Customer information from CRM API
        uri:
          url: "http://localhost:5000/crm/customer/customer_id/{{
customer_id }}"
          method: GET
          headers:
            Authorization: "Bearer {{ access_token }}"
          return_content: yes
        register: api_response_customer

      # 4. 从检索的数据中设置事实
      - name: Set package facts
        set_fact:
          package_name: "{{ api_response_product.json.product_name
}}"
```

```
          package_comment: "{{ api_response_product.json.comment
}}"
          setup_cost: "{{
api_response_product.json.retail_setup_cost }}"
          monthly_cost: "{{ api_response_product.json.retail_cost
}}"

      # 5. □□□□□□□□
      - name: Generate UUID
        set_fact:
          uuid: "{{ 99999999 | random | to_uuid }}"

      - name: Generate Service UUID
        set_fact:
          service_uuid: "Service_{{ uuid[0:8] }}"

      # 6. □□□□□□□□□□□
      - name: Create account in OCS/CGRateS
        uri:
          url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
          method: POST
          body_format: json
          headers:
            Content-Type: "application/json"
          body:
            {
              "method": "ApierV2.SetAccount",
              "params": [{
                  "Tenant": "{{ crm_config.ocs.ocsTenant }}",
                  "Account": "{{ service_uuid }}",
                  "ActionPlanIds": [],
                  "ActionPlansOverwrite": true,
                  "ExtraOptions": {
                    "AllowNegative": false,
                    "Disabled": false
                  },
                  "ReloadScheduler": true
              }]
            }
          status_code: 200
        register: ocs_response

      - name: Verify OCS account creation
        assert:
```

```
      that:
        - ocs_response.status == 200
        - ocs_response.json.result == "OK"

    # 7. □□□□□□
    - name: Add 0 Monetary Balance
      uri:
        url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
        method: POST
        body_format: json
        body:
          {
            "method": "ApierV1.AddBalance",
            "params": [{
                "Tenant": "{{ crm_config.ocs.ocsTenant }}",
                "Account": "{{ service_uuid }}",
                "BalanceType": "*monetary",
                "Categories": "*any",
                "Balance": {
                    "ID": "Initial Balance",
                    "Value": 0,
                    "ExpiryTime": "+4320h",
                    "Weight": 1,
                    "Blocker": true
                }
            }]
          }
        status_code: 200
      register: balance_response

    # 8. □ CRM □□□□□□□
    - name: Get current date and time in ISO 8601 format
      command: date --utc +%Y-%m-%dT%H:%M:%S%z
      register: current_date_time

    - name: Add Service via API
      uri:
        url: "http://localhost:5000/crm/service/"
        method: PUT
        body_format: json
        headers:
          Content-Type: "application/json"
          Authorization: "Bearer {{ access_token }}"
        body:
```

```
            {
              "customer_id": "{{ customer_id }}",
              "product_id": "{{ product_id }}",
              "service_name": "{{ package_name }} - {{
service_uuid }}",
              "service_type": "generic",
              "service_uuid": "{{ service_uuid }}",
              "service_billed": true,
              "service_taxable": true,
              "service_provisioned_date": "{{
current_date_time.stdout }}",
              "service_status": "Active",
              "wholesale_cost": "{{
api_response_product.json.wholesale_cost | float }}",
              "retail_cost": "{{ monthly_cost | float }}"
            }
          status_code: 200
        register: service_creation_response

      # 9. □□□□□□□□
      - name: Add Setup Cost Transaction via API
        uri:
          url: "http://localhost:5000/crm/transaction/"
          method: PUT
          headers:
            Content-Type: "application/json"
            Authorization: "Bearer {{ access_token }}"
          body_format: json
          body:
            {
              "customer_id": {{ customer_id | int }},
              "service_id": {{
service_creation_response.json.service_id | int }},
              "title": "{{ package_name }} - Setup Costs",
              "description": "Setup costs for {{ package_comment
}}",
              "invoice_id": null,
              "retail_cost": "{{ setup_cost | float }}"
            }
          return_content: yes
        register: transaction_response

      # 10. □□□□□□□
      - include_tasks: post_provisioning_tasks.yaml
```

```
    rescue:

    # □□/□□□□
    - name: Print all vars for debugging
      debug:
        var: hostvars[inventory_hostname]

    - name: Remove account in OCS
      uri:
        url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
        method: POST
        body_format: json
        body:
          {
            "method": "ApierV2.RemoveAccount",
            "params": [{
                "Tenant": "{{ crm_config.ocs.ocsTenant }}",
                "Account": "{{ service_uuid }}",
                "ReloadScheduler": true
            }]
          }
        status_code: 200
      ignore_errors: True
      when: service_uuid is defined

    - name: Delete Service from CRM if it was created
      uri:
        url: "http://localhost:5000/crm/service/service_id/{{
service_creation_response.json.service_id }}"
        method: DELETE
        headers:
          Authorization: "Bearer {{ access_token }}"
        status_code: 200
      ignore_errors: True
      when: service_creation_response is defined

    - name: Fail if not intentional deprovision
      assert:
        that:
          - action == "deprovision"
```

Topup/Recharge 模式

用于向现有服务添加信用、数据或时间。

```yaml
- name: Service Topup Playbook
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: Include vars of crm_config
      ansible.builtin.include_vars:
        file: "../../crm_config.yaml"
        name: crm_config

    # 1. 取服务信息
    - name: Get Service information from CRM API
      uri:
        url: "http://localhost:5000/crm/service/service_id/{{
service_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
      register: api_response_service

    # 2. 取产品信息（要充值的内容）
    - name: Get Product information from CRM API
      uri:
        url: "http://localhost:5000/crm/product/product_id/{{
product_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
      register: api_response_product

    # 3. 提取服务详细信息
    - name: Set service facts
      set_fact:
        service_uuid: "{{ api_response_service.json.service_uuid
}}"
        customer_id: "{{ api_response_service.json.customer_id }}"
        package_name: "{{ api_response_product.json.product_name
}}"
        topup_value: "{{ api_response_product.json.retail_cost }}"
```

```
    # 4. 在计费系统中执行操作（免费充值）
    - name: Execute Action to add credits
      uri:
        url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
        method: POST
        body_format: json
        body:
          {
            "method": "APIerSv1.ExecuteAction",
            "params": [{
                "Tenant": "{{ crm_config.ocs.ocsTenant }}",
                "Account": "{{ service_uuid }}",
                "ActionsId": "Action_Topup_Standard"
            }]
          }
        status_code: 200
      register: action_response

    - name: Verify action executed successfully
      assert:
        that:
          - action_response.status == 200
          - action_response.json.result == "OK"

    # 5. 重置任何触发的限制
    - name: Reset ActionTriggers
      uri:
        url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
        method: POST
        body_format: json
        body:
          {
            "method": "APIerSv1.ResetAccountActionTriggers",
            "params": [{
                "Tenant": "{{ crm_config.ocs.ocsTenant }}",
                "Account": "{{ service_uuid }}",
                "Executed": false
            }]
          }
        status_code: 200

    # 6. 更新服务日期
    - name: Calculate new expiry date
      command: "date --utc +%Y-%m-%dT%H:%M:%S%z -d '+30 days'"
```

```
      register: new_expiry_date

    - name: Update Service with new expiry
      uri:
        url: "http://localhost:5000/crm/service/{{ service_id }}"
        method: PATCH
        headers:
          Authorization: "Bearer {{ access_token }}"
          Content-Type: "application/json"
        body_format: json
        body:
          {
            "service_deactivate_date": "{{ new_expiry_date.stdout
}}",
            "service_status": "Active"
          }

    # 7. 可选：发送通知
    - name: Send Notification SMS
      uri:
        url: "http://sms-gateway/api/send"
        method: POST
        body_format: json
        body:
          {
            "source": "CompanyName",
            "destination": "{{ customer_phone }}",
            "message": "Your service has been topped up. New
expiry: {{ new_expiry_date.stdout }}"
          }
        status_code: 201
      ignore_errors: True
```

CPE 配置模式

用于配置客户场所设备（路由器、调制解调器、ONT）。

```yaml
- name: CPE Provisioning Playbook
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: Include vars of crm_config
      ansible.builtin.include_vars:
        file: "../../crm_config.yaml"
        name: crm_config

    # 1. 取 CPE 的库存项目
    - name: Set CPE inventory ID from hostvars
      set_fact:
        cpe_inventory_id: "{{ hostvars[inventory_hostname]['WiFi
Router CPE'] | int }}"
      when: "'WiFi Router CPE' in hostvars[inventory_hostname]"

    # 2. 从库存中取 CPE 详细信息
    - name: Get Inventory data for CPE
      uri:
        url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ cpe_inventory_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
      register: api_response_cpe

    # 3. 取客户站点信息
    - name: Get Site info from API
      uri:
        url: "{{ crm_config.crm.base_url
}}/crm/site/customer_id/{{ customer_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
      register: api_response_site

    # 4. 用位置更新 CPE 库存
    - name: Patch CPE inventory item with location
      uri:
```

```
        url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ cpe_inventory_id }}"
        method: PATCH
        body_format: json
        headers:
          Authorization: "Bearer {{ access_token }}"
        body:
          {
            "address_line_1": "{{
api_response_site.json.0.address_line_1 }}",
            "city": "{{ api_response_site.json.0.city }}",
            "state": "{{ api_response_site.json.0.state }}",
            "latitude": "{{ api_response_site.json.0.latitude }}",
            "longitude": "{{ api_response_site.json.0.longitude
}}"
          }
        status_code: 200

    # 5. □□□□
    - name: Set CPE hostname
      set_fact:
        cpe_hostname: "CPE_{{ cpe_inventory_id }}"
        cpe_username: "admin_{{ cpe_inventory_id }}"

    - name: Generate random password
      set_fact:
        cpe_password: "{{ lookup('pipe', 'cat /dev/urandom | tr -
dc a-zA-Z0-9 | head -c 16') }}"

    # 6. □□ WiFi □□
    - name: Set WiFi SSID
      set_fact:
        wifi_ssid: "Network_{{ cpe_inventory_id }}"

    - name: Generate WiFi password
      set_fact:
        word_list:
          - apple
          - cloud
          - river
          - mountain
          - ocean

    - name: Create WiFi PSK
```

```
      set_fact:
        random_word: "{{ word_list | random }}"
        random_number: "{{ 99999 | random(start=10000) }}"

    - name: Combine WiFi PSK
      set_fact:
        wifi_psk: "{{ random_word }}{{ random_number }}"

    # 7. □□□□□□
    - name: Set config filename
      set_fact:
        config_name: "{{ cpe_hostname }}_{{ lookup('pipe', 'date
+%Y%m%d%H%M%S') }}.cfg"
        config_dest: "/tmp/{{ cpe_hostname }}_{{ lookup('pipe',
'date +%Y%m%d%H%M%S') }}.cfg"

    - name: Create config from template
      template:
        src: "templates/cpe_router_config.j2"
        dest: "{{ config_dest }}"

    # 8. □□□□□□□
    - name: Read config file
      ansible.builtin.slurp:
        src: "{{ config_dest }}"
      register: config_content

    # 9. □□□□□□□
    - name: Patch CPE inventory with config
      uri:
        url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ cpe_inventory_id }}"
        method: PATCH
        body_format: json
        headers:
          Authorization: "Bearer {{ access_token }}"
        body:
          {
            "itemtext3": "{{ wifi_ssid }}",
            "itemtext4": "{{ wifi_psk }}",
            "management_url": "{{ cpe_hostname }}",
            "management_username": "{{ cpe_username }}",
            "management_password": "{{ cpe_password }}",
            "config_content": "{{ config_content.content |
```

```
b64decode }}",
            "inventory_notes": "Provisioned: {{ lookup('pipe',
'date +%Y-%m-%d') }}"
          }
        status_code: 200

    # 10. 将配置发送给支持团队
    - name: Email configuration to support
      uri:
        url: "https://api.mailjet.com/v3.1/send"
        method: POST
        body_format: json
        headers:
          Content-Type: "application/json"
        body:
          {
            "Messages": [{
              "From": {
                "Email": "provisioning@example.com",
                "Name": "Provisioning System"
              },
              "To": [{
                "Email": "support@example.com",
                "Name": "Support Team"
              }],
              "Subject": "CPE Config - {{ cpe_hostname }}",
              "Attachments": [{
                "ContentType": "text/plain",
                "Filename": "{{ config_name }}",
                "Base64Content": "{{ config_content.content }}"
              }]
            }]
          }
        user: "{{ mailjet_api_key }}"
        password: "{{ mailjet_api_secret }}"
        force_basic_auth: true
        status_code: 200
```

自动续费模式

使用 CGRateS ActionPlans 配置自动定期收费或续费。

```
# 部分充值 playbook 设置自动续费

# 1. 规范化 auto_renew 参数
- name: Normalize auto_renew to boolean
  set_fact:
    auto_renew_bool: "{{ (auto_renew | string | lower) in ['true',
'1', 'yes'] }}"

# 2. 创建自动续费的操作
- name: Create Action for AutoRenew
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.SetActions",
        "params": [{
          "ActionsId": "Action_AutoTopup_{{ service_uuid }}_{{
product_id }}",
          "Overwrite": true,
          "Actions": [
            {
              "Identifier": "*http_post",
              "ExtraParameters": "{{ crm_config.crm.base_url
}}/crm/provision/simple_provision_addon/service_id/{{ service_id
}}/product_id/{{ product_id }}"
            },
            {
              "Identifier": "*cdrlog",
              "BalanceType": "*generic",
              "ExtraParameters": "
{\"Category\":\"^activation\",\"Destination\":\"Auto Renewal\"}"
            }
          ]
        }]
      }
    status_code: 200
  register: action_response
  when: auto_renew_bool

# 3. 创建每月 ActionPlan
- name: Create ActionPlan for Monthly Renewal
```

```
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.SetActionPlan",
        "params": [{
          "Id": "ActionPlan_Monthly_{{ service_uuid }}_{{
product_id }}",
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "ActionPlan": [{
            "ActionsId": "Action_AutoTopup_{{ service_uuid }}_{{
product_id }}",
            "Years": "*any",
            "Months": "*any",
            "MonthDays": "*any",
            "WeekDays": "*any",
            "Time": "*monthly",
            "StartTime": "*now",
            "Weight": 10
          }],
          "Overwrite": true,
          "ReloadScheduler": true
        }]
      }
    status_code: 200
  when: auto_renew_bool

# 4. □ ActionPlan □□□□□
- name: Assign ActionPlan to account
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV2.SetAccount",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "ActionPlanIds": ["ActionPlan_Monthly_{{ service_uuid
}}_{{ product_id }}"],
          "ActionPlansOverwrite": true,
```

```
            "ReloadScheduler": true
          }]
        }
      status_code: 200
  when: auto_renew_bool

# 5. 如果禁用自动续费，则移除 ActionPlan
- name: Remove ActionPlan from account
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.RemoveActionPlan",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Id": "ActionPlan_Monthly_{{ service_uuid }}_{{
product_id }}"
        }]
      }
    status_code: 200
  ignore_errors: true
  when: not auto_renew_bool
```

可重用任务

可重用任务□小型、自包含的 playbooks，可以被多个 plays 包含。它们促进代码重用和一致性。

欢迎电子邮件任务

`task_welcome_email.yaml` - 向新客户发送欢迎电子邮件。

```
# 此任务期望这些变量由父 play 设置:
# - api_response_customer (客户详细信息)
# - package_name (产品名称)
# - monthly_cost (定期费用)
# - setup_cost (一次性费用)

- name: Set email configuration
  set_fact:
    mailjet_api_key: "{{ lookup('env', 'MAILJET_API_KEY') }}"
    mailjet_api_secret: "{{ lookup('env', 'MAILJET_SECRET') }}"
    email_from: "noreply@example.com"
    recipients: []

- name: Set email subject and sender name
  set_fact:
    email_subject: "Welcome to our service!"
    email_from_name: "Customer Service Team"

- name: Prepare list of recipients from customer contacts
  loop: "{{ api_response_customer.json.contacts }}"
  set_fact:
    recipients: "{{ recipients + [{'Email': item.contact_email,
'Name': item.contact_firstname ~ ' ' ~ item.contact_lastname}] }}"

- name: Get first contact name
  set_fact:
    first_contact: "{{
api_response_customer.json.contacts[0].contact_firstname }}"

- name: Send welcome email
  uri:
    url: "https://api.mailjet.com/v3.1/send"
    method: POST
    body_format: json
    headers:
      Content-Type: "application/json"
    body:
      {
        "Messages": [{
          "From": {
            "Email": "{{ email_from }}",
            "Name": "{{ email_from_name }}"
          },
```

```
            "To": "{{ recipients }}",
            "Subject": "{{ email_subject }}",
            "TextPart": "Dear {{ first_contact }}, welcome! Your
service is ready.",
            "HTMLPart": "Dear {{ first_contact }},<br/><h3>Welcome!
</h3><br/>Your {{ package_name }} service is now active.
<br/>Monthly cost: ${{ monthly_cost }}<br/>Setup fee: ${{
setup_cost }}<br/>If you have any questions, contact
support@example.com"
          }]
        }
      user: "{{ mailjet_api_key }}"
      password: "{{ mailjet_api_secret }}"
      force_basic_auth: true
      status_code: 200
    register: email_response
```

后配置任务

`post_provisioning_tasks.yaml` - 在每次配置后运行的标准清理和通知。

```
# 此文件在大多数配置 playbooks 的末尾包含
# 它处理常见的后配置操作

- include_tasks: task_notify_ocs.yaml
```

`task_notify_ocs.yaml` 可能包含:

```
- name: Notify OCS of provisioning completion
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "APIerSv1.ReloadCache",
        "params": [{
          "ArgsCache": "*all"
        }]
      }
    status_code: 200
  ignore_errors: true
```

常见□作

与库存合作

检索库存详细信息：

```
- name: Get SIM Card inventory ID
  set_fact:
    sim_inventory_id: "{{ hostvars[inventory_hostname]['SIM Card']
| int }}"
  when: "'SIM Card' in hostvars[inventory_hostname]"

- name: Get SIM Card details
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ sim_inventory_id }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: sim_response

- name: Extract SIM details
  set_fact:
    iccid: "{{ sim_response.json.iccid }}"
    imsi: "{{ sim_response.json.imsi }}"
    ki: "{{ sim_response.json.ki }}"
```

□□□□□□□□□□

```
- name: Assign SIM to customer
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ sim_inventory_id }}"
    method: PATCH
    headers:
      Authorization: "Bearer {{ access_token }}"
    body_format: json
    body:
      {
        "customer_id": {{ customer_id }},
        "service_id": {{ service_id }},
        "item_state": "Assigned"
      }
    status_code: 200
```

日期和时间□作

获取当前日期/时间:

```
- name: Get current date and time in ISO 8601 format
  command: date --utc +%Y-%m-%dT%H:%M:%S%z
  register: current_date_time

- name: Get today's date only
  set_fact:
    today: "{{ lookup('pipe', 'date +%Y-%m-%d') }}"
```

计算未来日期:

```
- name: Calculate expiry date 30 days from now
  command: "date --utc +%Y-%m-%dT%H:%M:%S%z -d '+30 days'"
  register: expiry_date

- name: Calculate date 90 days in future
  command: "date --utc +%Y-%m-%d -d '+{{ days }} days'"
  register: future_date
  vars:
    days: 90
```

生成随机值

UUID 和标识符:

```
- name: Generate UUID
  set_fact:
    uuid: "{{ 99999999 | random | to_uuid }}"

- name: Generate service identifier
  set_fact:
    service_uuid: "SVC_{{ uuid[0:8] }}"
```

随机密码:

```
- name: Generate secure password
  set_fact:
    password: "{{ lookup('pipe', 'cat /dev/urandom | tr -dc a-zA-
Z0-9 | head -c 16') }}"
```

可记忆的密码短语：

```
- name: Set word list
  set_fact:
    words:
      - alpha
      - bravo
      - charlie
      - delta
      - echo

- name: Generate passphrase
  set_fact:
    word: "{{ words | random }}"
    number: "{{ 99999 | random(start=10000) }}"

- name: Combine into passphrase
  set_fact:
    passphrase: "{{ word }}{{ number }}"
```

与 **CGRateS/OCS** 合作

创建账户：

```
- name: Create billing account
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV2.SetAccount",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "ActionPlanIds": [],
          "ActionPlansOverwrite": true,
          "ExtraOptions": {
            "AllowNegative": false,
            "Disabled": false
          },
          "ReloadScheduler": true
        }]
      }
    status_code: 200
  register: account_response
```

□□□□□

```
- name: Add data balance
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Categories": "*any",
          "Balance": {
            "ID": "Data Package",
            "Value": 10737418240,
            "ExpiryTime": "+720h",
            "Weight": 10
          }
        }]
      }
    status_code: 200
```

□□□□□

```
- name: Execute charging action
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "APIerSv1.ExecuteAction",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "ActionsId": "Action_Standard_Charge"
        }]
      }
    status_code: 200
```

获取账户信息：

```
- name: Get account details
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV2.GetAccount",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}"
        }]
      }
    status_code: 200
  register: account_info
```

与属性配置文件合作：

```
- name: Get AttributeProfile
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "APIerSv1.GetAttributeProfile",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "ID": "ATTR_{{ service_uuid }}"
        }]
      }
    return_content: yes
    status_code: 200
  register: attr_response
  ignore_errors: true

- name: Extract attribute value
  set_fact:
    phone_number: "{{ attr_response.json.result.Attributes |
json_query(\"[?Path=='*req.PhoneNumber'].Value[0].Rules\") | first
}}"
  when: attr_response is defined
```

条件逻辑

检查变量□否存在:

```
- name: Use custom value or default
  set_fact:
    monthly_cost: "{{ custom_cost | default(50.00) }}"

- name: Only run if variable is defined
  debug:
    msg: "Service UUID is {{ service_uuid }}"
  when: service_uuid is defined
```

布尔条件:

```
- name: Provision equipment
  include_tasks: configure_cpe.yaml
  when: provision_cpe | default(false) | bool

- name: Skip if deprovision
  assert:
    that:
      - action != "deprovision"
  when: action is defined
```

多个条件:

```
- name: Complex conditional task
  uri:
    url: "{{ endpoint }}"
    method: POST
  when:
    - service_uuid is defined
    - customer_id is defined
    - action != "deprovision"
    - enable_feature | default(true) | bool
```

循环和迭代

简单循环:

```
- name: Create multiple balances
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Account": "{{ service_uuid }}",
          "BalanceType": "{{ item.type }}",
          "Balance": {
            "Value": "{{ item.value }}"
          }
        }]
      }
  loop:
    - { type: "*voice", value: 3600 }
    - { type: "*data", value: 10737418240 }
    - { type: "*sms", value: 100 }
```

□□ API □□□

```
- name: Get all customer sites
  uri:
    url: "{{ crm_config.crm.base_url }}/crm/site/customer_id/{{
customer_id }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
  register: sites_response

- name: Configure equipment at each site
  debug:
    msg: "Configuring site at {{ item.address_line_1 }}"
  loop: "{{ sites_response.json }}"
```

□□□□

□□ ignore_errors□

```
- name: Optional SMS notification
  uri:
    url: "http://sms-gateway/send"
    method: POST
    body: {...}
  ignore_errors: true
```

□□□□□□□□

```
- name: Verify API response
  assert:
    that:
      - response.status == 200
      - response.json.result == "OK"
    fail_msg: "API call failed: {{ response.json }}"
```

□□□□□□□

```
- name: Try to get existing service
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/service/service_uuid/{{ service_uuid }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
  register: service_lookup
  failed_when: false

- name: Create service if it doesn't exist
  uri:
    url: "{{ crm_config.crm.base_url }}/crm/service/"
    method: PUT
    body: {...}
  when: service_lookup.status == 404
```

最佳实践

变量命名

使用描述性、一致的名称：

```
# 好
service_uuid: "SVC_12345"
customer_name: "John Smith"
monthly_cost: 49.99

# 坏
svc: "SVC_12345"
name: "John Smith"
cost: 49.99
```

按来源前缀变量：

```
api_response_customer: {...}
api_response_product: {...}
cgr_account_info: {...}
```

调试

打印变量以进行故障排除：

```
- name: Print all variables
  debug:
    var: hostvars[inventory_hostname]

- name: Print specific variable
  debug:
    msg: "Service UUID: {{ service_uuid }}"

- name: Print API response
  debug:
    var: api_response_product.json
```

验证

始终验证关键 API 响应：

```
- name: Create account
  uri:
    url: "{{ billing_endpoint }}"
    method: POST
    body: {...}
  register: response

- name: Verify account creation
  assert:
    that:
      - response.status == 200
      - response.json.result == "OK"
    fail_msg: "Failed to create account: {{ response.json }}"
```

幂等性

设计任务以安全地重新运行：

```
# 首先检查资源是否存在
- name: Check if account exists
  uri:
    url: "{{ ocs_endpoint }}/get_account"
    method: POST
    body: {"Account": "{{ service_uuid }}"}
  register: account_check
  failed_when: false

# 仅在不存在时创建
- name: Create account
  uri:
    url: "{{ ocs_endpoint }}/create_account"
    method: POST
    body: {...}
  when: account_check.status == 404
```

安全性

绝不要硬编码凭据：

```
# 坏
mailjet_api_key: "abc123def456"

# 好 - 使用环境变量
mailjet_api_key: "{{ lookup('env', 'MAILJET_API_KEY') }}"

# 好 - 使用配置文件
mailjet_api_key: "{{ crm_config.email.api_key }}"
```

始终使用 HTTPS 和身份验证：

```
- name: Call external API
  uri:
    url: "https://api.example.com/endpoint"
    method: POST
    headers:
      Authorization: "Bearer {{ access_token }}"
    validate_certs: yes
```

文档

记录复杂逻辑：

```
# 计算部分月份的按比例收费
# 如果客户在 15 号注册，而计费在 1 号，
# 则按剩余天数收取 50% 的月费
- name: Calculate days until end of month
  command: "date -d 'last day of this month' +%d"
  register: days_in_month

- name: Get current day
  command: "date +%d"
  register: current_day

- name: Calculate pro-rata amount
  set_fact:
    days_remaining: "{{ (days_in_month.stdout | int) -
(current_day.stdout | int) }}"
    pro_rata_cost: "{{ (monthly_cost | float) * (days_remaining |
float) / (days_in_month.stdout | float) }}"
```

测试 Playbooks

测试方法

1. 首先进行干运行：在非生产系统上测试
2. 验证变量：使用调试任务确认所有必需变量均已存在
3. 检查响应：在继续之前验证 API 响应
4. 回滚测试：故意使任务失败以验证回收块工作
5. 去配置测试：使用 `action: "deprovision"` 测试以验证清理

示例测试 playbook：

```yaml
- name: Test Service Provisioning
  hosts: localhost
  gather_facts: no

  tasks:
    - name: Verify required variables
      assert:
        that:
          - product_id is defined
          - customer_id is defined
          - access_token is defined
        fail_msg: "Missing required variables"

    - name: Test API connectivity
      uri:
        url: "http://localhost:5000/crm/health"
        method: GET
      register: health_check

    - name: Verify health check
      assert:
        that:
          - health_check.status == 200
```

常见陷阱

缺少类型转换：

```yaml
# 错误 - 可能是字符串
customer_id: "{{ customer_id }}"

# 正确 - 确保为整数
customer_id: {{ customer_id | int }}
```

未处理未定义的变量：

```
# 错误 - 如果未定义则失败
service_uuid: "{{ service_uuid }}"

# 正确 - 提供默认值
service_uuid: "{{ service_uuid | default('') }}"
```

忘记验证：

```
# 错误 - 不检查响应
- name: Create account
  uri: ...
  register: response

# 正确 - 验证响应
- name: Create account
  uri: ...
  register: response

- name: Verify creation
  assert:
    that:
      - response.json.result == "OK"
```

配置工作流

通常，Omnitouch 员工将与客户合作：

1. 定义产品要求
2. 开发必要的 Ansible playbooks 以自动化配置过程
3. 在暂存环境中测试 playbooks
4. 部署到生产

这确保每项服务都以一致和可靠的方式部署，减少错误的风险，并确保所有必要步骤以正确的顺序完成。

Ansible 变量

传递给 Ansible playbooks 的变量包括：

产品变量

来自 OmniCRM 产品配置，定义服务应如何设置。

库存变量

从库存中选择，包括调制解调器、SIM 卡、IP 地址块或配置所需的电话号码等项目。

系统变量

由 OmniCRM 自动添加：

- `product_id` - 正在配置的产品
- `customer_id` - 接收服务的客户
- `service_id` - 正在修改的服务（用于充值/更改）
- `access_token` - 用于 API 身份验证的 JWT

去配置

当服务不再需要时，**Ansible Playbooks** 也用于使用 `rescue` 块模式去配置服务。这：

- 移除任何配置
- 将库存释放回池中
- 删除计费账户
- 确保系统保持干净

去配置方法

去配置主要有两种触发方式：

1. 自动回滚（配置失败）

当主配置块中的任何任务失败时，回收部分自动执行以清理部分更改。

2. 手动去配置

在运行 playbook 时设置 `action: "deprovision"`，故意触发回收块以移除服务。

去配置模式

所有 OmniCRM playbooks 遵循此结构以安全清理:

```yaml
- name: Service Provisioning Playbook
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: Main block
      block:
        # 所有配置任务在这里
        - name: Create OCS account
          uri: ...

        - name: Create service record
          uri: ...
          register: service_creation_response

        - name: Add transaction
          uri: ...

      rescue:
        # 去配置/回滚任务在这里
        - name: Print all vars for debugging
          debug:
            var: hostvars[inventory_hostname]

        # 清理任务按相反顺序执行
        - name: Remove account in OCS
          uri: ...

        - name: Delete Service from CRM
          uri: ...

        - name: Fail if not intentional deprovision
          assert:
            that:
              - action == "deprovision"
```

完整去配置示例

以下 `play_simple_service.yaml` 的完整示例:

```
rescue:
  # 1. 调试信息
  - name: Print all vars for Deprovision/Rollback
    debug:
      var: hostvars[inventory_hostname]

  - name: Get today's date
    set_fact:
      today: "{{ lookup('pipe', 'date +%Y-%m-%d') }}"

  # 2. 如果可用，□取服务信息
  - name: Try to get Service information from CRM API to
Deprovision
    uri:
      url: "http://localhost:5000/crm/service/service_id/{{
service_creation_response.json.service_id }}"
      method: GET
      headers:
        Authorization: "Bearer {{ access_token }}"
      return_content: yes
    register: api_response_service
    ignore_errors: True
    when: service_creation_response is defined and
service_creation_response.json is defined and
service_creation_response.json.service_id is defined

  - name: Print api_response_service
    debug:
      var: api_response_service
    when: api_response_service is defined

  # 3. 为清理设置 service_uuid
  - name: Set service_uuid facts for Deprovision
    set_fact:
      service_uuid: "{{ api_response_service.json.service_uuid }}"
    when:
      - service_uuid is not defined
      - api_response_service is defined
      - api_response_service.json is defined

  # 4. 移除 OCS 账户
  - name: Remove account in OCS
    uri:
```

```
      url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
      method: POST
      body_format: json
      headers:
        Content-Type: "application/json"
        Authorization: "Bearer {{ access_token }}"
      body:
        {
          "method": "ApierV2.RemoveAccount",
          "params": [{
            "Tenant": "{{ crm_config.ocs.ocsTenant }}",
            "Account": "{{ service_uuid }}",
            "ReloadScheduler": true
          }]
        }
      status_code: 200
    register: response
    ignore_errors: True
    when: service_uuid is defined

  - name: Print response from OCS account removal
    debug:
      var: response
    when: response is defined

  # 5. □ CRM □□□□□□
  - name: Delete Service from CRM if it was created
    uri:
      url: "http://localhost:5000/crm/service/service_id/{{
service_creation_response.json.service_id }}"
      method: DELETE
      headers:
        Authorization: "Bearer {{ access_token }}"
      status_code: 200
    register: delete_service_response
    ignore_errors: True
    when: service_creation_response is defined and
service_creation_response.json is defined and
service_creation_response.json.service_id is defined

  - name: Print delete service response
    debug:
      var: delete_service_response
    when: delete_service_response is defined
```

```
  # 6. 确定这是故意的还是失败
  - name: Set status to "Success" if Manual deprovision / Fail if
failed provision
    assert:
      that:
        - action == "deprovision"
      fail_msg: "Provisioning failed and was rolled back"
      success_msg: "Service deprovisioned successfully"
```

关键去配置概念

ignore_errors: True

大多数清理任务使用 `ignore_errors: True`，因为资源可能不存在（例如，如果配置在创建它们之前失败）。

条件执行

使用 `when` 子句仅清理已创建的资源：

```
when: service_creation_response is defined
```

反向顺序

清理按创建的反向顺序进行：

1. 首先删除依赖资源（交易、库存分配）
2. 删除服务记录
3. 删除 OCS/计费账户
4. 释放任何被占用的资源

最终断言

最终断言区分：

- 故意去配置（`action == "deprovision"`）→ 成功
- 配置失败（未设置操作）→ 失败

去配置不同资源类型

OCS/CGRateS 账户：

```
- name: Remove account in OCS
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV2.RemoveAccount",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "ReloadScheduler": true
        }]
      }
    status_code: 200
  ignore_errors: True
  when: service_uuid is defined
```

CRM 服务记录:

```
- name: Delete Service from CRM
  uri:
    url: "http://localhost:5000/crm/service/service_id/{{
service_id }}"
    method: DELETE
    headers:
      Authorization: "Bearer {{ access_token }}"
    status_code: 200
  ignore_errors: True
  when: service_id is defined
```

库存项目（返回池）：

```
- name: Release SIM card back to inventory pool
  uri:
    url: "http://localhost:5000/crm/inventory/inventory_id/{{
sim_inventory_id }}"
    method: PATCH
    headers:
      Authorization: "Bearer {{ access_token }}"
    body_format: json
    body:
      {
        "customer_id": null,
        "service_id": null,
        "item_state": "Available"
      }
    status_code: 200
  ignore_errors: True
  when: sim_inventory_id is defined
```

ActionPlans□□□□□□□

```
- name: Remove ActionPlan from account
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.RemoveActionPlan",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Id": "ActionPlan_Monthly_{{ service_uuid }}_{{
product_id }}"
        }]
      }
    status_code: 200
  ignore_errors: True
```

□□□□□

```
- name: Release payment authorization
  uri:
    url: "http://localhost:5000/crm/payments/release/{{
authorization_id }}"
    method: POST
    headers:
      Authorization: "Bearer {{ access_token }}"
    body_format: json
    body:
      {
        "metadata": {
          "release_reason": "provisioning_failed"
        }
      }
    return_content: yes
  ignore_errors: True
  when: authorization_id is defined
```

手动去配置工作流

要手动去配置服务：

1. 确定要去配置的服务
2. 运行原始配置 playbook，设置 `action: "deprovision"`
3. Playbook 立即进入回收块
4. 所有清理任务执行
5. 服务被干净地移除

示例 **API** 调用：

```
POST /crm/provision/run_playbook
{
  "product_id": 123,
  "customer_id": 456,
  "service_id": 789,
  "action": "deprovision"
}
```

部分清理场景

场景 **1**: **OCS** 账户已创建，服务创建失败

- OCS 账户存在
- 服务记录不存在
- 回收块移除 OCS 账户
- 没有服务可删除（安全跳过）

场景 **2**: 服务已创建，交易失败

- OCS 账户存在
- 服务记录存在
- 交易不存在
- 回收块移除 OCS 账户和服务
- 没有交易可作废（从未创建）

场景 **3**: 完整配置，付款捕获失败

- OCS 账户存在
- 服务记录存在
- 付款已授权但未捕获
- 回收块:
  - 释放付款授权（客户未收费）
  - 移除服务记录
  - 移除 OCS 账户

去配置最佳实践

1. 广泛使用 ignore_errors

清理应宽容 - 如果资源不存在，则不失败。

2. 使用 when 子句检查资源是否存在

仅在资源已创建时尝试清理:

```
when: service_creation_response is defined
```

3. 打印调试信息

始终包括调试任务以帮助故障排除失败的配置：

```
- name: Print all vars for debugging
  debug:
    var: hostvars[inventory_hostname]
```

4. 以反向依赖顺序清理

在父项之前删除子项：

- 交易在服务之前
- 服务在 OCS 账户之前
- 库存分配在库存释放之前

5. 处理付款授权

始终在回收块中释放付款���权，以避免对失败的配置向客户收费。

6. 清理后重新加载调度程序

在移除 OCS 资源时，包括 `ReloadScheduler: true` 以确保 CGRateS 立即更新。

去配置与删除

去配置（通过 playbook 回收）：

- 从所有系统中移除服务
- 释放库存
- 取消定期收费
- 保留审计记录
- 推荐的方法

直接删除（通过 API）：

- 仅删除 CRM 记录
- 不清理 OCS 账户
- 不释放库存
- 可能留下孤立的资源
- 不推荐用于生产

回滚和错误处理

Ansible 的 **block/rescue** 特性在配置和去配置过程中用于优雅地处理错误。如果在配置的任何点任务失败，回收部分将自动回滚更改，以返回到一致状态。这确保了可靠性并减少了部分或失败部署的风险。

错误处理示例

```
- name: Main block
  block:
    - name: Step 1: Create account
      uri: ...
      register: response

    - name: Verify step 1
      assert:
        that:
          - response.status == 200

    - name: Step 2: Create service
      uri: ...

  rescue:
    # 如果任何断言失败或任何任务出错:
    # 1. 执行跳转到回收块
    # 2. 清理任务执行
    # 3. Playbook 失败并显示错误信息
    - name: Cleanup partial changes
      uri: ...
```

有关配置系统、工作流和身份验证的完整详细信息，请参见 `concepts_provisioning`。

OmniCRM API

OmniCRM 中的所有功能都可以通过 API 访问 - 没有仅在 UI 中可用的功能。

这使您能够将 OmniCRM 与其他系统集成或自动化任务。

API 是一个 RESTful API，并使用多种身份验证方法进行安全保护，包括 JWT 令牌、API 密钥和 IP 白名单。

API 使用 Swagger 进行文档编制，这是一个允许轻松阅读、理解和测试 API 功能的工具。

API 文档可在以下 URL 获取：

> <https://yourcrm/crm/docs/>

身份验证方法

OmniCRM 支持三种身份验证方法，每种方法都针对不同的用例设计：

1. **JWT Bearer** 令牌 - 用于交互式用户会话（Web UI、移动应用）

2. **API** 密钥 - 用于服务器到服务器的集成和自动化脚本
3. **IP** 白名单 - 用于受信任的内部系统（配置服务器、监控工具）

JWT Bearer 令牌身份验证

这□用户会话的主要身份验证方法。用户使用电子邮件和密码登录，接收 JWT 令牌，并在后续请求中使用它。

用例：

- Web UI 身份验证
- 移动应用身份验证
- 短期程序化访问

如何进行身份验证：

要登录，请将以下结构的 JSON 主体作为 POST 请求发送到 `/crm/auth/login`：

```
{
    "email": "youruser@yourdomain.com",
    "password": "yourpassword"
}
```

API 将返回一个包含 `token` 字段的 JSON 对象，该字段用于验证所有未来的请求。此外，响应还包括一个 `refresh_token`，可以在令牌过期时用于刷新令牌，以及用户的权限和角色。

您可以通过 Swagger 页面测试此功能，选择 `/auth/login` 端点，填写您的用户名和密码，然后单击 `Try it out` 按钮。

要授权会话，请复制令牌值并单击 Swagger 页面右上角的“Authorize”按钮。在“Value”字段中粘贴令牌，前面加上 `Bearer`，然后单击“Authorize”。

现在，所有后续请求将使用此令牌进行身份验证。

API 密钥身份验证

API 密钥为服务器到服务器的集成和自动化脚本提供安全、长期的身份验证，而无需用户密码。

用例：

- 自动化配置系统
- 监控和警报工具
- 与外部系统的集成
- 定时任务和 cron 作业

API 密钥的工作原理：

API 密钥在 `crm_config.yaml` 文件中配置，并与特定角色和权限相关联。每个 API 密钥都□一个安全的随机字符串（最小 32 个字符），在通过 `X-API-KEY` 头传递时用于验证请求。

配置 **API** 密钥：

API 密钥必须由具有服务器访问权限的管理员添加到 `crm_config.yaml` 中：

```
api_keys:
  your-secure-api-key-here-minimum-32-chars:
    roles:
      - admin
    description: "Provisioning automation system"
  another-api-key-for-monitoring-system:
    roles:
      - view_customer
      - view_service
    description: "Monitoring and alerting"
```

使用 **API** 密钥:

在请求的 `X-API-KEY` 头中包含 API 密钥:

```
curl -X GET "https://yourcrm.com/crm/customers" \
  -H "X-API-KEY: your-secure-api-key-here-minimum-32-chars"
```

Python 示例:

```
import requests

crm_url = 'https://yourcrm.com'
api_key = 'your-secure-api-key-here-minimum-32-chars'

headers = {
    "Content-Type": "application/json",
    "X-API-KEY": api_key
}

# □取客户
response = requests.get(crm_url + '/crm/customers',
headers=headers)
for customer in response.json()['data']:
    print(customer)
```

最佳实践:

- 使用加密安全的随机生成器生成 API 密钥 (`openssl rand -base64 48`)

- 为不同的系统使用不同的 API 密钥
- 在 `description` 字段中记录每个 API 密钥的目的
- 定期轮换 API 密钥
- 永远不要将 API 密钥提交到版本控制
- 为每个 API 密钥分配最小必要权限

IP 白名单身份验证

IP 白名单允许特定 IP 地址在无需身份验证的情况下访问 API。这对于位于私有网络上的受信任的内部系统非常有用。

用例：

- 内部配置服务器
- 管理 VLAN 上的网络监控系统
- 在受控基础设施上运行的 Ansible 剧本

配置 **IP** 白名单：

将受信任的 IP 地址添加到 `crm_config.yaml`：

```
ip_whitelist:
  - 192.168.1.100
  - 10.0.0.0/24
  - 172.16.50.10
```

安全考虑：

- 仅在私有、安全的网络上使用 IP 白名单
- 永远不要将公共 IP 地址列入白名单
- 使用尽可能具体的 IP 范围
- 记录每个 IP 被列入白名单的原因
- 定期审核白名单中的 IP

使用 **Python** 的示例 **API** 调用

以下□如何使用 JWT 令牌身份验证登录并检索客户列表的示例：

```python
import requests

crm_url = 'https://yourcrm.com'
session = requests.Session()

print("Provisioning data to server: " + str(crm_url))

headers = {
    "Content-Type": "application/json"
}

# □取���份验证令牌
response = session.post(crm_url + '/crm/auth/login', json={
    "email": "youruser@yourdomain.com",
    "password": "yourpassword"
}, headers=headers)

print(response.status_code)
print(response.json())
assert response.status_code == 200

headers['Authorization'] = 'Bearer ' + response.json()['token']
print("Authenticated to CRM successfully")

# □取客户
response = session.get(crm_url + '/crm/customers',
headers=headers)
for customer in response.json()['data']:
    print(customer)
```

API 性能监控

所有 API 请求都自动跟踪并提供全面的指标。有关 API 性能指标的详细信息，包括请求速率、响应时间、错误率和特定端点的统计信息，请参见 Monitoring & Metrics。

指标端点以 Prometheus 格式在 `/crm/metrics` 提供，用于监控和警报。

相关文档

- System Architecture - 整体系统设计和组件关系
- Monitoring & Metrics - 完整的指标参考，包括 API、数据库和集成指标
- Authentication Flows - 详细的身份验证和授权流程
- RBAC - 基于角色的访问控制和权限

CRM 服务 / 产品收费说明

注意

有关产品定义、供应、附加组件和去供应的完整端到端指南，包含详细的 Ansible 示例和定价策略，请参见 完整产品生命周期指南 `<guide_product_lifecycle>`。

有关使用 SIM 卡供应移动服务的详细信息，请参见 `SIM 卡供应 <concepts_sim_provisioning>`。

产品与服务概述

产品（菜单项）：

产品就像餐厅菜单上的特定菜肴，例如“意大利培根意面”。

它有明确的描述、成分列表（如意大利面、奶油、鸡蛋、奶酪和培根）和价格。

在 OmniCRM 中，产品同样包含所包含内容的详细信息——功能、规格和定价。

通常，客户可能想要修改，例如“不要洋葱”或“加点额外的奶酪”。在 OmniCRM 中，这对应于在创建之前自定义服务。对服务的自定义或修改的程度由您（运营商）定义。

在 OmniCRM 中，客户或员工可能会修改产品，以更好地满足特定客户的需求，例如升级他们的互联网□□□度或添加特定功能。这种自定义反映在提供的特定服务中。

产品本质上□客户可以选择订购的产品，类似于从菜单上阅读和选择菜肴。

产品目录（餐厅菜单）：

产品目录就像餐厅的整个菜单，列出了所有可用的菜肴——从开胃菜到甜点。

它餐厅（或在您的情况下，服务提供商）所提供的一切的完整集合。

在商业上下文中，产品目录为客户提供所有可用的产品，以便他们可以选择最符合其需求的产品。

服务（准备好的菜肴）：

当客户从菜单上点餐时，菜肴在厨房中准备。这类似于从产品创建服务。

在 OmniCRM 中，当客户选择产品时，会创建该产品的实例并作为服务交付。

它□专门为该客户定制和准备的，就像为顾客准备的餐点。

例如，当某人从产品目录中选择“互联网铜牌计划”时，供应系统会“烹饪”出该计划的实例，使用的成分（IP 地址、调制解调器和端口）——即，激活该计划并将其交付给特定客户。

捆绑产品（组合餐）：

产品目录还可能提供捆绑，例如组合餐，其中包括开胃菜、主菜和甜点，以特价提供。

在 OmniCRM 中，捆绑产品将多个单独的产品组合成一个方便的包——例如“家庭必需品捆绑包”，其中包括以折扣价提供的互联网、有线电视和电话服务。

一旦选择，该捆绑包就会变成多个为客户量身定制的服务。

产品定义

产品□用于创建服务/附加组件/折扣/附加服务等的模板。

在定义中，我们包括：

- 关于产品的信息（功能、包含内容、条款和条件、合同长度、图标等），这些信息显示给 CRM 的用户（客户或 □□）。
- 关于谁可以购买该产品的业务逻辑（□□或 □□），□否依赖于已供应的父服务（例如，仅向拥有移动服务的客户提供移动附加组件），□否可以通过自助服务直接由客户订购，或仅由客户服务代理订购，以及何时可以购买该产品（允许产品仅在设定的时间段内可用）。
- 当要包含库存项目（例如调制解调器或 SIM 卡）时，这些项目被指定为库存项目列表，例如以下服务需要分配 SIM 卡和电话号码：

  `['SIM Card', 'Phone Number']` 这些与 CRM 中定义的 `库存项目 <administration_inventory>` 相关。
- 引用 Ansible Playbook 以供应服务 `供应 Play <concepts_ansible>` 以及要传递给 Ansible 的变量。这些要传递的变量□魔法的，因为它们可能□由我们添加到的产品定义的变量，如 `service_id`，或者它们可能□ `ICCID`

- `MSISDN`□□□

- **standalone** - 在创建新服务时显示为基本服务选项
- **addon** - 在向现有服务添加时显示
- **bundle** - 作为捆绑服务选项显示（像附加组件一样供应给现有服务）
- **promo** - 特别促销产品

这些类别纯粹□组织性的，并不决定什么被供应。实际的供应行为完全由引用的 Ansible playbook 中的逻辑决定。

例如：- `standalone` 产品通常会创建一个新的服务对象 - `addon` 或 `bundle` 产品通常会添加到现有服务中 - 但这取决于编写 playbook 的实施者 - 您可以从附加组件创建多个服务对象，或者根据需要从独立产品修改现有服务

类别仅控制 UI 流程以及客户/员工看到产品选项的位置。

服务类型

`service_type` 字段对提供的服务类型进行分类。

这些完全由用户定义，但常见值包括：

- **mobile** - 提供语音、短信和数据的手机服务
- **fixed** - 固定无线或有线互联网服务
- **fixed-voice** - 固定电话语音服务（VoIP、固定电话）
- **hotspot** - 移动热点或租赁设备
- **dongle** - USB 调制解调器或加密狗服务
- **voice** - 仅语音服务
- **data** - 仅数据服务

与类别一样，服务类型可以根据您的产品进行自定义。它们有助于：

- 过滤哪些附加组件适用于哪些基本服务
- 在客户门户中组织产品
- 匹配库存要求
- 确定供应工作流

示例：拥有 `mobile` 服务的客户可以看到移动附加组件，而拥有 `fixed` 服务的客户则可以看到固定线路附加组件。

□□□□

□□□

3. ⚙ 配置 - 续订设置、客户类型和依赖关系
4. 供应 - Ansible playbook 配置和库存要求
5. 可用性 - 日期范围和系统时间戳

定价选项卡组织：

定价选项卡将成本字段分组为逻辑部分：

- 经常性成本 - 每月零售和批发成本并排显示
- 设置成本 - 向客户收取的一次性激活费用
- 税 - 税率配置及自动计算

编辑模式功能：

- 图标选择器 - 以视觉方式搜索和选择 FontAwesome 图标
- 库存项目选择器 - 从可用库存项目类型中选择
- 日期/时间选择器 - 轻松选择可用窗口
- 货币格式化 - 成本字段自动添加 $ 前缀
- 下拉选择器 - 为类别和布尔字段提供预定义选项

图标选择器：

在编辑图标字段时，会出现一个可搜索的图标选择器界面，允许您以视觉方式浏览和选择数千个 FontAwesome 图标。

功能：* 按关键字搜索图标（例如，“扳手”、“移动”、“wifi”） * 实时预览图标外观 * 显示图标类名称以供参考 * 下拉选择以快速访问

配置选项卡：

配置选项卡将产品行为设置组织成逻辑组。

配置部分：

- 续订设置：
  - 自动续订 - 默认续订行为（提示/□/否）
  - 允许自动续订 - □否允许客户启用自动续订
  - 合同天数 - 最小合同长度（例如，30 表示每月，365 表示每年）
- 客户类型：
  - 住宅 - 可供消费者客户使用
  - 商业 - 可供商业客户使用
- 依赖关系：
  - 依赖列表 - 在添加此产品之前所需的产品 ID 或服务类型
  - 用于附加组件依赖关系（例如，移动附加组件需要活动的移动服务）

供应选项卡：

供应选项卡处理 Ansible 自动化和库存要求。

供应字段：

- 供应 **Play**：
  - Ansible playbook 的名称（不��� .yaml 扩展名）
  - 必须存在于 OmniCRM-API/Provisioners/plays/ 目录中
  - 在创建、更新或去供应服务时调用
- 供应 **JSON** 变量：
  - 作为 JSON 传递给 Ansible playbook 的默认变量
  - 可以在供应期间被覆盖
  - Playbook 接收这些变量以及 customer_id、product_id、service_id、access_token
- 库存项目列表：
  - 多选选择器，显示可用的库存项目类型
  - 示例：SIM 卡、电话号码、调制解调器路由器、IPv4 地址
  - 客户/员工在订购时从可用库存中选择特定项目
  - 选定的库存 ID 作为变量名与库存类型一起传递给 playbook

可用性选项卡：

可用性选项卡控制何时可以购买该产品并显示系统元数据。

可用性设置：

- 可用开始：
  - 产品可供购买的日期/时间
  - 留空表示立即可用
  - 有助于提前宣布新产品
- 可用结束：
  - 产品不再可供购买的日期/时间
  - 留空表示无限期可用
  - 适用于限时促销或终止产品
- 系统元数据（只读）：
  - 创建 - 产品首次创建的时间戳
  - 最后修改 - 最近更新的时间戳
  - 由系统自动维护

模态□作：

- 查看模式：
  - ���闭 - 关闭模态
  - 克隆产品 - 创建一个带有“_clone”后缀的副本
  - 编辑产品 - 切换到编辑模式
- 编辑/创建模式：
  - 取消 - 放弃更改并关闭
  - 保存更改 - 创建或更新产品（大按钮以示强调）

□□□□

□□□

税收应用：

当从该产品创建交易时，税率会自动复制到交易中，并计算税额。例如：

- 产品税率为 10%，零售成本为 $50.00 → 交易税额为 $5.00
- 产品税率为 0%（免税） → 交易税额为 $0.00
- 手动交易覆盖 → 员工可以根据每笔交易更改税率

客户可见性和访问权限

- **enabled** - 此产品□否处于活动状态并可供购买（设置为 false 以隐藏而不删除）
- **residential** - 住宅（消费者）客户□否可以购买此产品
- **business** - 商业（商业）客户□否可以购买此产品
- **customer_can_purchase** - 客户□否可以通过门户自助购买（true）或仅员工可以添加（false）
- **available_from** - 此产品可供购买的日期/时间（可选）
- **available_until** - 此产品不再可供购买的日期/时间（可选，适用于限时优惠）

合同和续订

- **contract_days** - 最小合同长度（以天为单位）（例如，30 表示每月，365 表示每年，0 表示无最低合同）
- **auto_renew** - 默认续订行为：
  - `prompt` - 每次询问客户□否续订
  - `true` - 自动续订，无需询问
  - `false` - 需要手动续订

- **allow_auto_renew** - 客户□否可以启用自动续订（对一次性购买设置为 false）

面向客户的内容

- **terms** - 在购买前显示给客户的条款和条件（包括限制、到期规则、使用条件）
- **features_list** - 显示给客户的功能和包含内容列表（Python 列表格式：`['Feature 1', 'Feature 2']`）

供应配置

- **provisioning_play** - 供应此服务的 Ansible playbook 名称（不带 .yaml 扩展名）。必须存在于 `OmniCRM-API/Provisioners/plays/` 目录中。
- **provisioning_json_vars** - 作为 JSON 传递给 Ansible playbook 的默认变量。这些可以在供应时被覆盖。playbook 接收这些变量以及 `customer_id`、`product_id`、`service_id` 和 `access_token`。
- **inventory_items_list** - 此产品所需的库存项目列表（例如，`['SIM Card', 'Mobile Number']`）。当客户下单时，他们将被提示从��用库存中选择特定项目。选定的库存 ID 与库存类型作为变量名一起传递给供应 playbook。
- **relies_on_list** - 在添加此产品之前必须存在的产品 ID 或服务类型列表。用于附加组件依赖关系（例如，移动附加组件需要活动的移动服务）。

系统元数据

- **created** - 产品创建的时间戳（自动设置）
- **last_modified** - 产品最后更新的时间戳（自动更新）

示例产品定义

独立产品（移动 **SIM**）

```
{
  "product_id": 1,
  "product_slug": "Mobile-SIM",
  "product_name": "Mobile SIM Only",
  "category": "standalone",
  "service_type": "mobile",
  "provisioning_play": "play_psim_only",
  "provisioning_json_vars": "{\"iccid\": \"\", \"msisdn\": \"\"}",
  "inventory_items_list": "['SIM Card', 'Mobile Number']",
  "retail_cost": 0,
  "retail_setup_cost": 0,
  "wholesale_cost": 3,
  "wholesale_setup_cost": 1,
  "contract_days": 0,
  "residential": true,
  "business": true,
  "enabled": true,
  "customer_can_purchase": true,
  "icon": "fa-solid fa-sim-card",
  "features_list": "['Australian Phone Number (04xxx)', 'Fastest speeds', 'Best coverage', 'Roaming on the Mainland']",
  "terms": "Must be activated within 6 months. All credit lost if service is not used for 12 months.",
  "comment": "Physical SIM card for use with Mobile Phones"
}
```


这个独立产品需要两个库存项目（SIM 卡和移动号码），并在供应时创建一个新服务。

附加产品（每月数据计划）


```
{
  "product_slug": "norfone-mobile-prepaid-mini",
  "product_name": "Norfone Mini Plan",
  "category": "addon",
  "service_type": "mobile",
  "provisioning_play": "play_topup_charge_then_action",
  "provisioning_json_vars": "",
  "inventory_items_list": "[]",
  "retail_cost": 30,
  "retail_setup_cost": 0,
  "wholesale_cost": 5.84,
  "contract_days": 30,
  "residential": true,
  "business": false,
  "enabled": true,
  "customer_can_purchase": true,
  "auto_renew": "prompt",
  "icon": "fa-solid fa-sim-card",
  "features_list": "['8GB of Ultra fast data', 'Unlimited Calls & Texts to Norfone users', '100 Minutes of Talk to Australia', '100 SMS to Australia', '30 Day Expiry']",
  "terms": "Credit expires after 30 days. Once data, voice or sms is used up, you will need to top up to continue using the service.",
  "comment": "Our smallest plan for light users"
}
```


这个附加产品不需要库存，并应用于现有服务。它向客户收费并向他们的服务添加信用/余额。

捆绑产品（老年人套餐）


```
{
  "product_slug": "Bundle-Seniors",
  "product_name": "Seniors Bundle",
  "category": "bundle",
  "service_type": "fixed",
  "provisioning_play": "play_seniors_package",
  "provisioning_json_vars": "{\"IPTV_Service_ID\":
\"SeniorBundle\"}",
  "inventory_items_list": "['Modem Router']",
  "retail_cost": 30,
  "retail_setup_cost": 0,
  "wholesale_cost": 10,
  "wholesale_setup_cost": 11,
  "contract_days": 180,
  "residential": true,
  "business": false,
  "enabled": true,
  "icon": "fa-solid fa-person-walking-with-cane",
  "features_list": "['20Mbps Download', '5Mbps Upload', 'Unlimited
Data', 'Home Voice', 'TV: Extra +£5 per month', '£60 Installation
Fee']",
  "terms": "6 Month Contract, must show senior citizen's card to
qualify",
  "comment": "20Mbps/2Mbps GPON Service + IPTV + Phone"
}
```


这个捆绑产品通过一个 playbook 供应多个服务（互联网 + IPTV + 电话）。它需要一个库存项目（调制解调器路由器）。

附加产品（简单充值）

```
{
  "product_slug": "Mobile-Topup-5",
  "product_name": "PAYG £5 Topup",
  "category": "addon",
  "service_type": "mobile",
  "provisioning_play": "play_topup_monetary",
  "provisioning_json_vars": "{\"service_id\": \"\"}",
  "inventory_items_list": "[]",
  "retail_cost": 5,
  "retail_setup_cost": 0,
  "wholesale_cost": 0,
  "contract_days": 0,
  "residential": true,
  "business": false,
  "enabled": true,
  "customer_can_purchase": true,
  "icon": "fa-solid fa-coins",
  "features_list": "['£5 credit', 'Valid for 180 days']",
  "terms": "Valid for 180 days or until all credit is used. See
our website for full rates",
  "comment": "£5 to use for Calls, SMS & Data"
}
```

这个附加产品简单地向现有服务添加货币信用。不需要库存，并使用 `service_id` 来识别要充值的服务。

变量如何传递给 **Ansible**

理解变量如何从产品定义通过 API 流向 Ansible playbook 对于编写有效的供应 playbooks 至关重要。

变量来源和合并

当创建供应作业时，变量来自多个来源，并按以下顺序合并（后来的来源覆盖早期的）：

1. 产品的 **provisioning_json_vars** - 来自产品定义的默认变量
2. 请求体 - 在 API 调用中传递的变量（可以覆盖产品默认值）
3. 系统添加的变量 - 由供应系统自动添加

4. 库存选择 - 选定库存项目的 ID（如果 `inventory_items_list` 不为空）

变量合并过程

系统从所有来源合并变量，后来的来源覆盖早期的。这允许在供应时进行灵活的自定义。

例如，如果您的产品有：

```
"provisioning_json_vars": "{\"monthly_cost\": 50, \"data_gb\": 100}"
```

而您的 API 请求包括：

```
{
  "product_id": 10,
  "customer_id": 456,
  "monthly_cost": 45,
  "custom_param": "value"
}
```

最终传递给 Ansible 的 `extra_vars` 将：

```
{
    "monthly_cost": 45,          # 从请求中覆盖
    "data_gb": 100,              # 来自 provisioning_json_vars
    "product_id": 10,            # 来自请求
    "customer_id": 456,          # 来自请求
    "custom_param": "value",     # 来自请求
    "access_token": "eyJ..."     # 系统添加
}
```

系统添加的变量

供应系统自动添加：

- `access_token` - 用于将 API 调用认证回 CRM 的 JWT 令牌（直接提供用于 IP/API 密钥认证，或通过 `refresh_token` 为用户认证生成）

- `initiating_user` - 触发供应的用户 ID（或自动化系统的第一个管理员）
- 请求体中的任何字段（`product_id`、`customer_id`、`service_id` 等）

库存变量

当产品需要库存项目时（例如，`inventory_items_list: "['SIM Card', 'Mobile Number']"`），流程如下：

1. **UI/API** 提示选择 - 用户从可用库存中选择特定库存项目
2. 库存 **ID** 被添加到变量中 - 选定的库存项目 ID 被添加，库存类型作为变量名
3. **Playbook** 访问库存 **ID** - 供应 playbook 然后可以从 CRM API 获取完整的库存详细信息

例如，如果用户选择：- SIM 卡，库存 ID: 789 - 移动号码，库存 ID: 101

传递给 playbook 的变量包括：- `SIM Card`: 789 - `Mobile Number`: 101

Playbook 然后可以使用这些 ID 从 CRM API 获取完整的库存记录（ICCID、IMSI、MSISDN 等），并使用该信息在网络设备上供应服务。

Ansible 如何接收变量

供应系统将所有合并的变量作为 `extravars` 传递给 Ansible playbook。在 playbook 内部，这些变量通过 Ansible 的标准变量系统可用，并可以在任务中使用。

可以使用 `{{ variable_name }}` 语法直接在 playbook 任务中引用变量。例如，`{{ product_id }}`、`{{ customer_id }}`、`{{ monthly_cost }}` 等。

传递给附加产品的变量

当供应附加产品时，系统自动传递：

- `product_id` - 正在供应的附加产品的 ID
- `customer_id` - 拥有该服务的客户
- `service_id` - 此附加组件要添加到的服务的 ID（对附加组件至关重���）
- `access_token` - 用于 API 调用的认证令牌
- 来自 `provisioning_json_vars` 的任何变量
- 来自 API 请求的任何其他变量

示例附加产品供应流程

当客户将“£5 充值”附加组件添加到他们的移动服务（service_id: 123）时，playbook 接收到的变量包括：

- `product_id`: 45（充值产品）
- `customer_id`: 456（客户）
- `service_id`: 123（要添加信用的服务）
- `access_token`: 认证令牌
- 以及来自产品的 `provisioning_json_vars` 的任何变量

Playbook 然后使用这些变量：

1. 从 **CRM API** 获取服务详细信息，使用 `service_id`
2. 从服务记录中提取服务 **UUID** 和其他信息
3. 向计费系统（**OCS**）添加信用，使用服务 UUID
4. 在 **CRM** 中记录交易，用于计费目的

此流程允许附加组件准确识别要修改的服务并适当地应用更改。

独立产品与附加产品变量的区别

独立产品接收：

- `product_id` - 正在供应的产品
- `customer_id` - 订购服务的客户
- 如果产品需要库存项目，则为库存项目 ID（例如，`SIM Card`、`Mobile Number`）
- `access_token` - 用于 API 认证

附加产品接收：

- `product_id` - 正在供应的附加产品
- `customer_id` - 拥有该服务的客户
- `service_id` - 要修改的现有服务的 **ID**（这□关键区别）
- `access_token` - 用于 API 认证

关键区别在于 `service_id` - 这告诉 playbook 要修改或添加到哪个现有服务。

捆绑产品

捆绑产品的供应方式类似于附加产品，但它们的 playbook 可能会创建多个服务记录。它们接收与附加产品相同的变量，包括：

- `product_id` - 捆绑产品
- `customer_id` - 客户
- `service_id` - 父服务（如果适用）
- 如果需要，则为库存项目 ID（例如，`Modem Router`）
- `access_token` - 用于 API 认证

捆绑 playbook（例如，`play_seniors_package`）然后创建多个相关服务（互联网、IPTV、电话）并将它们链接在一起。

服务

服务属于客户的产品实例，客户为此付费。

它本质上指向 `OCS`（在线计费系统）帐户的链接，该帐户处理费用生成以及帐户的实际余额和使用情况。OCS 由 CGRateS 提供支持，管理货币余额、单位余额（数据、语音、短信）、用于自动续订的 ActionPlans 和用于支出限制的 ThresholdS。

添加服务：产品选择和过滤

在向客户添加服务（无论新的独立服务还现有服务的附加组件）时，系统在旋转界面中显示可用产品。显示的产品根据几个标准进行过滤：

独立服务的产品过滤

在为客户创建新服务时，UI 根据以下条件过滤产品：

1. 客户类型 - 产品被分类为：
   - 个人（住宅）：`residential = true` 或 `business = false` 的产品
   - 商业：`business = true` 的产品
2. 类别 - 产品分为：
   - 服务计划：`category` = `standalone` 或 `bundle` 的产品
   - 附加组件：`category` = `addon` 的产品（在单独的旋转中显示）
3. 可用性 - 产品仅在以下情况下显示：

- `enabled = true` - 产品处于活动状态且未禁用
- 当前日期在 `available_from` 和 `available_until` 之间 - 产品在其可用窗口内
- `customer_can_purchase = true`（如果客户自助购买） - 产品允许直接客户购买

注意

API 级别过滤：API 自动根据启用状态和可用日期在两个级别过滤产品：

- 购买/选择端点（`/crm/product/`）- 由附加组件模态和计划列表用于产品选择。自动过滤以仅显示在其可用日期范围内启用的产品。这确保客户和员工只能选择当前可供购买的产品。
- 管理端点（`/crm/product/paginated`）- 由产品管理页面使用。显示所有产品，包括禁用和超出可用日期的产品，允许管理员管理完整的产品目录，包括非活动产品。

将 `include_disabled=true` 传递给基础产品端点以绕过过滤（仅供管理使用）。

UI 显示单独的旋转：

- 个人服务计划 - 面向消费者客户的住宅产品
- 商业服务计划 - 面向商业客户的商业产品
- 个人附加组件 - 面向住宅的附加组件包
- 商业附加组件 - 面向商业的附加组件包

附加服务的产品过滤

在向 现有服务 添加附加组件时，应用额外的过滤：

1. 服务类型匹配 - 仅显示匹配 `service_type` 的附加组件：
   - 如果现有服务的 `service_type = "mobile"`，则仅显示 `service_type = "mobile"` 的附加组件
   - 这确保移动客户仅看到移动附加组件，互联网客户仅看到互联网附加组件等。
2. 依赖检查 - 如果附加组件有 `relies_on_list`：
   - 系统检查客户□否拥有所需的产品/服务
   - 仅显示依赖关系满足的附加组件
3. 相同客户类型过滤 - 附加组件仍按 `residential` 与 `business` 进行过滤，以匹配客户类型

示例过滤场景

对于拥有现有移动服务（`service_type = "mobile"`）的商业客户：

- 显示的独立产品：所有商业独立/捆绑产品（`business = true``category != "addon"`）
- 显示的附加产品：仅商业移动附加组件（`business = true`，`category = "addon"`，`service_type = "mobile"`）
- 隐藏的产品：住宅产品、其他服务类型的附加组件（互联网、语音等）、禁用产品

服务字段

服务模型包含跟踪已供应服务实例及其与客户、产品和计费系统关系的字段。

基本服务信息

- **service_id** - 系统自动分配的唯一标识符（只读）
- **customer_id** - 链接到拥有此服务的客户（创建后只读）
- **product_id** - 链接到创建此服务的产品（创建后只读）
- **service_name** - 显示给客户的名称（可编辑）
- **service_type** - 服务类型：移动、互联网、VoIP、IPTV、捆绑等（可编辑）
- **service_uuid** - 用于 OCS/CGRateS 计费的唯一标识符（只读，自动生成）
- **icon** - 用于在自助服务门户中显示的 FontAwesome 图标类（可编辑）

服务状态和日期

- **service_status** - 当前状态：活动、非活动、暂停等（可编辑）
- **service_provisioned_date** - 服务首次供应的时间（自动设置，只读）
- **service_active_date** - 服务变为活动的时间（可编辑）
- **service_deactivate_date** - 服务到期或将被停用的时间（可编辑）
- **contract_end_date** - 合同承诺的结束日期（可编辑）

计费和定价

- **retail_cost** - 向客户收取的每月经常性费用（可编辑）
- **wholesale_cost** - 提供服务的成本（可编辑）
- **service_billed** - 此服务□否出现在发票上（可编辑，默认：true）
- **service_taxable** - 此服务□否适用税费（可编辑，默认：true）
- **invoiced** - 此服务□否已开具发票（由计费系统自动设置）
- **promo_code** - 创建服务时使用的促销代码（可编辑）

客户可见性

- **service_visible_to_customer** - 客户□否可以在自助服务门户中看到此服务（可编辑，默认：true）
- **service_usage_visible_to_customer** - 客户□否可以查看使用情况/余额详细信息（可编辑，默认：true）

供应配置

- **provisioning_play** - 用于供应此服务的 Ansible playbook（从产品继承，只读）
- **provisioning_json_vars** - 传递给供应 playbook 的变量（从产品继承，只读）
- **deprovisioning_play** - 在去供应服务时运行的 Ansible playbook（只读）
- **deprovisioning_json_vars** - 去供应 playbook 的变量（只读）

服务关系

- **bundled_parent** - 如果此服务□捆绑的一部分，则为父服务的 service_id（只读）
- **site_id** - 链接到服���交付的物理站点/位置（可编辑）

备注和元数据

- **service_notes** - 关于服务的内部备注，仅供员工参考（可编辑）
- **created** - 服务创建的时间戳（自动设置，只读）
- **last_modified** - 最后更新的时间戳（自动更新，只读）

可编辑与只读字段

通过 **API/UI** 可编辑：

服务可以通过 `PATCH /crm/service/{service_id}` 更新这些字段：

- service_name、service_type、service_status
- service_notes
- retail_cost、wholesale_cost
- service_billed、service_taxable
- service_visible_to_customer、service_usage_visible_to_customer
- service_active_date、service_deactivate_date、contract_end_date
- icon、promo_code、site_id

只读（自动设置）：

这些字段在创建后无法直接修改：

- service_id、customer_id、product_id
- service_uuid（在供应期间生成）
- service_provisioned_date
- provisioning_play、provisioning_json_vars
- deprovisioning_play、deprovisioning_json_vars
- bundled_parent
- invoiced（由计费系统管理）
- created、last_modified（自动管理）

将产品供应到服务

供应过程通过一系列协调步骤将产品（模板）转换为服务（特定于客户的实例），涉及 Web UI、API 和 Ansible playbooks。

高级供应流程

1. 预供应设置 - 在 API 中创建产品，配置供应，并编写和测试相应的 Ansible playbooks
2. 服务选择 - 从客户页面，员工或客户选择“添加服务”
3. 产品过滤 - 显示的产品根据以下条件过滤：
   - 客户类型（住宅/商业）
   - 现有服务（用于附加组件依赖关系中的 `relies_on_list`）
   - 可用日期（`available_from`/`available_until`）
   - `enabled` 和 `customer_can_purchase` 标志
4. 自定义 - 选项覆盖供应变量（用于价格调整、自定义配置等）
5. 库存选择 - 如果产品需要库存（`inventory_items_list` 不为空），用户选择特定项目（例如，选择哪个 SIM 卡，哪个电话号码）
6. 供应启动 - 当单击“供应”按钮时，API 创建一个供应作业

详细的 **API** 和 **Ansible** 集成流程

当供应服务时，发生以下序列：

步骤 **1**：创建供应作业

API 接收供应请求并创建一个供应作业，包含：

- `provisioning_play` - Ansible playbook 的名称（例如，`play_psim_only`）
- `provisioning_json_vars` - 来自产品或请求覆盖的变量的 JSON 字符串
- `customer_id` - 订购服务的客户 ID
- `product_id` - 正在供应的产品 ID
- `service_id` - （可选）用于附加组件的现有服务 ID
- 库存选择 - 选定库存项目的 ID

步骤 **2**：变量组装

供应服务合并来自多个来源的变量，按以下顺序：

1. 产品的 `provisioning_json_vars`（来自产品定义的默认值）
2. 请求体参数（可以覆盖产品默认值）
3. 系统添加的变量：
   - `access_token` - 用于 API 认证回 CRM 的 JWT 令牌
   - `initiating_user` - 触发供应的用户 ID
   - `customer_id`、`product_id`、`service_id`
4. 库存选择 - 作为 `{inventory_type: inventory_id}` 对添加

示例合并变量：

```
{
    "customer_id": 123,
    "product_id": 456,
    "service_id": 789,              # 仅适用于附加组件
    "SIM Card": 1001,               # 来自库存选择
    "Mobile Number": 1002,          # 来自库存选择
    "monthly_cost": 30,             # 来自 provisioning_json_vars
    "data_gb": 50,                  # 来自 provisioning_json_vars
    "access_token": "eyJ...",       # 系统添加的用于 API 回调
    "initiating_user": 5            # 系统添加的
}
```

步骤 **3**：创建供应记录

在数据库中创建一个 `Provision` 记录，包含：

- `provision_id` - 用于跟踪的唯一标识符
- `provisioning_play` - playbook 文件名
- `provisioning_json_vars` - 合并的变量作为 JSON 字符串
- `task_count` - playbook 中的任务数量（从 YAML 中提取）
- `provisioning_status` - 状态代码（最初设置为 1 = 运行，然后更新为 0 = 成功，2 = 失败，或如果仍在进行中可能保持为 1）
- `product_id`、`customer_id`、`service_id` - 上下文引用

步骤 **4**：后台 **playbook** 执行

API 生成一个后台线程：

1. 从 `OmniCRM-API/Provisioners/plays/{playbook_name}.yaml` 加载 playbook YAML
2. 调用 `ansible_runner.run()`，参数包括：
   - `playbook` - 加载的 YAML 文件的路径
   - `extravars` - 所有合并的变量（传递给 Ansible）
   - `inventory` - 设置为 `'localhost,'`（本地执行）
   - `event_handler` - 自定义处理程序，用于捕获任务执行事件
3. 实时监控 playbook 执行

步骤 **5**：事件捕获和日志记录（`ProvisioningEventHandler`）

每当 Ansible 任务执行时，事件会被捕获并存储为 `Provision_Event` 记录：

- `event_name` - 来自 playbook 的任务名称
- `event_number` - 序列号
- `provisioning_status` - 指示任务结果的状态代码：
  - **0** = 成功 - 任务成功完成
  - **1** = 运行中 - 任务正在执行
  - **2** = 失败 - 导致供应停止的严重故障
  - **3** = 失败（忽略） - 任务失败但错误被忽略（`ignore_errors: true` 在 playbook 中）
- `provisioning_result_json` - 任务结果，敏感数据已被编辑

事件处理程序会自动从日志中剥离密码、密钥、机密和其他敏感数据。

步骤 **6**: **Ansible playbook** 执行（`Provisioners/plays/*.yaml`）

Ansible playbook 在本地运行，通常执行以下操作:

1. 获取产品定义 - 使用 `{{ access_token }}` 对 `/crm/product/product_id/{{ product_id }}` 进行 GET 请求
2. 获取客户信息 - 对 `/crm/customer/customer_id/{{ customer_id }}` 进行 GET 请求
3. 处理库存项目（如果需要） - 对每个选定项目进行 GET 请求 `/crm/inventory/inventory_id/{{ inventory_id }}` 以检索完整详细信息（ICCID、MSISDN、序列号等）
4. 配置外部系统 - 进行 API 调用以:
    - HSS（家庭用户服务器）进行用户供应
    - IMS（IP 多媒体子系统）进行语音注册
    - CGRateS/OCS 创建帐户、计费配置、费率计划
    - ENUM 服务器进行电话号码映射
    - 网络设备（路由器、交换机等）
5. 添加设置成本（如果适用） - 对 `/crm/transaction/` 进行 POST 请求以记录一次性费用
6. 向客户收费 - 对 OCS/CGRateS 进行 POST 请求以收取 retail_setup_cost（如果配置）
7. 创建 **OCS** 帐户 - 对 OCS/CGRateS 进行 POST 请求以使用 UUID 创建计费帐户
8. 配置经常性收费 - 在 OCS/CGRateS 中为每月经常性收费创建 Actions 和 ActionPlans
9. 创建服务记录 - 对 `/crm/service/` 进行 PUT/POST 请求以在 CRM 中创建服务记录:

```
{
  "customer_id": 123,
  "product_id": 456,
  "service_name": "Mobile SIM - 0412345678",
  "service_uuid": "generated-uuid-for-ocs",
  "service_status": "Active",
  "service_type": "mobile",
  "retail_cost": 30,
  "wholesale_cost": 5,
  "provisioning_play": "play_psim_only",
  "provisioning_json_vars": "{...}"
}
```

10. 分配库存 - 对 `/crm/inventory/inventory_id/{{ inventory_id }}` 进行 PATCH 请求，以将库存标记为“分配”给服务

11. 发送通知（可选） - 向客户发送电子邮件或短信，提供服务详细信息

步骤 **7**：完成和状态更新

当 playbook 完成时：

- 成功：`Provision.provisioning_status` 更新为 **0**（成功）
- 严重故障：`Provision.provisioning_status` 更新为 **2**（失败），并向 `crm_config.provisioning.failure_list` 发送故障电子邮件
- 非关键故障：任务失败且 `ignore_errors: true` 的任务标记为状态 **3**（失败但被忽略），并不会停止供应

已供应的服务现在在 CRM 中可见，并对客户处于活动状态（如果供应成功）。

关键区别：独立产品与附加产品与捆绑产品的供应

独立产品（`category: standalone`）：

- 接收 `customer_id` 和 `product_id`
- 通常需要库存项目（SIM 卡、电话号码、调制解调器）
- 通过 API PUT `/crm/service/` 创建新服务记录
- 在外部系统上供应新资源（HSS、OCS、网络设备）
- 示例：新移动 SIM 激活、新互联网连接

附加产品（`category: addon`）：

- 接收 `customer_id`、`product_id` 和 **`service_id`**（要修改的现有服务）
- 通常不需要库存（或仅需最少库存）
- 修改现有 服务或向现有 OCS 帐户添加费用
- 可能在 OCS 上执行操作（添加数据包、添加信用、启用功能）
- 不创建新服务记录（或创建链接到父服务的子服务记录）
- 示例：每月数据计划充值、国际漫游包、额外信用

捆绑产品（`category: bundle`）：

- 在接收的变量方面与附加产品类似
- 可能需要一些库存项目（例如，家庭捆绑所需的调制解调器）
- 创建 多个 相关服务记录（互联网 + 电视 + 电话）
- 在不同系**??**之间供应多个资源
- 在 CRM 中将服务链接在一起以实现统一计费/管理
- 示例：家庭捆绑（互联网 + IPTV + VoIP 电话）

供应 **playbook** 要求

为了使 playbook 正常工作，它必须：

1. 位于 `OmniCRM-API/Provisioners/plays/{playbook_name}.yaml`
2. 通过 **Ansible** 的 **`extravars`** 接受变量（通过 `{{ variable_name }}` 访问）
3. 使用 **`Authorization: Bearer {{ access_token }}`** 头进行 **API** 调用认证
4. 优雅地处理故障，在适当的地方使用 `rescue` 块和 `ignore_errors`
5. 为独立产品创建服务记录，或修改附加产品的现有服务
6. 如果选择了库存，则分配库存
7. 在发生关键错误时通过 **`fail`** 模块返回有意义的错误消息

Playbooks 中可用的常见变量

每个 playbook 接收这些变量：

- `customer_id` - 整数，订购服务的客户
- `product_id` - 整数，正在供应的产品

- `service_id` - 整数（仅限附加组件/捆绑），要修改的现有服务
- `access_token` - 字符串，CRM API 认证的 JWT 令牌
- `initiating_user` - 整数，触发供应的用户
- 以及任何库存项目 ID：`{{ inventory_type }}: inventory_id`
- 以及来自 `provisioning_json_vars` 的任何变量
- 以及请求中传递的任何变量

Playbooks 可以使用这���变量：

- 获取完整的产品详细信息：`GET /crm/product/product_id/{{ product_id }}`
- 获取客户详细信息：`GET /crm/customer/customer_id/{{ customer_id }}`
- 获取库存详细信息：`GET /crm/inventory/inventory_id/{{ SIM_Card }}`
- 创建交易：`POST /crm/transaction/`
- 创建服务：`PUT /crm/service/`
- 更新服务：`PATCH /crm/service/{{ service_id }}`
- 分配库存：`PATCH /crm/inventory/inventory_id/{{ inventory_id }}`

示例：简单附加产品 **playbook** 流程

对于移动数据充值附加产品：

1. Playbook 接收：`customer_id`、`product_id`、`service_id`、`access_token`
2. 获取服务详细信息：`GET /crm/service/{{ service_id }}` 以获取 `service_uuid`
3. 获取产品详细信息：`GET /crm/product/product_id/{{ product_id }}` 以获取定价和数据量
4. 在 OCS 中向客户收费：`POST to CGRateS` 从余额中扣除 retail_cost
5. 在 OCS 中添加数据信用：`POST to CGRateS` 添加数据余额并设置到期
6. 在 CRM 中记录交易：`POST /crm/transaction/` 记录收费详细信息
7. 成功完成

整个过程在 `Provision` 和 `Provision_Event` 表中进行跟踪，以便于调试和审计。

OCS 参与

`OCS`（在线计费系统），通过 CGRateS 实现，处理所有实时计费和服务使用跟踪。CRM 服务记录充当指向 OCS 帐户的指针，该帐户管理：

- 经常性收费 - 每月费用、DID 租金、订阅费用
- 基于使用的计费 - 每分钟语音通话、每 MB 数据、每条短信收费
- 余额管理 - 货币余额（预付信用）和单位余额（数据 GB、语音分钟、短信计数）
- 余额转换 - 将货币余额转换为单位余额（例如，花费 $30 获取 10GB 数据包）
- 帐户状态 - 根据信用限额和阈值的状态（活动、暂停、禁用）

CRM 服务记录包含元数据和配置（客户、产品、定价、可见性），而 OCS 包含实时计费状态（余额、使用情况、收费）。

检索服务使用情况和余额

服务使用信息从 OCS/CGRateS 检索，并实时显示给客户和员工。

如何检索使用情况

当请求服务的使用情况时（通过 UI 或 API），发生以下流程：

1. **API** 请求 - 前端调用 `GET /crm/service/{service_id}` 或在 UI 中查看服务详细信息
2. 服务查找 - API 从数据库中检索服务记录，提取 `service_uuid`
3. **CGRateS API** 调用 - 系统向 CGRateS 发出两个调用：
    i. **Get_Balance(service_uuid)** - 检索帐户余额，使用 `BalanceMap`
        - 返回按类型组织的余额：数据、语音、短信、货币、��据加密狗
        - 每个余额包括：ID、值、到期日期、权重、目标 ID
        - 系统添加人类可读字段：`custom_Name_hr`、`custom_Expiration`、`custom_Description_String`
    ii. **Get_ActionPlans(service_uuid)** - 检索活动的自动续订行动计划（在下一节中介绍）
4. 响应合并 - CGRateS 数据合并到服务响应中：

```
{
  "service_id": 123,
  "service_name": "Mobile Service",
  "service_uuid": "abc-123-def",
  "cgrates": {
    "BalanceMap": {
      "DATA": [{
        "ID": "DATA_10GB",
        "Value": 5368709120,
        "ExpirationDate": "2025-02-01T00:00:00Z",
        "custom_Name_hr": "10GB Data Pack",
        "custom_Expiration": "Feb 1, 2025",
        "custom_Description_String": "5 GB remaining"
      }],
      "VOICE": [{
        "ID": "VOICE_UNLIMITED",
        "Value": 999999999,
        "custom_Name_hr": "Unlimited Calls",
        "custom_Description_String": "Unlimited minutes"
      }],
      "MONETARY": [{
        "ID": "PREPAID_CREDIT",
        "Value": 25.50,
        "custom_Description_String": "$25.50 credit"
      }]
    },
    "ActionPlans": [...]
  }
}
```

5. **UI** 显示 - 前端组件显示使用数据:

   - **ServiceUsage.js** - 主要使用显示组件，每 3 秒自动刷新
   - **UsageCard.js** - 每种余额类型的摘要卡
   - **UsageProgress.js** - 显示使用/剩余百分比的进度条
   - 余额以颜色编码并格式化以提高可读性

使用数据结构

`BalanceMap` 中的每个余额包含:

CGRateS 原生字段：

- `ID` - 余额的唯一标识符（例如，“DATA_10GB_2025_01”）
- `Value` - 余额金额：
  - 对于数据：字节（5368709120 = 5 GB）
  - 对于语音：秒（3600 = 1 小时）
  - 对于短信：计数（100 = 100 条消息）
  - 对于货币：货币单位（25.50 = $25.50）
- `ExpirationDate` - 余额到期的 ISO 8601 时间戳
- `Weight` - 余额消耗的优先级（优先消耗较高的余额）
- `DestinationIDs` - 此余额适用的目标（例如，["AU", "INTERNATIONAL"]）

人类可读字段（由 **CRM** 添加）：

- `custom_Name_hr` - 从 ID 提取的人类可读名称
- `custom_Expiration` - 格式化的到期日期（例如，“2025 年 1 月 15 日”或“11 天后”）
- `custom_Description_String` - 人类可读的余额描述：
  - 数据：“5 GB 剩余”或“10 GB 总计”
  - 语音：“60 分钟剩余”或“无限制”
  - 短信：“50 条短信剩余”
  - 货币：“$25.50 信用”

使用可见性控制

服务使用可见性由两个字段控制：

- **service_visible_to_customer** - 如果为 false，则服务在客户的自助服务门户中完全隐藏
- **service_usage_visible_to_customer** - 如果为 false，则服务可见但使用/余额详细信息隐藏（客户可以看到他们有服务，但不知道使用了多少）

这允许运营商：

- 隐藏内部/测试服务
- 显示服务存在而不透露使用情况（对无限计划或敏感服务有用）
- 完全透明的使用显示（默认）

实时使用更新

Web UI 自动刷新使用数据：

- 间隔：每 3 秒（在 ServiceUsage 组件中可配置）
- 方法：轮询 `GET /crm/service/{service_id}`，从 CGRateS 获取实时数据
- 效率：仅活动服务视图刷新；列表视图使用缓存数据

这确保客户和员工在使用发生时看到近实时的余额更新。

经常性收费 / 自动续订

经常性收费，例如每月服务费或每个 DID 收费，首先在 `OCS` 中作为 Actions 创建，格式为 `Action_ServiceUUID_ServiceName_WhatitDoes`。

对于每月 $60 的 GPON 服务，包括 1TB 的使用，Action 可能看起来像这样：

`Action_kj49-adsf-1234-9742_60_GPON_1TB_MonthlyExpiry`

1. 将货币余额重置为 $0
2. 向 CRM 发送 HTTP POST 请求以供应某些内容
3. 添加 1TB 使用的信用，设置到期为 1 个月

如果我们想让 MRC 变为经常性（我们确实想），那么我们将创建一个名为 `ActionPlan_{{ service_uuid }}_Monthly_Charge` 的 `ActionPlan`，其时间设置为 □□ 触发一次，并将 `ActionPlan` 分配给帐户。

我们可以根据 □/□/□ 参数设置到期日期，以确定 MRC 停止的时间，例如对于固定的 12 个月服务，在此之后停止。

由于 Actions 和 `ActionPlans` 都□服务特有的，它们不会与任何其他服务共享任何内容。

这意味着我们可以使用调整后的值进行供应，而不会影响其他服务。

附加组件和附加服务

附加组件/附加服务，例如购买额外数据、漫游包、国际通话分钟等，处理方式基本相同。创建一个 Action 来执行所需操作，例如收取货币值，然后授予具有设置到期的单位余额。

与使用 `ActionPlans` 自动将其添加到帐户不同，我们只需从 Ansible 中触发刚创建的 Action 的 `ExecuteAction` 一次。

添加预付货币余额

对于预付货币余额，例如 $10 PAYG 计划，这作为货币余额添加，但优先级更高。

这些服务的默认余额的信用限额将为 $0。

信用限额 / 防止过度支出

`ThresholdS` 用于在每个帐户上设置给定时间段的最大支出。

对于 PAYG / 预付客户，这为 $0。

通过 **CRM** 与 **OCS** 交互

对于每项服务，您可以在 CRM API 中查看来自 `OCS` 的 `Balances` 和 `ActionPlans`、`Actions` 和 `ThresholdS`。

可以根据需要从 CRM API 中删除 `ActionPlans`，通过 Ansible Playbooks 进行操作。可以根据需要从 CRM 中添加 `ActionPlans`，通过添加附加组件/服务并通过 Ansible Playbooks 进行操作。

可以禁用 `OCS` 帐户，这将停止 `ActionPlans` 的执行以及服务的消费。

对于信用限额，按产品政策设置 `ThresholdS` 值。

在 **CRM** 中查看和管理 **ActionPlans**

ActionPlans（自动续订配置）通过 CRM 界面显示和管理，允许员工和客户查看即将到来的自动续订并进行管理。

如何检索和显示 **ActionPlans**

在 CRM 中查看服务时，ActionPlans 会自动获取并显示：

1. **API** 调用 - 当调用 `GET /crm/service/{service_id}` 时，API：

   - 从数据库中检索服务记录
   - 提取 `service_uuid`（OCS 帐户标识符）
   - 调用 CGRateS API 通过服务 UUID 获取 ActionPlans
   - 这内部调用 `Get_ActionPlans(service_uuid)` 在 CGRateS 上

2. **ActionPlan** 数据结构 - 每个返回的 ActionPlan 包含：

```
{
    "ActionPlanId": "ServiceID_abc-123-
def__ProductID_456__...",
    "PlanName": "Monthly_Renewal_Plan",
    "NextExecTime": "2025-02-01T00:00:00Z",
    "custom_NextExecTime_hr": "in 11 days",
    "ActionPlanId_split_dict": {
        "ServiceID": "abc-123-def",
        "ProductID": 456,
        "CustomerID": 789,
        ...
    }
}
```

   - `ActionPlanId` - 包含编码服务/产品/客户信息的唯一标识符
   - `PlanName` - 行动计划的名称（通常□续订 playbook 名称）
   - `NextExecTime` - 下次执行 ActionPlan 的 ISO 时间戳
   - `custom_NextExecTime_hr` - 人类可读的执行时间（例如，“11 天后”、“明天”、“2025 年 2 月 1 日”）
   - `ActionPlanId_split_dict` - 从 ActionPlanId 中解析的组件的字典

3. **UI** 显示 - ActionPlans 在 **ActionPlansTable** 组件中显示：

   表格列：

   - 产品名称 - 通过从 ActionPlanId 中查找 `ProductID` 获取
   - 费用 - 显示来自产品定义的 `retail_cost`
   - 续订日期 - 显示 `custom_NextExecTime_hr`（人类可读）
   - □作 - 两个按钮：

- 立即续订 - 立即供应附加组件/续订（绕过等待计���执行）
- 移除自动续订 - 取消自动续订

当不存在 **ActionPlans** 时：

- 表格显示消息："此服务未启用自动续订"
- 客户可以添加自动续订的附加组件以启用自动续订

管理 ActionPlans

员工和客户可以通过 UI 管理 ActionPlans：

移除 **ActionPlan**（取消自动续订）：

1. 单击 ActionPlansTable 中的"移除自动续订"按钮
2. 确认模态出现："您确定要移除此自动续订吗？"
3. 确认后，前端调用：`DELETE /crm/oam/remove_action_plan/{action_plan_id}`
4. API 通过 `ocs.Remove_ActionPlan()` 从 CGRateS 中移除 ActionPlan
5. 记录活动："从服务 {service_id} 中移除 ActionPlan {ActionPlanId}"
6. ActionPlan 从表中消失

立即续订（手动续订）：

1. 单击 ActionPlansTable 中的"立即续订"按钮
2. 确认模态出现："您确定要立即续订吗？"
3. 确认后，系统：
   - 从 ActionPlanId 中提取 product_id
   - 为该产品创建新的供应作业
   - 立即供应附加组件（运行供应 playbook）
   - 服务在不等待计划续订的情况下接收附加组件的好处
4. 供应状态模态显示进度
5. 成功后，余额立即更新

添加自动续订：

通过供应具有 `auto_renew` 设置的附加产品来启用自动续订：

- `auto_renew = "true"` 的产品 - 在供应期间自动创建 ActionPlans

- `auto_renew = "prompt"` 的产品 - 询问客户□否希望自动续订（模态对话框）
- `auto_renew = "false"` 的产品 - 从不创建 ActionPlans（一次性购买）

供应 playbook 在 CGRateS 中创建 ActionPlan，具有：

- 编码服务、产品和客户 ID 的唯一 ActionPlanId
- 续订计划（每月、每年、自定义间隔）
- 要执行的操作（通常□重新供应相同的附加组件）
- 到期日期（如果合同有固定期限）

ActionPlan 命名约定

ActionPlans 遵循标准化的命名约定，以编码元数据：

格式：

```
ServiceID_{service_uuid}__ProductID_{product_id}__CustomerID_{custome
```

示例：

```
ServiceID_abc-123-
def__ProductID_456__CustomerID_789__MonthlyRenewal
```

此编码允许 CRM：

- 确定 ActionPlan 属于哪个服务
- 查找产品详细信息（名称、定价）以供显示
- 跟踪客户所有权
- 解析续订类型和计划

CRM 会自动将这些组件解析为 `ActionPlanId_split_dict`，以便于访问。

服务视图中的 ActionPlans

在 CRM 中查看服务时，ActionPlans 表在服务详细信息中显示：

员工视图（`ServiceView.js`）：

- 显示完整的 ActionPlans 表，包含所有管理选项
- 显示产品名称、费用、续订日期
- 允许移除和立即续订

客户自助服务视图：

- 以简化视图显示即将到来的续订
- 显示下一个续订日期和金额
- 可能允许客户禁用自动续订（可根据产品配置）

空状态：

- 如果没有 ActionPlans：“此服务未启用自动续订”
- 建议添加自动续订的附加组件以启用自动续订

ActionPlans 和即将到期的服务

CRM 包含一个即将到期服务的端点，识别需要续订的服务：

`GET /crm/service/expiring?threshold=7`

这返回在阈值天数内到期的服务，并包括：

- 余额即将到期的服务
- 没有活动 ActionPlans 的服务（需要手动续订）
- 即将执行的 ActionPlans 的服务

这允许运营商：

- 向客户发送续订提醒
- 识别面临风险的客户（即将到期且没有自动续订）
- 监控即将到来的自动续订
- 主动管理服务连续性

供应系统

OmniCRM 使用 **Ansible** 来自动化客户服务的供应、配置和去供应。供应系统旨在灵活，允许复杂的工作流程，同时保持一致性和可靠性。

另请参见：`SIM 卡供应 <concepts_sim_provisioning>`，获取有关移动 SIM 供应的详细信息，包括物理 SIM 和 eSIM。

注意

有关产品到服务旅程的完整演练，包括详细的 Ansible 剧本示例、定价策略和真实场景，请参见 完整产品生命周期指南 <guide_product_lifecycle>。

概述

当产品被订购或需要配置服务时，OmniCRM 创建一个 供应作业，该作业执行一个或多个 Ansible 剧本。这些剧本与各种后端系统（OCS/CGRateS、网络设备、API 等）交互，以完全供应服务。

供应系统支持两种主要工作流程：

1. 标准供应 - 通过 UI/API 由员工或客户触发
2. 简单供应 - 由 OCS 等外部系统触发以进行自动化操作

供应状态值

供应作业和单个任务可以具有以下状态：

- 状态 **0**（成功） - 供应作业成功完成
- 状态 **1**（运行中） - 供应作业或任务当前正在执行
- 状态 **2**（失败 - 严重） - 发生了严重故障，导致供应失败
- 状态 **3**（失败 - 被忽略） - 任务失败但设置了 `ignore_errors: true`，因此供应继续

当供应作业失败时，OmniCRM 会向配置的失败通知列表发送电子邮件通知，包含详细的错误信息。

产品如何驱动供应

产品 定义□供应内容及其方式的蓝图。当用户选择要供应的产品时，系统从产品定义中读取几个关键字段以确定该做什么。

供应中使用的产品字段

产品定义包含：

- `provisioning_play` - 要执行的 Ansible 剧本的名称（不带 .yaml 扩展名）

- `provisioning_json_vars` - 包含要传递给 Ansible 的默认变量的 JSON 字符串
- `inventory_items_list` - 必须分配的库存类型列表（例如，`['SIM Card', 'Mobile Number']`）
- `product_id`、`product_name`、定价字段 - 自动传递给剧本

示例产品定义

```
{
  "product_id": 1,
  "product_slug": "Mobile-SIM",
  "product_name": "Mobile SIM Only",
  "provisioning_play": "play_psim_only",
  "provisioning_json_vars": "{\"iccid\": \"\", \"msisdn\": \"\"}",
  "inventory_items_list": "['SIM Card', 'Mobile Number']",
  "retail_cost": 0,
  "retail_setup_cost": 0,
  "wholesale_cost": 3,
  "wholesale_setup_cost": 1
}
```

从产品到供应作业

当供应被启动时，系统：

1. 加载 **`provisioning_play`** 中指定的剧本

   系统查找 `OmniCRM-API/Provisioners/plays/play_psim_only.yaml`

2. 将多个来源的变量合并到 **`extra_vars`** 中：

   i. 来自 **`provisioning_json_vars`**：`{"iccid": "", "msisdn": ""}`
   ii. 来自请求体：用户/API 提供的任何其他变量
   iii. 来自产品字段：`product_id`、`customer_id` 等
   iv. 来自身份验证：`access_token` 或 `refresh_token` 的设置

3. 根据 **`inventory_items_list`** 分配库存

   在运行剧本之前，UI/API 提示选择库存：

- **SIM** 卡 - 用户从库存中选择一个可用的 SIM
- 手机号码 - 用户选择一个可用的电话号码

选择的库存 ID 被添加到 `extra_vars` 中，库存类型作为键：

```
extra_vars = {
    "product_id": 1,
    "customer_id": 456,
    "SIM Card": 789,          # 选择的 SIM 的 inventory_id
    "Mobile Number": 101,     # 选择的电话号码的 inventory_id
    "iccid": "",              # 来自 provisioning_json_vars
    "msisdn": "",             # 来自 provisioning_json_vars
    "access_token": "eyJ..."
}
```

4. 通过 `hostvars[inventory_hostname]` 将所有内容传递给 **Ansible**

在剧本内部，变量可以访问如下：

```
- name: Get inventory_id for SIM Card
  set_fact:
    inventory_id_sim_card: "{{ hostvars[inventory_hostname]
['SIM Card'] | int }}"
  when: "'SIM Card' in hostvars[inventory_hostname]"
```

剧本如何使用库存变量

一旦剧本获得库存 ID，它将从 API 检索完整的库存详细信息：

```
- name: Get SIM Card Details from Inventory
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ inventory_id_sim_card }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: sim_card_response

- name: Extract ICCID and IMSI from inventory
  set_fact:
    iccid: "{{ sim_card_response.json.iccid }}"
    imsi: "{{ sim_card_response.json.imsi }}"

- name: Get Phone Number Details from Inventory
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ inventory_id_phone_number }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: phone_number_response

- name: Extract MSISDN
  set_fact:
    msisdn: "{{ phone_number_response.json.msisdn }}"
```

然后，剧本可以使用这些值来：

- 在 HSS 上供应 SIM 卡与 IMSI
- 在计费系统中配置电话号码
- 将库存项目分配给客户
- 使用这些详细信息创建服务记录

真实世界示例：移动 SIM 供应

从 `play_psim_only.yaml`，以下它如何使用产品和库存数据：

```
- name: Get Product information from CRM API
  uri:
    url: "{{ crm_config.crm.base_url }}/crm/product/product_id/{{
product_id }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: api_response_product

- name: Set package facts from product
  set_fact:
    package_name: "{{ api_response_product.json.product_name }}"
    package_comment: "{{ api_response_product.json.comment }}"
    setup_cost: "{{ api_response_product.json.retail_setup_cost
}}"
    monthly_cost: "{{ api_response_product.json.retail_cost }}"

- name: Set inventory_id_sim_card if SIM Card was selected
  set_fact:
    inventory_id_sim_card: "{{ hostvars[inventory_hostname]['SIM
Card'] | int }}"
  when: "'SIM Card' in hostvars[inventory_hostname]"

- name: Set inventory_id_phone_number if Mobile Number was
selected
  set_fact:
    inventory_id_phone_number: "{{ hostvars[inventory_hostname]
['Mobile Number'] | int }}"
  when: "'Mobile Number' in hostvars[inventory_hostname]"

- name: Get SIM Card details from inventory
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ inventory_id_sim_card }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: sim_inventory_response

- name: Get Phone Number details from inventory
  uri:
```

```
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ inventory_id_phone_number }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: phone_inventory_response

- name: Extract values from inventory
  set_fact:
    iccid: "{{ sim_inventory_response.json.iccid }}"
    imsi: "{{ sim_inventory_response.json.imsi }}"
    msisdn: "{{ phone_inventory_response.json.msisdn }}"
    ki: "{{ sim_inventory_response.json.ki }}"
    opc: "{{ sim_inventory_response.json.opc }}"

- name: Provision subscriber on HSS
  uri:
    url: "http://{{ hss_server }}/subscriber/{{ imsi }}"
    method: PUT
    body_format: json
    body:
      {
        "imsi": "{{ imsi }}",
        "msisdn": "{{ msisdn }}",
        "ki": "{{ ki }}",
        "opc": "{{ opc }}",
        "enabled": true
      }
    status_code: 200

- name: Assign inventory to customer
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ inventory_id_sim_card }}"
    method: PATCH
    headers:
      Authorization: "Bearer {{ access_token }}"
    body_format: json
    body:
      {
        "customer_id": {{ customer_id }},
        "item_state": "Assigned"
```

```
      }
    status_code: 200
```

这演示了完整的流程：

1. 产品定义指定 `provisioning_play: "play_psim_only"`
2. 产品要求 `inventory_items_list: ['SIM Card', 'Mobile Number']`
3. 用户在供应过程中选择库存项目
4. 库存 ID 作为 `extra_vars` 传递给剧本
5. 剧本从 API 检索完整的库存详细信息
6. 剧本使用库存数据配置网络设备
7. 剧本将库存标记为分配给客户

回滚和清理：最佳实践模式

关键最佳实践：同一个剧本应使用 Ansible 的 `block` 和 `rescue` 结构处理失败的供应回滚和有意的去供应。

剧本结构

来自 `play_psim_only.yaml`：

```yaml
- name: OmniCore Service Provisioning 2024
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: Main block
      block:
        # --- 供应任务 ---
        - name: Get Product information
          uri: ...

        - name: Create account in OCS
          uri: ...

        - name: Provision subscriber on HSS
          uri: ...

        - name: Create service record
          uri: ...

        # ... 还有很多供应任务 ...

      rescue:
        # --- 清理任务 ---
        # 此部分在以下情况下运行:
        # 1. 块中的任何任务失败（回滚）
        # 2. action == "deprovision"（有意清理）

        - name: Get Inventory items linked to this service
          uri:
            url: "{{ crm_config.crm.base_url
}}/crm/inventory/customer_id/{{ customer_id }}"
            method: GET
          register: inventory_api_response
          ignore_errors: True

        - name: Return inventory to pool
          uri:
            url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ item.inventory_id }}"
            method: PATCH
            body_format: json
```

```
      body:
        service_id: null
        customer_id: null
        item_state: "Used"
    with_items: "{{ inventory_api_response.json.data }}"
    ignore_errors: True

  - name: Delete Account from Charging
    uri:
      url: "http://{{ crm_config.ocs.OCS }}/jsonrpc"
      method: POST
      body:
        {
          "method": "ApierV1.RemoveAccount",
          "params": [{
              "Tenant": "{{ crm_config.ocs.ocsTenant }}",
              "Account": "{{ service_uuid }}"
          }]
        }
    ignore_errors: True

  - name: Delete Attribute Profile
    uri:
      url: "http://{{ crm_config.ocs.OCS }}/jsonrpc"
      method: POST
      body:
        {
          "method": "APIerSv1.RemoveAttributeProfile",
          "params": [{
              "ID": "ATTR_ACCOUNT_{{ service_uuid }}"
          }]
        }
    ignore_errors: True

  - name: Remove Resource Profile
    uri: ...
    ignore_errors: True

  - name: Remove Filters
    uri: ...
    ignore_errors: True

  - name: Deprovision Subscriber from HSS
    uri:
```

```
            url: "{{ item.key }}/subscriber/{{
item.value.subscriber_id }}"
            method: DELETE
          loop: "{{ hss_subscriber_data | dict2items }}"
          ignore_errors: True
          when:
            - deprovision_subscriber | bool == true

        - name: Patch Subscriber to Dormant State
          uri:
            url: "{{ item.key }}/subscriber/{{
item.value.subscriber_id }}"
            method: PATCH
            body:
              {
                "enabled": true,
                "msisdn": "9999{{ imsi[-10:] }}",   # 虚拟号码
                "ue_ambr_dl": 9999999,               # 不可用的高值
                "ue_ambr_ul": 9999999
              }
          loop: "{{ hss_subscriber_data | dict2items }}"
          when:
            - deprovision_subscriber | default(false) | bool ==
false

        # 最终断言确定成功或失败
        - name: Set status to "Success" if Manual deprovision /
Fail if failed provision
          assert:
            that:
              - action == "deprovision"
```

为什么这种模式是最佳实践

1. 无代码重复

相同的清理任务处理这两种情况:

- 失败的供应（回滚）：如果 `block` 中的任何任务失败，`rescue` 部分会自动执行
- 有意的去供应：当调用时 `action: "deprovision"`，剧本立即跳到 `rescue`

2. 完整的清理保证

当供应在中途失败时，`rescue` 部分确保：

- 所有创建的 OCS 账户被删除
- 所有配置的网络设备条目被移除
- 分配的库存返回到池中
- HSS 订阅者被删除或设置为休眠
- 不会在任何系统中留下部分供应

这防止了“孤立”资源，这些资源：

- 消耗库存但未被跟踪
- 创建未链接到服务的计费账户
- 在故障排除过程中造成混淆
- 浪费网络资源

3. 使用 ignore_errors 进行优雅的失□□处理

注意每个清理任务都使用 `ignore_errors: True`。这□故意的，因为：

- 在回滚期间，某些资源可能尚未创建
- 我们希望尝试所有清理任务，即使某些失败
- 最终断言确定整体成功/失败

例如，如果供应在“创建 OCS 账户”时失败，清理将尝试：

- 删除 OCS 账户（将失败，但被忽略）
- 移除属性配置文件（将失败，但被忽略）
- 返回库存（成功）
- 删除 HSS 订阅者（可能不存在，被忽略）

4. 区分去供应和回滚

`rescue` 末尾的最终断言很巧妙：

```
- name: Set status to "Success" if Manual deprovision / Fail if
failed provision
  assert:
    that:
      - action == "deprovision"
```

这意味着：

- 如果 `action == "deprovision"`：断言通过，剧本成功（状态 0）
- 如果 `action` 未设置或 `!= "deprovision"`：断言失败，剧本失败（状态 2）

因此，相同的清理代码根据意图导致不同的供应作业状态。

5. 基于服务类型的条件清理

某些清理任务使用条件来处理不同的场景：

```
- name: Deprovision Subscriber from HSS
  uri: ...
  when:
    - deprovision_subscriber | bool == true

- name: Patch Subscriber to Dormant State
  uri: ...
  when:
    - deprovision_subscriber | default(false) | bool == false
```

这允许灵活的清理：

- 完全删除：当 SIM 专用于客户时（`deprovision_subscriber: true`）
- 休眠状态：当 SIM 可重用并应保留在 HSS 中时（`deprovision_subscriber: false`）

如何使用此模式

对于供应：

```
{
  "product_id": 1,
  "customer_id": 456,
  "provisioning_play": "play_psim_only"
}
```

□□□□□□□□□□ `rescue` □□□□□

□□□□□□

```
{
  "service_id": 123,
  "service_uuid": "Service_abc123",
  "action": "deprovision",
  "provisioning_play": "play_psim_only"
}
```

□□□□□□ `rescue` □□□□□□□□□□□□□□

□□□□

□ □□□□□□□□□□□□□□□□□□□ □ □□□□□□□□□□□□□□□□□□ □ □□□□□□□□□□□□□□□ □ □□□□□□□□ □□□□□□□□□□□□□□ □ □□□□□□□□□□□□□□□□□□□□ □ □□□□□□□□□□ `action: "deprovision"` □□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□

`provisioning_json_vars` □□□□□□□□□□□□□□□□□□□□□□□□□□

```
{
  "provisioning_json_vars": "{\"monthly_cost\": 50,
\"data_limit_gb\": 100}"
}
```

□□□□□□□□□□□□□□□

```
{
  "product_id": 1,
  "customer_id": 456,
  "monthly_cost": 45,
  "data_limit_gb": 150
}
```

合并的 `extra_vars` 将使用覆盖的值。这允许：

- 针对特定客户的自定义定价
- 基于促销的不同数据限制
- 使用不同参数进行测试，而无需修改产品

没有库存的产品

并非所有产品都需要库存。例如，数据附加组件或功能切换可能具有：

```
{
  "product_id": 10,
  "product_name": "Extra 10GB Data",
  "provisioning_play": "play_local_data_addon",
  "provisioning_json_vars": "{\"data_gb\": 10}",
  "inventory_items_list": "[]"
}
```

在这种情况下，剧本接收：

```
extra_vars = {
    "product_id": 10,
    "customer_id": 456,
    "service_id": 123,   # 要添加数据的服务
    "data_gb": 10,
    "access_token": "eyJ..."
}
```

剧本只需将数据添加到现有服务中，而无需任何库存项目。

标准供应工作流程

标准供应在以下情况下启动：

- 员工通过 UI 向客户添加服务
- 客户通过自助服务门户订购服务
- API 直接调用 `PUT /crm/provision/`

当您点击“供应”时

以下用户点击“供应”按钮时发生的完整流程：

1. UI 显示产品选择

用户从产品目录中选择一个产品。产品包含：

- `provisioning_play` - 要运行的 Ansible 剧本
- `inventory_items_list` - 所需库存（例如，`['SIM Card', 'Mobile Number']`）
- `provisioning_json_vars` - 默认变量

2. 库存选择器（如果需要）

如果 `inventory_items_list` 不为空，则会出现一个模态窗口，显示每种库存类型的下拉菜单。用户必须选择可用的库存项目才能继续。

3. 点击供应按钮

JavaScript 发送 `PUT /crm/provision/` 请求：

```
PUT /crm/provision/
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
  "product_id": 42,
  "customer_id": 123,
  "SIM Card": 5001,
  "Mobile Number": 5002
}
```

4. API 接收请求

供应端点：

- 验证身份验证（Bearer 令牌、API 密钥或 IP 白名单）
- 检查用户□否具有 `CREATE_PROVISION` 权限
- 从令牌中提取 `initiating_user`
- 从数据库加载产品定义
- 检索剧本路径：`OmniCRM-API/Provisioners/plays/play_psim_only.yaml`

5. 变量合并

系统将来自多个来源的变量组合：

```
# 来自产品
product_vars = json.loads(product['provisioning_json_vars'])
# 来自请求体
request_vars = request_body_json
# 系统添加
system_vars = {
    'product_id': 42,
    'customer_id': 123,
    'access_token': auth_token,  # 见下面的身份验证部分
    'initiating_user': 7
}
# 最终合并
extra_vars = {**product_vars, **request_vars, **system_vars}
```

6. 创建供应记录

数据库记录以状态 1（运行中）创建：

```
provision = {
    'provision_id': 456,
    'customer_id': 123,
    'product_id': 42,
    'provisioning_play': 'play_psim_only',
    'provisioning_json_vars': json.dumps(extra_vars),
    'provisioning_status': 1,  # 运行中
    'task_count': 85,
    'initiating_user': 7,
    'created': '2025-01-10T14:30:00Z'
}
```

7. 生成后台线程

系统生成一个后台线程以异步执行剧本：

- 剧本：`play_psim_only.yaml`
- 变量：合并的 extra_vars
- 供应 ID：456（用于状态跟踪）
- 刷新令牌：在执行期间传递以进行令牌刷新

8. API 立即返回

响应返回到 UI，包含 `provision_id`：

```
{
  "provision_id": 456,
  "provisioning_status": 1,
  "message": "Provisioning job created"
}
```

9. UI 轮询更新

UI 开始每 3 秒轮询 `GET /crm/provision/provision_id/456` 以检查状态。响应包括：

```
{
  "provision_id": 456,
  "provisioning_status": 1,
  "task_count": 12,
  "provisioning_result_json": [
    {
      "event_number": 1,
      "event_name": "Get Product information from CRM API",
      "provisioning_status": 0,
      "timestamp": "2024-01-15T10:30:05"
    },
    {
      "event_number": 2,
      "event_name": "Assign SIM Card from inventory",
      "provisioning_status": 1,
      "timestamp": "2024-01-15T10:30:07"
    }
  ]
}
```


10. Ansible 在后台执行

剧本按顺序运行任务：

- 每个任务完成都会在数据库中创建 `Provision_Event` 记录
- 事件包括：任务名称、状态（0=成功，2=失败，3=失败但被忽略）、结果 JSON
- UI 显示实时进度，显示已完成的任务和当前正在运行的任务
- 失败的任务在事件详细信息中显示错误消息

在 **UI** 中跟踪：

在供应运行时（状态 1），用户可以查看：

- 服务详细信息页面 - 显示供应状态徽章（运行中/成功/失败）
- 活动日志 - 列出所有供应事件及其时间戳
- 供应详细视图 - 显示逐任务进度，并可展开/折叠以获取详细信息

示例显示：

供应状态：���行中（12 个任务中完成 8 个）

✓ 从 CRM API 获取产品信息 ✓ 获取客户详细信息 ✓ 从库存中分配 SIM 卡（ICCID: 8991101200003204510） ✓ 分配手机号码（555-0123） ⟳ 在 OCS/CGRateS 中创建账户（进行中...□ □ □□□□□□ □ □□□□□□ ...

11. 供应完成

最终状态设置：

- `provisioning_status: 0` - 成功
- `provisioning_status: 2` - 失败（严重错误）

UI 停止轮询并显示结果：

- 成功：绿色勾号，服务标记为活动，用户可以查看服务详细信息
- 失败：红色 X，显示错误消息，提供重试或联系支持的选项
- 电子邮件通知：如果失败，电子邮件发送到配置中的 `provisioning.failure_list`

身份验证和授权

用户跟踪

每个供应作业跟踪哪个用户发起了它：

- 用户发起：`initiating_user` 字段设置为其 JWT 令牌中的用户 ID
- **API** 密钥身份验证：使用第一个管理员用户 ID
- **IP** 白名单身份验证：使用第一个管理员用户 ID

权限检查

系统在允许供应之前检查权限：

- 员工需要 `CREATE_PROVISION` 权限
- 客户只能为自己的账户供应服务（`VIEW_OWN_PROVISION` 权限）

Ansible 如何通过 **CRM API** 进行身份验证

Ansible 剧本需要进行经过身份验证的 API 调用回 CRM（以获取产品详细信息、创建服务、更新库存等）。身份验证通过传递给剧本的 **Bearer** 令牌处理。

`access_token` 的来源取决于用于调用供应 API 的身份验证方法：

方法 **1**：用户登录（**Bearer** 令牌）

当用户通过 Web UI 登录时：

1. 用户进行身份验证：`POST /crm/auth/login`
2. 接收 JWT `access_token`（短期有效，15-30 分钟）和 `refresh_token`（长期有效）
3. 使用 Bearer 令牌在头中进行供应请求
4. 供应 API 从 `Authorization: Bearer ...` 头中提取令牌
5. 存储令牌以供供应期间使用
6. 作为 `access_token` 变量传递给 Ansible

实施流程：

系统从 Authorization 头中提取 Bearer 令牌，验证并解码，然后存储以供供应期间使用。然后将令牌作为额外变量传递给剧本，以便它可以对后续 API 调用进行身份验证。

方法 **2**：**API** 密钥（**X-API-KEY** 头）

对于使用 API 密钥的自动化系统：

1. 系统发出请求：`PUT /crm/provision/`，并带有 `X-API-KEY: your-api-key...` 头
2. 供应 API 验证 API 密钥□否与 `crm_config.yaml` 中的密钥匹配
3. 为第一个管理员用户动态生成新的 **JWT** 令牌
4. 存储令牌以供供应使用
5. 传递给 Ansible

为什么生成令牌？

API 密钥□字符串，而不□ JWT。剧本调用 API 端点时需要 JWT 身份验证。因此：

- 验证 API 密钥
- 如果有效且具有 `admin` 角色，则生成临时 JWT
- 使用第一个管理员用户的 ID 作为 JWT 主体
- 令牌允许剧本进行经过身份验证的 API 调用

实施流程：

系统验证 API 密钥与配置的密钥。如果有效且具有管理员角色，则使用管理员用户身份生成临时 JWT，然后存储以供供应使用。

方法 **3**: **IP** 白名单

对于受信任的内部系统在私有网络上：

1. 系统从白名单 IP（例如，192.168.1.100）发出请求
2. 供应 API 检查客户端 IP 是否与 `crm_config.yaml` 中的 `ip_whitelist` 匹配
3. 如果在白名单中，为第一个管理员用户生成新的 **JWT** 令牌
4. 存储令牌以供供应使用
5. 传递给 Ansible

实施流程：

系统检查客户端 IP 是否与白名单匹配。如果允许，则为管理员用户生成 JWT 并存储以供供应使用。

在剧本中使用令牌

剧本中的每个 API 调用都包括令牌：

```
- name: Get Product Details
  uri:
    url: "http://localhost:5000/crm/product/product_id/{{
product_id }}"
    headers:
      Authorization: "Bearer {{ access_token }}"

- name: Create Service Record
  uri:
    url: "http://localhost:5000/crm/service/"
    method: PUT
    headers:
      Authorization: "Bearer {{ access_token }}"
    body:
      customer_id: "{{ customer_id }}"
      service_name: "Mobile Service"
```

令牌过期和刷新

长时间运行的剧本（5-10 分钟）可能会超出 `access_token`（15-30 分钟过期）。对于用户发起的供应，系统传递 `access_token` 和 `refresh_token`：

```
refresh_token = request.cookies.get('refresh_token')
run_playbook(playbook_path, extra_vars, provision_id,
refresh_token=refresh_token)
```

如果 `access_token` 过期，剧本运行器可以：

1. 检测到 401 未授权响应
2. 使用 `refresh_token` 调用 `POST /crm/auth/refresh`
3. 接收新的 `access_token`
4. 重试失败的请求

对于 API 密钥/IP 白名单身份验证，生成的令牌可以具有更长的过期时间（1-2 小时），因为这些受信任的自动化系统。

供应过程

1. 作业创建

   当接收到供应请求时，系统：

   - 验证请求并检查权限
   - 加载产品定义中指定的 Ansible 剧本
   - 在数据库中创建状态为 1（运行中）的 `Provision` 记录
   - 从产品定义和请求体中提取变量
   - 捕获用于 API 访问的身份验证令牌

2. 令牌处理

   Ansible 剧本需要通过 CRM API 进行身份验证以检索数据并进行更改。供应系统以两种方式处理此问题：

   - **Bearer** 令牌（**JWT**）：对于用户发起的供应，来自请求的 `refresh_token` 用于在剧本执行期间生成新访问令牌
   - **API** 密钥/**IP** 身份验证：对于自动化系统，`access_token` 直接传递给剧本

3. 后台执行

剧本在后台线程中运行。这允许 API 立即返回，同时供应异步继续。

在执行期间：

- 每个任务的完成/失败都会创建一个 `Provision_Event` 记录
- 事件处理程序监控关键失败与被忽略的失败
- 实时状态更新写入数据库
- UI 可以通过 `GET /crm/provision/provision_id/<id>` 轮询更新

4. 剧本执行

   Ansible 剧本通常执行以下操作：

   - 从 API 检索产品信息
   - 从 API 检索客户信息
   - 分配库存项目（SIM 卡、IP 地址、电话号码等）
   - 在 OCS/OCS 中创建账户
   - 配置网络设备
   - 在 CRM API 中创建服务记录
   - 添加设置成本交易
   - 向客户发送欢迎电子邮件/SMS

5. 错误处理

   Ansible 剧本使用 `block` 和 `rescue` 部分进行回滚：

   - 如果关键任务失败，`rescue` 部分将移除部分供应
   - 标记为状态 3 的带有 `ignore_errors: true` 的任务不会导致作业失败
   - 严重错误（YAML 语法、连接失败）会创建一个特殊的错误事件，包含调试信息

示例：标准供应剧本

以下来自 `play_simple_service.yaml` 的示例：

```
- name: Simple Provisioning Play
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: Main block
      block:
      - name: Get Product information from CRM API
        uri:
          url: "http://localhost:5000/crm/product/product_id/{{
product_id }}"
          method: GET
          headers:
            Authorization: "Bearer {{ access_token }}"
          return_content: yes
          validate_certs: no
        register: api_response_product

      - name: Set package facts
        set_fact:
          package_name: "{{ api_response_product.json.product_name
}}"
          setup_cost: "{{
api_response_product.json.retail_setup_cost }}"
          monthly_cost: "{{ api_response_product.json.retail_cost
}}"

      - name: Generate Service UUID
        set_fact:
          service_uuid: "Service_{{ 99999999 | random | to_uuid
}}"

      - name: Create account in OCS
        uri:
          url: "http://{{ crm_config.ocs.OCS }}/jsonrpc"
          method: POST
          body_format: json
          body:
            {
              "method": "ApierV2.SetAccount",
              "params": [{
                  "Tenant": "{{ crm_config.ocs.ocsTenant }}",
```

```
                  "Account": "{{ service_uuid }}",
                  "ActionPlanIds": [],
                  "ExtraOptions": { "AllowNegative": false,
"Disabled": false }
                }]
              }
            status_code: 200
          register: response

      - name: Add Service via API
          uri:
            url: "http://localhost:5000/crm/service/"
            method: PUT
            body_format: json
            headers:
              Authorization: "Bearer {{ access_token }}"
            body:
              {
                "customer_id": "{{ customer_id }}",
                "product_id": "{{ product_id }}",
                "service_name": "Service: {{ service_uuid }}",
                "service_uuid": "{{ service_uuid }}",
                "service_status": "Active",
                "retail_cost": "{{ monthly_cost | float }}"
              }
            status_code: 200
          register: service_creation_response

      - name: Add Setup Cost Transaction
          uri:
            url: "http://localhost:5000/crm/transaction/"
            method: PUT
            headers:
              Authorization: "Bearer {{ access_token }}"
            body_format: json
            body:
              {
                "customer_id": {{ customer_id | int }},
                "service_id": {{
service_creation_response.json.service_id | int }},
                "title": "{{ package_name }} - Setup Costs",
                "retail_cost": "{{ setup_cost | float }}"
              }
          register: api_response_transaction
```

```
    rescue:
      - name: Remove account in OCS
        uri:
          url: "http://{{ crm_config.ocs.OCS }}/jsonrpc"
          method: POST
          body_format: json
          body:
            {
              "method": "ApierV2.RemoveAccount",
              "params": [{
                  "Tenant": "{{ crm_config.ocs.ocsTenant }}",
                  "Account": "{{ service_uuid }}"
              }]
            }
          status_code: 200

      - name: Fail the provision
        assert:
          that:
            - false
```

此剧本演示了典型流程：

1. 从 CRM API 获取产品详细信息
2. 生成唯一的服务 UUID
3. 在 OCS 中创建计费账户
4. 通过 CRM API 创建服务记录
5. 添加设置成本交易
6. 如果任何操作失败，`rescue` 部分将删除 OCS 账户

简单供应工作流程

简单供应在用于需要触发供应而无需用户交互的自动化系统。最常见的用例 OCS 通过 ActionPlans 触发附加供应。

简单供应端点

OmniCRM 提供两个简单的供应端点：

- `POST /crm/provision/simple_provision_addon/service_id/<id>/product_id/<id>`

  用于自动化附加供应（例如，定期收费、自动充值）

- `POST /crm/provision/simple_provision_addon_recharge/service_id/<id>/product_id/<id>`

  用于需要立即反馈的快速充值操作

简单供应的身份验证

简单供应端点使用 **IP** 白名单 或 **API** 密钥 进行身份验证：

- 请求的源 IP 会检查□否与 `crm_config.yaml` 中的 `ip_whitelist` 匹配
- 或者可以提供来自 `crm_config.yaml` 中的 `api_keys` 的 API 密钥
- 生成访问令牌并传递给剧本

示例：**OCS ActionPlan** 回调

OCS 可以配置为在执行定期操作时调用简单供应端点：

```
{
  "method": "ApierV1.SetActionPlan",
  "params": [{
    "Id": "ActionPlan_Service123_Monthly_Charge",
    "ActionsId": "Action_Service123_Add_Monthly_Data",
    "Timing": {
      "Years": [],
      "Months": [],
      "MonthDays": [1],
      "Time": "00:00:00Z"
    },
    "Weight": 10,
    "ActionTriggers": [
      {
        "ThresholdType": "*min_event_counter",
        "ThresholdValue": 1,
        "ActionsID": "Action_Service123_HTTP_Callback"
      }
    ]
  }]
}
```

该操作会发出 HTTP POST 到:

这触发了相关的剧本（例如，`play_topup_no_charge.yaml`），该剧本向服务添加数据/积分。

示例：简单充值剧本

来自 `play_topup_monetary.yaml`:

```yaml
- name: Mobile Topup Monetary - 2024
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: Get Product information from CRM API
      uri:
        url: "http://localhost:5000/crm/product/product_id/{{
product_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
      register: api_response_product

    - name: Get Service information from CRM API
      uri:
        url: "http://localhost:5000/crm/service/service_id/{{
service_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
      register: api_response_service

    - name: Set service facts
      set_fact:
        service_uuid: "{{ api_response_service.json.service_uuid
}}"
        customer_id: "{{ api_response_service.json.customer_id }}"
        package_name: "{{ api_response_product.json.product_name
}}"
        monthly_cost: "{{ api_response_product.json.retail_cost
}}"

    - name: Get Customer Payment Method
      uri:
        url: "http://localhost:5000/api/payments/methods/default?
customer_id={{ customer_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
```

```
      return_content: yes
    register: api_response_payment_method

  - name: Charge customer
    uri:
      url: "http://localhost:5000/api/payments/charge"
      method: POST
      headers:
        Authorization: "Bearer {{ access_token }}"
      body_format: json
      body:
        {
          "customer_id": "{{ customer_id | int }}",
          "amount": "{{ monthly_cost }}",
          "currency": "USD",
          "metadata": {
            "description": "{{ package_name }} topup",
            "service_id": "{{ service_id | int }}"
          }
        }
    register: api_response_payment

  - name: Add monetary balance to OCS
    uri:
      url: "http://{{ crm_config.ocs.OCS }}/jsonrpc"
      method: POST
      body_format: json
      body:
        {
          "method": "ApierV1.AddBalance",
          "params": [{
              "Tenant": "{{ crm_config.ocs.ocsTenant }}",
              "Account": "{{ service_uuid }}",
              "BalanceType": "*monetary",
              "Balance": {
                  "Value": "{{ monthly_cost | float * 100 }}",
                  "ExpiryTime": "+4320h"
              }
          }]
        }
      status_code: 200

  - name: Add Transaction to CRM
    uri:
```

```
        url: "http://localhost:5000/crm/transaction/"
        method: PUT
        headers:
          Authorization: "Bearer {{ access_token }}"
        body_format: json
        body:
          {
            "customer_id": {{ customer_id | int }},
            "service_id": {{ service_id | int }},
            "title": "{{ package_name }}",
            "retail_cost": "{{ monthly_cost | float }}"
          }

    - name: Send Notification SMS
      uri:
        url: "http://sms-gateway/SMS/plaintext/{{ api_key }}"
        method: POST
        body_format: json
        body:
          {
            "source_msisdn": "YourCompany",
            "destinatination_msisdn": "{{ customer_phone }}",
            "message_body": "Thanks for topping up {{ monthly_cost
}}!"
          }
        status_code: 201
      ignore_errors: True
```

此剧本：

1. 从 API 获取服务和产品详细信息
2. 检索客户的支付方式
3. 通过 Stripe API 向客户收费
4. 向 OCS 添加货币余额
5. 在 CRM 中记录交易
6. 发送确认 SMS（使用 `ignore_errors: True`，以便失败不会导致作业失败）

供应链

对于需要多个供应步骤的复杂产品，OmniCRM 支持 供应链。链按顺序执行多个剧本，在它们之间传递上下文。

示例用例：一个捆绑服务供应：

1. 基础互联网服务（创建主要服务记录）
2. IPTV 附加组件（使用步骤 1 的 service_id）
3. 静态 IP 附加组件（使用步骤 1 的 service_id）

供应服务会自动：

- 查询数据库以获取第一个剧本创建的 `service_id`
- 将其注入后续剧本的 `extra_vars`
- 将每个剧本跟踪为单独的 `Provision` 记录

失败原因和调试

当供应失败时，系统捕获详细信息以帮助诊断问题。

常见失败场景

关键任务失败（状态 **2**）

这些会导致整个供应作业失败：

- API 调用返回意外状态代码
- 断言失败（例如，`assert: that: response.status == 200`）
- 缺少必需的库存项目
- 网络设备无法访问
- 无效凭据或过期令牌
- OCS/OCS 错误

被忽略的失败（状态 **3**）

这些被记录但不会导致作业失败：

- 可选的 SMS/email 通知失败
- 非关键数据查找（标记为 `ignore_errors: True`）
- 在去供应期间的清理操作

致命错误

这些会阻止剧本的运行：

- 剧本中的 YAML 语法错误
- 剧本中的未定义变量
- 缺少剧本文件
- 与 Ansible 控制器的连接失败

当发生致命错误时，系统会创建一个特殊的错误事件，包含：

- Ansible 退出代码
- 完整的 stdout（包含语法错误详细信息）
- 完整的 stderr（包含运行时错误）
- 该���型失败的常见原因列表
- 传递给剧本的所有变量

错误通知电子邮件

当供应失败（状态 2）时，系统会自动向配置的失败通知列表（`crm_config.yaml` 中的 `provisioning.failure_list`）发送电子邮件。

电子邮件包括：

- 客户信息
- 产品/服务详细信息
- 颜色编码的任务结果：
  - 绿色：成功的任务
  - 橙色：失败但被忽略的任务
  - 红色：关键失败
- 对于关键失败：完整的调试输出，包括请求/响应主体
- 对于致命错误：Ansible 输出、错误消息和常见原因

监控供应作业

OmniCRM 提供全面的指标来监控供应性能。有关供应指标的完整详细信息，包括作业持续时间、成功率、任务计数和错误跟踪，请参见 监控与指标。

供应状态 API

要检查供应作业的状态：

```
GET /crm/provision/provision_id/<id>
Authorization: Bearer <token>
```

响应包括：

```
{
  "provision_id": 123,
  "customer_id": 456,
  "customer_name": "John Smith",
  "product_id": 10,
  "provisioning_status": 0,
  "provisioning_play": "play_psim_only",
  "playbook_description": "OmniCore Service Provisioning 2024",
  "task_count": 85,
  "provisioning_result_json": [
    {
      "event_number": 1,
      "event_name": "Get Product information from CRM API",
      "provisioning_status": 1,
      "provisioning_result_json": "{...}"
    },
    {
      "event_number": 2,
      "event_name": "Create account in OCS",
      "provisioning_status": 1,
      "provisioning_result_json": "{...}"
    }
  ]
}
```

列出供应作业

要获取所有供应作业的分页列表：

```
GET /crm/provision/?
page=1&per_page=20&sort=provision_id&order=desc
Authorization: Bearer <token>
```

支持过滤：

```
GET /crm/provision/?filters={"provisioning_status":
[2]}&search=Mobile
Authorization: Bearer <token>
```

这仅返回失败的作业（状态 2），其中描述包含“Mobile”。

最佳实践

剧本设计

- 始终使用 **block/rescue**：确保可以回滚部分供应
- 明智地使用 **ignore_errors**：仅用于真正可选的操作
- 记录重要变量：使用 `debug` 任务记录关键值以便于故障排除
- 验证响应：使用 `assert` 检查 API 响应□否如预期
- 幂等性：设计剧本以安全地重新运行

身份验证

- 用户发起的供应：始终在长时间运行的剧本中使用 refresh_token
- 自动化供应：使用 IP 白名单或 API 密钥与生成的访问令牌
- 令牌过期：refresh_token 确保访问令牌根据需要重新生成

错误处理

- 提供上下文：在错误消息中包含 customer_id、service_id 和操作详细信息
- 适当通知：关键失败触发电子邮件，但不要对预期失败进行垃圾邮件
- 调试信息：在 Provision_Event 记录中捕获完整的请求/响应主体

安全

- 验证输入：在供应之前检查 customer_id、product_id、service_id
- 权限检查：验证用户只能为授权客户供应
- 敏感数据：使用红action系统从日志中剥离密码/密钥
- **IP** 白名单：仅将简单供应端点限制为受信任的系统

性能

- 后台执行：绝不要阻塞 API 响应，等待供应
- 轮询间隔：UI 应每 2-5 秒轮询状态更新
- 并行任务：使用 Ansible 的原生并行性来处理独立操作
- 数据库更新：事件处理程序实时更新数据库，无需在执行期间查询

相关文档

- `concepts_ansible` - 一般 Ansible 供应概念
- `concepts_sim_provisioning` - SIM 卡供应（物理和 eSIM）
- `concepts_api` - CRM API 身份验证和使用
- `concepts_products_and_services` - 产品和服务定义
- `administration_inventory` - 供应的库存管理
- 监控与指标 - 供应性能指标和监控

客户活动日志

对客户、联系人、站点、服务和财务参数（如交易、发票和支付方式）所做的每个更改都记录在活动日志中。

这使我们能够跟踪对系统所做的更改、谁进行了更改以及何时进行的更改，这对于审计更改和追踪问题非常有用。例如，如果客户说他们从未收到发票，我们可以检查活动日志以查看□否已发送，或者如果某个联系人被删除，我们可以查看□谁在何时删除的。

活动日志□一个按时间顺序排列的更改列表，最新的更改位于顶部，较旧的更改位于列表的下方，可以通过标签进行过滤。

活动日志记录无法删除，但可以进行过滤，并且可以查看详细信息以了解更改了什么以及由谁更改。

□□□□

□□□

分配计划工作流程

分配计划功能允许员工通过从目录中选择产品并启动供应过程来为客户提供服务。这□在客户未自助供应时创建新服务的主要方法。

概述

分配计划涉及：

1. 选择客户
2. 从目录中选择产品
3. 配置库存要求（SIM卡、设备等）
4. 设置服务参数（自动续订、自定义字段）
5. 启动供应
6. 监控供应进度

此工作流程适用于所有服务类型：移动、互联网、IPTV和VoIP。

访问分配计划

从产品目录：

从客户页面：

从附加组件页面：

分配计划界面在模态框或专用页面中打开。

步骤工作流程

步骤 1：浏览产品目录

产品目录显示按类别和客户类型分组的可用产品。

视图切换：

> 类别： • 服务计划 • 附加组件

在个人和商业之间切换过滤产品，以仅显示可供所选客户类型使用的产品��

产品卡片：

产品以轮播或网格形式显示：

点击**“分配给客户”**以继续。

步骤 2：选择客户

如果尚未在客户页面上，系统会提示您选择客户。

客户搜索：

> 搜索客户：[John ▼]
>
> 匹配的客户：• John Smith（ID：123）• John Doe（ID：456）• Johnson Enterprises（ID：789）

输入以搜索：

- 客户名称
- 客户ID
- 电子邮件地址
- 电话号码

从下拉列表中选择客户。

警告

在继续之前，请确保您已选择正确的客户。将计划分配给错误的客户需要手动干预以进行更正。

步骤 3：配置库存（如有需要）

如果产品需要库存项（在`inventory_items_list`中定义），则会出现库存选择器。

库存选择器示例：

> 所需库存项：
>
> SIM卡 *
>
> 可用SIM卡： • SIM-00123 - ICCID: 8944...0001 (新) • SIM-00124 - ICCID: 8944...0002 (新) • SIM-00125 - ICCID: 8944...0003 (新)
>
> 手机号码 *
>
> 可用号码： • +44 7700 900123 (保留) • +44 7700 900124 (可用) • +44 7700 900125 (可用)

库存选择规则：

- 星号（*）表示必填字段
- 仅显示可用库存项（状态：“有货”或“新”）
- 下拉列表根据库存模板名称动态加载
- 一旦选择，项目会被暂时保留

发生的事情：

- 选定的库存项作为变量传递给供应剧本

- 在供应过程中，项目分配给服务和客户
- 项目状态从“有货”更改为“已分配”

步骤 4：配置自动续订（可选）

对于定期服务，您可能会被提示设置自动续订：

> 您想为此服务启用自动续订吗？
>
> 启用后，此服务将在每个计费周期结束时自动续订，并向客户的默认付款方式收费。
>
> [否] [□]

自动续订行为：

- 是：服务自动续订，客户每月收费
- 否：服务在合同期结束时到期，需要手动续订

最佳实践：对于消费者服务默认选择“□”，对于一次性服务或客户要求手动控制时选择“否”。

步骤 5：审查并确认

审查屏幕显示所有选择的内容，然后进行供应：

> 客户：John Smith（ID：123）产品：移动 - 20GB计划
>
> 库存： • SIM卡：SIM-00123（ICCID：8944...0001） • 手机号码：+44 7700 900123
>
> 定价： • 设置费：£0.00 • 月费用：£15.00
>
> 自动续订：□
>
> [取消] [确认并供应]

点击**“确认并供应”**以启动供应过程。

步骤 6：供应进度

供应模态框显示实时进度：

> ✓ 验证客户账户 ✓ 分配SIM卡（ICCID: 8944...0001） ✓ 分配手机号码（+44 7700 900123） ⟳ 配置OCS账户（进行中...□ □ □□□□□□ □ □□□□□□□□□

进度：6个任务中的3个已完成

模态框每0.2秒轮询供应API以获取状态更新。

进度指示器：

- ☑ 成功完成
- ⟳ 当前运行中
- □ □□□□□□□□
- ✗ 失败（如果发生错误）

步骤 **7**：完成

成功：

> 服务已成功为John Smith供应
>
> 服务ID：789 服务名称：移动 - +44 7700 900123 状态：活动
>
> [查看服务] [关闭]

点击**“查看服务”**以打开服务详细信息页面。

失败：

如果供应失败：

> 错误：无法连接到OCS
>
> 服务记录已创建，但供应未完成。请查看错误并重试。
>
> 供应ID：456
>
> [查看日志] [重试] [关闭]

- 查看日志：打开供应详细信息及错误消息
- 重试：再次尝试供应
- 关闭：退出模态框（服务记录仍然存在但未��活）

特殊情况

向现有服务添加附加组件

当将附加组件（类别：“addon”）分配给已经拥有服务的客户时：

1. 自动服务检测：
    - 系统查找客户的现有服务
    - 按service_type过滤（移动附加组件仅显示移动服务）
    - 如果客户有多个匹配的服务，提示选择其中一个
2. 不创建新服务：
    - 附加组件针对现有service_id进行供应
    - 使用现有服务的OCS账户
    - 库存（如有）分配给现有服务
3. 供应剧本：
    - 与独立服务不同的剧本
    - 通常向现有账户添加余额、功能或设备

示例：

> 现有移动服务： • 移动 - +44 7700 900123 (ID: 789) • 移动 - +44 7700 900456 (ID: 790)
>
> 哪个服务应该接收此附加组件？ [移动 - +44 7700 900123 ▼]
>
> [取消] [继续]

为商业客户供应

商业客户可能有额外要求：

- 站点选择 - 选择哪个商业地点接收服务
- 联系人分配 - 指定账单/技术联系人
- 自定义字段 - 账户号码、成本中心、采购订单号码

示例商业流程：

> 选择安装地点：[伦敦办公室 - 123 Main St ▼]
>
> 账单联系人：[Jane Doe - <jane@acme.com> ▼]
>
> 技术联系人：[Bob Smith - <bob@acme.com> ▼]
>
> 采购订单号码：[PO-2025-001234_____]
>
> [取消] [继续]

批量服务分配

用于将相同计划分配给多个客户（例如，大规模迁移）：

1. 使用CSV导入（如可用）
2. 或使用模板设置逐个分配
3. 库存必须以批量形式可用
4. 监控供应队列以避免过载

常见工作流程

工作流程 1：新移动服务

1. 客户走进商店想要移动服务
2. 员工打开产品 → 计划
3. 切换到个人客户类型
4. 选择**“移动 - 20GB计划”**
5. 点击**“分配给客户”**
6. 通过电话搜索客户：“+1234567890”
7. 从结果中选择**John Smith**
8. 从库存选择器中选择SIM卡
9. 选择可用手机号码
10. 启用自动续订
11. 确认并供应
12. 观察进度直到完成

13. 将SIM卡和欢迎包交给客户

工作流程 2：向现有客户添加互联网

1. 导航到客户页面：客户 → **John Smith**
2. 点击服务选项卡
3. 点击**“添加服务”**按钮
4. 浏览互联网计划
5. 选择**“光纤 - 100Mbps”**
6. 选择安装地点（如商业/多个地点）
7. 从库存中选择CPE调制解调器
8. 设置安装日期（如有需要）
9. 供应服务
10. 创建安装工单

���作流程 3：分配数据补充附加组件

1. 客户来电：“我需要更多数据”
2. 员工在全局搜索中搜索客户
3. 打开客户服务选项卡
4. 点击移动服务旁的“附加组件”
5. 选择“5GB数据提升”
6. 授权付款
7. 附加组件立即供应
8. 客户收到即时数据提升

故障排除

“没有可用产品”

- 原因：过滤器排除了所有产品
- 解决方法：
  - 切换客户类型（个人与商业）
  - 检查产品目录□否有启用的产品

- 验证产品□否符合客户资格

“没有可用库存”

- 原因：库存项缺货或全部分配
- 解决方法：
  - 向系统添加更多库存项
  - 检查项目状态（应为“新”或“有货”）
  - 验证库存模板名称□否符合产品要求

“未找到客户”

- 原因：客户不存在或搜索词不正确
- 解决方法：
  - 首先创建客户
  - 尝试不同的搜索词（ID、电子邮件、电话）
  - 检查□否有拼写错误

供应卡住

- 原因：剧本错误或外部系统无法访问
- 解决方法：
  - 等待超时（通常为5分钟）
  - 检查供应日志以获取具体错误
  - 验证OCS、网络系统□否在线
  - 修复问题后重试供应

供应成功但服务无法使用

- 原因：创建了OCS账户但网络未更新
- 解决方法：
  - 检查OCS□否有账户
  - 验证SIM在HLR/HSS中□否激活
  - 检查网络供应（RADIUS、DPI等）
  - 审查剧本任务以查找遗漏的步骤

最佳实践

在分配之前：

- 验证客户□否在文件中有有效的付款方式
- 确认客户对产品的资格（住宅与商业）
- 确保所需库存可用
- 与客户审查产品条款和定价

在分配期间：

- 在确认之前仔细检查客户选择
- 选择正确的库存项（检查序列号）
- 启用自动续订以方便（除非客户反对）
- 监控供应进度直到完成

在分配之后：

- 验证服务□否出现在客户的服务列表中
- 检查服务状态□否为“活动”
- 确认库存□否正确分配
- 向客户发送欢迎电子邮件或说明
- 如果可能，测试服务（拨打测试电话，检查数据）

对于附加组件：

- 确认附加组件与现有服务兼容
- 解释计费（一次性与定期）
- 在供应之前验证付款
- 检查附加组件供应后余额□否立即更新

相关文档

- `csa_add_service` - 服务概述
- `concepts_products_and_services` - 产品目录概念

- `administration_inventory` - 管理库存项
- `concepts_provisioning` - 供应系统详细信息
- `guide_product_lifecycle` - 包括供应在内的完整产品生命周期

修改服务

服务可以通过 `Self-Care Portal <self_care_portal>` 由最终客户修改，或通过管理员在管理门户进行修改。

一旦服务被提供，您可以修改其参数、添加增强功能或更改设置。

编辑服务参数

基本服务参数可以通过点击服务详情页面上的 编辑 按钮进行修改。

可编辑字段：

- 服务名称
- 服务状态（活动、非活动、暂停）
- 服务备注
- 自动续订设置

- 关联的库存项目
- 自定义字段

更改会立即保存并反映在客户的自助服务门户中。

添加服务附加功能

附加功能通过额外的功能、数据配额或硬件增强现有服务。常见的附加功能类型包括：

- 数据充值 - 额外的数据配额（例如，“5GB 数据提升”）
- 功能升级 - 额外的能力（例如，“国际通话”）
- 硬件租赁 - 设备附加（例如，“WiFi 6 调制解调器”）
- 高级服务 - 增强功能（例如，“静态 IP 地址”）

访问附加功能目录

从服务详情页面：

1. 导航到您想要增强的服务
2. 点击“添加附加功能”或“浏览附加功能”按钮
3. 附加功能目录打开，过滤以仅显示兼容的附加功能

自动过滤：

系统根据以下条件自动过滤附加功能：

- 服务类型 - 仅显示与服务类型匹配的附加功能（移动、互联网、VoIP 等）
- 客户类型 - 按住宅客户与商业客户进行过滤
- 服务兼容性 - 检查服务□否满足附加功能要求
- 可用性 - 仅显示启用的产品

例如，如果查看住宅移动服务，您将仅看到标记为住宅客户的移动附加功能。

附加功能显示

附加功能以交互式轮播的形式显示：

产品卡显示：

显示的信息：

- 产品图标
- 产品名称
- 功能列表（项目符号）
- 设置费用
- 每月/定期费用
- 条款和条件链接
- “添加到服务”按钮

配置附加功能

步骤 **1**：选择附加功能

点击所需的附加功能卡片，然后点击 **“添加到服务”**

步骤 **2**：选择库存（如需要）

如果附加功能需要物理库存（例如，硬件租赁），将出现库存选择器：

> 可用调制解调器： • Modem-12345 - TP-Link AX1800（新） • Modem-12346 - TP-Link AX1800（新） • Modem-12347 - Netgear RAX40（新）

选择要分配给此服务的特定库存项目。

步骤 **3**：配置自动续订（可选）

对于定期附加功能，您可能会被提示：

> 您想为此附加功能启用自动续订吗？
>
> [否] [□]

- 是 - 附加功能将在每个计费周期自动续订
- 否 - 一次性购买，客户必须手动续订

步骤 **4**：确认并配置

查看附加功能的详细信息并点击 **“确认”**

配置模态框出现，显示实时进度：

> ✓ 验证支付 ✓ 分配库存 ⟳ 配置服务（进行中...□ □ □□□□ □ □□□□□□□□

完成后，附加功能将出现在服务的附加功能列表中。

后台配置过程：

当您添加附加功能时，系统：

1. 验证客户可以购买附加功能
2. 运行附加功能的 Ansible 剧本（`provisioning_play`）
3. 将任何所需的库存项目分配给服务
4. 创建计费的交易记录
5. 更新服务配置（OCS、网络系统等）
6. 向客户发送确认通知

有关附加功能配置的技术细节，请参见 `Complete Product Lifecycle Guide - Adding Addons <guide_product_lifecycle>`。

查看活动附加功能

活动附加功能在服务详情页面的“附加功能”部分出现：

示例显示：

> 5GB 数据提升
> 添加日期：2025年1月10日 到期日期：2025年1月17日 状态：活动 费用：£5.00
>
> WiFi 6 调制解调器租赁
> 添加日期：2024年12月1日 库存：Modem-12345 自动续订： 每月： £10.00

移除附加功能

要从服务中移除附加功能：

1. 导航到服务详情页面
2. 在“活动附加功能”部分找到附加功能
3. 点击附加功能旁边的“移除”或 垃圾桶图标
4. 确认移除

发生的事情：

- 硬件附加功能 - 运行去配置剧本，库存标记为返回
- 虚拟附加功能 - 立即移除利益
- 自动续订附加功能 - 取消未来的续订
- 创建交易 - 如果适用，按比例退还任何金额

警告

移除硬件附加功能（调制解调器、CPE 等）通常需要归还设备。系统将标记库存以供返回，并可能向客户发送返回说明。

常见附加功能场景

场景 1：客户数据用尽

1. 客户联系支持：“我已经用完我的月度配额”
2. 员工导航到客户的移动服务

3. 点击“添加附加功能”
4. 选择“5GB 数据提升”
5. 立即配置
6. 客户收到即时数据充值

场景 **2**：企业需要静态 **IP**

1. 商业客户请求用于 VPN 的静态 IP
2. 员工打开客户的互联网服务
3. 浏览附加功能，选择“静态 IP 地址”
4. 系统从可用池（库存）中配置 IP
5. 在网络设备中配置路由
6. 企业收到 IP 配置详细信息

场景 **3**：光纤服务的设备租赁

1. 客户注册光纤互联网
2. 在配置过程中，客户选择调制解调器租赁
3. 员工添加“WiFi 6 调制解调器”附加功能
4. 从库存中选择可用调制解调器
5. 调制解调器寄送给客户
6. 每月 £10.00 的定期费用添加到账户

故障排除

“没有可用的兼容附加功能**”**

- 原因：没有附加功能与服务类型或客户类型匹配
- 解决方法：检查产品目录□否有与 `service_type` 和 `residential`/`business` 设置匹配的附加功能

附加功能配置失败

- 原因：配置剧本错误或库存不可用
- 解决方法：
  - 检查配置日志以获取具体错误
  - 验证库存项目□否有库存（如需要）

- 查看配置系统中的剧本日志

库存选择器没有显示项目

- 原因：没有所需类型的可用库存项目
- 解决方法：
  - 向系统添加库存项目
  - 检查现有项目□否全部分配或损坏
  - 验证库存模板名称□否与 `inventory_items_list` 完全匹配

自动续订未工作

- 原因：自动续订标志未设置或支付方式已过期
- 解决方法：
  - 验证服务□否具有 `auto_renew: true`
  - 检查客户□否在档案中有有效的支付方式
  - 查看计费系统中的计划作业

相关文档

- `guide_product_lifecycle` - 完整的附加功能配置流程
- `csa_add_service` - 创建新服务
- `concepts_products_and_services` - 产品和服务概念
- `administration_inventory` - 硬件附加功能的库存管理

□□□□

□□□

- 客户名称 - 关联客户（可点击链接）
- 服务类型 - 移动、IPTV、互联网、VoIP
- 状态 - 活动、非活动、暂停
- 开始日期 - 服务开始时间
- 结束日期 - 服务到期/结束时间
- 月费 - 零售价格
- 作 - 查看、编辑、附加功能、使用情况、删除

过滤服务

状态标签

通过标签栏速按状态过滤服务：

- 所有结果 - 显示所有服务，无论状态如何
- 活动 - 当前活动的服务（默认视图）
- 非活动 - 已取消或已过期的服务
- 暂停 - 暂时禁用的服务（因未付款、欺诈等）

点击标签会立即更新列表。活动标签会被高亮显示。

服务类型过滤器

按服务类型过滤：

选择一个或多个类型以仅显示匹配的服务。

用例：

- 移动 - 查看所有移动 SIM 服务
- 互联网 - 显示光纤、DSL 和固定无线
- **IPTV** - 显示电视订阅服务
- **VoIP** - 列出语音通信服务

客户过滤器

按客户搜索和过滤：

开始输入客户名称以查看匹配结果：

选择客户以仅显示他们的服务。

用例：快速查看特定客户的所有服务。

搜索

在所有服务字段中进行全局搜索：

搜索：

- 服务名称
- 服务 ID
- 客户名称
- 电话号码（针对移动服务）
- 服务 UUID

示例：搜索“0770”以查找所有电话号码中包含“0770”的服务。

排序服务

点击任何列标题按该字段排序：

可排序列：

- 服务 ID（默认：最新优先）
- 服务名称（按字母顺序）
- 客户名称（按字母顺序）
- 服务类型（按字母���序）
- 开始日期（按时间顺序）
- 结束日期（按时间顺序）
- 月费（按数值）

排序方向：

- 点击一次：升序（A-Z，最旧-最新，最低-最高）
- 点击两次：降序（Z-A，最新-最旧，最高-最低）
- 活动排序列显示 ▲ 或 ▼ 指示符

排序下拉菜单：

右上角的下拉菜单提供快速排序预设：

查看服务详情

点击任何服务名称以打开详细服务视图。

服务详情标签：

1. 概述 - 服务摘要、状态、日期、定价
2. 库存 - 分配的设备（SIM 卡、调制解调器等）
3. 交易 - 费用、信用、付款
4. 使用情况 - 数据/语音/SMS 使用统计
5. 附加功能 - 活动和可用的附加功能
6. 活动日志 - 变更历史

快速□作

从服务列表中，点击操作菜单（⋮）进行快速操作：

- 查看 - 打开服务详情
- 编辑 - 修改服务参数
- 附加功能 - 浏览并添加服务增强功能
- 使用情况 - 查看当前使用情况和余额
- 删除 - 取消/移除服务

批量操作

使用复选框选择多个服务以执行批量操作。

选择服务

单个选择：

点击每个服务旁边的复选框以选择。

全选：

点击表头中的复选框以选择所有可见服务。

> ☑ 服务 1 ☑ 服务 2 ☑ 服务 3

选择计数器：

界面显示已选择的服务数量：

批量操作

选择服务后，批量操作按钮会出现：

删除多个服务：

1. 选择要删除的服务
2. 点击“删除所选”按钮
3. 在模态框中确认删除

警告

批量删除永久性的，将会：

- 取消所有选定的服务
- 将库存标记为未分配
- 停止所有重复收费
- 创建活动日志条目

用例：

- 清理测试服务
- 取消已关闭客户账户的服务
- 移除重复或错误的条目

最佳实践：在进行批量操作之前使用过滤器缩小列表，以避免意外删除。

服务状态管理

状态类型

服务可以有三种状态：

活动

- 服务正常运行
- 收费适用
- 客户可以使用服务
- 显示为绿色徽章

非活动

- 服务已被取消或过期
- 不收取费用
- 客户无法使用服务
- 显示为灰色徽章
- 库存标记为待退回/翻新

暂停

- 服务暂时禁用
- 可能会收费，也可能不会（可配置）
- 客户无法使用服务
- 显示为橙色/黄色徽章
- 常见原因：未付款、欺诈调查、客户请求

??改服务状态

要更改服务的状态：

1. 打开服务详情
2. 点击“编辑”按钮
3. 更改“服务状态”下拉菜单
4. 点击“保存”

发生的事情：

- 活动 → 非活动：触发去配置剧本（如果已配置）
- 活动 → 暂停：在 OCS/网络中禁用服务，但保留记录
- 暂停 → 活动：重新启用服务，恢复计费
- 非活动 → 活动：可能触发重新配置（请谨慎操作）

服务使用情况视图

点击“使用情况”在操作菜单中打开使用情况模态框。

显示的信息：

- 余额 - 剩余信用或预付值
- 数据使用 - 使用与分配（进度条）
- 语音使用 - 使用的分钟与计划允许

- **SMS** 使用 - 发送的消息与允许
- 到期日期 - 当前余额/计划到期时间

□作:

- 充值 - 添加信用或数据（针对预付服务）
- 查看详情 - 查看详细使用情况分解
- 导出 - 下载使用报告（如果可用）

即将到来的自动续订

即将到来的自动续订视图提供一个集中界面，用于监控和管理您组织内所有计划的服务续订。

访问自动续订

导航到:

或直接:

这��显示所有计划自动续订的服务，按下一个续订日期排序。

自动续订概述

自动续订列表以表格形式显示计划续订，包含:

列:

- 客户 - 客户名称（可点击链接到客户概览）
- 服务 - 服务名称（可点击链接到客户概览）
- 产品 - 正在续订的产品/计划名称
- 费用 - 续订费用（来自产品零售价格）
- 续订时间 - 下次续订的日期和时间，采用人类可读格式
- 状态 - 服务状态（活动、暂停等）
- □作 - 立即续订或移除自动续订

示例显示:

自动续订的工作原理

当创建或修改服务时，自动续订在 CGRateS（计费系统）中被计划。系统：

1. 计划□作 - 在 CGRateS 中创建一个带有续订日期的 ActionPlan
2. 监控余额 - 在续订之前检查客户□否有足够的余额
3. 执行续订 - 在计划的日期，自动续订服务
4. 更新记录 - 创建交易，更新服务日期，并记录活动

数据源：

即将到来的自动续订视图直接查询 CGRateS，使用 `ApierV1.GetScheduledActions` API，并从 CRM 数据库中丰富客户和服务信息。

立即续订

要在计划日��之前手动触发续订：

1. 点击服务的 □【立即续订】按钮
2. 在确认模态框中查看续订详情：

   > 客户：Acme Corp 服务：FixedWireless_75628fa5 产品：家庭互联网 Mega 费用：$89.99
   >
   > 下次计划续订：2025年11月11日 上午10:45
   >
   > [取消] [确认续订]

3. 点击“确认续订”立即处理

4. 创建并执行一个配置作业
5. 在配置状态模态框中监控进度

发生的事情：

- 服务将根据产品的允许进行充值
- 在 CGRateS 中更新余额
- 创建交易记录
- 延长服务结束日期
- 创建活动日志条目

用例：

- 客户请求提前续订
- 通过重置允许解决服务问题
- 测试续订流程
- 客户希望在计划续订之前使用服务

移除自动续订

要取消服务的自动续订：

1. 点击服务的 （移除自动续订）按钮
2. 在模态框中确认删除：

> 这将移除以下内容的计划自动续订：
>
> 客户：Acme Corp 服务：FixedWireless_75628fa5 下次续订： 2025年11月11日 上午10:45
>
> 服务将不会自动续订。您需要手动续订，或者服务将在结束日期到期。
>
> [取消] [移除自动续订]

3. 点击 “移除自动续订” 以确认
4. ActionPlan 将从 CGRateS 中移除

5. 服务将自然到期，除非手动续订

发生的事情：

- ActionPlan 从 CGRateS 中删除
- 服务状态保持不变
- 服务将在当前结束日期到期
- 客户必须手动续订，否则服务将停止

用例：

- 客户正在取消服务
- 切换到手动续订流程
- 服务计划正在更改
- 客户请求停止自动收费

警告

移除自动续订意味着服务将到期，除非手动续订。客户将不会收到自动服务续订。

理解显示内容

客户列：

- 显示客户名称作为可点击链接
- 链接到客户概览页面
- 如果服务未链接到客户（孤立服务），则显示“N/A”

服务列：

- 显示服务名称/UUID
- 链接到客户概览页面
- 如果名称未设置，则显示服务 UUID
- 如果服务在数据库中找不到，则显示“N/A”

产品列：

- 来自 CGRateS ActionPlan 的产品名称

- 从计划操作元数据中提取
- 即使服务未找到，也始终显示产品名称

费用列：

- 来自 CRM 中产品表的零售成本
- 续订时将收取的金额
- 如果在数据库中找不到产品，则显示“N/A”

续订时间列：

- 下次续订���确切日期和时间
- 人类可读的相对时间（例如，“3小时后”，“2天后”）
- 了解时区（使用 CGRateS 服务器时区）

状态列：

- 来自 CRM 数据库的当前服务状态
- 活动（绿色） - 服务正常运行
- 暂停（橙色） - 服务暂时禁用
- 未知（灰色） - 在数据库中找不到服务或状态未设置

故障排除

服务在客户/服务列中显示**“N/A”**

- 原因： CGRateS 中的服务 UUID 与 CRM 数据库中的 service_uuid 不匹配
- 修复：
  - 验证服务□否存在于数据库中
  - 检查 service_uuid 格式□否匹配：`ServiceType_UUID`（例如，`FixedWireless_75628fa5`）
  - 服务可能已从 CRM 中删除，但仍在 CGRateS 中计划

费用显示**“N/A”**

- 原因： CGRateS 中的产品 ID 在 CRM 产品表中不存在
- 修复：
  - 验证产品□否存在于数据库中

- 检查 ActionPlan 中的 product_id 是否与 CRM 中的产品匹配
- 产品可能已被删除

“续订时间”未显示日期

- 原因：CGRateS 响应中未包含 NextRunTime
- 修复：
  - 检查 CGRateS ActionPlan 配置
  - 验证 ActionTiming 是否配置正确
  - 检查 CGRateS 日志以获取错误信息

“立即续订”失败

- 原因：各种配置错误
- 故障排除：
  - 检查配置状态模态框以获取错误详情
  - 验证 Ansible 剧本是否配置正确
  - 检查 OCS 连接性
  - 查看配置日志

自动续订已移除但仍显示

- 原因：缓存延迟或 CGRateS 同步问题
- 修复：
  - 点击 “刷新” 按钮以重新加载数据
  - 验证 ActionPlan 是否确实在 CGRateS 中被移除
  - 检查 CGRateS API 连接性

分页

为了提高性能，服务以页面形式显示：

> ◄ 上一页 1 [2] 3 4 5 下一页 ►
>
> 每页项目数：[10 ▼]
>
> - 10

> - 25
> - 50
> - 100

控件：

- 上一页/下一页 - 导航页面
- 页码 - 跳转到特定页面
- 每页项目数 - 调整每页显示多少服务

性能提示：使用过滤器减少总结果，而不□增加每页项目数。

服务徽章和指示器

视觉指示器帮助快速识别服务状态：

状态徽章：

自动续订指示器：

启用自动续订的服务显示：

即将到期：

在 7 天内到期的服务显示：

逾期：

有未付款余额的服务显示：

CGRateS 集成（高级）

对于与 CGRateS（计费和计费引擎）集成的服务，管理员可以直接从服务视图管理高级配置。这包括属性、过滤器和查看活��会话。

注意

CGRateS 集成功能需要 **cgrates_api_access** 权限。默认情况下，只有管理员可以访问这些功能。请参见 `rbac` 进行权限配置。

自动配置与手动管理

在正常操作中，CGRateS 属性和过滤器在初始服务配置工作流期间由 Ansible 自动配置。当创建新服务时，配置剧本：

- 在 CGRateS 中创建服务账户
- 配置属性（IMSI、MSISDN、账户标识符、速度配置文件等）
- 设置过滤器以确保正确计费
- 应用适当的计费计划

然而，在服务配置后，您可能需要修改这些配置：

- 速度配置文件更改 - 客户升级/降级带宽（MaxBitrateDL/UL）
- 策略调整 - 更改 QoS 策略或流量整形规则（PcefPolicyName）
- 电话号码更改 - 更新 MSISDN 或其他标识符
- 故障排除 - 修复配置错误或测试不同设置
- 特殊配置 - 应用不属于标准配置的自定义属性

手动管理界面允许管理员直接进行这些更改，而无需重新运行整个配置工作流。这对于以下情况特别有用：

- 快速更改 - 修改单个属性而无需等待配置
- 测试 - 尝试不同的配置
- 客户支持 - 在支持电话中即时解决问题
- 自定���配置 - 应用不在模板覆盖范围内的服务特定设置

警告

对 CGRateS 配置的手动更改绕过标准配置工作流。确保您了解更改的影响，因为不正确的配置可能会影响计费和服务功能。所有更改都会记录到客户活动记录中，以供审计。

访问 **CGRateS** 功能

在查看或编辑已在 CGRateS 中配置的服务时，服务表单底部会出现三个可折叠部分：

- **CGRateS** 属性 - 配置服务特定属性
- **CGRateS** 过滤器 - 定义服务的过滤规则
- 活动会话 - 查看实时活动会话

每个部分默认□折叠的，以保持界面整洁。点击部分标题以展开并查看/编辑配置。

CGRateS □□

□□□

属性字段：

- 路径 - 要修改的字段（例如，`*req.Account`、`*req.IMSI`）
- 类型 - 值的设置方式：

    - `*constant` - 设置固定值
    - `*variable` - 使用 RSRParser 从事件字段捕获值
    - `*composed` - 附加值而不覆盖
    - `*usage_difference` - 计算两个字段之间的持续时间
    - `*sum` - 求多个值的和
    - `*value_exponent` - 计算字段的指数
- 规则 - 要应用的值（每个属性可以有多个规则）

示例属性配置：

```
{
  "Path": "*req.Account",
  "Type": "*constant",
  "Value": [{"Rules": "Mobile_SIM_474a380a"}]
}
```

常见用例：

- 为计费设置账户标识符
- 将 IMSI/MSISDN 映射到服务
- 配置带宽限制（MaxBitrateDL/UL）
- 设置策略名称（PcefPolicyName）
- 转换或丰富计费事件

活动日志记录：

所有属性修改都会记录到客户的活动记录中，详细记录所做的更改。

CGRateS 过滤器

过滤器定义匹配规则，以确定在计费过程中何时应用此服务的配置。

过滤器 **ID** 格式：`FLTR_ACCOUNT_{service_uuid}`

示例：对于 UUID 为 `Mobile_SIM_c2880638` 的服务，过滤器 ID 为 `FLTR_ACCOUNT_Mobile_SIM_c2880638`

管理过滤器：

1. 打开服务编辑视图
2. 展开 **“CGRateS 过滤器”** 部分
3. 点击 **“编辑过滤器”** 按钮
4. 添加/修改/删除过滤规则
5. 点击 **“保存过滤器”**

过滤规则字段：

- 元素 - 要匹配的字段（例如，`~*req.Account`、`~*req.Destination`）
- 类型 - 匹配类型：
  - `*string` - 精确字符串匹配
  - `*prefix` - 以指定值开头
  - `*suffix` - 以指定值结尾
  - `*empty` - 字段为空
  - `*exists` - 字段存在
  - `*notexists` - 字段不存在
  - `*timings` - 匹配时间/日期模式
  - `*destinations` - 匹配目的地模式
  - `*rsr` - RSR 字段匹配
  - `*gt` / `*gte` / `*lt` / `*lte` - 数值比较
- 值 - 要匹配的值（每个规则可以有多个值）

示例过滤器配置：

```
{
  "Element": "~*req.Account",
  "Type": "*string",
  "Values": ["Mobile_SIM_474a380a"]
}
```

常见用例：

- 确保属性仅适用于特定账户
- 按目的地过滤（国内与国际）

- 基于时间的过滤（高峰与非高峰）
- 按服务类型或类别过滤

活动日志记录：

所有过滤器修改都会记录到客户的活动记录。

活动会话

查看此服务的实时活动会话。这显示正在进行的通话、数据会话或其他当前进行的计费事件。

查看活动会话：

1. 打开服务编辑视图
2. 展开“活动会话”部分
3. 查看活动会话列表
4. 点击任何会话的“查看详情”以查看完整会话数据
5. 点击“刷新”以重新加载会话列表

显示的会话信息：

- 设置时间 - 会话开始时间
- 使用情况 - 当前会话持续时间（以秒为单位）
- 目的地 - 被拨打的号码或目的地

会话详情模态框：

点击“查看详情”会打开一个模态框，显示：

- 基本信息：
  - CGRID（会话 ID）
  - 账户（服务 UUID）
  - 设置时间
  - 当前使用情况/持续时间
  - 目的地
  - 类别
- 完整会话数据：
  - 会话的完整 JSON 表示
  - 所有 CGRateS 会话字段
  - 实时会话状态
  - 可滚动的 JSON 查看器以进行检查

用例：

- 监控活动通话或数据会话
- 故障排除计费问题

- 验证会话□否正确计费
- 检查会话属性和值
- 审计活动服务使用情况

刷新率：

会话在您展开部分时按需获取。点击“刷新”以获取最新的会话数据。

注意

仅显示与此服务账户（服务 UUID）匹配的会话。过滤器 `*string:~*req.Account:{service_uuid}` 会自动应用。

CGRateS API 代理

所有 CGRateS 操作（属性、过滤器、会话）使用 OmniCRM API 代理端点：

端点： `POST /crm/ocs/proxy`

必需字段：

- `method` - CGRateS API 方法（例如，`APIerSv1.GetAttributeProfile`）
- `params` - 方法的参数数组
- `customer_id` - 客户 ID（用于活动日志记录）
- `service_id` - 服务 ID（用于活动日志记录）

可选字段：

- `tenant` - CGRateS 租户（默认为配置值）

示例请求：

```
{
  "method": "APIerSv1.GetAttributeProfile",
  "params": [{"ID": "ATTR_ACCOUNT_Mobile_SIM_c2880638"}],
  "customer_id": 123,
  "service_id": 456
}
```

租户配置：

租户会自动从 OmniCRM 配置文件（`crm_config.yaml`）中的 `ocs.ocsTenant` 设置。这确保所有 CGRateS 操作都使用正确的租户，而无需在前端硬编码值。

权限要求：

需要 `cgrates_api_access` 权限。默认情况下，此权限授予 `admin` 角色。

活动日���记录：

所有非 GET 的 CGRateS API 操作都会自动记录到客户的活动记录中，包括：

- 调用的 API 方法
- 使用的租户
- 发送的完整参数
- 执行操作的服务 ID
- 执行操作的用户
- 时间戳

这创建了 CGRateS 配置更改的完整审计跟踪。

故障排除 **CGRateS** 集成

访问 **CGRateS** 功能时出现“权限被拒绝”

- 原因：用户缺少 `cgrates_api_access` 权限
- 修复：将权限授予用户的角色（通常仅限管理员的功能）

属性或过滤器未加载

- 原因： CGRateS 连接问题或配置文件不存在
- 修复：
  - 检查配置中的 CGRateS 服务器连接性
  - 验证租户配置否正确
  - 检查浏览器控制台否有 API 错误
  - 配置文件可能尚不存在（将显示空表单）

更改未保存

- 原因：验证错误或 CGRateS API 错误
- 修复：
  - 检查必填字段（路径、类型、元素）
  - 验证 JSON 格式□否正确
  - 检查活动日志以获取错误详情
  - 查看 CGRateS 日志

没有显示活动会话

- 原因：此服务当前没有活动会话
- 修复：
  - 如果服务未在使用中，这□正常的
  - 尝试在发起会话（通话、数据等）后刷新
  - 验证服务 UUID □否与 CGRateS 中的账户匹配

会话详情未实时更新

- 原因：会话数据□按需获取的，而不□实时的
- 修复：点击“刷新”按钮以获取最新的会话数据

活动日志未显示 **CGRateS** 更改

- 原因：仅记录非 GET 操作（读取不记录）
- 修复：这□设计使然 - 仅写入/修改会创建活动条目

常见工作流程

工作流程 **1**：查找客户的服务

1. 点击 服务类型过滤器（可选）
2. 点击 客户过滤器
3. 输入客户名称
4. 从下拉列表中选择客户
5. 查看客户的服务

工作流程 2：识别即将到期的服务

1. 点击“活动”标签
2. 按“结束日期”排序（升序）
3. 即将到期的服务优先显示
4. 联系客户进行续订

工作流程 3：清理测试服务

1. 在搜索框中搜索“测试”
2. 查看结果以确认它们□测试数据
3. 选择所有测试服务
4. 点击“删除所选”
5. 确认删除

工作流程 4：暂停未付款客户

1. 导航到客户账户
2. 查看服务标签
3. 选择所有活动服务
4. 将状态更改为“暂停”
5. 保存更改

工作流程 5：查看移动服务使用情况

1. 按 服务类型：移动 过滤
2. 点击服务名称以打开详���
3. 点击“使用情况”标签
4. 查看数据/语音/SMS 消耗
5. 确定重度用户或超额使用

故障排除

服务未出现在列表中

- 原因：状态过滤器隐藏结果
- 修复：点击“所有结果”标签以显示所有状态

无法通过搜索找到服务

- 原因：搜索词与存储数据不匹配
- 修复：
  - 尝试部分搜索（例如，“0770”而不□完整号码）
  - 使用客户过滤器
  - 检查□否有拼写错误

批量删除按钮被禁用

- 原因：未选择任何服务
- 修复：检查您想要删除的服务旁边的复选框

排序无效

- 原因：列不可排序
- 修复：只有带有 ▲▼ 图标的列□可排序的

页面加载缓慢

- 原因：要显示的服务过多
- 修复：
  - 应用过滤器以减少结果集
  - 减少每页项目数
  - 使用搜索缩小结果

相关文档

- `csa_add_service` - 添加新服务
- `csa_modify` - 修改服务和添加附加功能
- `concepts_products_and_services` - 产品和服务概念
- `basics_customers` - 客户管理

服务使用和余额跟踪

服务使用系统提供实时监控客户的数据、语音、短信和货币余额消耗。此功能与 OCS（在线计费系统）集成，以向客户（通过 `Self-Care Portal <self_care_portal>`）和员工显示当前使用情况、剩余配额和余额到期信息。

概述

使用跟踪使得：

- 实时余额显示 - 查看当前使用情况和剩余配额
- 多种余额类型 - 同时跟踪数据、语音、短信和货币余额
- 到期监控 - 查看余额何时到期
- 余额细分 - 各个余额桶的详细视图
- 自动刷新 - 每 3 秒自动更新使用情况

访问服务使用情况

从服务列表：

1. 导航到 服务 → 服务列表
2. 点击某个服务旁边的 □作菜单（⋮）
3. 选择 "使用情况"

从服务详情：

1. 打开服务的详细页面
2. 点击 "使用情况" 标签

从客户页面：

1. 打开客户概览
2. 导航到 服务 标签
3. 点击任何服务旁边的 "查看使用情况"

使用情况模态框或页面打开，显示实时□□耗数据。

使用情况显示

使用界面显示每种余额类型的摘要卡和详细进度条。

摘要卡

顶部行显示每种余额类型的快速查看卡：

卡片信息：

- □□□□ - □□□□

- 充值 - 在周期中购买的额外数据
- 加密狗服务 - 热点设备的预付数据

多个桶：

服务通常有多个数据余额：

- 每月配额（每月到期）
- 奖励数据（在活动期间到期）
- 充值数据（到期时间更短，优先消耗）

语音余额

跟踪电话通话分钟数。

单位：分钟 (min)

显示格式：

> 进度: 125 min / 500 min (25%)

通话时长跟踪：

- 来电（如果收费）
- 外拨电话
- 国际电话（如果适用，单独桶）
- 高收费号码

计算：

语音使用量通过内部通话时长（以纳秒为单位）计算，转换为分钟以供显示。

短信余额

跟踪短信使用情况。

单位：短信 (msgs)

显示格式：

> 进度: 45 / 250 (18%)

消息类型:

- 标准短信（160 个字符）
- 长短信（多个段落）
- 彩信（如果单独跟踪）

货币余额

跟踪预付信用或账户余额。

单位：货币 (£, $, €, 等)

显示格式:

> 进度: £15.50 / £20.00 (77%)

使用情况:

- 预付账户使用货币余额支付使用费用
- 客户使用服务时余额减少
- 可以通过支付或代金券充值
- 如果在有效期内未使用，可能会过期

???细余额细分

点击任何卡片上的 "更多信息" 或点击进度条以展开详细细分。

扩展视图:

> □ 每月配额 20GB
> 剩余: 12.5 GB 已使用: 7.5 GB 到期: 2025年1月25日（15天） 权重: 10
>
> □ 奖励数据 5GB
> 剩余: 5.0 GB 已使用: 0 GB 到期: 2025年1月31日（21天） 权重: 20

> 充值数据 3GB
> 剩余: 0 GB 已使用: 3.0 GB 到期: 2025年1月18日（已过期） 权重: 30
>
> 总剩余: 17.5 GB

余额桶字段:

- **ID/**名称 - 余额桶的标识符
- 剩余 - 此特定桶中剩余的金额
- 已使用 - 从此桶中消耗的金额
- 到期日期 - 此余额到期的时间
- 权重 - 优先顺序（权重高的优先消耗）

权重系统

余额具有权重值，决定消耗顺序:

- 权重高 = 优先消耗
- 权重低 = 最后消耗

示例权重:

- 充值数据: 权重 30（优先消耗，过期时间短）
- 奖励数据: 权重 20（第二消耗）
- 每月配额: 权重 10（最后消耗，过期时间长）

这确保了即将到期的余额在更持久的余额之前使用。

有关余额消耗规则、权重策略以及 CGRateS 如何优先处理余额的完整详细信息，请参见 CGRateS Actions and Topup Behaviors。

实时更新

使用数据每 **3** 秒 自动刷新一次。

更新内容:

- 当前余额金额

- 使用进度条
- 到期计时器
- 各个桶的详细信息

用户体验：

- 无需重新加载页面
- 平滑更新，无闪烁
- 刷新期间加载覆盖
- 状态徽章显示当前服务状态

用例：

- 在通话期间监控客户使用情况
- 观察余额实时减少，客户使用服务时
- 购买后立即验证充值

不同服务类型中的使用情况

移动服务

显示所有四种余额类型：

- 数据（GB）
- 语音（分钟）
- 短信（消息）
- 货币（货币）

示例：

> 数据：12.5GB 剩余 语音：125 min 剩余 短信：45 msgs 剩余 货币：£15.50 剩余

固定无线/互联网

通常只显示：

- 数据（GB 或 TB）
- 货币（如果□预付）

示例：

> 数据：450GB / 500GB 剩余 货币：£45.00（预付信用）

热点/加密狗服务

显示加密狗特定的数据跟踪：

- 数据（消耗与预付）
- 货币（预付余额）

显示模式：

当 `dongle=true` 时，组件隐藏语音和短信，仅显示相关的数据和货币余额。

故障排除

使用情况显示为 **0 / 0**

- 原因：服务未与 OCS 或 CGRateS 集成
- 修复：
  - 验证服务□否在 OCS 中配置
  - 检查 OCS API 连接
  - 查看服务 UUID 映射

使用情况未更新

- 原因：轮询停止或 OCS 无法访问
- 修复：
  - 刷新页面
  - 检查浏览器控制台□否有错误
  - 验证 OCS API □否在线

余额显示不正确

- 原因：OCS 数据不匹配或缓存问题
- 修复：
  - 强制 OCS 余额刷新
  - 检查待处理交易
  - 验证 OCS 配置

到期日期缺失

- 原因：余额没有设置到期
- 修复：
  - 某些余额设置为永不过期（无限有效期）
  - 检查 OCS 中的余额配置

多个余额令人困惑

- 原因：添加了多个充值或奖励数据
- 修复：
  - 使用详细细分视图查看所有桶
  - 按权重排序以查看消耗顺序
  - 查看各个到期日期

与 OCS/CGRateS 的集成

使用数据来自**OCS**（在线计费系统），通常 CGRateS。有关余额如何工作的全面信息，包括余额类型、消耗规则、滚存行为和操作配置，请参见 CGRateS Actions and Topup Behaviors。

数据流：

1. 用户打开使用视图
2. OmniCRM 调用 `GET /crm/service/{service_id}`
3. API 通过服务 UUID 查询 OCS
4. OCS 返回余额映射：

```
{
  "BalanceMap": {
    "*data": [
      {
        "ID": "monthly_data_20GB",
        "Value": 13421772800,
        "ExpiryTime": "2025-01-25T23:59:59Z",
        "Weight": 10
      }
    ],
    "*voice": [
      {
        "ID": "monthly_voice_500min",
        "Value": 7500000000000,
        "ExpiryTime": "2025-01-25T23:59:59Z",
        "Weight": 10
      }
    ],
    "*sms": [
      {
        "ID": "monthly_sms_250",
        "Value": 250,
        "ExpiryTime": "2025-01-25T23:59:59Z",
        "Weight": 10
      }
    ],
    "*monetary": [
      {
        "ID": "prepaid_credit",
        "Value": 1550,
        "ExpiryTime": "2025-02-25T23:59:59Z",
        "Weight": 10
      }
    ]
  }
}
```


5. UI 将值转换为显示单位（字节 → GB，纳秒 → 分钟）
6. 渲染进度条和卡片
7. 每 3 秒继续轮询

OCS 余额类型映射：

OCS 返回带有类型前缀的余额数据，这些前缀映射到 UI 显示：

- `*data` → 数据卡（互联网使用）
- `*voice` → 语音卡（通话分钟）
- `*sms` → 短信卡（文本消息）
- `*monetary` → 货币卡（预付信用）

每种余额类型可以有多个桶（例如，每月配额 + 奖励数据 + 充值数据），所有这些都在详细分视图中显示。

余额值转换：

- 数据：字节 → GB（除以 1024^3）
- 语音：纳秒 → 分钟（除以 60×10^9）
- 短信：计数（无转换）
- 货币：美分 → 货币（除以 100）

自动续订和作计划

启用自动续订的服务在 OCS 中有作计划。有关操作如何工作、余额操作和配置操作计划的详细信息，请参见 CGRateS Actions and Topup Behaviors。

什么是作计划?

操作计划在 CGRateS 中按计划执行的任务，自动在特定时间执行：

- 向账户添加余额（自动充值）
- 续订每月配额
- 应用定期收费
- 过期旧余额

自动续订的工作原理：

1. 服务配置：
    - 当服务创建时，`auto_renew = true`
    - 配置剧本在 OCS 中创建操作计划
    - 操作计划配置为每月运行（或按计费周期）

2. □作计划配置：

   操作计划包含：

   - 账户 **ID** - 服务 UUID
   - □作 - 要做什么（添加数据、语音、短信、货币余额）
   - 计划 - 何时执行（例如，每月的第 1 天 00:00 UTC）
   - 金额 - 添加多少余额

3. 自动执行：

   - OCS 在计划时间执行操作计划
   - 向账户添加余额��例如，20GB 数据，500 分钟语音）
   - 为新余额设置到期日期（例如，30 天）
   - 客户通过文件中的支付方式收费

4. 查看□作计划：

   - 导航到 OCS 视图中的服务详情
   - 列出操作计划及下次执行时间
   - 显示：计划名称、下次运行日期、操作细节

示例□作计划：

```
{
  "ActionPlanId":
"ProductID_MonthlyPlan__ProductName_20GB_Mobile__ActionPlan_Monthly_F
  "NextExecTime": "2025-02-01T00:00:00+00:00",
  "ActionName_hr": "每月续订",
  "PlanName": "20GB 移动",
  "ActionFrequency_hr": "每月计划",
  "custom_NextExecTime_hr": "在 22 天内"
}
```

管理自动续订：

- 启用 - 在服务创建或修改时设置
- 禁用 - 从 OCS 中删除操作计划（服务保留现有余额，但不会自动续订）
- 修改 - 通过服务修改更改续订金额或频率

手动续订：

如果禁用自动续订，客户必须手动：

- 在余额到期前充值
- 或在余额耗尽时暂停服务

在 **UI** 中查看：

服务标签显示自动续订状态：

> 下次续订：2025年2月1日（在22天内）续订金额：£15.00

最佳实践

对于支持人员：

- 在回答“我的服务为什么慢？”的电话之前检查使��情况
- 在充值后验证余额以确认成功
- 使用详细细分查看过期桶
- 监控高使用客户以防止超支

对于客户（自助服务）：

- 定期检查使用情况以避免用尽
- 在余额到期前充值
- 理解权重系统以了解哪个余额优先消耗
- 如果使用情况似乎不正确，请联系支持

对于管理员：

- 配置适当的余额到期
- 设置权重值以优先考虑即将到期的余额
- 监控 OCS 连接以确保准确报告
- 查看余额配置与产品提供匹配

□□□□

- CGRateS Actions and Topup Behaviors - □□□□

客户服务

支持和故障排除的用户模拟

模拟允许授权的工作人员暂时以其他用户的身份登录，以排除问题、验证配置或准确查看用户的体验。此功能对于客户支持至关重要，但需要适当的权限，并且会进行全面审计。

在模拟客户时，工作人员可以访问 `Self-Care Portal <self_care_portal>`，与客户看到的完全相同，从而实现准确的故障排除和支持。

另见：`RBAC <rbac>` 以获取权限配置， `Customers <basics_customers>` 以获取客户管理， `Self-Care Portal <self_care_portal>` 以获取客户门户功能。

目的

用户模拟提供：

1. 故障排除 — 精确查看客户所见以诊断问题
2. 验证 — 确认服务配置和权限正常工作
3. 培训 — 从客户的角度演示功能
4. 支持 — 帮助客户在不需要共享屏幕的情况下导航系统
5. 审计跟踪 — 所有模拟会话都会记录以确保安全和合规

所需权限

要模拟用户，您必须拥有以下权限之一：

- `can_impersonate` — 专为支持人员提供的模拟权限
- `admin` — 完全管理访问权限（包括模拟权利）

没有这些权限的用户无法访问模拟功能。

如何模拟用户

通过 **Web UI**:

1. 导航到客户 — 在 CRM 中找到客户
2. 选择联系人 — 查看客户的联系人列表
3. 点击“以用户身份登录” — 每个有用户帐户的联系人旁边会出现按钮
4. 确认模拟 — 系统可能会提示确认
5. 会话开始 — 您现在以该用户身份登录

通过 **API**:

开始模拟会话:

端点: `POST /auth/impersonate`

所需权限: `can_impersonate` 或 `admin`

请求:

```
{
  "user_id": 42
}
```

响应：

```
{
  "success": true,
  "impersonating_user_id": 1,
  "target_user_id": 42,
  "impersonation_start": "2025-01-04T15:30:00Z",
  "access_token": "new_token_for_impersonated_user",
  "refresh_token": "new_refresh_token"
}
```

返回的令牌用于模拟用户的会话。

模拟期间发生的事情

当您模拟用户时：

- 完整上下文 — 您看到的系统与目标用户看到的完全相同：
  - 他们的仪表板和导航
  - 他们的客户数据（如果□客户用户）
  - 他们的权限和访问控制
  - 他们的服务、发票和使用情况
- 会话跟踪 — 系统跟踪两个身份：
  - `impersonating_user_id` — 您的真实用户 ID
  - `target_user_id` — 您正在模拟的用户
  - `impersonation_start` — 模拟开始的时间
- 指示器 — UI 显示一个横幅，表明您正在模拟某人：
  - "您当前以 [用户名] 登录"
  - "点击此处停止模拟"
- 审计日志 — 所有操作都记录了两个用户 ID：
  - 操作显示为由目标用户执行
  - 审计日志记录实际执行操作的用户（模拟用户）
  - 在 `ImpersonationLog` 表中维护完整的审计跟踪

停止模拟

通过 **Web UI**:

1. 点击横幅 — 点击页面顶部的模拟横幅
2. 或导航 — 转到用户菜单并选择“停止模拟”
3. 确认 — 会话结束，您返回到自己的帐户

通过 **API**:

端点: `POST /auth/stop_impersonation`

请求：不需要请求体（经过身份验证的请求）

响应:

```
{
  "success": true,
  "impersonation_end": "2025-01-04T15:45:00Z",
  "duration_seconds": 900,
  "access_token": "your_original_token",
  "refresh_token": "your_original_refresh_token"
}
```

您的原始会话被恢复。

模拟审计日志

所有模拟会话都记录在 `ImpersonationLog` 表中，包含:

- **impersonating_user_id** — 执行模拟的工作人员
- **target_user_id** — 被模拟的客户或用户
- **impersonation_start** — 开始时间戳
- **impersonation_end** — 结束时间戳（会话停止时）
- **impersonation_duration** — 持续时间（以秒为单位）

这为所有模拟会话提供了完整的问责制，并允许:

- 安全审计 — 审查谁在何时模拟了谁
- 合规报告 — 演示对提升访问权限的正确使用
- 调查 — 跟踪模拟会话期间的操作
- 监控 — 识别异常的模拟模式

查看模拟日志：

管理员可以通过 API 查询模拟日志：

`GET /auth/impersonation_logs?user_id={user_id}&start_date={date}&end_date={date}`

按以下条件过滤：

- 模拟用户（执行模拟的人）
- 目标用户（被模拟的人）
- 日期范围
- 持续时间

最佳实践

1. 最小化持续时间 — 仅在解决问题所需的时间内进行模拟
2. 记录目的 — 在客户活动日志中注明为什么需要模拟
3. 通知客户 — 让客户知道您可能需要查看他们的帐户（隐私政策）
4. 验证身份 — 在通过他们的帐户进行模拟之前确认用户身份
5. 审查日志 — 定期审计模拟日志以查找异常模式
6. 限制权限 — 仅向需要的支持人员授予 `can_impersonate`
7. 培训 — 确保工作人员理解责任和审计影响

安全考虑

- 完全访问 — 模拟授予对目标用户帐户的完全访问
- 无需密码 — 模拟绕过身份验证（仅基于权限）
- 记录的□作 — 模拟期间的所有操作都归因于目标用户在应用日志中（但审计日志显示真实执行者）

- 会话隔离 — 模拟创建一个新会话；不会影响目标用户的活动会话
- 时间限制 — 模拟会话应有限制时间（可配置）
- **MFA** 绕过 — 模拟绕过 2FA 要求（使用模拟者的身份验证）

限制

- 无法模拟管理员 — 根据配置，可能无法模拟其他管理员
- 相同权限 — 您获得目标用户的权限，而不□两个用户权限的并集
- 会话限制 — 每个工作人员一次只能有一个模拟会话
- 审计要求 — 无法禁用或隐藏模拟日志

故障排除常见问题

问题：“以用户身份登录”按钮未出现

- 解决方案：验证您□否拥有 `can_impersonate` 或 `admin` 权限
- 解决方案：确认联系人□否有关联的用户帐户

问题：模拟失败，出现权限错误

- 解决方案：检查目标用户□否为管理员（可能受到限制）
- 解决方案：验证您的模拟权限□否有效

问题：无法停止模拟

- 解决方案：使用 API 端点停止模拟：`POST /auth/stop_impersonation`
- 解决方案：清除浏览器 cookies 并使用您的凭据重新登录

问题：□作未正确记录

- 解决方案：验证模拟会话□否处于活动状态（检查横幅）
- 解决方案：查看审计日志 - 操作记录了两个用户 ID

定义产品和 CGRateS 配置

本指南解释了如何在 OmniCRM 中定义产品及其相关的 CGRateS 计费配置。

产品及其 CGRateS 操作通常在 Python 脚本中一起定义，这些脚本配置了 CRM 和 OCS（在线计费系统）。

概述

定义完整产品涉及在 两个系统 中进行配置：

1. **CGRateS/OCS** - 定义计费行为（操作、费率、目的地）
2. **OmniCRM** - 定义产品元数据（价格、名称、特性）

这些配置故意松散关联的，允许您通过定义的 Ansible Plays 与 CGrateS 进行交互。

重要: CGRateS 能力

CGRateS 是一个极其强大和灵活的计费引擎。本指南展示了 一种常见的配置模式 用于 OmniCRM 产品，但 CGRateS 能做更多事情：

- 高级计费场景：时段费率、客户等级、基于通话质量的路由、突发计费
- 复杂的漫游逻辑：漫游时 MO 与 MT 通话/SMS 的不同费率、网络特定定价
- 遗留协议支持：用于 2G/3G 漫游的 CAMEL（MO-MT SMS、USSD、用于 LTE/5G 的 Diameter
- 复杂的计费：基于预留的计费、信用控制、多层次回退
- 动态路由：基于成本的路由、供应商选择、LCR（最低成本路由）

这只是 **CRM** 指南 - 它专注于与 OmniCRM 的配置系统良好配合的简单产品定义模式。有关高级 CGRateS 配置，请查阅 CGRateS 文档 或查看 CGRateS 操作和充值行为 以获取余额方法和计费策略。

完整产品定义工作流程

步骤 1：对 CRM API 进行身份验证

```
import requests
import json

crm_url = 'https://crm.example.com/'
session = requests.Session()

headers = {
    "Content-Type": "application/json"
}

# 取身份验证令牌
response = session.post(crm_url + '/crm/auth/login', json={
    "email": "admin@example.com",
    "password": "your_password"
}, headers=headers)

assert response.status_code == 200
headers['Authorization'] = 'Bearer ' + response.json()['token']
print("对 CRM 身份验证成功")
```

步骤 2：定义库存模板（可选）

库存模板定义了在配置过程中可以分配的物理项目类型（SIM 卡、路由器等）。

```
inventory_list_new = []

# 定义 SIM 卡库存模板
inventory_list_new.append({
    "item": "SIM Card",
    "itemtext1_label": "ICCID",
    "itemtext2_label": "IMSI",
    "wholesale_cost": 0.2,
    "retail_cost": 1,
    "allow_dropdown_staff": True,
    "allow_dropdown_customer": True,
    "icon": "fa-solid fa-sim-card"
})

# 定义手机号码库存模板
inventory_list_new.append({
    "item": "Mobile Number",
    "itemtext1_label": "E.164 Mobile Number",
    "itemtext2_label": "Type",
    "wholesale_cost": 0.2,
    "retail_cost": 3,
    "icon": "fa-solid fa-phone"
})

# 检查库存模板是否存在，创建或更新
for inventory_template in inventory_list_new:
    # 取现有模板
    inventory_list_existing = session.get(
        crm_url + '/crm/inventory/item_template/',
        headers=headers
    )

    # 检查模板是否已存在
    if inventory_template['item'] in [x['item'] for x in
inventory_list_existing.json()]:
        # 更新现有模板
        inventory_template_id = [
            x['inventory_template_id']
            for x in inventory_list_existing.json()
            if x['item'] == inventory_template['item']
        ][0]

        response = session.patch(
```

```
            crm_url + '/crm/inventory/item_template/' +
str(inventory_template_id),
            json=inventory_template,
            headers=headers
        )
        assert response.status_code == 200
        print(f"更新库存模板: {inventory_template['item']}")
    else:
        # 创建新模板
        response = session.put(
            crm_url + '/crm/inventory/item_template',
            json=inventory_template,
            headers=headers
        )
        assert response.status_code == 200
        print(f"创建库存模板: {inventory_template['item']}")
```

库存模板字段:

- **item**（必填） - 模板名称
- **itemtext1_label** - 第一个自定义字段的标签
- **itemtext2_label** - 第二个自定义字段的标签
- **wholesale_cost** - 您的成本
- **retail_cost** - 面向客户的成本
- **allow_dropdown_staff** - 在员工下拉菜单中显示
- **allow_dropdown_customer** - 在客户下拉菜单中显示
- **icon** - Font Awesome 图标类

步骤 3: 连接到 CGRateS

```
import cgrateshttpapi
import time

OCS_Obj = cgrateshttpapi.CGRateS("ocs.example.com", "2080")

tenant = "your_tenant_name"
tpid = str(tenant) + "_" + str(int(time.time()))
```

步骤 4：在 CGRateS 中定义目的地

目的地 定义了余额可以使用的地方。它们必须在创建引用它们的操作之前定义。

有两种类型：

1. 地理目的地 - 语音/SMS 到特定地点的号码缀（例如，`Dest_AU_Mobile`，`Dest_International_UK`）
2. **PLMN** 目的地 - 数据使用和漫游的网络代码（例如，`Dest_PLMN_OnNet`，`Dest_PLMN_US_Verizon`）

关键规则：

- 语音/**SMS** 余额 → 使用地理目的地（被呼叫的号码）
- 数据余额 → 使用 PLMN 目的地（客户连接的网络）

有关完整目的地定义示例，包括：

- 地理目的地（移动、固定电话、免费电话、国际）
- PLMN 目的地（本网、漫游、区域）
- PLMN 格式规则（`mccXXX.mncYYY`）
- 何时使用每种方法

请参见：CGRateS 操作 - 定义目的地

步骤 5：定义 CGRateS 作

作 将余额添加到客户帐户的机制。在将 CRM 产品链接到操作之前，必须在 CGRateS 中定义该操作。

关键：操作在初始系统配置期间定义（而不在配置期间）。它们指定要添加的余额、到期时间、回滚行为等。

请参见 CGRateS 操作和充值行为 以获取有关以下内容的完整文档：

- 如何使用 Python 定义操作（`OCS_Obj.SendData`）
- 理解 `*topup` 与 `*topup_reset`（回滚行为）
- 单位计算（数据的字节、语音的纳秒、SMS 的计数）
- 字段参考（BalanceId、BalanceType、DestinationIDs、Units、ExpiryTime、Weight）
- 完整的工作示例（多余额计划、数据附加、语音附加）
- 余额消耗规则和优先级

快速回滚参考：

- `*topup` - 添加到现有余额（启用回滚）。用于：预付费附加、数据包
- `*topup_reset` - 替换现有余额（无回滚）。用于：具有固定配额的月度计划

步骤 6：在 CRM 中创建产品

现在，CGRateS 中的操作已定义，在 CRM 中创建相应的产品。

示例：独立 SIM 服务

```
product_list_new = []

product_list_new.append({
    "category": "standalone",
    "provisioning_play": "play_psim_only",
    "relies_on_list": "",
    "contract_days": 0,
    "retail_cost": 0,
    "retail_setup_cost": 0,
    "product_slug": "Mobile-SIM",
    "product_name": "仅限移动 SIM",
    "comment": "用于移动电话的物理 SIM 卡",
    "provisioning_json_vars": "{\"iccid\" : \"\", \"msisdn\" :
\"\"}",
    "terms": "必须在 6 个月内激活。如果服务在 12 个月内未使用，则所有信用将丢失。",
    "service_type": "mobile",
    "residential": True,
    "business": True,
    "enabled": True,
    "inventory_items_list": "['SIM Card', 'Mobile Number']",
    "icon": "fa-solid fa-sim-card",
    "features_list": "['澳大利亚电话号码 (61xxx)', '最快的速度', '最佳覆盖']",
    "wholesale_cost": 3,
    "wholesale_setup_cost": 1,
    "customer_can_purchase": True
})
```

示例：月度计划附加

```
product_list_new.append({
    "category": "addon",
    "provisioning_play": "play_topup_charge_then_action",
    "relies_on_list": "",
    "contract_days": 30,
    "retail_cost": 59.00,
    "retail_setup_cost": 0,
    "product_slug": "au-premium-plan-1",  # 链接到 Action_au-premium-
plan-1
    "product_name": "澳大利亚高级计划 1",
    "comment": "在澳大利亚 100GB 数据，3000 分钟，3000 SMS，6GB 漫游数据",
    "provisioning_json_vars": "",
    "terms": "后付费计划。每 30 天自动续订。",
    "service_type": "mobile",
    "residential": True,
    "business": True,
    "enabled": True,
    "icon": "fa-solid fa-mobile",
    "features_list": "['在澳大利亚 100GB 数据', '在澳大利亚 3000 分钟', '在澳大利亚
3000 SMS', '6GB 漫游数据']",
    "wholesale_cost": 45.00,
    "wholesale_setup_cost": 0,
    "auto_renew": "True",
    "customer_can_purchase": True
})
```

示例：一次性充值附加

```
product_list_new.append({
    "category": "addon",
    "provisioning_play": "play_topup_charge_then_action",
    "relies_on_list": "",
    "contract_days": 0,
    "retail_cost": 30,
    "retail_setup_cost": 0,
    "product_slug": "au-data-addon-20gb",  # 链接到 Action_au-data-addon-20gb
    "product_name": "澳大利亚数据附加 20GB",
    "comment": "在澳大利亚 20GB 数据",
    "provisioning_json_vars": "",
    "terms": "预付费充值。立即收费。",
    "service_type": "mobile",
    "residential": True,
    "business": True,
    "enabled": True,
    "icon": "fa-solid fa-mobile",
    "features_list": "['在澳大利亚 20GB 数据']",
    "wholesale_cost": 22,
    "wholesale_setup_cost": 0,
    "auto_renew": "False",
    "customer_can_purchase": True
})
```

产品字段描述:

- **category**（必填） - 产品类别:
  - `standalone` - 完整服务（创建新服务记录）
  - `addon` - 添加到现有服务
  - `bundle` - 多个服务的包
- **provisioning_play**（必填） - 要执行的 Ansible playbook（例如，`play_topup_charge_then_action`，`play_simple_service`）
- **relies_on_list** - 此产品依赖的 product_slugs 的逗号分隔列表
- **contract_days** - 合同持续天数（0 = 无合同，30 = 每月）
- **retail_cost**（必填） - 面向客户的每月/定期成本
- **retail_setup_cost** - 向客户收取的一次性设置费用
- **product_slug**（必填） - 链接到 **CGRateS** 作的唯一标识符（约定：`Action_{product_slug}`）
- **product_name**（必填） - 显示名称

- **comment** - 简短描述
- **provisioning_json_vars** - 传递给 playbook 的变量的 JSON 字符串
- **terms** - 条款和条件文本
- **service_type** - 服务分类（移动、加密狗、固定等）
- **residential** - 面向住宅客户
- **business** - 面向商业客户
- **enabled** - 产品处于活动状态并可购买
- **inventory_items_list** - 所需库存项目的 Python 列表字符串
- **icon** - Font Awesome 图标类
- **features_list** - 特性要点的 Python 列表字符串
- **wholesale_cost** - 您的定期成本
- **wholesale_setup_cost** - 您的一次性成本
- **auto_renew** - 自动续订的 "True" 或 "False" 字符串
- **customer_can_purchase** - 客户可以通过自助服务门户购买

另请参见：产品生命周期

步骤 7：通过 API 创建或更新产品

```
# □取现有产品
product_list_existing = session.get(crm_url + '/crm/product',
headers=headers)
existing_products = product_list_existing.json()

# ���理每个产品
for product in product_list_new:
    print(f"处理产品: {product['product_slug']}")

    # 检查产品是否已存在
    if product['product_slug'] in [x['product_slug'] for x in
existing_products]:
        print(f"产品已存在: {product['product_slug']}")

        # □取现有产品的 product_id
        product_id = [
            x['product_id']
            for x in existing_products
            if x['product_slug'] == product['product_slug']
        ][0]

        product['product_id'] = product_id

        # 更新现有产品
        response = session.patch(
            crm_url + '/crm/product/' + str(product_id),
            json=product,
            headers=headers
        )
        print(f"状态: {response.status_code}")
        assert response.status_code == 200
        print(f"更新产品: {product['product_slug']}")
    else:
        # 创建新产品
        print(f"创建新产品: {product['product_slug']}")
        response = session.put(
            crm_url + '/crm/product',
            json=product,
            headers=headers
        )
        print(f"状态: {response.status_code}")
```

```
        assert response.status_code == 200
        print(f"创建产品: {product['product_slug']}")
```

将产品链接到□作

CRM 产品与 CGRateS 操作之间的关键链接□ 命名约定:

```
# 在 CGRateS 中
ActionsId = "Action_au-premium-plan-1"

# 在 CRM 中
product_slug = "au-premium-plan-1"

# 在 playbook 中(自动构建)
cgr_action_name = "Action_" + product_slug
# 结果: "Action_au-premium-plan-1"
```

Playbooks 如何执行□作

注意：使用 `ExecuteAction` 只□添加余额的一种模式。您也可以直接在 playbook 中调用 `SetBalance` 或其他 CGRateS API。然而，`OmniCRM-API/Provisioners/plays/play_topup_charge_then_action.yaml` 使用 ExecuteAction 模式，因为这□一个常见的、可重用的方法，将余额逻辑（操作）与配置逻辑（Playbooks）分开。

当使用此模式配置产品时，playbook 从 product_slug 构建操作名称:

```
# 在 play_topup_charge_then_action.yaml 中
- name: 设置 cgr_action_name 事实
  set_fact:
    cgr_action_name: "Action_{{
api_response_product.json.product_slug }}"

# 执行操作
- name: 执行 CGRateS 操作
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "APIerSv1.ExecuteAction",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "ActionsId": "{{ cgr_action_name }}"
        }]
      }
```

使用此模式时：操作 必须在 **CGRateS** 中存在，否则执行将失败并出现 "找不到操作" 错误。

另请参见: Ansible Playbooks 指南

完整示例：定义新产品

让我们逐步创建一个完整的新产品 "50GB 数据包":

1. 在 CGRateS 中定义□作

请参见步骤 5 中的 简单数据附加 示例，了解如何定义数据附加操作。对于我们的 50GB 示例，操作将遵循相同的模式，但单位不同:

- **ActionsId**: `"Action_50gb-data-pack"` (与下面的 product_slug 匹配)
- **BalanceType**: `"*data"`
- **DestinationIDs**: `"Dest_PLMN_OnNet"` (用于本网数据使用)
- **Units**: `50 * 1024 * 1024 * 1024` (50GB 的字节数)

- **Identifier**: `"*topup"` (启用回滚)

2. 在 CRM 中创建产品

```
product_50gb_data = {
    "category": "addon",
    "provisioning_play": "play_topup_charge_then_action",
    "relies_on_list": "",
    "contract_days": 0,
    "retail_cost": 25.00,
    "retail_setup_cost": 0,
    "product_slug": "50gb-data-pack",  # 必须与 ActionsId 匹配，不带 "Action_"
    "product_name": "50GB 数据包",
    "comment": "有效期 30 天的 50GB 高速数据",
    "provisioning_json_vars": "",
    "terms": "数据在 30 天后过期。如果在到期之前再次充值，未使用的数据将回滚。",
    "service_type": "mobile",
    "residential": True,
    "business": True,
    "enabled": True,
    "icon": "fa-solid fa-database",
    "features_list": "['50GB 高速数据', '30 天有效', '启用回滚']",
    "wholesale_cost": 20.00,
    "wholesale_setup_cost": 0,
    "auto_renew": "False",
    "customer_can_purchase": True
}

# 在 CRM 中创建
response = session.put(
    crm_url + '/crm/product',
    json=product_50gb_data,
    headers=headers
)
assert response.status_code == 200
print("产品创建成功: 50gb-data-pack")
```

3. 验证链接

现在可以配置该产品:

1. 客户购买 "50GB 数据包"（product_slug: `50gb-data-pack`）
2. Playbook `play_topup_charge_then_action` 执行
3. Playbook 构建操作名称: `Action_50gb-data-pack`
4. Playbook 调用 `APIerSv1.ExecuteAction`，操作 ID 为 `Action_50gb-data-pack`
5. CGRateS 找到该操作并执行
6. 客户收到 50GB 余额

另请参见：从 Playbooks 进行计费和支付

常见产品模式

这些模式展示了典型的 CRM 产品配置及其与 CGRateS 操作的链接。有关相应的操作定义，请参见 CGRateS 操作和充值行为。

模式 1：多余额月度计划

用例：综合月度计划，包括数据、语音、SMS、漫游

CRM 产品：

```
{
    "product_slug": "premium-monthly-plan",  # 链接到 Action_premium-monthly-plan
    "product_name": "高级月度计划",
    "category": "addon",
    "retail_cost": 59.00,
    "auto_renew": "True",
    "contract_days": 30,
    "provisioning_play": "play_topup_charge_then_action",
    "features_list": "['100GB 数据', '5000 分钟', '5000 SMS', '包含漫游']"
}
```

作：使用 `*topup_reset` 和多种余额类型。请参见 示例 1：多余额月度计划（Python）。

模式 2：简单数据附加（回滚）

用例：一次性数据购买，如果再次充值则回滚

CRM 产品:

```
{
    "product_slug": "10gb-addon",  # 链接到 Action_10gb-addon
    "product_name": "10GB 数据附加",
    "category": "addon",
    "retail_cost": 15.00,
    "auto_renew": "False",  # 一次性购买
    "contract_days": 0,
    "provisioning_play": "play_topup_charge_then_action",
    "features_list": "['10GB 数据（启用回滚）', '30 天有效']"
}
```

□作：使用 `*topup` 以实现回滚行为。请参见 CGRateS 操作 - 回滚。

模式 3：固定月度计划（无回滚）

用例：月度计划始终重置为相同的金额

CRM 产品:

```
{
    "product_slug": "fixed-50gb-monthly",  # 链接到 Action_fixed-
50gb-monthly
    "product_name": "固定 50GB 月度计划",
    "category": "addon",
    "retail_cost": 35.00,
    "auto_renew": "True",
    "contract_days": 30,
    "provisioning_play": "play_topup_charge_then_action",
    "features_list": "['50GB 数据', '每月重置', '无回滚']"
}
```

□作：使用 `*topup_reset` 防止回滚。请参见 CGRateS 操作 - 重置行为。

高级：充电器和计费计划

重要：本节涵盖基本的按使用计费。CGRateS 支持极其复杂的计费场景，包括：

- 差异化漫游费率：漫游时 MO 与 MT 通话/SMS 的不同价格
- 特定协议计费：用于 2G/3G 的 CAMEL（MO-MT SMS、USSD），用于 4G/5G 的 Diameter
- 上下文感知定价：基于时间、地点、客户细分、网络���量的费率
- 多维计费：结合网络类型、漫游状态、目的地和时段

以下示例展示了适合大多数 OmniCRM 部署的简单、固定费率的 PAYG 配置。

有关按使用计费（超出包含余额的使用费用），您需要定义计费计划，以指定收费金额。

计费工作原理

当客户使用服务（数据、语音、SMS）时：

1. CGRateS 检查单位余额（例如，包含的数据 GB）
2. 如果存在单位余额，则从中扣除使用量
3. 如果没有单位余额或余额耗尽，CGRateS 将回退到货币余额
4. 计费计划 确定价格（例如，$0.10 每 MB）
5. 计算出的费用从客户的货币余额中扣除

示例：设置按使用计费的数据费用

```
# 1. 定义目的地费率（价格）
OCS_Obj.SendData({
    "method": "ApierV2.SetTPDestinationRate",
    "params": [{
        "TPid": tpid,
        "ID": "DR_DATA_PAYG",
        "DestinationRates": [{
            "DestinationId": "Dest_PLMN_OnNet",
            "RateId": "RT_DATA_$0_10_PER_MB",
            "RoundingMethod": "*up",
            "RoundingDecimals": 4,
            "MaxCost": 0,
            "MaxCostStrategy": ""
        }]
    }]
})

# 2. 定义实际费率
OCS_Obj.SendData({
    "method": "ApierV2.SetTPRate",
    "params": [{
        "TPid": tpid,
        "ID": "RT_DATA_$0_10_PER_MB",
        "RateSlots": [{
            "ConnectFee": 0,           # 无连接费用
            "Rate": 0.10,              # 每单位 $0.10
            "RateUnit": "1048576",     # 1 MB（1024 * 1024 字节）
            "RateIncrement": "1048576", # 按 MB 计费
            "GroupIntervalStart": "0s"
        }]
    }]
})

# 3. 定义使用此费率的计费计划
OCS_Obj.SendData({
    "method": "ApierV2.SetTPRatingPlan",
    "params": [{
        "TPid": tpid,
        "ID": "RP_PAYG_DATA",
        "RatingPlanBindings": [{
            "DestinationRatesId": "DR_DATA_PAYG",
```

```
            "TimingId": "*any",
            "Weight": 10
        }]
    }]
})

# 4. 将计费计划链接到计费配置文件（针对特定帐户）
OCS_Obj.SendData({
    "method": "ApierV2.SetTPRatingProfile",
    "params": [{
        "TPid": tpid,
        "Tenant": tenant,
        "Category": "data",
        "Subject": "*any",  # 或特定帐户标识符
        "RatingPlanActivations": [{
            "ActivationTime": "2024-01-01T00:00:00Z",
            "RatingPlanId": "RP_PAYG_DATA",
            "FallbackSubjects": ""
        }]
    }]
})

# 5. 加载资费计划
OCS_Obj.SendData({
    "method": "APIerSv1.LoadTariffPlanFromStorDb",
    "params": [{
        "TPid": tpid,
        "DryRun": False,
        "Validate": True,
        "APIOpts": {}
    }]
})
```

这会做什么：

- 当客户没有数据余额（或用完余额）时，数据使用费用为 **$0.10** 每 **MB**
- 收费从他们的货币余额中扣除（`*monetary`）
- 计费增量为 1 MB（向上取整）

示例：语音通话费率

```
# 为不同目的地定义不同的费率
OCS_Obj.SendData({
    "method": "ApierV2.SetTPRate",
    "params": [{
        "TPid": tpid,
        "ID": "RT_VOICE_DOMESTIC_$0_30_PER_MIN",
        "RateSlots": [{
            "ConnectFee": 0.15,           # $0.15 连接费用
            "Rate": 0.30,                 # 每分钟 $0.30
            "RateUnit": "60s",            # 按分钟计费
            "RateIncrement": "60s",       # 四舍五入到整分钟
            "GroupIntervalStart": "0s"
        }]
    }]
})

OCS_Obj.SendData({
    "method": "ApierV2.SetTPRate",
    "params": [{
        "TPid": tpid,
        "ID": "RT_VOICE_INTL_$1_50_PER_MIN",
        "RateSlots": [{
            "ConnectFee": 0.25,           # $0.25 连接费用
            "Rate": 1.50,                 # 每分钟 $1.50
            "RateUnit": "60s",
            "RateIncrement": "1s",        # 按秒计费（更准确）
            "GroupIntervalStart": "0s"
        }]
    }]
})
```

充电器配置文件（可选）

充电器将计费应用于特定类型的事件。对于大多数用例，默认充电器就足够了：

```
OCS_Obj.SendData({
    "method": "APIerSv1.SetChargerProfile",
    "params": [{
        "Tenant": tenant,
        "ID": "Charger_Default",
        "FilterIDs": [],
        "AttributeIDs": ["*constant:*req.RequestType:*none"],
        "Weight": 999
    }]
})
```

另请参见：

- CGRateS 操作和充值行为 - 余额管理方法（单位、货币、混合）和计费配置
- CGRateS 目的地 - 如何定义地理和 PLMN 目的地

最佳实践

1. 使用一致的命名约定

```
# 好 - 清晰的关系
ActionsId = "Action_premium-plan-100gb"
product_slug = "premium-plan-100gb"

# 坏 - 没有清晰的关系
ActionsId = "Action_Plan_A"
product_slug = "premium_100"
```

2. 在产品之前定义作

始终在创建 CRM 产品之前创建 CGRateS 操作。如果在执行时尝试执行不存在的操作，配置 playbook 将失败。

3. 包含 CDR 日志录

始终添加 `*cdrlog` 操作以跟踪余额添加：

```
{
    "Identifier": "*cdrlog",
    "BalanceType": "*generic",
    "ExtraParameters": "
{\"Category\":\"^activation\",\"Destination\":\"Product Name\"}",
    "Weight": 80
}
```

4. 使用描述性余额 ID

```
# 好
"BalanceId": "Premium_Data_100GB"
"BalanceId": "AU_Voice_Domestic__" + str(units)

# 坏
"BalanceId": "data1"
"BalanceId": "balance"
```

5. 清晰地记录单位

```
# 好 - 显示计算
Units = 100 * 1024 * 1024 * 1024  # 100GB 的字节数
Units = 3000 * 60 * 1000000000    # 3000 分钟的纳秒数

# 坏 - 魔法数字
Units = 107374182400
```

6. 战略性地使用权重

较高的权重 = 执行和消耗的优先级更高：

```
# 执行顺序（较高的权重 = 首先执行）
{"Identifier": "*reset_account", "Weight": 700}
{"Identifier": "*topup_reset", "Weight": 90}
{"Identifier": "*cdrlog", "Weight": 80}

# 余额消耗（较高的权重 = 首先消耗）
"BalanceWeight": 1200  # 高级/国内余额
"BalanceWeight": 1100  # 漫游余额
"BalanceWeight": 1000  # 国际余额
```

7. 首先使用小值进行测试

在定义新产品时，首先使用小余额进行测试：

```
# 测试
Units = 100 * 1024 * 1024  # 100MB 用于测试

# 生产
Units = 100 * 1024 * 1024 * 1024  # 100GB
```

故障排除

错误：“找不到□作”

症状：Playbook 失败并显示“找不到操作”或“ActionsId 不存在”

原因：操作未在 CGRateS 中定义，或命名不匹配

解决方案：

1. 验证操作□否存在：查询 CGRateS 以获取该操作
2. 检查命名：确保 `ActionsId = "Action_" + product_slug`
3. 如果缺失，请定义该操作

错误：“找不到目的地”

症状：操作执行但未创建余额

原因： DestinationIDs 引用不存在的目的地

解决方案：

1. 首先在 CGRateS 中定义目的地
2. 使用 `*any` 作为通用目的地
3. 检查目的地 ID 的拼写

产品创建但未添加余额

症状：产品成功配置但客户没有余额

原因：操作存在但没有余额添加操作

解决方案：

1. 验证操作□否具有 `*topup` 或 `*topup_reset` 操作
2. 检查单位值□否正确（不□ 0）
3. 验证到期时间□否已过

余额未回滚

症状：使用回滚但未使用的余额消失

原因：使用 `*topup_reset` 而不□ `*topup`

解决方案：

- 将 `Identifier` 从 `*topup_reset` 更改为 `*topup`
- 确保 BalanceId 在□□值之间保持一致

另请参见：CGRateS 操作和充值行为

总结

本指南展示了在 OmniCRM 中配置产品与 CGRateS 的一种常见方法。这里展示的模式适用于典型的移动虚拟网络运营商（MVNO）用例，包括预付费/后付费计划、数据包和漫游。

请记住： CGRateS 能够处理比这里涵盖的更复杂的场景。如果您需要高级功能，例如：

- 漫游语音和 SMS 的差异化 MO/MT 定价
- 针对 2G/3G 遗留设备的基于 CAMEL 的计费
- 时段或季节性费率变化
- 基于客户等级的定价（青铜/白银/黄金）
- 基于网络质量或 QoS 的计费
- 多供应商的最低成本路由

...请直接查阅 CGRateS 文档，并独立于 CRM 产品定义进行计费计划配置。

相关文档

- CGRateS 操作和充值行为 - 有关操作类型和余额行为的详细指南
- Ansible Playbooks 指南 - Playbooks 如何执行操作
- 从 Playbooks 进行计费和支付 - 配置期间的支付处理
- 产品生命周期 - 管理产品的生命周期
- 概念：产品和服务 - 产品基础知识
- CGRateS 官方文档 - 完整的 CGRateS 参考

短信广播系统

OmniCRM中的短信广播系统使移动网络运营商能够向特定地理区域内的移动设备发送紧急警报和公共警告。短信广播用于AMBER警报、天气警告、海啸警报和其他紧急通知的重要公共安全功能。

主要优势：与标准SMS消息不同，短信广播消息将在静音、余额不足或漫游的手机上发出声音警报。由于这广播消息，因此可以在几秒钟内向每个携带移动电话的居民发送警报。

Omnitouch警告链接（**OWL**）

Omnitouch警告链接（OWL）平台为灾害管理专业人员和移动网络运营商提供了全面的解决方案：

- 短信广播实体（**CBE**） - 供授权用户创建和广播紧急警告消息的安全基于Web的应用程序
- 短信广播中心（**CBC**） - 符合标准的网络集成组件，连接到蜂窝网络（2G/3G/4G/5G）以分发消息

OWL旨在用于任何具有Web浏览器的设备（Chrome/Firefox/Safari/Edge），例如计算机、笔记本电脑、平板电脑或手机。

概述

短信广播（也称为公共警告系统或PWS）允许运营商：

- 发送紧急警报 - 向区域内的所有设备分发关键安全信息
- 目标地理区域 - 通过控制哪些基站广播消息，向特定跟踪区域或网络小区广播
- 支持多种语言 - 同时提供多种语言的警报消息（每种语言最多500个字符）
- 管理警报生命周期 - 创建、更新、批准、监控和删除广播消息
- 与外部系统集成 - 通过多个蜂窝网络接口与CBC（短信广播中心）基础设施连接
- 双因素认证 - 使用基于时间的一次性密码（TOTP）确保安全的批准过程
- 双人规则 - 在消息传输之前，需第二个人的批准（可选要求）

与SMS不同，短信广播不需要订阅者列表，可以立即到达特定地理区域内的所有兼容设备，使其非常适合时间敏感的公共安全警报。在大多数网络中，向所有设备广播的时间少于10秒。

使用案例

短信广播用于：

- 紧急警告 - 自然灾害（地震、海啸、洪水、火灾）
- **AMBER**警报 - 儿童绑架通知
- 天气警报 - 严重天气警告、龙卷风警报
- 公共安全 - 恐怖威胁、化学泄漏、撤离
- 测试消息 - 系统测试和公众意识活动（每月测试消息不会提醒用户，但会被接收）

在紧急情况下，地理定位通常至关重要。例如，海啸警报应告知靠近沿海地区的人们寻求更高的地方，而不应发送给远离危险的内陆地区的人。通过控制哪些基站广播紧急警告消息，传输范围可以限制在适当的地理区域内。

紧急警告消息生命周期

传输紧急警告消息需要快速行动，同时提供准确的信息和身份验证以确保消息的有效性。

消息生命周期包括四个阶段：

1. 消息定义 - 消息类型、消息内容、过期设置
2. 消息定位 - 地理区域和基站广播
3. 消息批准 - 授权/验证操作员身份和第二个人的批准（如有必要）
4. 消息审核 - 传输前的最终确认

广播后，消息可以被监控，随着情况的发展进行更新，并在立即危险减退时停止。

阶段**1**：消息定义

此步骤定义要广播的紧急警告消息的基本参数：

消息标识符

不同类型的消息有不同的标识符，接收手机会以不同的方式处理。例如，每月测试消息不应提醒实际用户，但他们仍会收到消息。

注意： OWL系统中可用的每个消息模板已经嵌入了适当的消息标识符（MI），因此系统用户在创建警报消息时无需手动选择此项。

消息文本

文本主体限制为500个字符，包含要显示给最终用户的消息。可以通过在第一种语言下方添加第二种语言文本来提供多种语言的消息。请记住，

500□□□

“添加新区域”功能允许定义可以保存以供将来使用的自定义目标区域。

阶段3：批准

批准过程验证发出紧急警告消息的人□否有权这样做。

双因素认证

通过以下方式使用基于时间的一次性密码（TOTP）：

- 物理令牌（如RSA SecurID）
- 基于应用的解决方案（Google Authenticator、Authy、Microsoft Authenticator或其他兼容TOTP的应用）

将创建或批准草稿消息警报的用户必须在其智能手机上安装一个身份验证器应用程序，以生成系统将请求的授权代码。

设置2FA

首次配置2FA时：

1. 在智能手机上安装身份验证器应用程序（Google Authenticator、Authy、Microsoft Authenticator等）
2. 导航到您的OWL帐户设置，并使用身份验证器应用程序扫描QR码
3. 输入验证码以确认设置
4. 将备份代码保存在安全位置
5. 在紧急情况之前测试代码生成

有关详细的2FA设置说明，请参见`双因素认证 <2fa>`。

如果您更换移动设备或应用程序与您的OWL帐户停止同步，请联系您的系统管理员以获得帮助。管理员可以从用户和角色 → 用户页面重置2FA令牌。

双人规则

在流程要求第二个人批准的情况下，发出警报的人必须输入另一人的基于时间���一次性密码，才能允许流程继续。这提供了监督并最大限度地降低了滥用风险。

细粒度用户角色

可以配置单个用户角色以：

- 仅允许某些用户发送预定义消息
- 将消息定位限制在特定区域

- 需要额外的批准工作流

阶段4：审核

完成消息定义、定位和批准阶段后，操作员必须在最终广播前审核消息。一旦对消息细节满意，他们可以传输消息。

传输速度：在大多数网络中，向网络内所有设备广播的时间少于10秒。

阶段5：监控和更新

一旦消息广播开始，操作员可以监控和管理传输的消息。

网络反馈

蜂窝网络返回有关已广播消息的基站的信息。如果某个基站离线或不可用，这将反馈给操作员。

自动重传

如果任何离线基站在紧急警告仍然有效时再次可用，所有连接到该基站的手机将收到消息。

在飞行中的更新

广播后，消息可以：

- 随着情况的发展进行更新
- 修改新的消息主体内容
- 在任何时候召回/停止

历史记录

所有关于历史消息的信息都可以查看和审计。

消息结构

每条短信广播消息由以下部分组成：

消息配置

- 消息标识符 - 警报类型的唯一标识符（例如，4370用于ETWS地震，4371用于ETWS海啸）
- 类别 - 警报类别（正常、紧急、高、极端）
- 重复周期 - 广播重复之间的秒数

- 广播次数 - 广播消息的次数
- 警告周期 - 警告有效的持续时间（以秒为单位）
- 通道指示符 - 用于广播的通道类型

本地化消息

每条CBC消息可以包括多种语言变体：

- 语言 - ISO语言代码（en、es、fr、zh等）
- 消息主体 - 该语言的警报文本（最多1395个字符）

系统会自动广播所有语言变体，使接收者能够以其首选语言查看警报。

跟踪区域

定义警报的地理定位：

- 跟踪区域 - 地理标识符（小区ID、跟踪区域代码）
- 运营商 - 移动网络运营商代码（MCC-MNC）
- **RAT**类型 - 无线接入技术（LTE、5G、UMTS、GSM）

可以指定多个跟踪区域以覆盖更大区域或多个运营商。

创建短信广播消息

通过**Web UI**：

1. 导航到短信广播 - 从主导航访问CBC管理界面
2. 点击“创建警报” - 打开消息创建表单
3. 配置消息参数：
    - 消息标识符（例如，4370用于地震警报）
    - 类别（正常、高、极端）
    - 重复周期（通常为5-60秒）
    - 广播���数（999用于连续，或特定计数）
    - 警告周期（以秒为单位的持续时间）
    - 通道指示符（通常为“基本”）

4. 添加本地化消息：
    - 点击“添加语言”
    - 从下拉菜单中选择语言
    - 输入消息文本（GSM7最大1395个字符，Unicode更少）
    - 对其他语言重复此操作
5. 定义跟踪区域：
    - 点击“添加跟踪区域”
    - 输入跟踪区域代码
    - 选择运营商（MCC-MNC组合）
    - 选择RAT类型（LTE、5G等）
    - 对其他地理区域重复此操作

6. 审核并创建 - 验证所有细节并点击“创建警报”

通过**API**:

端点: `PUT /crm/cbc/`

所需权限: `CREATE_CBC_MESSAGE`

请求体:

```
{
  "messageIdentifier": "4370",
  "category": "emergency",
  "repetitionPeriod": 10,
  "numberOfBroadcasts": 999,
  "warningPeriodSec": 3600,
  "channelIndicator": "basic",
  "localized_messages": [
    {
      "language": "en",
      "messageBody": "EARTHQUAKE WARNING: Magnitude 6.5 earthquake
detected. Take cover immediately. Drop, Cover, Hold On."
    },
    {
      "language": "es",
      "messageBody": "ADVERTENCIA DE TERREMOTO: Terremoto de
magnitud 6.5 detectado. Cúbrase inmediatamente. Agáchese, Cúbrase,
Agárrese."
    }
  ],
  "tracking_areas": [
    {
      "tracking_area": "12345",
      "operator": "310-410",
      "rat_type": "LTE"
    },
    {
      "tracking_area": "12346",
      "operator": "310-410",
      "rat_type": "5G"
    }
  ]
}
```


□□□

```
{
  "cbc_message_id": 123,
  "cbc_unique_id": "550e8400-e29b-41d4-a716-446655440000",
  "messageIdentifier": "4370",
  "category": "emergency",
  "repetitionPeriod": 10,
  "numberOfBroadcasts": 999,
  "warningPeriodSec": 3600,
  "channelIndicator": "basic",
  "initiating_user": 5,
  "approving_user": null,
  "created": "2025-01-10T14:30:00Z",
  "localized_messages": [...],
  "tracking_areas": [...]
}
```

消息立即发送到短信广播中心进行传输。

管理现有消息

查看所有消息

端点: `GET /crm/cbc/`

所需权限: `VIEW_CBC_MESSAGE`

返回所有CBC消息的列表及其状态、时间戳和配置。

更新消息

端点: `PATCH /crm/cbc/{cbc_message_id}`

所需权限: `UPDATE_CBC_MESSAGE`

更新消息内容、跟踪区域或广播参数。更新的消息会重新发送到CBC。

请求体:

```
{
  "cbc_message_id": 123,
  "numberOfBroadcasts": 500,
  "localized_messages": [
    {
      "language": "en",
      "messageBody": "UPDATED: Earthquake warning still in effect.
Aftershocks possible."
    }
  ]
}
```

删除消息

端点: `DELETE /crm/cbc/{cbc_message_id}`

所需权限: `DELETE_CBC_MESSAGE`

从数据库中删除消息并尝试在CBC上取消它。

批准工作流

短信广播消息支持高风险警报的可选批准工作流:

1. 发起用户 - 创建警报的工作人员（`initiating_user`字段）
2. 批准用户 - 在广播前批准警报的经理（`approving_user`字段）

如果需要批准:

- 消息以`approving_user = null`创建
- 消息保持在“待批准”状态
- 批准用户审核消息并批准或拒绝
- 批准后，设置`approving_user`并广播消息

该工作流可根据组织政策进行配置。

消息标识符

标准消息标识符遵循3GPP TS 23.041:

地震和海啸警告系统（**ETWS**）:

- **4370** - ETWS地震警告
- **4371** - ETWS海啸警告
- **4372** - ETWS地震和海啸联合警告
- **4373-4378** - ETWS其他紧急类型
- **4379** - ETWS测试消息

商业移动警报系统（**CMAS**）/无线紧急警报（**WEA**）:

- **4352** - 总统警报
- **4353-4355** - 极端警报
- **4356-4359** - 严重警报
- **4360-4363** - AMBER警报
- **4364-4367** - 公共安全消息
- **4368-4369** - 州/地方测试
- **4380-4381** - 测试消息

自定义范围:

- **0-999** - 保留用于运营商特定警报
- **1000-4095** - 自定义消息类型

与短信广播中心的集成

短信广播实体（CBE）需要一种机制将消息传递给各个蜂窝网络。OWL短信广播中心（CBC）连接到每个蜂窝网络，以向公众发送紧急警告消息。

多网络冗余
如果一个蜂窝网络运营商不可用（停机或没有覆盖），如果另一个运营的蜂窝网络可用，用户仍将通过其他可用网络接收紧急警告消息。

按运营商的**CBC**实例

为了确保竞争网络之间没有连接，OWL为每个MNO运行一个单独的CBC实例；它不在运营商之间共享。

OWL CBC配置

CBC URL在`crm_config.yaml`中配置:

```
cbc_url: "http://cbc.example.com:8080"
```

消息传输:

当创建或更新CBC消息时:

1. OmniCRM CBE将消息存储在其数据库中
2. 消息格式化为CBC API
3. 向`{cbc_url}/alerts/send`发送HTTP POST请求
4. CBC通过适当的接口（CBSP、SBc-AP或N50）连接到蜂窝网络
5. CBC确认接收并开始广播
6. 目标跟踪区域内的设备接收警报

消息删除:

当消息被删除时，CBE尝试在CBC上取消它以停止正在进行的广播。

蜂窝网络集成点

不同代的蜂窝网络（2G/3G/4G/5G）各自具有独特的接口，用于连接到/从短信广播中心。根据每个蜂窝网络使用的技术，必须配置正确的接口。

OWL短信广播中心支持2G、3G、4G和5G短信广播接口，并已与许多常用的蜂窝网络组件集成。

CBSP - 2G/3G - 基站控制器（**BSC**）

短信广播服务协议（CBSP）接口将CBC与控制2G（GSM）基站的基站控制器（BSC）连接起来。

- 用于2G和3G短信广播消息，结合无线网络控制器/基站控制器部署
- 可以根据BSC供应商配置为客户端或服务器安排

- 必须在网络中的所有BSC与OWL CBC之间建立连接
- 接口持续监控，并发出警报以指示CBSP链接□否已断开

注意：3GPP定义了用于独立RNC部署的服务区广播协议（SABP）。如果在组合RNC/BSC上不支持CBSP，则可以在需要时使用此协议，但可能需要来自RNC供应商的额外测试和支持。

SBc-AP - 4G/5G非独立 - MME/IWF

SBc-AP接口将OWL CBC与为4G和5G eNodeB/gNodeBs提供服务的MME（移动管理实体）连接。

- 用于4G LTE网络
- 也用于非独立5G（截至2025年大多数部署）
- 必须在网络中的所有MME与OWL CBC之间建立连接
- 接口持续监控，并发出警报以指示SBc-AP链接□否已断开

N50 - 5G独立 - AMF

对于独立5G网络，N50接口将OWL CBC与为5G gNodeBs提供服务的AMF（接入和移动管理功能）连接。

- 接口存在于OWL CBC中
- 由于2025年可用的商业5G SA网络数量较少，因此尚未与第三方AMF进行广泛测试
- 随着5G SA部署变得更加普遍，将全面支持

MNO网络要求

OWL CBC与移动网络运营商的网络之间必须建立网络，以达到上述接口。

这逐案处理，但通常需要：

- CBC与MNO网络之间的专用交叉连接/光纤
- 每个接口逻辑上分开
- 连接到蜂窝网络中的每个集成点（MME、RNC、BSC）

支持的网络设备

OWL CBC已与主要供应商的常用蜂窝网络组件进行了测试和集成：

基站数据集成

OWL支持自动数据抓取来自：

- 诺基亚**NetAct**
- 华为**U2000 / U2020**
- 中兴**NetNumen / ZXPOS**
- 爱立信**ENM**

或者，基站数据可以定期通过电子邮件提供给Omnitouch运营团队。

用户管理和访问控制

基于角色的访问控制（**RBAC**）

OWL系统使用基于���色的访问控制（RBAC）：人（注册用户）被分配一个或多个角色，每个角色□一组权限的集合。权限□访问的最小单位（例如，创建草稿消息警报）。注册用户的有效访问□所有分配角色的权限的并集。

RBAC组件：

- 用户 - 登录OWL系统的真实人员
- 权限 - 微能力（例如，批准草稿消息、创建消息、查看报告）
- 角色 - 权限的命名集合（例如，消息批准者、消息创建者）
- 分配 - 用户接收一个或多个角色；权限聚合

RBAC的好处：

1. 数据保护 - 用户仅查看和执行他们被允许的操作
2. □作适应性 - 角色反映工作职能（管理员、消息创建者、消息批准者）
3. 简单管理 - 通过分配角色授予访问权限；避免逐用户的微观管理

系统权限

系统权限通常遵循CRUD模式，具有四个选项：

- 查看 - 阅读或浏览消息和报告
- 创建 - 创建或添加消息警报
- 更新 - 编辑或修改草稿消息警报

- 删除 - 删除或移除草稿消息警报

核心**CBC**权限：

- `CREATE_CBC_MESSAGE` - 创建新的广播消息
- `VIEW_CBC_MESSAGE` - 查看现有消息及其状态
- `UPDATE_CBC_MESSAGE` - 修改消息内容或广播参数
- `DELETE_CBC_MESSAGE` - 删除消息并取消广播

根据您组织的公共安全职责将这些权限分配给角色。

有关用户角色和权限管理的全面信息，请参见用户和角色 `<rbac>`。

用户帐户管理

首次登录

当新用户在系统中设置时，系统管理员会提供登录凭据：

1. 导航到系统管理员提供的OWL登录URL
2. 输入您的用户名和密码
3. 如果您的帐户启用了2FA，系统将提示您进行设置（请参见上面的2FA部分）

最佳实践：用户应在首次登录后立即更改初始密码以确保安全。导航到您的用户个人资料设置以更改密码。

有关身份验证流程的更多详细信息，请参见身份验证流程和管理员控制 `<authentication_flows>`。

密码重置

用户可以重置自己的密码：

1. 从系统登录页面，选择“忘记密码”
2. 按照电子邮件验证过程进行操作
3. 设置新密码

管理员密码重置：

系统管理员可以从用户和角色 → 用户页面重置用户的密码：

1. 导航到用户和角色 → 用户

2. 选择需要重置密码的用户
3. 点击重置密码
4. 系统生成一个安全的随机临时密码（12个字符）
5. 临时密码显示给管理员
6. 通过安全渠道（电话或面对面，绝不通过电邮件）将临时密码提供给用户
7. 用户可以使用临时密码登录
8. 重要：用户应在登录后立即更改此密码

有关更多详细信息，请参见用户和角色 `<rbac>`。

添加新系统用户

系统管理员可以通过导航到用户和角色 → 用户添加新用户：

每个新注册用户所需的信息：

- 名字
- 姓氏
- 电子邮件
- 密码（临时）
- 手机号码
- 角色

最佳实践：实施与您组织现有政策和程序一致的新系统用户申请表。仅在适当批准后添加用户。

编辑用户个人资料

系统管理员可以更改现有用户的权限和访问权限：

1. 导航到用户和角色 → 用户
2. 选择要编辑的用户
3. 根据需要修改分配的角色
4. 保存更改

删除系统用户

系统管理员可以从用户和角色 → 用户中删除用户。

警告：一旦用户从OWL系统中删除，此操作无法撤销。在执行此任务之前请务必小心。

查看所有系统用户

在任何时候，系统管理员都可以通过导航到用户和角色 → 用户查看所有注册用户的列表，并查看每个用户的权限和角色。

警报消息历史

OWL系统保留所有发送的警报消息记录，以便审计和支持操作审查。

访问消息历史：

导航到管理短信广播以查看：

- 所有先前发送的消息
- 消息状态（活动、过期、取消）
- 传输时间戳
- 发起和批准用户
- 地理定位详细信息
- 消息内容和参数

这有助于：

- 审计系统
- 审查NDMO和其他警报机构使用的操作程序
- 了解消息的有效性
- 合规和报告要求

短信广播地图可视化

短信广播地图提供了一个可视化界面，用于规划地理覆盖区域并识别哪些基站将广播警报。

访问地图

导航到：

或直接：

所需权限： `VIEW_CBC_MESSAGE` 或 `CREATE_CBC_MESSAGE`

地图功能

地图显示：

基站位置：

- 塔图标 - 基站位置以基站图标显示
- 扇区指示器 - 显示天线扇区方向的方向箭头
- 塔名称 - 站点标识符和名称
- 覆盖数据 - 从 `cellSites.txt` 文件加载

绘图工具：

- 多边形工具 - 绘制自定义覆盖区域
- 圆形工具 - 创建圆形广播区域
- 矩形工具 - 定义矩形覆盖区域

覆盖分析：

- 站点列表 - 表格显示绘制区域内的所有基站
- 区域管理 - 保存和加载预定义的覆盖区域
- 站点计数 - 实时计数覆盖区域内的塔数

使用地图

步骤**1**：加载基站数据

地图在页面加载时自动加载基站位置，来自 `/cellSites.txt`。此文件包含：

- **lat/lng** - 塔坐标
- **site_name** - 标识符
- **sectors** - 天线方向（以度为单位）

步骤**2**：绘制覆盖区域

选择绘图工具：

1. 点击多边形以绘制自定义形状
    - 点击以添加点
    - 双击以完成
    - 创建精确的覆盖边界
2. 点击圆形以绘制圆形区域
    - 点击中心点
    - 拖动以设置半径
    - 适合快速半径基础警报
3. 点击矩形以绘制矩形区域
    - 点击一个角落
    - 拖动到对角角落
    - 对于网格对齐覆盖高效

步骤**3**：审核覆盖站点

绘制后，系统：

- 计算哪些基站落在该区域内
- 在地图下方的表格中显示列表
- 显示站点名称和坐标
- 提供受影响塔的计数

示例覆盖表：

步骤**4**：导出覆盖数据

使用覆盖数据：

- 识别广播的跟踪区域代码
- 根据塔数估计警报覆盖范围
- 通过了解受影响区域来规划紧急响应
- 保存区域定义以供将来使用

工作流集成

地图可视化与广播创建集成：

规划警报：

1. 打开**CBC**地图以可视化受影响区域
2. 使用多边形工具绘制覆盖区域
3. 审核覆盖站点并调整边界
4. 记录受影响基站的跟踪区域代码
5. 创建**CBC**消息并进行适当定位

示例：沿海海啸警告

区域管理

保存区域：

预定义的覆盖区域可以保存以供常见场景使用：

- 沿海地区 - 用于海啸/风暴潮警告
- 野火区域 - 用于撤离警报
- 城市中心 - 用于人口密集地区的AMBER警报
- 高速公路走廊 - 用于交通/天气警报

区域数据格式：

区域存储在`site_data.json`中：

```
[
  {
    "area": "Coastal Region",
    "polygon": [
      {
        "coord": [
          {"lat": -33.8688, "lng": 151.2093},
          {"lat": -33.8650, "lng": 151.2070},
          {"lat": -33.8600, "lng": 151.2150},
          {"lat": -33.8688, "lng": 151.2093}
        ]
      }
    ]
  }
]
```

加载已保存的区域：

1. 从下拉菜单中选择区域
2. 多边形自动在地图上显示
3. 站点列表更新以显示覆盖的塔
4. 如有需要，修改多边形

地图配置

Google Maps API：

地图需要在环境中配置Google Maps API密钥：

```
VITE_GOOGLE_API_KEY=your_google_maps_api_key_here
```

基站数据：

使用您网络的基站位置更新`public/cellSites.txt`：

```
# 格式: lat,lng,name,sector1,sector2,sector3
-33.8688,151.2093,SYD_CENTRAL,0,120,240
-33.8650,151.2070,SYD_HARBOUR,45,165,285
```

从以下来源获取基站坐标：

- 网络规划工具
- 塔安装记录
- 使用GPS的现场调查
- 无线网络控制器（RNC）配置

地图样式：

可以在`mapStyles.js`中配置自定义地图样式，以：

- 突出紧急服务
- 显示地形特征
- 强调人口中心
- 匹配组织品牌

使用案例

紧急规划：

- 为已知危险区域预先规划覆盖区域
- 测试不同的定位策略
- 估计人口覆盖范围
- 与紧急服务协调

警报验证：

- 确认消息将到达预定区域
- 识别覆盖盲区
- 避免对邻近区域的过度警报
- 验证跟踪区域代码

网络分析：

- 可视化基站分布
- 识别覆盖重叠
- 规划基站部署以改善警报覆盖
- 优化广播效率

最佳实践

根据与全球客户的经验，Omnitouch建议所有OWL部署遵循以下最佳实践。

消息内容：

- 保持消息简洁明了（单页警报不超过360个字符）
- 对于紧急警报使用全大写字母（提高可读性）
- 包含具体的行动项（“立即撤离”，“寻求庇护”）
- 避免技术术语
- 在紧急情况之前使用实际设备测试消息

语言支持：

- 始终以该地区的主要语言提供消息
- 为多元文化地区提供额外语言
- 确保翻译符合文化适宜性
- 测试特殊字符和Unicode支持

地理定位：

- 使用最小必要的跟踪区域以避免警报疲劳
- 在设置重复周期时考虑人口密度
- 在紧急情况之前测试地理定位
- 维护准确的跟踪区域文档

测试：

- 使用测试消息标识符（4379，4380-4381）进行演练
- 定期安排系统测试
- 验证CBC集成□否正常工作

- 培训员工应对紧急程序

警报疲劳：

- 仅用于真正的紧急情况
- 避免过度广播
- 设置适当的重复周期
- 适当使用严重性级别

用户帐户安全

2FA令牌安全：

- 保护物理2FA令牌，如建筑访问卡
- 立即报告丢失或被盗的令牌
- 定期测试2FA以确保其正常工作
- 在设置2FA时保存备份代码并安全离线存储
- 如果需要2FA重置，请联系系统管理员

有关2FA设置和恢复的更多信息，请参见`双因素认证 <2fa>`。

数据维护

预定义目标区域：

随着地理边界的变化、开发的发生和风险区域的转移，需要审查预定义目标区域。Omnitouch建议每年由NDMO审查这些数据，并在适用时获得其他警报机构的支持。

[illegible]以使用几种常见的GIS平台或Google Earth定义边界的更新，然后通过电子邮件提供给Omnitouch运营团队，后者将对系统进行更改。

预定义消息模板：

预定义消息模板应至少每年由NDMO审查，并在适用时获得其他警报机构的支持，以确保：

- 消息内容仍然准确反映危险和行动呼吁
- 联系信息和说明□最新的
- 语言翻译保持准确

- 消息语气和紧迫性适当
- 对紧急服务或程序的引用□最新的

可以通过电子邮件将更新提供给Omnitouch运营团队，后者将对系统进行更改。

基站数据：

- 每当网络发生变化时，审查和更新基站数据
- 典型更新频率：每月或每季度
- 与MNO网络规划团队协调
- 在主要网络升级或扩展后验证准确性

系统架构

所有Omnitouch产品都旨在支持地理分布的部署。

部署选项

所有组件可以作为以下内容运行：

- 容器（**K8s**） - Kubernetes编排的容器化部署
- 虚拟机 - VMware、Proxmox、HyperV
- 私有云 - 本地云基础设施
- 公共云 - AWS、GCP
- 裸金属 - 直接硬件部署

分布式架构

分布式架构允许：

- 本地灾害管理办公室访问OWL CBE并分发消息，即使某个地区与国家网络隔离
- 每个运营商/每个国家有多个短信广播实体和多个短信广播中心
- 本地灾害响应机构（市政府、警察、消防等）在主要NDMO（国家灾害管理办公室）失去访问权限时向其区域发出警报

这在MNO分布其蜂窝网络与本地BSC/MME资源时特别有价值。

CBE和**CBC**网络

为了确保竞争网络之间没有连接，每个MNO都有一个单独的CBC实例（不共享）。

CBE与CBC之间的网络要求：

- CBE和CBC之间的所有流量都□加密的
- 基于相互证书的身份验证
- CBE与CBC之间通过TLS在TCP端口443上连接
- MNO与NDMO或托管CBE的机构之间需要协调

访问考虑

托管CBE的机构需要为最终用户定义访问程序（例如，Citrix 、VPN等），并考虑到系统必须在非理想场景下可访问，例如：

- 公共电力网络的大规模停电
- 电信网络故障
- 影响基础设施的自然灾害

部署要求

CBE虚拟机要求（**NDMO /** 政府）

3个虚拟机：

- **2**个**CBE**虚拟机（理想情况下位于不同的数据中心/可用性区域）
- **1**个监控虚拟机

每个虚拟机要求：

- 存储：50GB
- **CPU**：2个虚拟CPU
- **RAM**：8GB
- □作系统：Omnitouch提供的基础操作系统
- 网络：允许TCP端口443上的流量到CBC虚���机，以进行TLS流量控制

CBC虚拟机要求（MNO）

3个虚拟机：

- 2个CBC虚拟机（理想情况下位于不同的数据中心/可用性区域）
- 1个监控虚拟机

每个虚拟机要求：

- 存储：50GB
- CPU：2个虚拟CPU
- RAM：8GB
- 作系统：Omnitouch提供的基础操作系统
- 连接性：到蜂窝网络中的每个集成点（MME、RNC、BSC）
- 网络：允许来自CBE虚拟机的流量在TCP端口443上进行TLS流量控制

集成步骤

部署OWL涉及以下步骤：

1. 寻找新运营商 - 确定参与的移动网络运营商
2. 项目管理员（设置） - 建立项目治理和管理
3. 选择CBE的托管位置 - 确定CBE将托管的位置
4. 定义用户和消息流/程序 - 建立批准工作流和用户角色
5. 定义目标区域和消息模板的多边形 - 预配置常见场景
6. IP地址分配 - 为NDMO和MNO分配IP地址
7. 设置站点到站点的VPN - 建立与Omnitouch团队的安全连接
8. 将CBE虚拟机部署到NDMO - 安装短信广播实体
9. 将CBC虚拟机部署到MNO - 在每个运营商处安装短信广播中心
10. 配置网络元素 - 设置与CBC的连接
11. CBE和CBC虚拟机之间的网络 - 建立安全通信
12. CBC虚拟机与网络元素之间的网络 - 连接到BSC/MME/AMF
13. 设置API访问NMS - 配置基站数据集成
14. 监控设置和测试 - 验证监控和警报
15. 测试测试警报的验证/测试 - 进行系统测试

16. 公共警告测试 - 进行端到端的公共测试

字符限制

短信广播消息根据编码具有严格的字符限制：

GSM 7位编码（英语、基本拉丁字符）：

- 单页：93个字符
- 多页：15页 × 93 = 最大1395个字符

Unicode UCS-2编码（非拉丁脚本、表情符号）：

- 单页：41个字符
- 多页：15页 × 41 = 最大615个字符

OWL平台：

- 消息文本限制为500个字符
- Web UI显示剩余字符计数，并在接近限制时发出警告

监控和日志

短信广播活动被记录以供审计：

- 消息创建、更新和删除事件
- 发起和批准用户
- 时间戳和消息标识符
- CBC API响应和错误
- 地理定位详细信息

通过活动日志或数据库查询访问日志：

```
SELECT * FROM cbc
WHERE created >= '2025-01-01'
ORDER BY created DESC;
```

与移动设备的集成

短信广播消息由兼容的移动设备接收：

设备支持：

- 大多数2015年及之后的智能手机支持短信广播
- 功能手机可能支持有限
- 设备必须连接到网络（无需数据/SMS余额）
- 即使在网络拥堵时SMS失败也能正常工作

用户体验：

- 警报显示为全屏通知
- 播放独特的警报音
- 警报在被确认之前持续存在
- 不需要用户订阅
- 对于总统/极端警报，用户无法阻止

测试设备接收：

要验证设备是否能够接收警报：

1. 发送测试消息（标识符4379或4380-4381）
2. 确保设备在目标跟踪区域内
3. 检查设备在设置中已启用短信广播
4. 验证多个设备型号和操作系统版本

额外功能

OWL平台可以通过可选功能扩展，以补充短信广播消息：

大规模文本/SMS

向个人发送常规SMS以进行补充通知。

- □□□□

- 语音呼叫系统
- 广播系统
- 物理警报和警笛
- 电子标牌

可以在设计阶段探索定制选项。

定制和维护

定期测试

应定期对解决方案进行定期测试，以确保：

- 解决方案和所有组件正常工作
- 所有员工熟悉发出紧急警告消息所需的流程和程序
- 集成点保持正常
- 消息模板□最新和有效的

推荐测试：

- 使用标识符4379、4380-4381进行每月测试消息
- 每季度进行全面系统测试，包括批准工作流
- 每年进行公共警告测试，并提前通知
- 定期为授权用户进行培训

基站数据维护

当运营商添加或删除基站，或更改基站的跟踪区域/标识符时，必须与Omnitouch团队共享这些信息，以确保映射工具数据保持准确。

自动数据集成

OWL支持自动数据抓取来自：

- 诺基亚NetAct
- 华为U2000 / U2020
- 中��NetNumen / ZXPOS

- 爱立信ENM

手动数据更新

或者，基站数据可以定期通过电子邮件以各种格式提供给Omnitouch运营团队。

更新频率：每当网络发生变化时，审查和更新基站数据，通常为每月或每季度。

预定义目标区域

随着地理边界的变化、开发的发生和风险区域的转移，需要审查在定位阶段使用的预定义目标区域。

年度审查：预定义目标区域应由NDMO（国家灾害管理办公室）每年审查。

更新过程：更新边界可以使用几种常见的GIS平台或Google Earth定义，然后提供给Omnitouch运营团队。

预定义消息模板

年度审查：预定义消息模板应每年由NDMO审查，以确保：

- 消息内容反映当前的紧急程序
- 语言翻译准确
- 消息标识符适当
- 联系信息和说明□最新的

更新过程：更新可以通过电子邮件提供给Omnitouch运营团队。

消息批准流程

不同地区对发出消息和批准流程有不同的要求。

双人规则：在实际可行的情况下，建议使用双人规则，以确保消息提交的监督。

细粒度用户角色：可以配置单个用户角色以：

- 仅允许某些用户发送预定义消息
- 将定位限制在特定区域
- ���要额外的批准步骤

- 最大限度地降低滥用风险

2FA / 安全维护

Omnitouch运营团队可以提供支持:

- 重置2FA令牌
- 重新发放丢失/损坏/过期的令牌
- 安全审计和令牌管理

令牌安全程序:

当发放令牌时，详细说明了如果令牌丢失或失踪应采取的步骤。接受令牌的人必须遵循这些程序，以确保系统不被滥用。

丢失令牌的立即行动:

1. 立即向授权人员报告丢失的令牌
2. 在系统中停用令牌
3. 进行安全审查
4. 在安全验证后发放新令牌

外部API集成

提供一整套API供短信广播实体使用，以允许第三方系统与CBE集成和交互。

API功能:

- 报告/监控 - 验证传输消息的状态和覆盖范围
- 消息创建 - 以编程方式创建和广播新消息
- 健康检查 - 定期检查系统健康状况并生成例行测试消息流量
- 状态查询 - 检索消息状态、交付统计信息和系统指标

请参阅下面的API参考部分以获取详细的端点文档。

API参考

所有CBC端点都需要身份验证和适当的权限。

创建消息：

`PUT /crm/cbc/`

获取所有消息：

`GET /crm/cbc/`

更新消息：

`PATCH /crm/cbc/{cbc_message_id}`

删除消息：

`DELETE /crm/cbc/{cbc_message_id}`

请参阅`/crm/docs/`中的Swagger文档以获取详细的API规范。

全局搜索

全局搜索功能提供了一个统一的搜索界面，可以快速查找整个 OmniCRM 数据库中的客户、联系人、服务、库存和站点。

另见：`Customers <basics_customers>`，`Inventory <administration_inventory>`，`Service Management <csa_service_management>`。

访问全局搜索

从 **CRM** 的任何地方：

点击顶部导航栏中的搜索图标或导航到：

全局搜索页面将显示一个大型搜索框和过滤选项。

工作原理

全局搜索在五种数据类型之间执行跨实体搜索：

搜索内容：

1. 客户 - 客户名称
2. 联系人 - 名字、姓氏、电子邮件地址、电话号码
3. 站点 - 站点名称
4. 库存 - 序列号、ICCID、标识符（itemtext1、itemtext2）
5. 服务 - 服务名称、服务 UUID

搜索行为：

- 部分匹配 - 搜索包含您的查询的术语（例如，“Smith”匹配“John Smith”和“Smithson”）
- 不区分大小写 - “john”匹配“John”、“JOHN”和“john”
- 多个实体 - 单次搜索返回所有实体类型的结果
- 分页结果 - 默认每页显示 10 个结果

执行搜索

基本搜索

1. 在搜索框中输入您的搜索词
2. 点击**“搜索”**或按回车

示例搜索词：

- 客户名称：`"Acme Corp"`
- 电话号码：`"+1234567890"`或`"1234567890"`
- 电子邮件：`"john@example.com"`或`"john"`
- 序列号：`"ICCID8944"`或仅`"8944"`
- 服务 UUID：`"123e4567-e89b"`

包括已关闭账户

默认情况下，搜索仅返回开放客户账户的结果。

要搜索包括已关闭账户在内的所有账户：

1. 勾选**“包括已关闭账户”**复选框
2. 再次点击**“搜索”**

这将搜索：

- 状态为`customer_status = "Closed"`的客户
- 与已关闭客户关联的联系人、服务、站点和库存

已关闭账户搜索的用例：

- 查找历史客户记录
- 定位已停用服务的设备
- 查找旧电话号码或服务
- 恢复客户数据以便重新激活

理解搜索结果

结果显示格式

结果以可滚动列表显示：

> John Smith 客户 ID: 123 类型: customer
>
> John Smith (联系人) 客户 ID: 123 类型: contact
>
> Mobile - +44 7700 900123 客户 ID: 123 类型: service

每个结果显示：

- 名称/标题 - 主要标识符（可点击链接）
- 客户 ID - 该记录所属的父客户
- 类型 - 实体类型（客户、联系人、站点、库存、服务）

结果类型解释

客户结果：

点击打开客户概览页面，显示所有详细信息、服务、联系人等。

联系人结果：

点击打开客户页面，联系人标签处于活动状态，滚动到特定联系人。

站点结果：

点击打开客户页面，站点标签处于活动状态。

库存结果：

点击打开客户页面，库存标签处于活动状态。如果库存未分配（没有 customer_id），则链接到主库存列表。

服务结果：

点击打开客户页面，服务标签处于活动状态，突出显示特定服务。

从结果导航

所有搜索结果都可点击链接，直接导航到相关页面：

链接模式：

- `/customers/{customer_id}` - 客户记录
- `/customers/{customer_id}#4` - 联系人（标签 4）
- `/customers/{customer_id}#2` - 站点（标签 2）
- `/customers/{customer_id}#8` - 库存（标签 8）
- `/customers/{customer_id}#3` - 服务（标签 3）
- `/inventory-items-list` - 未分配库存

哈希（`#`）片段在客户页面加载时自动选择正确的标签。

分页

结果按每页 10 项进行分页：

> 显示结果 11-20，共 47 项

使用以下方式浏览页面：

- 上一页/下一页按钮
- 页码 - 点击特定页面
- 键盘 - 左/右箭头（如果已实现）

常见搜索场景

场景 **1**：通过电话查找客户

用户来电，提供电话号码。

> 结果：• John Smith (联系人) - 客户 ID: 123 • Mobile - 555-0123 (服务) - 客户 ID: 123

点击任一结果以访问客户账户。

场景 **2**：定位 **SIM** 卡

技术人员需要找出哪个客户拥有特定的 SIM。

> 结果：• 89445380000000001234 (库存) - 客户 ID: 456

点击结果以查看 SIM 分配、客户详细信息。

场景 **3**：查找不活跃客户

需要定位一个 6 个月前关闭账户的客户。

> ☑ 包括已关闭账户
>
> 结果：• Acme Corporation (客户) - 客户 ID: 789

场景 **4**：通过电子邮件搜索

客户发送电子邮件给支持，工作人员需要查找他们的账户。

> 结果：• John Smith (联系人) - 客户 ID: 123

场景 **5**：通过 **UUID** 查找服务

配置日志显示服务 UUID，需要找出哪个客户。

> 结果：• Mobile - +44 7700 900123 (服务) - 客户 ID: 456

搜索提示

获得最佳结果：

- 使用部分术语 - “Smith”比“John Smith”更好，以获得更广泛的���果
- 尝试变体 - 如果“John”无效，尝试电话或电子邮件
- 包括已关闭账户 - 在搜索历史数据时
- 对设备要具体 - 使用完整的序列号进行库存搜索
- 搜索服务 **UUID** - 当其他标识符未知时

按实体搜索的内容：

客户：

- 仅客户名称（不包括地址、备注或其他字段）

联系人：

- 名字
- 姓氏
- 电子邮件地址
- 电话号码

站点：

- 仅站点名称

库存：

- itemtext1（通常□ ICCID、序列号、MAC 地址）
- itemtext2（通常□ IMSI、次要标识符）
- *□□□□□□ itemtext3-20 □□□□□*

服务：

- 服务名称
- 服务 UUID

不被搜索的内容：

- 客户地址
- 客户备注
- 交易描述

- 发票详细信息
- 配置日志
- 活动日志条目
- 库存备注（超出 itemtext1/2）

API 参考

全局搜索端点

```
GET /utilities/search_everything?
search=Smith&page=1&per_page=10&search_closed_records=false
Authorization: Bearer <token>
```

查询参数：

- `search`（必需） - 搜索词
- `page`（可选） - 页码（默认：1）
- `per_page`（可选） - 每页结果数（默认：10）
- `search_closed_records`（可选） - 包括已关闭账户（默认：false）

响应：

```
{
  "data": [
    {
      "id": 123,
      "name": "John Smith",
      "customer_id": 123,
      "type": "customer"
    },
    {
      "id": 456,
      "name": "John Smith",
      "customer_id": 123,
      "type": "contact"
    },
    {
      "id": 789,
      "name": "Mobile - +44 7700 900123",
      "customer_id": 123,
      "type": "service"
    }
  ],
  "pagination": {
    "current_page": 1,
    "per_page": 10,
    "total_pages": 5,
    "total_items": 47
  }
}
```


搜索逻辑（后端）：

后端在所有实体表之间执行 SQL UNION:

```sql
-- Customers
SELECT customer_id AS id,
       customer_name AS name,
       customer_id,
       'customer' AS type
FROM customer
WHERE customer_name LIKE '%Smith%'
  AND customer_status = 'Open'

UNION ALL

-- Contacts
SELECT contact_id AS id,
       CONCAT(contact_firstname, ' ', contact_lastname) AS name,
       customer_id,
       'contact' AS type
FROM customer_contact
WHERE (contact_firstname LIKE '%Smith%' OR
       contact_lastname LIKE '%Smith%' OR
       contact_email LIKE '%Smith%' OR
       contact_phone LIKE '%Smith%')

UNION ALL

-- Sites
SELECT site_id AS id,
       site_name AS name,
       customer_id,
       'site' AS type
FROM customer_site
WHERE site_name LIKE '%Smith%'

UNION ALL

-- Inventory
SELECT inventory_id AS id,
       itemtext1 AS name,
       customer_id,
       'inventory' AS type
FROM inventory
WHERE itemtext1 LIKE '%Smith%' OR
      itemtext2 LIKE '%Smith%'
```

```
UNION ALL

-- Services
SELECT service_id AS id,
       service_name AS name,
       customer_id,
       'service' AS type
FROM customer_service
WHERE service_name LIKE '%Smith%' OR
      service_uuid LIKE '%Smith%'
```

结果随后被分页并返回。

性能考虑

搜索性能：

- 搜索使用带通配符的 LIKE 查询（`%term%`）
- 当前未实现全文索引
- 大型数据库（>100k 客户）可能会遇到搜索速度变慢
- 每页结果限制为 10 个以提高性能

优化提示：

- 使用具体的搜索词以减少结果集
- 使用已关闭账户过滤器以减少搜索范围
- 考虑在经常搜索的字段上添加数据库索引

故障排除

未找到结果（但记录存在）

- 原因：搜索词与存储的数据格式不匹配
- 示例：
  - 电话存储为“+44 7700 900123”，搜索“07700900123”将不匹配
  - 电子邮件存储为“<john.smith@example.com>”，搜索“john”将不匹配

- 解决方法：尝试变体，使用肯定存在的部分匹配

搜索太慢

- 原因：大型数据库，跨多个表的复杂查询
- 解决方法：
  - 使用更具体的搜索词
  - 限制为仅开放账户（取消勾选已关闭账户）
  - 联系管理员以获取数据库索引

结果链接到错误的客户

- 原因：多个客户/联系人具有相同名称
- 解决方法：使用客户 ID 进行区分，或通过唯一标识符（电子邮件、电话）进行搜索

已关闭账户未出现

- 原因：未勾选“包括已关闭账户”复选框
- 解决方法：勾选该框并重新搜索

相关文档

- `basics_customers` - 客户管理
- `basics_navigation` - 一般导航
- `administration_inventory` - 库存搜索

充值和补充系统

OmniCRM 充值系统提供一个 自助服务预付费充值门户，供客户通过 `Self-Care Portal <self_care_portal>` 添加信用或延长服务有效期。此功能通常用于：

- 移动数据服务 - 预付费 SIM 卡和仅数据服务
- 热点服务 - WiFi 热点 Dongle 和便携式互联网设备
- 固定无线服务 - 预付费互联网接入

另见： `SIM Card Provisioning <concepts_sim_provisioning>` 以获取有关移动服务配置和管理 SIM 库存的信息。

概述

充值系统允许客户通过一个简化的多步骤结账流程购买额外的服务天数，并集成了 Stripe 支付处理。

主要特点：

- 自助服务客户门户（无需员工干预）
- 灵活的持续时间选择（1-30 天）
- 购买前实时使用情况显示
- Stripe 支持的安全支付处理
- 如果充值失败，自动退款
- 发票和交易生成
- 服务激活的配置系统集成

访问充值门户

充值门户通过 公共 **URL** 访问，客户可以在不登录 CRM 的情况下访问：

客户如何访问：

- 当余额低时通过 SMS 发送的直接链接
- 印刷材料上的二维码

- 自助服务门户上的链接
- 客户支持共享

门户会根据客户请求的 IP 地址或 IMSI 自动检测客户的服务。

充值流程

充值流程由 **4** 个步骤 组成：

步骤 **1**：数据选择

客户选择他们想要购买的服务天数。

界面：

- 滑块控制 - 选择 1 到 30 天
- 实时价格计算 - 根据选择显示总费用
- 到期日期显示 - 计算并显示服务何时到期
- 当前使用情况显示 - 在充值前显示剩余余额/到期

示例显示：

定价配置：

- 每天的价格通过环境变量 `VITE_TOPUP_PRICE_PER_DAY` 配置
- 默认：每天下 $10 美元
- 货币通过 `VITE_CURRENCY_CODE` 设置

步骤 **2**：账单信息

客户提供交易的联系信息：

- 名字
- 姓氏
- 电子邮件地址

这些信息用于：

- 发票生成
- 支付收据电子邮件
- 交易记录
- 退款处理（如有需要）

步骤 **3**：支付

通过 供应商托管的支付表单（Stripe Elements, PayPal SDK）进行安全支付处理。

支持的支付方式：

- 信用卡（Visa, Mastercard, Amex）
- 借记卡
- 数字钱包（Apple Pay, Google Pay, PayPal）□□□□□□□□□

安全特性：

- 符合 PCI 的支付供应商集成
- OmniCRM 中不存储卡详细信息
- 支持 3D Secure 身份验证
- 加密支付传输

支付流程：

1. 显示安全支付表单并输入卡信息
2. 客户输入支付详细信息
3. 通过配置的供应商处理支付
4. 卡立即收费
5. 处理支付成功/失败

注意

如果支付成功但充值配置失败（例如，网络错误，OCS 无法访问），系统会自动向客户的支付方式发起 全额退款。

步骤 **4**：完成

成功屏幕：

> 您的服务已延长。新到期日期：2025年1月17日
>
> 收据已发送至：<customer@example.com> 交易 ID: TXN-123456

失败屏幕：

如果充值失败，系统会显示错误并自动处理退款：

> 我们无法完成您的充值。您的支付已被退款。
>
> 错误：无法连接到计费系统
>
> 请重试或联系支持。

后端处理

当客户完成支付时，以下操作会自动发生：

1. 支付验证

系统验证：

- 支付意图状态为 `succeeded`
- 支付金额与选择的天数匹配（`days × price_per_day`）
- 支付意图未被处理过（防止双重充值）

2. 充值□作

```
- API 端点: POST /oam/topup_dongle
- 验证 service_uuid 和 IMSI
- 调用 OCS/CGRateS 添加余额
- 创建配置作业（play_topup_hotspot）
```

3. 记录创建

系统创建多个数据库记录：

- **HotspotTopup** 记录 - 跟踪充值交易
  - payment_intent_id

    - service_uuid
    - imsi
    - 购买的天数
    - topup_amount
    - 状态（成功/失败/退款）
- 交易记录 - 财务交易
    - 标题："热点充值 - 7 天"
    - 金额：topup_amount（正数）
    - 链接到 service_id 和 customer_id
- 发票记录 - 支付发票
    - 包含充值交易
    - 立即标记为已支付
    - 支付参考：Stripe payment_intent_id
- 支付交易 - 抵消信用交易
    - 标题："支付 [发票标题]"
    - 金额：topup_amount（负数 - 信用）
    - 将发票支付链接到客户账户

4. 配置作业

创建一个配置作业，使用 playbook `play_topup_hotspot`，其：

- 接受 `days` 变量
- 连接到 OCS/CGRateS API
- 为账户添加余额
- 更新服务到期日期

示例 playbook 结构：

```yaml
- name: 为热点服务充值
  hosts: localhost
  tasks:
    - name: 向 OCS 添加余额
      uri:
        url: "{{ ocs_api_url }}/add_balance"
        method: POST
        body:
          imsi: "{{ imsi }}"
          days: "{{ days }}"
          service_uuid: "{{ service_uuid }}"
```

低余额通知

系统可以在客户余额低时发送自动通知：

SMS 通知：

当 OCS 事件触发时（`Action_Balance_Low`，`Action_Balance_Out`，`Action_Balance_Expired`）：

电子邮件通知：

在 OCS/CGRateS 行动计划中配置以发送余额警报。

通知触发器：

- `Action_Balance_Low` - 余额低于阈值（例如，剩余 2 天）
- `Action_Balance_Out` - 余额耗尽
- `Action_Balance_Expired` - 服务到期

每个通知都包含充值门户链接，以便客户轻松访问。

故障排除

常见问题

“支付系统不可用”

- 原因：Stripe 库加载失败或无效的可发布密钥
- 解决方案：
  - 检查 `VITE_STRIPE_PUBLISHABLE_KEY` 是否设置正确
  - 验证 Stripe 账户是否处于活动状态
  - 检查浏览器控制台是否有 JavaScript 错误

“充值失败。退款中...”

- 原因：配置作业失败（OCS 无法访问，playbook 错误等）
- 解决方案：
  - 检查配置日志：`GET /crm/provision/provision_id/<id>`
  - 验证 OCS/CGRateS API 是否可访问
  - 检查 playbook `play_topup_hotspot.yaml` 是否有错误
  - 检查 Ansible 日志

“支付意图已处理”

- 原因：客户尝试重用相同的支付（例如，成功后刷新）
- 解决方案：这是为了防止双重计费而设计的。客户如有需要应重新开始充值。

“支付意图金额不匹配”

- 原因：UI 计算与后端验证之间不匹配
- 解决方案：
  - 验证 `VITE_TOPUP_PRICE_PER_DAY` 是否与后端期望一致（默认 $10）
  - 检查货币配置是否一致
  - 清除浏览器缓存并重试

监控充值

查看充值记录：

查询 `HotspotTopup` 表以查看所有充值尝试：

```
SELECT
  hotspot_topup_id,
  service_uuid,
  days,
  topup_amount,
  status,
  payment_intent_id,
  created
FROM hotspot_topup
WHERE status = 'Failed'
ORDER BY created DESC;
```

检查配置状态：

```
GET /crm/provision/provision_id/<provision_id>
```

显示充值配置作业的详细状态。

Stripe 控制面板：

在您的 Stripe 控制面板中监控支付、退款和失败交易，地址为 <https://dashboard.stripe.com>

安全考虑

支付安全：

- 所有卡数据由 Stripe 处理（符合 PCI Level 1）
- OmniCRM 数据库中不存储敏感支付数据
- 支付意图防止未经授权的收费
- 客户端和服务器端均进行金额验证

防欺诈：

- 重复支付意图检测防止双重计费
- IP 地址跟踪以进行使用相���性

- IMSI 验证确保充值到正确的服务
- 自动退款限制财务风险

访问控制：

- 充值门户□公开的（按设计 - 客户需要访问）
- 使用端点需要有效的服务识别（IP 或 IMSI）
- 后端验证防止任意服务充值
- 管理员可以通过 CRM 界面查看所有充值记录

最佳实践

对于运营商：

1. 测试退款流程 - 定期测试失败场景以确保退款正常工作
2. 监控失败的充值 - 设置高失败率的警报
3. 保持 **playbooks** 简单 - 充值 playbooks 应快速且可靠
4. 验证 **OCS** 连接性 - 确保 OCS API 始终可访问
5. 审查定价 - 根据需要更新 `VITE_TOPUP_PRICE_PER_DAY`

对于客户：

1. 收藏充值 **URL** - 需要时快速访问
2. 保存低余额通知 - SMS 包含直接链接
3. 保持电子邮件更新 - 收据发送至档案中的电子邮件
4. 出行前检查到期 - 在离开覆盖区域前充值

对于开发人员：

1. 处理 **Stripe webhooks** - 实现 webhook 处理程序以获取支付状态更新
2. 实现幂等性 - 在处理之前始终检查 payment_intent_id
3. 广泛记录 - 充值失败需要详细的故障排除信息
4. 测试错误路径 - 验证退款自动化□否正常工作
5. 监控性能 - 配置��询应在 <5 秒内完成

相关文档

- `payments_process` - 一般支付处理
- `concepts_provisioning` - 配置系统概述
- `Payment System Guide <payment_system_guide>` - 支付供应商集成详细信息
- `payments_transaction` - 交易管理
- `payments_invoices` - 发票处理

完整产品生命周期指南

本指南提供了 OmniCRM 中产品生命周期的端到端流程，从创建产品定义到服务配置、添加附加组件和取消配置。我们将涵盖定价策略、Ansible 集成，并在整个过程中提供实际示例。

概述：产品到服务的旅程

OmniCRM 中产品的生命周期遵循以下阶段：

1. 产品定义 - 管理员创建包含定价和配置规则的产品模板
2. 服务创建 - 客户订购产品，系统配置服务实例
3. 服务生命周期 - 客户使用服务，添加附加组件/充值，修改服务
4. 取消配置 - 服务终止，释放资源

理解定价：批发与零售

OmniCRM 中的每个产品和服务都有两个定价维度：批发和零售。

批发成本

批发成本代表交付服务的实际成本：

- 基础设施和带宽成本
- 许可费用
- 设备成本
- 运营费用

零售成本

零售成本□向客户收取的金额。

设置成本

批发和零售都有一次性配置费用的变体：

- `wholesale_setup_cost` - 您的配置成本
- `retail_setup_cost` - 向客户收取的激活费用

示例：

```
{
  "retail_cost": 15.00,
  "wholesale_cost": 5.00,
  "retail_setup_cost": 0.00,
  "wholesale_setup_cost": 1.00
}
```

阶段 1：创建产品定义

产品定义要配置的内容和如何向客户收费的模板。

创建移动 SIM 产品

让我们创建一个每月提供 20GB 数据的预付费移动 SIM 产品。

步骤 1：导航到产品管理

从管理员 UI，转到 产品 → 创建产品。

步骤 2：定义基本信息

```
{
  "product_name": "预付费移动 20GB",
  "product_slug": "prepaid-mobile-20gb",
  "category": "standalone",
  "service_type": "mobile",
  "enabled": true,
  "icon": "fa-solid fa-sim-card",
  "comment": "预付费移动 SIM，提供 20GB 数据，无限通话和短信"
}
```

字段说明：

- `product_name` - 客户可见的名称，显示在目录中
- `product_slug` - 用于 API 调用和链接的 URL 安全标识符
- `category` - "standalone" 表示这创建了一个新服务（与附加组件/捆绑相比）
- `service_type` - 将相关产品分组，用于附加组件过滤
- `enabled` - 必须为 true，才能订购产品
- `icon` - 在 UI 中显示的 FontAwesome 图标
- `comment` - 供员工参考的内部备���

步骤 **3**：设置定价

```
{
  "retail_cost": 15.00,
  "wholesale_cost": 5.00,
  "retail_setup_cost": 0.00,
  "wholesale_setup_cost": 1.00,
  "contract_days": 30
}
```

定价细分：

- 每位客户的月收入：£15.00
- 每月交付成本：£5.00
- 每月利润率：£10.00（200% 加价，67% 利润率）
- 设置利润：-£1.00（补贴以吸引客户）
- 合同期限：30 天（每月续订）

步骤 **4**：定义客户资格

```
{
  "residential": true,
  "business": false,
  "customer_can_purchase": true,
  "available_from": "2025-01-01T00:00:00Z",
  "available_until": null
}
```

- 住宅客户可以订购

- 商业客户不能（不同的产品线）
- 自助购买已启用
- 从 2025 年 1 月 1 日起可用
- 没有结束日期（持续优惠）

步骤 **5**：配置自动续订

```
{
  "auto_renew": "prompt",
  "allow_auto_renew": true
}
```

- `"prompt"` - 在购买时询问客户□否希望自动续订
- `"true"` - 自动续订，无需询问
- `"false"` - 永不自动续订（仅手动充值）
- `allow_auto_renew: true` - 客户可以稍后启用/禁用自动续订

步骤 **6**：指定库存要求

库存要求定义在配置此产品时必须分配哪些物理或虚拟资源。这□将您的产品目录与您的 库存管理系统 `<administration_inventory>` 连接起来的关键步骤。

```
{
  "inventory_items_list": "['SIM Card', 'Mobile Number']"
}
```

什么是库存项目？

库存项目□存储在 OmniCRM 库存系统中的可追踪资源。每个项目都有：

- 类型 - 由库存模板定义（例如，“SIM 卡”，“手机号码”，“调制解调器”）
- 唯一属性 - 序列号、MAC 地址、电话号码等
- 状态 - 库存中、已分配、退役等
- 位置 - 物理或逻辑位置

库存要求的工作原理：

`inventory_items_list` —一个包含库存类型名称的 Python 列表（作为字符串）。每个名称必须完全匹配现有的`库存模板 <administration_inventory>` 名称。

示例库存要求：

```
# 移动 SIM 产品
inventory_items_list: "['SIM Card', 'Mobile Number']"

# 固定互联网服务
inventory_items_list: "['Modem Router', 'Static IP Address']"

# 数字服务（没有物理项目）
inventory_items_list: "[]"

# 固定无线与 CPE
inventory_items_list: "['Fixed Wireless CPE', 'IPv4 Address',
'IPv6 Prefix']"
```

库存选择器流程

当用户配置具有库存要求的产品时，系统强制执行强制选择过程：

1. 点击配置按钮

选择产品后，用户点击“配置”。系统会检查 `inventory_items_list`，而不立即配置。

2. 库存选择器模态框出现

如果需要库存，将出现一个模态对话框，每种库存类型都有一个单独的下拉菜单：

3. 过滤可用库存

每种库存类型的下拉菜单仅显示以下项目：

- 正确类型 - 完全匹配库存模板名称
- 可用状态 - `item_state` 为 "New" 或 "In Stock"（而不 "Assigned" 或 "Damaged"）
- 未分配 - `service_id` 和 `customer_id` 为 NULL
- 在指定位置有库存 - 可选择按仓库/商店位置过滤

示例下拉选项：

对于“SIM 卡”库存类型，下拉菜单可能显示：

每个选项显示：

- 库存 ID 或参考编号
- 主要标识符（`itemtext1` - 例如，SIM 的 ICCID，电话的号码）
- 当前地点（`item_location`）

4. 选择必需项以继续

关键规则：在选择所有必需的库存项目之前，无法继续配置。

- “继续”按钮在所有下拉菜单都有选择之前被禁用
- 用户必须为每种库存类型选择一个项目
- 系统在继续之前验证选择

5. 选定的库存传递给 Ansible

一旦用户点击“继续”，所选库存 ID 将作为变量传递给 Ansible 剧本：

```
# 用户选择：
# - SIM 卡 inventory_id: 5001
# - 手机号码 inventory_id: 5002

# 传递给 Ansible 的变量：
{
    "product_id": 42,
    "customer_id": 123,
    "SIM Card": 5001,              # 库存 ID
    "Mobile Number": 5002,          # 库存 ID
    "access_token": "eyJ..."
}
```

注意：变量名称与库存类型完全匹配。剧本使用 `hostvars[inventory_hostname]['SIM Card']` 访问库存 ID。

6. 剧本获取完整库存详细信息

Ansible 剧本使用库存 ID 获取完整详细信息：

```
- name: 从库存获取 SIM 卡详细信息
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ hostvars[inventory_hostname]['SIM
Card'] }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
  register: api_response_sim

- name: 提取 ICCID 和 IMSI
  set_fact:
    iccid: "{{ api_response_sim.json.itemtext1 }}"
    imsi: "{{ api_response_sim.json.itemtext2 }}"
```

现在剧本拥有所有 SIM 详细信息（ICCID、IMSI 等）以在 HSS 中配置用户。

7. 库存状态更改为“已分配”

在创建服务记录后，剧本更新库存以将其链接到服务：

```
- name: 将 SIM 卡分配给服务
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ hostvars[inventory_hostname]['SIM
Card'] }}"
    method: PATCH
    body:
      {
        "service_id": "{{
service_creation_response.json.service_id }}",
        "customer_id": "{{ customer_id }}",
        "item_state": "Assigned"
      }
```

重要：库存分配发生在剧本执行期间作为特定任务，而不□在点击配置按钮时。这意味着：

- 双重分配风险：在点击“配置”和库存被分配之间，另一个用户理论上可以选择相同的库存项目
- 最佳实践：对于高容量操作，实施库存锁定或使用数据库事务
- 失败时回滚：如果剧本在库存分配之前失败，库存将保持未分配状态并可供重用

为什么不早点分配?

在点击“配置”时不分配库存□因为:

1. 需要服务 ID: 在剧本中创建服务之前，`service_id` 不存在
2. 回滚简单性: 如果配置早期失败（例如，OCS 账户创建失败），则无需清理库存
3. 灵活性: 剧本可以根据条件逻辑决定□否分配库存

处理失败的配置:

当配置在库存分配后失败时，救援块应释放库存:

```
rescue:
  - name: 失败时释放库存
    uri:
      url: "{{ crm_config.crm.base_url }}/crm/inventory/inventory_id/{{ hostvars[inventory_hostname]['SIM Card'] }}"
      method: PATCH
      body:
        {
          "service_id": null,
          "customer_id": null,
          "item_state": "In Stock"
        }
    when: service_id is defined  # 仅在服务被创建时
```

这确保库存不会因不存在或失败的服务而留在“已分配”状态。

当库存列表为空时

如果 `inventory_items_list: "[]"`（空列表），则完全跳过库存选择器，立即进行配置。这在以下情况下很常见:

- 数字产品 - 软件许可证、VPN 帐户
- 服务附加组件 - 不需要新硬件的数据充值
- 虚拟服务 - 不消耗可追踪资源

示例: “5GB 数据提升”附加组件具有 `inventory_items_list: "[]"`，因为它只□向现有服务添加余额，而无需新硬件。

库存模板设置

在 `inventory_items_list` 中使用库存类型之前，您必须创建库存模板：

1. 导航到 管理 → 库存 → 模板
2. 创建具有确切名称的模板（例如，“SIM 卡”）
3. 定义字段：
   - `itemtext1_label`: "ICCID"
   - `itemtext2_label`: "IMSI"
   - `itemtext3_label`: "PUK Code"
4. 将此类型的库存项目添加到库存中

有关创建和管理库存模板的完整详细信息，请参见`库存管理 <administration_inventory>`。

相同类型的多个项目

虽然 `inventory_items_list` 一个数组，但拥有重复类型（例如，`"['SIM Card', 'SIM Card']"`）不推荐，因为这可能会导致 UI 和剧本变量命名中的混淆。

对于需要多个相似项目的场景：

选项 **1**：创建不同的库存模板名称

```
# 双 SIM 手机服务
inventory_items_list: "['Primary SIM Card', 'Secondary SIM Card',
'Mobile Number']"
```

创建单独的模板：“Primary SIM Card”和“Secondary SIM Card”，字段相同但名称不同。

选项 **2**：使用单个捆绑库存项目

```
# 双 SIM 套件
inventory_items_list: "['Dual SIM Kit', 'Mobile Number']"
```

其中“Dual SIM Kit”库存模板具有两个 SIM 的字段（`itemtext1`：主 ICCID，`itemtext2`：次 ICCID 等）。

常见库存场景

移动服务：

```
inventory_items_list: "['SIM Card', 'Mobile Number']"
```

- SIM 卡：带有 ICCID/IMSI 的物理或 eSIM
- 手机号码：电话号码（MSISDN）

固定互联网：

```
inventory_items_list: "['Modem Router', 'Static IP Address']"
```

- 调制解调器路由器：带有 MAC 地址的 CPE 设备
- 静态 IP 地址：来自地址池的 IPv4

固定无线：

```
inventory_items_list: "['Fixed Wireless CPE', 'IPv4 Address',
'IPv6 Prefix']"
```

- CPE：客户场所设备（天线、调制解调器）
- IPv4：公共 IP 地址
- IPv6 前缀：/56 或 /64 前缀

注意：预约和调度不是库存项目。使用单独的调度/日历系统进行安装预约。

VoIP 服务：

```
inventory_items_list: "['DID Number']"
```

- DID 号码：直接拨入电话号码

注意： SIP 用户名、密码和帐户配置由配置剧本程序生成，而不□从库存中选择。

GPON/光纤：

```
inventory_items_list: "['ONT Device', 'GPON Port', 'IPv4 Address',
'Fiber Drop Cable']"
```

- ONT 设备：带有序列号的光网络终端
- GPON 端口：与光纤连接的 OLT 上的特定端口
- IPv4 地址：公共或私有 IP
- 光纤下缆：从街道到场所的物理光纤电缆（跟踪资产管理）

设备租赁：

```
inventory_items_list: "['Rental Modem']"
```

- 跟踪哪个调制解调器与哪个客户
- 取消时恢复设备很重要

为什么库存要求很重要

1. 防止双重分配

没有库存跟踪，您可能会意外地：

- 将同一 SIM 卡分配给两个客户
- 将相同的 IP 地址分配给多个服务
- 将相同的设备序列号发货到不同位置

库存选择器确保每个项目仅分配给一个服务。

2. 审计跟踪

库存分配创建完整的审计跟踪：

- 哪个 SIM 卡与哪个客户
- 何时分配
- 哪个服务使用哪个电话号码
- 设备历史（谁拥有它，何时，出于什么服务）

3. 资源规划

跟踪库存水平：

- 当 SIM 卡即将用完时发出警报
- 在缺货之前重新订购
- 根据 CPE 可用性计划技术人员日程
- 管理 IP 地址空间分配

4. 成本跟踪

将批发成本链接到特定项目：

- 跟踪每个 SIM 卡的成本
- 计算设备折旧
- 识别丢失或被盗的项目
- 准确的 COGS（销售成本）

5. 取消配置

当服务被取消时，库存可以：

- 释放回库存（SIM 卡、调制解调器）
- 退役（损坏设备）
- 退还给供应商（租赁设备）
- 在宽限期内保留（电话号码在释放之前）

故障排除库存选择器问题

问题：“没有可用库存”消息出现

原因：

- 数据库中不存在所需类型的库存项目
- 所有项目已“分配”给其他服务
- 项目标记为“损坏”或“停用”
- 库存模板名称不完全匹配（区分大小写）

解决方案：

1. 验证库存模板是否存在：管理 → 库存 → 模板

2. 检查模板名称□否完全匹配（包括空格、大小写）
3. 添加此类型的库存项目：管理 → 库存 → 添加项目
4. 验证项目□否处于“新建”或“库存中”状态
5. 检查项目□否未分配（`service_id` 应为 NULL）

问题：库存选择器未出现

原因：

- `inventory_items_list` 为空：`"[]"`
- `inventory_items_list` 为 NULL 或未设置
- 产品类别为“附加组件”，并继承父服务库存

解决方案：

- 如果需要库存，请设置 `inventory_items_list: "['Type1', 'Type2']"`
- 验证产品定义□否正确保存
- 检查 API 响应以确认产品包含 inventory_items_list

问题：剧本失败，出现“未找到库存”

原因：

- 剧本引用错误的变量名称
- 库存 ID 未正确传递
- 在选择和配置之间删除了库存

解决方案：

- 验证剧本使用正确的变量：`hostvars[inventory_hostname]['SIM Card']`
- 检查变量□否为整数：`{{ hostvars[inventory_hostname]['SIM Card'] | int }}`
- 在剧本中添加缺少库存的错误处理

有关创建模板、添加项目和管理库存水平的完整详细信息，请参见`库存管理 <administration_inventory>`。

步骤 **7**：定义功能和条款

功能和条款□面向客户的营销和法律内容，帮助客户理解他们所购买的内容及相关义务。

```
{
  "features_list": "20GB 高速数据。无限通话和短信。包括欧盟漫游。无合同。30 天到期",
  "terms": "信用在 30 天后到期。数据、通话和短信仅在到期期间有效。适用公平使用政策。有关完整条款，请参见网站。"
}
```

目的和商业价值

功能列表 - 营销与销售：

功能列表服务于多个关键业务功能：

1. 产品差异化 - 帮助客户快速比较产品并选择合适的产品
    - “预付费移动 20GB”与“预付费移动 50GB” - 功能清楚地显示差异
    - 如果没有功能，客户只会看到价格，错过价值主张
2. 营销沟通 - 关键卖点显著展示
    - “包括欧盟漫游”吸引国际旅行者
    - “无合同”吸引不愿意承诺的客户
    - 功能驱动购买决策
3. 客户期望 - 明确设定包含的内容
    - 减少支持电话（“这包括通话吗？”→清楚列出���
    - 防止误解和退款请求
    - 通过透明度建立信任
4. 自助服务 - 使客户能够自选合适的产品
    - 客户阅读功能，理解产品，做出明智选择
    - 减少销售人员的解释需求
    - 加快购买流程
5. **SEO** 和可发现性 - 功能可以被索引以供搜索
    - 客户搜索“无限通话移动计划”→产品出现
    - 提高产品目录的可搜索性

条款和条件 - 法律与合规：

条款服务于法律和运营目的：

1. 法律保护 - 保护企业免受争议和责任
    - “信用在 30 天后到期” - 客户不能在 31 天时要求退款
    - “适用公平使用政策” - 防止滥用（在移动计划上连接整个办公室）

    - 创建具有约束力的协议
2. 期望管理 - 防止客户不满
    - “仅在到期期间有效” - 客户知道使用截止日期
    - “不可退款”（对于附加组件） - 防止欺诈性购买
    - 减少退款和投诉
3. 法规合规 - 符合法律要求
    - 消费者保护法要求明确条款
    - 电信法规要求披露
    - 可以引用 GDPR/隐私条款
4. 运营边界 - 定义服务范围和限制
    - “受网络覆盖限制” - 不对死区负责
    - “速度可能会有所不同” - 管理对“高达”速度的期望
    - “���备必须归还” - 确保租赁设备的回收
5. 审计跟踪 - 证明客户已被告知
    - 客户在购买时接受条款
    - 系统记录接受时间戳
    - 在争议或法律程序中可辩护

实际示例：

客户购买“无限通话和短信”计划，然后用于电话营销（每天 10,000 个电话）。如果没有条款：

- 客户：“你说无限！”
- 提供商：“我们的意思□个人使用.....”
- 客户：“这不□你宣传的内容！”
- 结果：争议，潜在的监管投诉，品牌损害

有了条款：“适用公平使用政策。服务仅供个人使用。禁止商业使用。”

- 提供商：指向客户接受的条款
- 客户不能声称无知
- 暂停服务的法律依据
- 争议以提供商有利的方式解决

功能列表格式：

理解正确的格式至关重要，因为不当格式会破坏 **UI** 显示。功能可能会作为一长串字符串出现，而不□以项目符号显示，或者根本不显示。

`features_list` 字段可以以两种方式格式化：

选项 **1**：以句点分隔的字符串（推荐）

功能由句点和空格（". "）分隔。UI 在此分隔符上拆分并将每个功能呈现为项目符号。

为什么选择这种格式？

- 编辑简单 - 只需在它们之间输入句点
- 无���转义特殊字符
- 在所有 UI 组件中可靠工作
- 易于更新而不会破坏 JSON 语法

正确与错误：

选项 **2**：**JSON** 数组字符串

```
"['20GB 高速数据', '无限通话和短信', '包括欧盟漫游']"
```

UI 也可以解析 JSON 数组。请注意，这□一种包含 **JSON** 的字符串，而不□数据库中的实际 JSON 数组。

为什么存在这种格式？

- 允许功能中包含句点（例如，“高达 100Mbps。受可用性限制。”）
- 从脚本/API 进行程序生成更容易
- 从使用数组的外部产品目录导入

重要：这必须□有效的 Python 列表语法作为字符串。每个项目周围用单引号，整个字符串用双引号。

使用哪种格式？

- 以句点分隔 - 用于在 UI 中手动创建产品（更简单，出错率更低）
- **JSON** 数组 - 用于基于 API/脚本的产品创建（对于复杂功能更强大）

这两种格式在 UI 中产生相同的输出 - 只□影响您输入数据的方式。

功能在 **UI** 中出现的位置：

1. 产品目录（客户视图）

当客户浏览可用产品时，功能显示为每个产品卡上的项目符号：

2. 产品详细信息页面

点击“查看详细信息”显示完整的产品信息，包括：

- 产品名称和图��
- 定价（每月费用、设置费用）
- 完整功能列表（项目符号）
- 条款和条件（见下文）
- 可用性和资格

3. 配置确认

在配置期间，功能显示供用户在确认之前查看：

> 功能：• 20GB 高速数据 • 无限通话和短信 • 包括欧盟漫游 • 无合同 • 30 天到期
>
> 成本：£15.00/月 设置：£0.00
>
> [取消] [确认并配置]

4. 服务详细信息（配置后）

服务激活后，功能显示在服务详细信息页面供客户参考。

条款和条件格式：

`terms` 字段□可以包含换行符的纯文本：

条款在 **UI** 中出现的位置：

1. 产品详细信息页面

条款显示在一个可折叠的部分中，点击时展开：

2. 订单确认

在配置期间，复选框要求用户接受条款：

> [配置] 按钮在选中之前被禁用

3. 发票

服务条款可能作为脚注包含在发票中以便于理解。

最佳实践：

- 功能：保持简洁（每个不超过 50 个字符），关注关键好处
- 条款：包括关键法律要求、到期政策、公平使用政策
- 两者：产品更改时更新以保持客户知情

步骤 8：链接 **Ansible** 配置剧本

```
{
  "provisioning_play": "play_local_mobile_sim",
  "provisioning_json_vars": "{
    \"days\": 30,
    \"data_gb\": 20,
    \"voice_minutes\": \"unlimited\",
    \"sms_count\": \"unlimited\"
  }"
}
```

- `provisioning_play` - Ansible 剧本的名称（不带 .yaml 扩展名）
- `provisioning_json_vars` - 传递给剧本的默认变量
- 剧本必须存在于：`OmniCRM-API/Provisioners/plays/play_local_mobile_sim.yaml`

完整产品定义

```
{
  "product_name": "预付费移动 20GB",
  "product_slug": "prepaid-mobile-20gb",
  "category": "standalone",
  "service_type": "mobile",
  "enabled": true,
  "icon": "fa-solid fa-sim-card",
  "comment": "预付费移动 SIM，提供 20GB 数据，无限通话和短信",

  "retail_cost": 15.00,
  "wholesale_cost": 5.00,
  "retail_setup_cost": 0.00,
  "wholesale_setup_cost": 1.00,
  "contract_days": 30,

  "residential": true,
  "business": false,
  "customer_can_purchase": true,
  "available_from": "2025-01-01T00:00:00Z",
  "available_until": null,

  "auto_renew": "prompt",
  "allow_auto_renew": true,

  "inventory_items_list": "['SIM Card', 'Mobile Number']",

  "features_list": "[
    '20GB 高速数据',
    '无限通话和短信',
    '包括欧盟漫游',
    '无合同',
    '30 天到期'
  ]",
  "terms": "信用在 30 天后到期。数据、通话和短信仅在到期期间有效。适用公平使用政策。",

  "provisioning_play": "play_local_mobile_sim",
  "provisioning_json_vars": "{
    \"days\": 30,
    \"data_gb\": 20,
    \"voice_minutes\": \"unlimited\",
    \"sms_count\": \"unlimited\"
```

```
  }"
}
```

创建附加组件产品

附加组件增强或修改现有服务。它们分为两种类型：虚拟附加组件（没有物理资源）和 硬件附加组件（需要库存）。

示例 **1**：虚拟附加组件（**5GB** 数据提升）

一个数字附加组件，向现有移动服务添加数据：

```
{
  "product_name": "5GB 数据提升",
  "product_slug": "5gb-data-boost",
  "category": "addon",
  "service_type": "mobile",
  "enabled": true,
  "icon": "fa-solid fa-plus",
  "comment": "向现有移动服务添加 5GB 额外数据",

  "retail_cost": 5.00,
  "wholesale_cost": 1.50,
  "retail_setup_cost": 0.00,
  "wholesale_setup_cost": 0.00,
  "contract_days": 0,

  "residential": true,
  "business": true,
  "customer_can_purchase": true,

  "auto_renew": "false",
  "allow_auto_renew": false,

  "inventory_items_list": "[]",
  "relies_on_list": "",

  "features_list": "5GB 高速数据。有效期 7 天",
  "terms": "数据在 7 天后或用尽时到期。不可退款。",

  "provisioning_play": "play_topup_charge_then_action",
  "provisioning_json_vars": "{
    \"data_gb\": 5,
    \"days\": 7
  }"
}
```

示例 **2**：硬件附加组件（调制解调器租赁）

一个为现有光纤服务提供物理设备的附加组件：

```
{
  "product_name": "WiFi 6 调制解调器租赁",
  "product_slug": "wifi6-modem-rental",
  "category": "addon",
  "service_type": "internet",
  "enabled": true,
  "icon": "fa-solid fa-router",
  "comment": "为光纤服务添加 WiFi 6 调制解调器 - 租赁",

  "retail_cost": 10.00,
  "wholesale_cost": 3.00,
  "retail_setup_cost": 0.00,
  "wholesale_setup_cost": 45.00,
  "contract_days": 30,

  "residential": true,
  "business": true,
  "customer_can_purchase": true,

  "auto_renew": "true",
  "allow_auto_renew": true,

  "inventory_items_list": "['Rental Modem']",
  "relies_on_list": "",

  "features_list": "WiFi 6 (802.11ax)。双频 2.4GHz + 5GHz。支持多达 40 个设备。家长控制",
  "terms": "设备租赁。必须在服务取消时归还，否则将收取 £150 的更换费用。设备仍为提供商的财产。",

  "provisioning_play": "play_addon_assign_modem",
  "provisioning_json_vars": "{
    \"device_type\": \"modem_router\",
    \"requires_configuration\": true
  }"
}
```

附加组件的关键区别：

- `category: "addon"` - 应用于现有服务，而不□独立的
- `contract_days: 0`（虚拟）或 `30`（定期租赁） - 计费频率
- `inventory_items_list: "[]"`（虚拟）或 `"['Rental Modem']"`（硬件） - 物理资源

- `auto_renew: "false"`（一次性）或 `"true"`（租赁） - 递归行为
- `relies_on_list: ""` - 为空表示适用于任何匹配 `service_type` 的服务

为什么硬件附加组件需要库存：

硬件附加组件需要 `inventory_items_list`，因为：

1. 跟踪设备 - 知道哪个调制解调器与哪个客户
2. 防止缺货 - 如果没有库存中的调制解调器，则无法配置附加组件
3. 回收 - 当客户取消时，知道要回收哪些设备
4. 成本跟踪 - 将批发成本链接到特定序列号
5. 折旧 - 跟踪设备在租赁期间的价值
6. 保修 - 通过序列号识别缺陷单元

附加组件配置流程与库存：

当客户将“WiFi 6 调制解调器租赁”添加到其光纤服务时：

1. 选择附加组件 - 客户点击“添加到服务”
2. 库存选择器出现 - 与独立服务相同：
3. 处理付款 - 收取 £10.00 的月租费
4. 调制解调器分配 - 库存更新：
   - `service_id`：链接到光纤服务
   - `customer_id`：链接到客户
   - `item_state`："Assigned"
5. 触发发货 - 通知履行系统发货调制解调器
6. 安装 - 客户收到调制解调器，插入 ONT
7. 递归计费 - 每月收取 £10，直到附加组件被取消

取消配置硬件附加组件：

当客户取消调制解调器租赁时：

1. 发起取消 - 客户点击“移除附加组件”
2. 开始退货流程：
   - 发送带有退货说明的电子邮件
   - 生成预付费运输标签

    - 在罚款之前有 14 天的宽限期
3. 设备归还：
    - 库存更新：`item_state` = "In Stock"（经过翻新后）
    - 或 `item_state` = "Damaged"（如果有缺陷）
    - 一旦翻新，链接到下一个客户
4. 未归还：
    - 14 天后，收取 £150 的更换费用
    - 库存标记：`item_state` = "Lost"
    - 收回批发成本（£45）+ 更换价值

附加组件的定价：

附加组件的定价可以与独立服务不同：

- 虚拟附加组件通常没有设置费用
- 硬件附加组件可能有设备的批发设置费用
- 定期租赁附加组件使用 `contract_days` 作为计费频率

阶段 2：配置过程

当客户订购“预付费移动 20GB”产品时，OmniCRM 通过 Ansible 协调配置。

配置流程图

客户订购 → 库存选择 → 创建配置作业 ↓ ↓ 付款授权 ← 变量组装 ← 执行 Ansible 剧本 ↓ ↓ 创建服务记录 → OCS 账户设置 → 库存分配 → 服务激活

步骤配置流程

1. 客户发起订单

从客户页面：

- 员工点击“添加服务”
- 从产品轮播中选择“预付费移动 20GB”
- 显示产品详细信息和定价

2. 库存选择

系统提示所需库存：

- **SIM** 卡 - 下拉菜单显示库存中可用的 SIM 卡
  - 示例：“SIM-00123 - ICCID: 8944...”
- 手机号码 - 下拉菜单显示可用的电话号码
  - 示例：“+44 7700 900123”

员工或客户从可用库存中选择项目。

3. 确认定价

系统显示最终定价：

- 设置费用：£0.00（免费激活）
- 每月费用：£15.00
- 今日到期：£15.00（第一个月）
- 续订日期：从今天起 30 天

如果启用了自动续订提示，客户选择：

☐ 每 30 天自动续订此服务

4. 点击配置按钮

当点击“配置”时，API：

- 创建状态为“运行”（status=1）的 `Provision` 记录
- 合并来自产品 + 请求 + 库存选择的变量
- 生成后台线程以执行 Ansible 剧本
- 返回 provision_id 以供 UI 状态跟踪

5. 变量组装

系统合并来自多个来源的变量：

来自产品：

```
{
  "days": 30,
  "data_gb": 20,
  "voice_minutes": "unlimited",
  "sms_count": "unlimited"
}
```

来自请求:

```
{
  "product_id": 42,
  "customer_id": 123,
  "SIM Card": 5001,
  "Mobile Number": 5002
}
```

系统添加:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "initiating_user": 7
}
```

最终传递给 **Ansible** 的变量:

```
{
    "product_id": 42,
    "customer_id": 123,
    "SIM Card": 5001,
    "Mobile Number": 5002,
    "days": 30,
    "data_gb": 20,
    "voice_minutes": "unlimited",
    "sms_count": "unlimited",
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "initiating_user": 7
}
```

6. 执行 Ansible 剧本

剧本 `play_local_mobile_sim.yaml` 使用这些变量执行。

理解 Ansible 配置剧本

让我们检查一个实际的配置剧本，以了解幕后发生的事情。

移动 SIM 配置剧本示例

位��: `OmniCRM-API/Provisioners/plays/play_local_mobile_sim.yaml`

高层结构:

```
- name: 移动 SIM 配置
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: 主块
      block:
        # 1. 加载配置
        # 2. 从 API 取产品详细信息
        # 3. 从 API 取客户详细信息
        # 4. 从 API 取库存详细信息
        # 5. 在 OCS 中创建账户（CGRateS）
        # 6. 向 OCS 添加余额和配额
        # 7. 在 CRM 中创建服务记录
        # 8. 将库存分配给服务
        # 9. 记录交易
        # 10. 发送欢迎通知

      rescue:
        # 失败时回滚
        # - 删除 OCS 账户
        # - 释放库存
        # - 记录错误
```

详细剧本逐步讲解：

任务 **1**：加载配置

```
- name: 包含 crm_config 的变量
  ansible.builtin.include_vars:
    file: "../../crm_config.yaml"
    name: crm_config
```

加载系统配置，包括：

- OCS/CGRateS URL 和凭据
- CRM 基础 URL
- 租户配置

任务 **2**：获取产品详细信息

```
- name: 从 CRM API 获取产品信息
  uri:
    url: "{{ crm_config.crm.base_url }}/crm/product/product_id/{{
product_id }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
    return_content: yes
  register: api_response_product
```

这做了什么：

- 调用 `GET /crm/product/product_id/42`
- 检索完整的产品定义
- 存储在 `api_response_product` 变量中

为什么：即使我们从产品中获得了 `provisioning_json_vars`，我们仍然获取整的产品以获取：

- 最新定价（可能在订单开始时已更改）
- 服务命名所需的产品名称
- 文档所需的功能列表

- 用于利润跟踪的批发成本

任务 **3**：设置包事实

```
- name: 设置包事实
  set_fact:
    package_name: "{{ api_response_product.json.product_name }}"
    monthly_cost: "{{ api_response_product.json.retail_cost }}"
    setup_cost: "{{ api_response_product.json.retail_setup_cost
}}"
```

提取常用值以便于阅读。

任务 **4**：获取库存详细信息

```
- name: 从 CRM API 获取 SIM 信息
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ hostvars[inventory_hostname]['SIM
Card'] }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
  register: api_response_sim

- name: 从库存响应中设置 IMSI
  set_fact:
    imsi: "{{ api_response_sim.json.itemtext2 }}"
    iccid: "{{ api_response_sim.json.itemtext1 }}"
```

这做了什么：

- 查找 SIM 卡库存 ID 5001
- 检索 SIM 详细信息：
  - `itemtext1` = ICCID（SIM 卡号码）
  - `itemtext2` = IMSI（用户身份）
- 对手机号码库存执行相同操作（检索电话号码）

为什么这很重要：

- IMSI 需要在 HSS（家庭用户服务器）中配置用户
- ICCID 记录在服务备注中以便于故障排除
- 电话号码（MSISDN）显示给客户并用于路由

任务 **5**：生成服务 **UUID**

```
- name: 生成 UUID 事实
  set_fact:
    uuid: "{{ 99999999 | random | to_uuid }}"

- name: 设置服务 UUID
  set_fact:
    service_uuid: "Local_Mobile_SIM_{{ uuid[0:8] }}"
```

这做了什么：

- 生成随机 UUID
- 创建类似 `Local_Mobile_SIM_a3f2c1d8` 的 service_uuid

为什么：

- 服务 UUID □ OCS/CGRateS 中的唯一标识符
- 用于所有计费操作
- 必须在所有服务中全局唯一

任务 **6**：在 **OCS** 中创建账户

```yaml
- name: 在 OCS 中创建账户
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    headers:
      Content-Type: "application/json"
    body:
      {
        "method": "ApierV2.SetAccount",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "ActionPlanIds": [],
          "ExtraOptions": {
            "AllowNegative": false,
            "Disabled": false
          },
          "ReloadScheduler": true
        }]
      }
  register: ocs_create_response
```

这做了什么：

- 调用 CGRateS JSON-RPC API
- 使用 service_uuid 创建新账户
- 将账户设置为活动状态（未禁用）
- 防止负余额（预付费模式）

为什么：

- OCS 账户□所有计费发生的地方
- 余额（数据、语音、短信、资金）存储在这里
- 使用情况实时跟踪和计费

任务 **7**：添加数据余额

```
- name: 向账户添加 20GB 数据余额
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Tenant": "{{ crm_config.ocs.ocsTenant }}",
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Balance": {
            "ID": "DATA_20GB_Monthly",
            "Value": 21474836480,
            "ExpiryTime": "+720h",
            "Weight": 10,
            "DestinationIDs": "*any"
          }
        }]
      }
```

这做了什么：

- 向账户添加 20GB 数据余额
- 值：21474836480 字节（20 * 1024 * 1024 * 1024）
- 在 720 小时（30 天）内到期
- 权重 10（优先消耗高权重）

任务 **8**：添加无限语音和短信

```
- name: 添加无限语音
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Account": "{{ service_uuid }}",
          "BalanceType": "*voice",
          "Balance": {
            "ID": "VOICE_Unlimited",
            "Value": 999999999,
            "ExpiryTime": "+720h"
          }
        }]
      }
```

- 添加 999,999,999 秒的语音（基本上□无限的）
- 在 30 天内到期

任务 **9**: 在 **CRM** 中创建服务记录

```
- name: 通过 API 添加服务
  uri:
    url: "{{ crm_config.crm.base_url }}/crm/service/"
    method: PUT
    body_format: json
    headers:
      Authorization: "Bearer {{ access_token }}"
    body:
      {
        "customer_id": "{{ customer_id }}",
        "product_id": "{{ product_id }}",
        "service_name": "Mobile - {{ phone_number }}",
        "service_type": "mobile",
        "service_uuid": "{{ service_uuid }}",
        "service_status": "Active",
        "service_provisioned_date": "{{ provision_datetime }}",
        "retail_cost": "{{ monthly_cost }}",
        "wholesale_cost": "{{
api_response_product.json.wholesale_cost }}",
        "icon": "fa-solid fa-sim-card"
      }
  register: service_creation_response
```

这创建了什么：

- 与客户链接的服务记录
- 通过 `service_uuid` 链接到 OCS
- 存储零售和批发成本
- 将状态设置为“活动”
- 返回 `service_id` 以供后续操作

任务 **10**：将库存分配给服务

```
- name: 将 SIM 卡分配给服务
  uri:
    url: "{{ crm_config.crm.base_url
}}/crm/inventory/inventory_id/{{ hostvars[inventory_hostname]['SIM
Card'] }}"
    method: PATCH
    body_format: json
    headers:
      Authorization: "Bearer {{ access_token }}"
    body:
      {
        "service_id": "{{
service_creation_response.json.service_id }}",
        "customer_id": "{{ customer_id }}",
        "item_state": "Assigned"
      }
```

这做了什么：

- 更新 SIM 卡库存记录
- 将 `service_id` 设置为将 SIM 链接到服务
- 将状态从“库存中”更改为“已分配”
- 对手机号码库存重复此操作

为什么：

- 跟踪哪个 SIM 分配给哪个客户
- 防止库存的双重分配
- 启用库存报告和审计

任务 **11**：记录设置费用交易

```
- name: 添加设置费用交易
  uri:
    url: "{{ crm_config.crm.base_url }}/crm/transaction/"
    method: PUT
    body_format: json
    headers:
      Authorization: "Bearer {{ access_token }}"
    body:
      {
        "customer_id": "{{ customer_id }}",
        "service_id": "{{
service_creation_response.json.service_id }}",
        "title": "{{ package_name }} - 设置",
        "description": "激活费用",
        "retail_cost": "{{ setup_cost }}",
        "wholesale_cost": "{{
api_response_product.json.wholesale_setup_cost }}"
      }
```

这做了什么：

- 记录 £0.00 的设置费用（零售）到客户
- 记录 £1.00 的批发成本
- 创建用于开票的交易记录

任务 **12**：救援块（错误处理）

```
rescue:
  - name: 在失败时删除 OCS 中的账户
    uri:
      url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
      method: POST
      body:
        {
          "method": "ApierV2.RemoveAccount",
          "params": [{
            "Account": "{{ service_uuid }}"
          }]
        }

  - name: 使配置失败
    fail:
      msg: "配置失败，已回滚 OCS 账户"
```

这做了什么：

- 如果任何任务失败，救援块执行
- 删除部分创建的 OCS 账户
- 将库存释放回“库存中”
- 使配置作业失败并显示错误消息

为什么：

- 防止 OCS 中孤立的账户
- 确保在错误时干净回滚
- 维护数据一致性

配置完成：创建了什么

成功配置后，系统已完成：

1. OCS 账户（CGRateS）：

- 账户 ID: `Local_Mobile_SIM_a3f2c1d8`
- 余额：
  - 20GB 数据（在 30 天内到期）

    - 无限语音（999M 秒，到期在 30 天内）
    - 无限短信（999M 条消息，到期在 30 天内）

2. CRM 服务记录：

- 服务 ID：1234
- 客户：John Doe（customer_id：123）
- 产品：预付费移动 20GB（product_id：42）
- 服务名称：“Mobile - +44 7700 900123”
- 服务 UUID：`Local_Mobile_SIM_a3f2c1d8`
- 状态：活动
- 每月费用：£15.00（零售），£5.00（批发）
- 利润：£10.00/月

3. 库存分配：

- SIM 卡 5001：分配给服务 1234，客户 123
- 手机号码 5002：分配给服务 1234，客户 123

4. 交易记录：

- 创建设置费用交易
- 记录第一个月费用

5. 客户现在可以：

- 在自助服务门户中查看服务
- 查看 20GB 数据余额
- 拨打电话和发送短信
- 充值或添加附加组件
- 实时查看使用情况

阶段 3：添加附加组件和充值

服务激活后，客户可以购买附加组件以增强其服务。

附加组件配置流程

假设客户已使用 20GB 配额中的 18GB，并希望购买“5GB 数据提升”附加组件。

1. 客户导航到服务

- 打开“Mobile - +44 7700 900123”服务页面
- 查看当前使用情况：20GB 中的 18GB（90%）
- 点击“添加附加组件”或“充值”

2. 系统过滤可用附加组件

仅显示以下附加组件：

- `category = "addon"`
- `service_type = "mobile"`（与服务类型匹配）
- `residential = true`（如果客户□住宅客户）
- `enabled = true`

客户看到：“5GB 数据提升 - £5.00”

3. 客户选择附加组件

- 点击“5GB 数据提升”
- 确认购买 £5.00
- 系统捕获付款授权

4. 附加组件配置启动

系统调用 `play_topup_charge_then_action.yaml`，变量如下：

```
{
    "product_id": 43,  # 5GB 数据提升产品
    "customer_id": 123,
    "service_id": 1234,  # 现有服务
    "access_token": "eyJ...",
    "data_gb": 5,  # 来自 provisioning_json_vars
    "days": 7      # 来自 provisioning_json_vars
}
```

与独立服务的关键区别：

- 包含 `service_id`（现有服务以修改）
- 不需要库存
- 不创建服务（修改现有服务）

附加组件配置剧本逐步讲解

任务 **1**：获取服务详细信息

```
- name: 从 CRM API 获取服务信息
  uri:
    url: "http://localhost:5000/crm/service/service_id/{{
service_id }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
  register: api_response_service

- name: 设置服务事实
  set_fact:
    service_uuid: "{{ api_response_service.json.service_uuid }}"
    customer_id: "{{ api_response_service.json.customer_id }}"
```

为什么：

- 需要 `service_uuid` 以向正确的 OCS 账户添加余额
- 验证服务□否存在且处于活动状态
- 确保服务属于客户

任务 **2**：向客户收费

```yaml
- name: 获取用户的默认付款方式
  uri:
    url: "http://localhost:5000/api/payments/methods/default?
customer_id={{ customer_id }}"
    method: GET
    headers:
      Authorization: "Bearer {{ access_token }}"
  register: api_response_payment_method

- name: 获取默认付款方式 ID
  set_fact:
    payment_method_id: "{{
api_response_payment_method.json.data.payment_method_id }}"

- name: 向用户收费
  uri:
    url: "http://localhost:5000/api/payments/charge"
    method: POST
    body_format: json
    headers:
      Authorization: "Bearer {{ access_token }}"
    body:
      {
        "customer_id": "{{ customer_id | int }}",
        "amount": 5.00,
        "currency": "USD",
        "payment_method_id": "{{ payment_method_id }}",
        "metadata": {
          "description": "5GB 数据提升",
          "service_id": "{{ service_id | int }}",
          "product_id": "{{ product_id | int }}",
          "invoice": true
        }
      }
  register: charge_response

- name: 确保付款成功
  assert:
    that:
      - charge_response.json.success == true
```

这做了什么:

- 查找客户的默认 Stripe 付款方式
- 向卡收费 £5.00
- 记录批发成本 £1.50 以进行利润跟踪
- 创建与服务相关的交易
- 添加到下一个发票中
- 如果付款失败，则失败配置

为什么先收费：

- 在确认付款之前不交付信用
- 防止欺诈
- 将付款与附加组件配置匹配

任务 **3**：向 **OCS** 添加数据余额

```
- name: 向账户添加 5GB 数据余额
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body_format: json
    body:
      {
        "method": "ApierV1.AddBalance",
        "params": [{
          "Account": "{{ service_uuid }}",
          "BalanceType": "*data",
          "Balance": {
            "ID": "DATA_5GB_Boost_{{ uuid }}",
            "Value": 5368709120,
            "ExpiryTime": "+168h",
            "Weight": 20
          }
        }]
      }
```

这做了什么：

- 向账户添加 5GB（5368709120 字节）
- 在 168 小时（7 天）内到期

- 权重 20（优先消耗高权重 - 提升优先于每月配额）

客户余额在附加组件后：

- 原始每月：剩余 2GB（在 25 天内到期）
- 新提升：5GB（在 7 天内到期）
- 可用总量：7GB
- 使用顺序：首先消耗提升，然后每月配额

任务 **4**：记录交易

```
- name: 添加附加组件交易
  uri:
    url: "http://localhost:5000/crm/transaction/"
    method: PUT
    body_format: json
    headers:
      Authorization: "Bearer {{ access_token }}"
    body:
      {
        "customer_id": "{{ customer_id }}",
        "service_id": "{{ service_id }}",
        "title": "5GB 数据提升",
        "description": "额外 5GB 数据，有效期 7 天",
        "retail_cost": 5.00,
        "wholesale_cost": 1.50
      }
```

这做了什么：

- 记录 £5.00 的费用到客户
- 记录 £1.50 的批发成本
- 将交易链接到服务以供报告

完整附加组件流程摘要

1. 客户从过滤列表中选择附加组件
2. 付款获授权并收费
3. 数据余额添加到 OCS 账户

4. 在 CRM 中记录交易
5. 客户立即看到更新的余额：可用 7GB

财务跟踪：

- 服务每月收费：£15 零售，£5 批发
- 附加组件购买：£5 零售，£1.50 批发

自动续订：递归附加组件

某些附加组件可以设置为自动续订（每月数据计划、订阅等）。

产品配置：

```
{
  "product_name": "每月 10GB 数据计划",
  "category": "addon",
  "retail_cost": 10.00,
  "contract_days": 30,
  "auto_renew": "true",
  "provisioning_play": "play_topup_charge_then_action"
}
```

配置创建□作计划：

```
- name: 为自动续订创建操作计划
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body:
      {
        "method": "ApierV1.SetActionPlan",
        "params": [{
          "Id": "ServiceID_{{ service_uuid }}__ProductID_{{
product_id }}__MonthlyRenewal",
          "ActionPlan": [{
            "ActionsId": "Action_{{ product_slug }}",
            "Years": "*any",
            "Months": "*any",
            "MonthDays": "*any",
            "WeekDays": "*any",
            "Time": "00:00:00",
            "Weight": 10
          }],
          "Overwrite": false,
          "ReloadScheduler": true
        }]
      }
```

这做了什么:

- 在 OCS 中创建计划任务
- 每 30 天执行 `Action_{{ product_slug }}`
- 该操作向客户收费并重新应用数据余额
- 继续直到客户取消

客户管理:

- 客户在服务视图中看到“下次续订: 2025 年 2 月 1 日 - £10.00”
- 可以点击“取消自动续订”以停止未来���费
- 可以点击“立即续订”以立即应用下个月的配额

阶段 4：取消配置服务

当客户取消服务时，系统必须干净地移除所有资源。

取消配置触发器

取消配置可以通过以下方式触发：

1. 客户取消 - 客户点击“取消服务”
2. 管理□作 - 员工标记服务以进行停用
3. 未付款 - 服务因未续订而到期
4. 合同结束 - 固定期限合同到期

取消配置流程

1. 客户发起取消

- 导航到服务
- 点击“取消服务”
- 系统提示：“您确定吗？任何剩余余额将被没收。”
- 客户确认

2. 宽限期（可选）

某些运营商实施宽限期：

- 服务标记为“待取消”
- 在 7-30 天内保持活动状态
- 客户可以在宽限期内撤回取消
- 宽限期后自动取消配置

3. 创建取消配置作业

系统创建配置作业，内容如下：

```
{
    "action": "deprovision",
    "service_id": 1234,
    "customer_id": 123,
    "service_uuid": "Local_Mobile_SIM_a3f2c1d8"
}
```

调用在 `service.deprovisioning_play` 中指定的剧本或原始剧本的救援块。

4. Ansible 取消配置剧本

```
- name: 取消配置移动服务
  hosts: localhost
  tasks:
    - name: 禁用 OCS 账户
      uri:
        url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
        method: POST
        body:
          {
            "method": "ApierV2.SetAccount",
            "params": [{
              "Account": "{{ service_uuid }}",
              "ExtraOptions": { "Disabled": true }
            }]
          }

    - name: 删除操作计划（停止自动续订）
      uri:
        url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
        method: POST
        body:
          {
            "method": "ApierV1.RemoveActionPlan",
            "params": [{
              "Id": "ServiceID_{{ service_uuid }}__*"
            }]
          }

    - name: 更新 CRM 中的服务状态
      uri:
        url: "http://localhost:5000/crm/service/{{ service_id }}"
        method: PATCH
        body:
          {
            "service_status": "Deactivated",
            "service_deactivate_date": "{{ current_datetime }}"
          }

    - name: 将库存释放回库存
      uri:
        url: "http://localhost:5000/crm/inventory/inventory_id/{{
sim_card_id }}"
        method: PATCH
```

```
          body:
            {
              "service_id": null,
              "customer_id": null,
              "item_state": "Decommissioned"
            }
```

这做了什么：

1. 禁用 **OCS** 账户 - 停止所有收费，阻止使用
2. 删除□作计划 - 取消自动续订
3. 更新 **CRM** 服务 - 状态为“已停用”，记录日期
4. 释放库存 - SIM 标记为“退役”，可供重用（翻新后）

5. 取消配置后

系统执行清理：

- 客户不再在自助服务门户中看到服务
- 服务仍保留在 CRM 中以供历史报告
- 保留交易和发票以便于会计
- 库存可以翻新并重用
- OCS 账户可以在保留期后归档

部分与完全取消配置

部分取消配置（暂停）：

- 用于未付款或临时暂停
- 禁用 OCS 账户但不删除
- 保留余额
- 收到付款时可以重新启用

```
- name: 暂停服务
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body:
      {
        "method": "ApierV2.SetAccount",
        "params": [{
          "Account": "{{ service_uuid }}",
          "ExtraOptions": { "Disabled": true }
        }]
      }
```

完全取消配置（永久取消）：

- 用于永久取消
- 完全删除 OCS 账户
- 放弃余额
- 无法重新启用

```
- name: 删除 OCS 账户
  uri:
    url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
    method: POST
    body:
      {
        "method": "ApierV2.RemoveAccount",
        "params": [{
          "Account": "{{ service_uuid }}"
        }]
      }
```

产品管理的最佳实践

产品生命周期管理

产品状态：

- `enabled: true` - 产品可用于新订单
- `enabled: false` - 产品已禁用，现有服务继续

禁用产品：

- 将产品标记为 `enabled: false` 以防止新订单
- 现有服务保持活动状态
- 客户仍然可以续订/修改现有服务
- 对于淘汰旧产品很有用

库存管理

库存状态：

- `New` - 新库存，准备分配
- `In Stock` - 可用于配置
- `Assigned` - 链接到客户服务
- `Decommissioned` - 可以翻新并重用
- `Damaged` - 需要修理或处置

重用库存：

取消配置后：

- SIM 卡：翻新并标记为“库存中”
- 电话号码：在端口期（30 天）后释放
- 设备：测试、翻新、标记为“已使用”

配置指标

监控：

- 配置成功率
- 平均配置时间
- 常见故障点
- 库存周转

Mailjet □ OmniCRM □□

OmniCRM □ **Mailjet** □□□□

电子邮件模板配置

OmniCRM 使用 **10** 种不同的电子邮件模板类型 进行自动化通信。每个模板配置如下：

- **from_email** - 发件人电子邮件地址
- **from_name** - 发件人显示名称
- **template_id** - Mailjet 模板 ID (数字)
- **subject** - 电子邮件主题行

模板类型和配置

客户欢迎邮件

当创建新客户账户时发送。

```
api_crmCommunicationCustomerWelcome:
  from_email: "support@yourcompany.com"
  from_name: "Your Company Support"
  template_id: 5977509
  subject: "欢迎来到您的公司"
```

发送时间：

- 新客户通过自助服务门户注册
- 员工创建新客户账户
- 客户首次激活服务

可用模板变量：

- `{{ var:customer_name }}` - 客户的全名
- `{{ var:email }}` - 客户的电子邮件地址
- `{{ var:company_name }}` - 您的公司名称
- `{{ var:login_url }}` - 自助服务门户链接
- `{{ var:support_url }}` - 支持页面链接

客户发票邮件

当生成发票并准备付款时发送。

```
api_crmCommunicationCustomerInvoice:
  from_email: "billing@yourcompany.com"
  from_name: "Your Company Billing"
  template_id: 6759851
  subject: "您的发票 - "
```

发送时间：

- 为账单周期自动生成发票
- 员工手动创建发票
- 客户请求发票副本

可用模板变量：

- `{{ var:customer_name }}` - 客户的全名
- `{{ var:invoice_number }}` - 发票 ID/编号
- `{{ var:invoice_date }}` - 发票发出日期
- `{{ var:due_date }}` - 付款截止日期
- `{{ var:total_amount }}` - 应付总金额
- `{{ var:invoice_url }}` - 查看/下载发票 PDF 的链接
- `{{ var:pay_url }}` - 在线支付发票的链接

发票附件：

发票 PDF 会自动附加到电子邮件中。

客户发票提醒

发送以提醒客户逾期发票。

```
api_crmCommunicationCustomerInvoiceReminder:
  from_email: "billing@yourcompany.com"
  from_name: "Your Company Billing"
  template_id: 6759852
  subject: "付款提醒 - 发票逾期"
```

发送时间：

- 发票逾期 X 天（可配置）
- 员工手动触发提醒
- 自动提醒工作流（如果已配置）

可用模板变量：

- `{{ var:customer_name }}`
- `{{ var:invoice_number }}`
- `{{ var:due_date }}`
- `{{ var:days_overdue }}`
- `{{ var:total_amount }}`
- `{{ var:pay_url }}`

员工用户欢迎邮件

当创建新员工用户账户时发送。

```
api_crmCommunicationUserWelcome:
  from_email: "admin@yourcompany.com"
  from_name: "Your Company Admin"
  template_id: 5977510
  subject: "欢迎加入团队"
```

发送时间：

- 管理员创建新员工用户
- 在用户管理中点击“发送欢迎邮件”按钮

可用模板变量：

- `{{ var:user_name }}` - 员工用户的全名
- `{{ var:email }}` - 员工用户的电子邮件
- `{{ var:role }}` - 分配的角色
- `{{ var:login_url }}` - 管理门户登录链接
- `{{ var:temp_password }}` - 临时密码（如适用）

- `{{ var:support_email }}` - IT 支持联系方式

用户密码重置

当用户请求重置密码时发送。

```
api_crmCommunicationUserPasswordReset:
  from_email: "noreply@yourcompany.com"
  from_name: "Your Company Security"
  template_id: 5977511
  subject: "密码重置请求"
```

发送时间：

- 用户在登录页面点击“忘记密码”
- 用户提交密码重置请求

可用模板变量：

- `{{ var:user_name }}`
- `{{ var:reset_url }}` - 有时间限制的密码重置链接（通常为 1 小时）
- `{{ var:expiry_time }}` - 重置链接过期时间

安全提示：

重置链接在配置的时间段后过期（默认 1 小时）。

用户密码重置成功

发送以确认密码已成功更改。

```
api_crmCommunicationUserPasswordResetSuccess:
  from_email: "noreply@yourcompany.com"
  from_name: "Your Company Security"
  template_id: 5977512
  subject: "密码已成功更改"
```

发送时间：

- 用户成功完成密码重置
- 新密码设置后立即发送

可用模板变量：

- `{{ var:user_name }}`
- `{{ var:change_date }}` - 密码更改的日期/时间
- `{{ var:ip_address }}` - 更改的 IP 地址（可选）
- `{{ var:support_email }}` - 如果更改未经授权的联系

用户密码更改

当用户从设置中更改密码时发送。

```
api_crmCommunicationUserPasswordChange:
  from_email: "noreply@yourcompany.com"
  from_name: "Your Company Security"
  template_id: 5977513
  subject: "密码更改通知"
```

发送时间：

- 用户从个人资料/设置中更改密码
- 管理员重置用户密码

可用模板变量：

- `{{ var:user_name }}`
- `{{ var:change_date }}`
- `{{ var:changed_by }}` - “自己”或管理员名称
- `{{ var:support_email }}`

电子邮件验证

发送以验证用户的电子邮件地址。

```yaml
api_crmCommunicationEmailVerification:
  from_email: "noreply@yourcompany.com"
  from_name: "Your Company"
  template_id: 5977514
  subject: "验证您的电子邮件地址"
```

发送时间：

- 创建新账户（客户或员工）
- 用户更改电子邮件地址
- 出于安全原因需要电子邮件验证

可用模板变量：

- `{{ var:user_name }}`
- `{{ var:verification_url }}` - 验证电子邮件的链接
- `{{ var:verification_code }}` - 手动输入的代码（替代链接）

余额过期通知

当客户的服务余额/配额过期时发送。

```yaml
api_crmCommunicationsBalanceExpired:
  from_email: "support@yourcompany.com"
  from_name: "Your Company Support"
  template_id: 5977515
  subject: "您的服务余额已过期"
```

发送时间：

- 预付余额过期
- 月度配额续订日期已过
- 服务到期日期已到

可用模板变量：

- `{{ var:customer_name }}`
- `{{ var:service_name }}` - 已过期服务的名称

- `{{ var:expiry_date }}`
- `{{ var:balance_type }}` - “数据”、“语音”、“货币”等
- `{{ var:renewal_url }}` - 续订/充值的链接

低余额警报

当客户的余额低于配置阈值时发送。

```
api_crmCommunicationsBalanceLow:
  from_email: "support@yourcompany.com"
  from_name: "Your Company Support"
  template_id: 5977516
  subject: "低余额警报"
```

发送时间：

- 余额低于阈值（例如，剩余 20%）
- 在服务计划或 OCS 中配置
- 实时监控触发警报

可用模板变量：

- `{{ var:customer_name }}`
- `{{ var:service_name }}`
- `{{ var:current_balance }}`
- `{{ var:threshold }}`
- `{{ var:balance_type }}`
- `{{ var:topup_url }}` - 添加余额的链接

eSIM 交付邮件

当 eSIM 被配置并准备激活时发送。

```yaml
api_crmCommunicationEsimDelivery:
  from_email: "support@yourcompany.com"
  from_name: "Your Company Support"
  template_id: 0
  subject: "您的 eSIM 已准备好"
```

发送时间：

- eSIM 在自助注册期间被配置
- eSIM 分配给客户服务

可用模板变量：

- `{{ var:customer_name }}` - 客户的全名
- `{{ var:iccid }}` - eSIM ICCID
- `{{ var:activation_code }}` - LPA 激活代码字符串
- `{{ var:has_qr_code }}` - 布尔值，如果附加了 QR 代码图像则为真
- `{{ var:has_activation_code }}` - 布尔值，如果激活代码可用则为真

内置回退：

当 `template_id` 为 `0`（或配置中缺少模板键）时，OmniCRM 使用内置品牌 HTML 电子邮件，eSIM QR 代码在线渲染。颜色和品牌名称由 `crm_config.yaml` 中的 `branding` 部分控制 — 有关详细信息，请参见 配置指南。

创建 **Mailjet** 电子邮件模板

对于每种电子邮件类型，您需要在 Mailjet 中创建相应的模板。

第 **1** 步：在 **Mailjet** 中创建模板

1. 登录 Mailjet 仪表板
2. 导航到 电子邮件模板 → 事务性模板
3. 点击 创建新模板
4. 选择 自己编码模板（适合高级用户）或 使用模板构建器

第 2 步：设计模板

使用 Mailjet 的拖放构建器或 HTML 编辑器设计您的电子邮件。

基本元素：

- 头部 - 公司徽标和品牌
- 问候 - 使用 `{{ var:customer_name }}` 或 `{{ var:user_name }}` 个性化
- 内容 - 主要消息正文
- 变量 - 从上面的列表中插入模板变量
- 行动呼吁 - 用户操作的按钮/链接
- 页脚 - 退订链接、公司地址、支持联系方式

示例模板（密码重置）：

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body { font-family: Arial, sans-serif; }
        .button { background-color: #4CAF50; color: white;
padding: 14px 28px; }
    </style>
</head>
<body>
    

    <h2>密码重置请求</h2>

    <p>你好 {{ var:user_name }},</p>

    <p>我们收到了重置您密码的请求。请点击下面的按钮以创建新密码：</p>

    <a href="{{ var:reset_url }}" class="button">重置密码</a>

    <p>此链接将在 {{ var:expiry_time }} 内过期。</p>

    <p>如果您没有请求此操作，请忽略此电子邮件。</p>

    <hr>
    <p style="font-size: 12px; color: #666;">
        您的公司 | support@yourcompany.com<br/>
        123 商务街，城市，国家
    </p>
</body>
</html>
```

第 3 步：获取模板 ID

1. 在 Mailjet 中保存模板
2. 注意 模板 **ID**（数字，例如 `5977509`）
3. 将此 ID 复制到 `crm_config.yaml`

第 4 步：测试模板

1. 在 Mailjet 中，使用 测试电子邮件 功能
2. 为所有变量提供示例值
3. 将测试电子邮件发送给自己
4. 验证格式、链接和品牌

第 5 步：在 OmniCRM 中配置

将模板配置添加到 `crm_config.yaml`：

```yaml
mailjet:
  api_key: your_api_key
  api_secret: your_secret

  api_crmCommunicationUserPasswordReset:
    from_email: "noreply@yourcompany.com"
    from_name: "Your Company Security"
    template_id: 5977511
    subject: "密码重置请求"
```

重启 OmniCRM API 以使更改生效：

```
cd OmniCRM-API
sudo systemctl restart omnicrm-api
```

联系人同步

OmniCRM 中的所有客户联系人会自动同步到 Mailjet。

同步内容：

- 联系人姓名
- 电子邮件地址
- 联系人类型（账单、技术等）
- 客户位置

- 自定义字段

同步频率：

联系人在以下情况下实时同步：

- 创建客户
- 添加/更新联系人
- 修改客户详细信息

故障排除

电子邮件未发送

- 原因：无效的 API 凭证、Mailjet 账户被暂停或模板 ID 错误
- 解决方法：
  - 验证 `crm_config.yaml` 中的 `api_key` 和 `api_secret`
  - 检查 Mailjet 账户状态和账单
  - 验证模板 ID 是否存在于 Mailjet
  - 检查 API 日志中的错误

模板变量未替换

- 原因：变量名称不匹配或 OmniCRM 中缺少数据
- 解决方法：
  - 验证变量名称完全匹配（区分大小写）
  - 使用 `{{ var:variable_name }}` 格式
  - 检查 OmniCRM 是否在 API 调用中传递变量数据
  - 在 Mailjet 中使用示例数据进行测试

发票 **PDF** 未附加

- 原因： PDF 生成失败或文件大小过大
- 解决方法：
  - 检查发票生成日志
  - 验证发票模板是否正确渲染

    - 确保 PDF 小于 15MB（Mailjet 限制）
    - 单独测试发票 PDF 生成

联系人未同步到 **Mailjet**

- 原因：超过 API 速率限制或同步服务未运行
- 解决方法：
    - 检查 Mailjet API 速率限制（每分钟 200 次调用）
    - 验证 OmniCRM-API 服务□否在运行
    - 检查同步日志中的错误
    - 手动触发同步以进行测试

相关文档

- `administration_configuration` - 完整的 Mailjet 配置参考
- `payments_invoices` - 发票生成和电子邮件发送
- `authentication_flows` - 密码重置和验证电子邮件
- `customer_care` - 自助服务门户欢迎邮件
- 监控与指标 - Mailjet API 指标、电子邮件发送跟踪和性能监控

深入阅读

- Mailjet 文档：<https://dev.mailjet.com/>
- Mailjet API 参考：<https://dev.mailjet.com/email/reference/>

更新日志

此文档包含对 OmniCRM 软件堆栈或其依赖项所做的最后 50 个更改。

注意：这不跟踪对单个客户设置的更改。

监控与指标

OmniCRM API 提供全面的基于 Prometheus 的指标，用于监控应用程序性能、业务操作和外部集成。所有指标都在 `/crm/metrics` 端点以 Prometheus 展示格式公开。

概述

指标系统跟踪：

- 配置□作 - 作业执行、持续时间、成功/失败率
- 数据库性能 - 查询时间、连接池健康
- 外部集成 - OCS/CGRateS、Stripe、Mailjet API 调用
- 后台作业 - 异步任务执行和性能
- **HTTP** 请求 - API 端点使用情况和响应时间（自动生成）

指标端点

URL: `http://your-omnicrm-api:5000/crm/metrics`

格式：Prometheus 展示格式

认证：指标端点可公开访问以供 Prometheus 抓取。在生产环境中，建议使用防火墙规则或反向代理认证来限制访问。

指标类别

1. 配置指标

配置指标跟踪执行 Ansible 剧本的情况，这些剧本用于配置服务、管理库存和配置外部系统。有关更多详细信息，请参见 Provisioning System 和 Ansible Playbooks。

omnicrm_provision_jobs_total

类型：计数器

标签：

- `status` - 作业完成状态：`success`，`failed`，`running`

描述：创建的配置作业总数。当配置作业以其最终状态完成时递增。

omnicrm_provision_job_duration_seconds

类型：直方图

标签：

- `playbook` - 执行的 Ansible 剧本名称
- `status` - 作业完成状态：`success`，`failed`

桶：`[1, 5, 10, 30, 60, 120, 180, 300, 600]` 秒（1秒到10分钟）

描述：配置作业完成所需的时间。记录整个剧本执行的持续时间，从开始到结束。

omnicrm_provision_jobs_active

类型：指标

描述：当前正在运行的配置作业数量。当作业开始时递增，当作业完成时递减。

omnicrm_provision_tasks_total

类型：计数器

标签：

- `playbook` - Ansible 剧本名称
- `status` - 任务结果：`ok`，`failed`，`ignored`

描述：在剧本中执行的 Ansible 任务总数。每个完成的单独任务（成功或失败）递增。

omnicrm_provision_errors_total

类型：计数器

标签：

- `error_type` - 错误类型: `fatal`, `task_failed`, `timeout`
- `playbook` - Ansible 剧本名称

描述：按类型统计的配置错误总数。当配置任务失败或执行期间发生致命错误时递增。

---

2. 数据库指标

数据库指标监控查询性能和连接池健康。OmniCRM 使用 SQLAlchemy 进行自动仪表化。有关数据模型的详细信息，请参见 System Architecture。

omnicrm_db_query_duration_seconds

类型：直方图

标签：

- `operation` - SQL 操作类型: `SELECT`, `INSERT`, `UPDATE`, `DELETE`
- `table` - 数据库表名称

桶：`[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0, 10.0]` 秒

描述：数据库查询执行时间。通过 SQLAlchemy 事件监听器在每个查询上自动跟踪。

omnicrm_db_pool_size

类型：指标

描述：当前数据库连接池大小。池中连接的总数（包括已检查和可用的连接）。

omnicrm_db_pool_overflow

类型：指标

描述：当前数据库连接池溢出。超出正常池大小限制创建的连接数量。

omnicrm_db_pool_connections_checked_out

类型：指标

描述：当前从池中检查出的连接数量，并被应用程序代码使用。

omnicrm_db_errors_total

类型：计数器

标签：

- `error_type` - 数据库错误类型
- `operation` - 导致错误的操作：`connection_error`，`query_error`，等等。

描述：数据库错误的总数。

状态：已定义但��当前代码中未积极使用（保留供将来使用）

---

3. OCS/CGRateS 指标

OCS（在线计费系统）指标跟踪与 CGRateS 计费引擎的交互。有关更多详细信息，请参见 Billing Overview 和 CGRateS Integration。

omnicrm_ocs_api_calls_total

类型：计数器

标签：

- `method` - OCS API 方法：`GetBalance`，`SetBalance`，`SetAccount`，`RemoveAccount`，等等。
- `status` - 调用结果：`success`，`failed`

描述：OCS/CGRateS API 调用的总数。每次异步 API 调用到 OCS 系统时递增。

omnicrm_ocs_api_duration_seconds

类型：直方图

标签：

- `method` - OCS API 方法：`GetBalance`，`SetBalance`，`GetAccount`，等等。

桶：`[0.1, 0.25, 0.5, 1.0, 2.0, 5.0, 10.0, 30.0]` 秒

描述：OCS/CGRateS API 调用持续时间。记录每个 API 调用所需的时间，包括网络延迟。

omnicrm_ocs_api_errors_total

类型：计数器

标签：

- `method` - 失败的 OCS API 方法
- `error_type` - 错误类别：`timeout`，`connection_error`，`json_error`，`http_error`，等等。

描述：OCS/CGRateS API 错误的总数。当 OCS API 调用因特定错误类型失败时递增。

omnicrm_ocs_balance_queries_total

类型：计数��

标签：

- `query_type` - 余额查询类型：`single_account`，`multiple_accounts`

描述：对 OCS 的余额查询总数。用于跟踪余额查询操作的使用情况。

omnicrm_ocs_action_plan_operations_total

类型：计数器

标签：

- `operation` - 行动计划操作：`create`，`remove`，`query`

描述：行动计划操作的总数。跟踪 CGRateS 行动计划的创建、删除和查询，以进行定期收费。

---

4. Stripe 支付指标

Stripe 指标跟踪支付处理操作。有关集成详细信息，请参见 Payment System Guide 和 Payment Methods。

omnicrm_stripe_api_calls_total

类型：计数器

标签：

- `operation` - Stripe 操作：`create_customer`，`charge`，`refund`，`update_payment_method`，等等。
- `status` - 操作结果：`success`，`failed`

描述：Stripe API 调用的总数。每次支付处理操作时递增。

omnicrm_stripe_api_duration_seconds

类型：直方图

标签：

- `operation` - Stripe 操作类型

桶：`[0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 30.0]` 秒

描述：Stripe API 调用持续时间，包括网络延迟。

omnicrm_stripe_api_errors_total

类型：计数器

标签：

- `operation` - 失败的 Stripe 操作
- `error_type` - 错误类别：`card_declined`，`network_error`，`api_error`，等等。

描述：Stripe API 错误的总数。当支付操作失败时递增。

omnicrm_stripe_payment_amount_cents

类型：计数器

标签：

- `payment_type` - 支付方向：`charge`，`refund`

描述：通过 Stripe 处理的总支付金额（以美分为单位）。用于跟踪交易量和收入。

omnicrm_stripe_payment_failures_total

类型：计数器

标签：

- `reason` - 失败原因：`card_declined`，`insufficient_funds`，`expired_card`，等等。

描述：按拒绝代码分类的 Stripe 支付失败总数。

---

5. Mailjet 邮件指标

Mailjet 指标跟踪电子邮件投递操作。有关配置详细信息，请参见 Mailjet Integration。

omnicrm_mailjet_api_calls_total

类型：计数器

标签：

- `email_type` - 邮件模板类型：`welcome`，`user_welcome`，`invoice`，`notification`
- `status` - 投递结果：`success`，`failed`

描述：Mailjet API 调用的总数。通过 `@track_mailjet_call` 装饰器跟踪。

omnicrm_mailjet_api_duration_seconds

类型：直方图

标签：

- `email_type` - 邮件模板类型

桶：`[0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 30.0]` 秒

描述：Mailjet API 调用持续[illegible]间。通过 Mailjet API 提交电子邮件所需的时间（不[illegible]投递时间）。

omnicrm_mailjet_api_errors_total

类型：计数器

标签：

- `email_type` - 邮件模板类型
- `error_type` - 错误类别

描述：Mailjet API 错误的总数。当电子邮件发送失败时递增。

omnicrm_mailjet_emails_sent_total

类型：计数器

标签：

- `email_type` - 邮件模板类型
- `template_id` - Mailjet 模板 ID

描述：通过 Mailjet 成功发送的电子邮件总数。与 API 调用不同，因为一次调用可以发送给多个收件人。

omnicrm_mailjet_email_recipients_total

类型：计数器

标签：

- `email_type` - 邮件模板类型

描述：所有已发送电子邮件的总收件人数量。

---

6. 后台作业指标

后台作业指标跟踪异步操作，如剧本链和计划任务。有关后台作业的详细信息，请参见 Provisioning System。

omnicrm_background_jobs_total

类型：计数器

标签：

- `job_type` - 作业类别: `playbook_single`, `playbook_chain`, `periodic_task`

描述: 启动的后台作业总数。通过 `BackgroundJobTimer` 上下文管理器跟踪。

omnicrm_background_jobs_active

类型: 指标

标签:

- `job_type` - 作业类别

描述: 当前正在运行的后台作业数量。在作业开始时递增，在作业完成时递减。

omnicrm_background_job_duration_seconds

类型: 直方图

标签:

- `job_type` - 作业类别
- `status` - 作业结果: `success`, `failed`

桶: `[1, 5, 10, 30, 60, 120, 180, 300, 600, 1800, 3600]` 秒 (1秒到1小时)

描述: 后台作业执行时间。包括多步骤操作的完整持续时间。

omnicrm_background_job_errors_total

类型: 计数器

标签:

- `job_type` - 作业类别
- `error_type` - 错误类别

描述: 后台作业错误的总数。当后台作业因异常失败时递增。

---

7. Flask HTTP 指标（自动生成）

这些指标由 `prometheus-flask-exporter` 库自动生成，跟踪所有对 API 的 HTTP 请求。

flask_http_request_duration_seconds

类型：直方图

标签：

- `method` - HTTP 方法：`GET`，`POST`，`PUT`，`DELETE`，等等。
- `endpoint` - Flask 路由名称
- `status` - HTTP 状态码

描述：所有 API 端点的 HTTP 请求持续时间。自动仪表化。

flask_http_request_total

类型：计数器

标签：

- `method` - HTTP 方法
- `endpoint` - Flask 路由名称
- `status` - HTTP 状态码

描述：按端点、方法和状态码统计的总 HTTP 请求数。

flask_http_request_exceptions_total

类型：计数器

标签：

- `method` - HTTP 方法
- `endpoint` - Flask 路由名称

描述：HTTP 请求中的总未处理异常。指示错误或意外错误。

---

8. API 错误指标（保留）

这些指标已定义但当前未仪表化。它们保留供将来使用。

omnicrm_api_errors_total

类型：计数器

标签：

- `endpoint` - API 端点
- `error_type` - 错误类别
- `status_code` - HTTP 状态码

状态：已定义但未积极使用

omnicrm_api_auth_failures_total

类型：计数器

标签：

- `auth_method` - 认证方法：`jwt`, `api_key`, `ip_whitelist`
- `reason` - 失败原因

状态：已定义但未积极使用

---

9. 应用程序信息指标

omnicrm_api

类型：信息

标签：

- `version` - API 版本字符串
- `environment` - 环境名称：`production`, `staging`, `development`

描述：OmniCRM API 信息指标。在应用程序启动时设置一次，包含版本和环境信息。

定期更新

update_db_pool_metrics(engine)

每 30 秒自动调用一次以更新数据库连接池指标。

Prometheus 配置

抓取配置

将 OmniCRM 添加到您的 `prometheus.yml`：

```
scrape_configs:
  - job_name: 'omnicrm-api'
    scrape_interval: 15s
    scrape_timeout: 10s
    static_configs:
      - targets: ['omnicrm-api:5000']
    metrics_path: '/crm/metrics'
```

示例警报

高配置失败率

```yaml
- alert: HighProvisionFailureRate
  expr: |
    (
      rate(omnicrm_provision_jobs_total{status="failed"}[5m]) /
      rate(omnicrm_provision_jobs_total[5m])
    ) > 0.1
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "高配置作业失败率"
    description: "{{ $value | humanizePercentage }} 的配置作业失败"
```

数据库连接池耗尽

```yaml
- alert: DatabasePoolExhausted
  expr: omnicrm_db_pool_overflow > 0
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: "检测到数据库连接池溢出"
    description: "连接池正在使用溢出连接，可能表示池大小过小"
```

慢数据库查询

```yaml
- alert: SlowDatabaseQueries
  expr: |
    histogram_quantile(0.99,
      rate(omnicrm_db_query_duration_seconds_bucket[5m])
    ) > 1.0
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "检测到慢数据库查询"
    description: "第 99 百分位查询时间为 {{ $value }}s"
```

OCS API 不可用

```yaml
- alert: OCSAPIDown
  expr: |
    (
      rate(omnicrm_ocs_api_calls_total{status="failed"}[5m]) /
      rate(omnicrm_ocs_api_calls_total[5m])
    ) > 0.5
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: "OCS API 失败率严重"
    description: "{{ $value | humanizePercentage }} 的 OCS API 调用失败"
```

Stripe 支付问题

```yaml
- alert: StripePaymentFailures
  expr: rate(omnicrm_stripe_payment_failures_total[5m]) > 5
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "Stripe 支付失败率上升"
    description: "{{ $value }} 每秒支付失败"
```

最佳实践

监控策略

1. 设置核心警报 - 配置关键指标的警报:
    - 配置失败率 > 10%
    - 数据库连接池耗尽
    - OCS/CGRateS API 失败
    - Stripe 支付处理错误

2. 跟踪业务指标 - 监控操作 KPI:
    - 通过 Stripe 处理的收入
    - 配置吞吐量
    - 客户电子邮件投递率
3. 性能监控 - 关注性能下降:
    - API 响应时间百分位
    - 数据库查询性能
    - 外部 API 延迟
4. 容量规划 - 使用指标进行预测:
    - 数据库连接池大小
    - 后台作业工作者扩展
    - API 服务器容量

指标保留

Prometheus 保留建议：

- **15** 天 - 高分辨率指标（15 秒抓取间隔）
- **90** 天 - 降采样指标（5 分钟聚合）
- **1** 年 - 长期聚合指标（1 小时聚合）

使用 Thanos、Cortex 或 VictoriaMetrics 进行长期���储和全局查询。

故障排除

指标未出现

默认情况下，`/metrics` 端点仅对内部（非公共地址空间）源公开。如果需要，您可以在 nginx 配置中更改此设置。

检查指标端点：

```
curl http://localhost:5000/crm/metrics
```

验证 **Prometheus** 是否可以抓取：

```
# 检查 Prometheus 目标页面
http://prometheus:9090/targets
```

缺少特定指标

某些指标仅在首次使用后创建：

- 配置指标在首次配置作业后出现
- Stripe 指标在首次支付后出现
- OCS 指标在首次计费操作后出现

高基数问题

如果 Prometheus 运行缓慢或消耗过多内存，请检查高基数标签：

```
# 计算每个指标的唯一时间序列
count by (__name__) ({__name__=~".+"})
```

具有 >10,000 个系列的指标可能表示基数问题。

指标摘要

总指标：34（31 个自定义 + 3 个自动生成的 Flask 指标）

按类型：

- 计数器：20
- 指标：5
- 直方图：8

- 信息: 1

按类别:

- 配置: 5 个指标
- 数据库: 5 个指标
- OCS/CGRateS: 5 个指标
- Stripe: 5 个指标
- Mailjet: 5 个指标
- 后台作业: 4 个指标
- HTTP/Flask: 3 个指标
- 应用程序信息: 1 个指标
- 保留（将来使用）: 2 个标

相关文档

- System Architecture - 整体系统设计和组件关系
- Provisioning System - 配置工作流和作业执行
- Ansible Playbooks - 剧本结构和执行
- API Documentation - API 端点和认证
- Billing Overview - 计费和收费系统
- Payment System Guide - Stripe 集成和支付处理
- Mailjet Integration - 电子邮件服务配置
- Administration Configuration - 系统配置选项

支付方式管理

OmniCRM 的支付方式系统允许客户和员工安全地管理支付卡，使用 多供应商支付处理（Stripe、PayPal 等）。支付方式支持自动计费服务、一次性付款和定期收费，而无需在 OmniCRM 中存储敏感的卡数据。

另请参见：`Payment System Guide <payment_system_guide>`，`Billing Overview <billing_overview>`，`Payment Processing <payments_process>`，`Invoices <payments_invoices>`。

概述

支付方式系统提供：

- 安全卡存储 - 由支付供应商（Stripe、PayPal）进行令牌化的卡片，永远不存储在 OmniCRM 中
- 多供应商支持 - 支持 Stripe 和 PayPal 支付方式
- 多张卡片 - 客户可以存储多种支付方式
- 默认选择 - 指定首选支付方式以进行自动收费
- 到期跟踪 - 监控和更新即将到期的卡片
- 自助服务 - 客户可以通过 `Self-Care Portal <self_care_portal>` 管理自己的卡片
- 员工管理 - 支持员工可以代表客户添加/删除卡片

支持的支付方式：

- 卡片（通过 Stripe 或 PayPal）
  - ���用卡（Visa、Mastercard、American Express、Discover）
  - 借记卡
  - 预付卡（如果卡网络支持）
- **PayPal** 账户（通过 PayPal 集成）

不存储在 **OmniCRM** 中：

卡片详细信息由支付供应商进行令牌化并安全存储。OmniCRM 仅存储：

- 支付供应商（stripe、paypal）
- 卡品牌（Visa、Mastercard 等）

- 后四位数字
- 到期月份/年份
- 持卡人姓名/昵称
- 供应商特定的支付方式令牌

访问支付方式

从客户页面：

1. 导航到 **Customers → [选择客户]**
2. 点击 **Billing** 标签
3. 滚动到 **Payment Methods** 部分

或者直接：

从即将到期的卡片仪表板：

查看所有即将到期的卡片客户：

这显示了系统范围内在接下来的 60 天内到期的卡片列表。

支付方式列表

支付方式表显示客户的所有存储卡片：

列描述：

- 昵称 - 卡片的友好名称（例如，“个人卡”，“工作 Visa”）
- 发行人 - 卡品牌和后四位数字
- 到期 - 到期月份/年份（MM/YYYY 格式）
- 添加 - 卡片添加到账户的日期
- 默认 - 勾选标记表示自动收费的默认支付方式

每张卡的□作：

每行��有一个操作菜单（⋮），选项包括：

- 设为默认 - 将此卡片设为默认支付方式
- 删除 - 从账户中移除卡片

添加支付方式

点击 **"Add Payment Method"** 打开安全支付模态框。

第一步：输入卡片详细信息

安全支付表单出现（由 Stripe Elements 或 PayPal SDK 提供支持）：

必填字段：

- 卡片信息 - 卡号、到期、CVC（由 Stripe 验证）
- 持卡人姓名 - 卡上的姓名
- 国家/地区 - 账单国家

可选字段：

- 卡片昵称 - 友好的标签以区分卡片

安全性：

- 卡片详细信息直接输入到供应商托管的安全 iframe（Stripe Elements / PayPal SDK）
- OmniCRM 从未看到或存储完整的卡号
- PCI DSS 合规性由支付供应商处理
- 实时验证防止无效卡号

第二步：提交并令牌化

当您点击 **"Add Payment Method"** 时：

1. 客户端验证：
   - 支付供应商验证卡号格式
   - 检查到期日期□否在未来
   - 验证 CVC 格式
2. 令牌化：
   - 卡片详细信息直接发送到支付供应商（而不□ OmniCRM）
   - 供应商创建一个安全令牌（例如，`pm_1A2B3C4D` 用于 Stripe）
   - 令牌返回给 OmniCRM
3. 服务器处理：
   - OmniCRM 将令牌保存到客户记录中，并附上供应商标识符
   - 存储后四位数字、品牌、到期和供应商名称以供显示
   - 完整的卡号从未接触 OmniCRM 服务器

第三步：确认

成功消息出现：

> 您的 Visa 卡（尾号 1234）已添加到您的账户中。

新卡片出现在支付方式表中。

自动默认选择：

- 如果这□客户的第一张卡，它会自动设为默认
- 如果客户已经有卡，新卡将作为非默认卡添加
- 客户可以在添加后更改默认设置

设置默认支付方式

默认支付方式用于：

- 自动定期服务收费
- 发票付款
- 充值和补充
- 一次性交易（除非另有说明）

更改默认设置：

1. 在支付方式表中找到您想要设为默认的卡片
2. 点击卡片旁边的 □作菜单（ ⋮ ）
3. 选择 **"Set as Default"**
4. 确认信息出现

   Visa 尾号 5678 现在□您的默认支付方式。

勾选标记移动到新选择的卡片。

视觉指示：

默认卡片显示：

在默认列中，通常带有绿色勾选标记徽章。

删除支付方式

移除过期、丢失或不再需要的卡片。

第一步：启动删除

1. 在支付方式表中找到要删除的卡片
2. 点击 □作菜单（ ⋮ ）
3. 选择 **"Delete"**

第二步：确认删除

确认模态框出现：

> 您确定要删除此支付方式吗?
>
> 卡片: Visa 尾号 1234 到期: 12/2026
>
> ⚠ 警告：如果这□您唯一的支付方式，您需要添加新的支付方式才能继续使用需要自动计费的服务。
>
> [取消] [删除支付方式]

点击 **"Delete Payment Method"** 以确认。

第三步：删除完成

成功消息：

卡片已从表中移除，并从支付供应商处删除。

重要限制：

- 如果存在其他卡片，则无法删除默认卡 - 首先将其他卡片设为默认
- 删除最后一张卡时的警告 - 需要付款的服务可能会被暂停
- 无法撤销 - 删除□永久的；客户如果需要必须重新添加卡片

管理即将到期的卡片

OmniCRM 跟踪卡片到期日期，并提供工具以主动更新即将到期的卡片。

即将到期的卡片仪表板

导航到 **Billing → Expiring Cards** 以查看系统范围内的列表：

> 客户 卡片 到期 距离操作天数 John Smith Visa **1234 02/2025 12 天 更新 **Acme Corp MC5678 03/2025 45 天 更新 **Jane Doe Amex**9012 01/2025 已过期 更新

筛选器：

- 到期范围 - 接下来的 30/60/90 天或已过期
- 客户类型 - 个人与企业
- 服务类型 - 按需付款方式过滤服务

操作：

- 更新 - 打开客户的支付方式页面以添加新卡
- 通知 - 向客户发送电子邮件提醒（如果已配置 Mailjet）

到期通知

如果已配置 Mailjet，将自动发送电子邮件：

- 到期前 **60** 天 - 第一次提醒
- 到期前 **30** 天 - 第二次提醒
- 到期前 **7** 天 - 最后警告
- 到期时 - 卡片已过期通知

客户可以点击电子邮件中的链接，通过自助服务门户更新他们的支付方式。

电子邮件模板变量：

Mailjet 模板接收：

- 客户姓名
- 卡品牌和后四位数字
- 到期日期
- 自助服务支付方式页面的链接

请参见 `integrations_mailjet` 以获取电子邮件模板配置。

更新即将到期的卡片

推荐工作流程：

1. 客户收到到期通知电子邮件
2. 客户登录自助服务门户
3. 导航到 **Billing → Payment Methods**
4. 点击 **"Add Payment Method"**
5. 输入新卡片详细信息（同一张卡的更新到期，或替换卡片）
6. 将新卡片设为默认
7. 删除旧的/过期的卡片

员工工作流程：

如果客户拨打支持电话：

1. 员工打开客户账户
2. 导航到 **Billing → Payment Methods**
3. 代表客户添加新卡（客户通过电话提供详细信息）
4. 将新卡片设为默认
5. 删除过期的卡片
6. 与客户确认

警告

切勿要求客户通过电子邮件或短信发送卡片详细信息。始终使用：

- 安全的自助服务门户进行自助服务
- 电话上由员工直接输入详细信息到系统
- 在零售地点亲自处理

卡片到期时会发生什么

当支付卡达到到期日期且未更新时：

立即影响：

1. 自动支付失败
    - 支付供应商拒绝使用过期卡的交易
    - 月度服务续订未能处理
    - 自动充值失败
    - 发票自动付款失败
2. 客户通知
    - 系统尝试收费卡片
    - 发送支付失败通知
    - 发送“更新支付方式”电子邮件，附有自助服务门户的链接
3. 服务状态变化
    - 后付费服务 - 可能在欠款的情况下暂时继续
    - 预付费服务 - 当余额耗尽时服务暂停
    - 自动续订服务 - 续订失败，服务可能到期

后续□作：

第 **1-3** 天（宽限期）：

- 服务正常继续
- 客户收到第一次支付失败通知
- 系统尝试重试（根据配置）

第 **4-7** 天：

- 第二次支付尝试（如果配置）
- 发送警告电子邮件
- 客户服务可能联系客户

第 **8-14** 天：

- 服务可能因未付款而暂停
- 暂停状态阻止使用但保留账户
- 客户可以通过更新支付方式和支付欠款来恢复

第 **15** 天及以后：

- 服务可能因未付款而终止
- 库存（SIM 卡、设备）标记为退回
- 发送最终通知
- 账户转交催收（如适用）

防止服务中断：

为避免服务中断：

- 在到期前 **30** 天更新卡片
- 添加多种支付方式以提高冗余性
- 启用支付失败警报
- 每周监控即将到期的卡片仪表板

到期后恢复服务：

如果因过期卡片暂停服务：

1. 添加新的有效支付方式

2. 设为默认
3. 支付欠款（如有）
4. 联系支持以重新激活服务
5. 服务在几分钟到几小时内恢复

支付方式安全

令牌化

OmniCRM 使用供应商令牌化确保安全：

1. 客户输入卡片 → 直接发送到支付供应商服务器
2. 供应商验证并令牌化 → 创建唯一令牌
3. 令牌存储在 **OmniCRM** → 从不存储完整卡号
4. 支付处理 → 令牌发送给供应商，供应商收费卡片

OmniCRM 存储的内容：

```
{
  "vendor": "stripe",
  "vendor_payment_method_id": "pm_1A2B3C4D5E6F",
  "payment_type": "card",
  "brand": "visa",
  "last4": "1234",
  "exp_month": 12,
  "exp_year": 2026,
  "name": "John Smith",
  "nickname": "Personal Card",
  "is_default": true
}
```

OmniCRM 不存储的内容：

- 完整的卡号
- CVV/CVC 代码
- 磁条数据
- PIN 号码

PCI 合规性

通过使用供应商托管的支付表单：

- 减少 **PCI** 范围 - 卡数据从未接触 OmniCRM 服务器
- 供应商托管字段 - 卡片输入发生在供应商的安全 iframe 中
- 无卡存储 - 使用令牌代替原始卡数据
- 安全传输 - 所有通信通过 HTTPS/TLS

请参见 `Payment System Guide <payment_system_guide>` 以获取支付供应商安全详细信息。

常见工作流程

工作流程 **1**：客户添加第一张支付方式

场景：新客户注册服务

1. 客户创建账户
2. 选择服务计划
3. 在结账时提示添加支付方式
4. 在 Stripe 模态框中输入卡片详细信息
5. 卡片令牌化并保存
6. 自动设为默认
7. 服务提供
8. 第一次收费处理

工作流程 **2**：客户更新即将到期的卡片

场景：信用卡即将到期

1. 客户收到电子邮件通知（到期前 60 天）
2. 登录自助服务门户
3. 导航到 **Billing → Payment Methods**
4. 查看当前卡片到期 12/2025
5. 点击 **"Add Payment Method"**

6. 输入到期 12/2028 的替换卡
7. 将新卡设为默认
8. 删除旧卡
9. 发送确认电子邮件

工作流程 **3**：员工通过电话帮助客户

场景：客户拨打电话：“我的卡被拒绝”

1. 客户拨打支持电话
2. 员工验证身份（安全问题）
3. 员工检查支付方式：卡片到期 01/2025
4. 员工：“您的卡片已过期。您有新卡吗？”
5. 客户通过电话提供新卡详细信息
6. 员工导航到 **Customers → [Customer] → Billing**
7. 点击 **"Add Payment Method"**
8. 输入客户朗读的卡片详细信息
9. 将新卡设为默认
10. 删除过期的卡片
11. 重试失败的支付
12. 与客户确认：“支付成功，服务恢复”

工作流程 **4**：拥有多张卡的企业客户

场景：公司希望为不同目的使用不同的卡片

1. 企业客户添加主卡（Visa 尾号 1111）
2. 将其设为每月服务收费的默认卡
3. 添加备用卡（Mastercard 尾号 2222）用于充值
4. 添加采购卡（Amex 尾号 3333）用于设备采购
5. 在进行充值时，在结账时手动选择 Mastercard
6. 默认的 Visa 仍用于自动每月计费

工作流程 5：管理即将到期的卡片（管理员）

场景：主动到期管理

1. 管理员导航到 **Billing → Expiring Cards**
2. 过滤器：“接下来的 30 天”
3. 看到 15 个客户的卡片即将到期
4. 选择所有 → **"Send Reminder Emails"**
5. Mailjet 向每个客户发送个性化电子邮件
6. 客户通过自助服务更新卡片
7. 管理员一周后查看列表
8. 联系尚未更新的剩余客户
9. 通过电话协助更新卡片

故障排除

“添加支付方式时卡被拒绝**”**

- 原因：支付供应商拒绝了卡片（资金不足、欺诈预防、发行人拒绝）
- 解决方案：
  - 尝试不同的卡片
  - 联系卡片发行人以授权交易
  - 确保卡片支持在线购买
  - 检查账单地址□否与档案中的卡片匹配

“添加支付方式时出错**”**（通用错误）

- 原因：支付供应商 API 错误或网络问题
- 解决方案：
  - 刷新页面并重试
  - 检查互联网连接
  - 验证系统设置中的支付供应商配置□否正确
  - 检查浏览器控制台以获取具体错误信息
  - 尝试不同的浏览器（禁用广告拦截器）

无法删除支付方式（按钮禁用）

- 原因：尝试删除默认卡，或者这□唯一的卡
- 解决方案：
  - 首先将其他卡设为默认
  - 如果这□唯一的卡，先添加新卡再删除

卡片显示已过期，但不在“即将到期的卡片”列表中

- 原因：卡片最近过期，缓存未刷新
- 解决方案：
  - 刷新页面
  - 检查即将到期的卡片仪表板上的过滤器
  - 过期的卡片可能会移动到不同的视图

新卡未立即出现

- 原因：添加卡片后页面未刷新
- 解决方案：
  - 支付方式表应自动刷新
  - 如果没有，手动刷新浏览器
  - 检查添加过程□否发生错误

支付模态框无法加载

- 原因：支付供应商 SDK 未加载、API 密钥问题或浏览器扩展阻止
- 解决方案：
  - 检查浏览器控制台中的错误
  - 禁用广告拦截器和跟踪保护
  - 验证系统设置中的支付供应商配置
  - 确保供应商 SDK 脚本加载（检查网络标签）
  - 尝试隐身/私密浏览模式

客户未收到到期通知

- 原因： Mailjet 未配置或电子邮件模板缺失
- 解决方案：

    - 验证 crm_config.yaml 中的 Mailjet 凭据
    - 检查□否存在卡片到期的电子邮件模板
    - 确认客户电子邮件地址有效
    - 检查 Mailjet 日志以获取投递失败信息

最佳实践

对于客户：

- 在服务激活之前添加支付方式以避免延误
- 保持至少 2 张卡片以提高冗余性
- 在到期前 30 天以上更新即将到期的卡片
- 删除旧的/过期的卡片以避免混淆
- 使用描述性昵称（“个人 Visa”，“工作 Amex”）
- 验证默认支付方式□否正确以进行自动计费

对于支持员工：

- 在访问支付方式之前验证客户身份
- 切勿要求客户通过电子邮件/SMS/聊天发送卡片详细信息
- 在通话期间立即处理卡片添加（不要延迟）
- 确认添加后新卡设为默认
- 仅在确认新卡有效后删除旧卡
- 更新过期卡片后测试支付（处理 £0.01 授权）

对于管理员：

- 每周监控即将到期的卡片仪表板
- 在到期前 60/30/7 天发送提醒电子邮件
- 保持支付供应商的测试/实时密钥分开用于开发与生产
- 确保配置 Mailjet 模板以进行到期通知
- 审查失败的支付报告以识别过期卡片
- 培训员工有关安全卡片处理程序

相关文档

- `Payment System Guide <payment_system_guide>` - 完整的支付 API 参考和架构
- `payments_process` - 使用存储支付方式处理支付
- `payments_invoices` - 使用默认卡进行自动发票付款
- `features_topup_recharge` - 使用支付方式的充值系统
- `basics_payment` - 一般支付和计费概念
- `customer_care` - 自助服务门户，供客户管理自己的卡片

OmniCRM 付款系统 API 指南

概述

OmniCRM 付款系统提供了一个全面的、与供应商无关 的支付处理基础设施。今天它支持 Stripe 和 PayPal，但模块化架构允许与任何支付提供商（Square、Adyen、Braintree 等）集成，而无需更改应用程序代码。

本文档涵盖了系统中所有可用的支付 API 和工作流程。

> □ **Ansible Playbook** 集成：有关在配置 Playbooks 中实现这些支付 API 的信息，请参见 从 **Playbooks** 进行收费和支付

目录

1. 模块化架构
2. 核心概念
3. 财务文档
4. 支付方式 API
5. 支付流程 API
6. 钱包 API
7. API 参考摘要
8. 常见用例

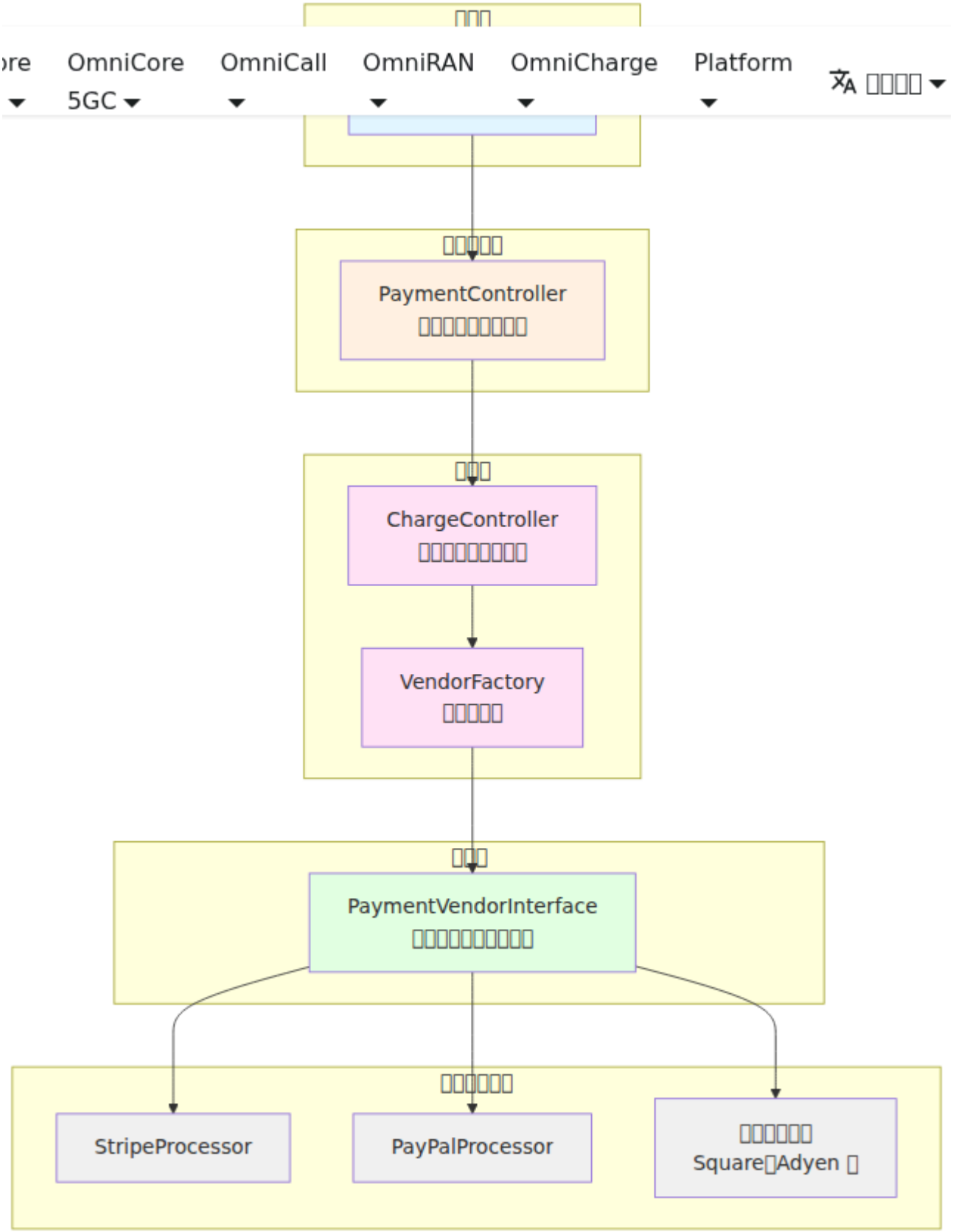
模块化架构

为什么与供应商无关?

该系统使用 抽象层 将业务逻辑与支付供应商的具体情况分离。这意味着：

- □ □□□□□□□□□□□□□□□□□□□□
- □ □□□□□□□□□□□□□

- □ □□□□□□□□□□
- □ □□□□□□□□□□□API □□



架构层

OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
PaymentController
ChargeController
VendorFactory
PaymentVendorInterface
StripeProcessor
PayPalProcessor
Square、Adyen 等

添加新的支付供应商

要添加新的提供商（例如 Square、Adyen、Braintree），请联系您的 OmniCRM 技术团队。

有关现有供应商（Stripe、PayPal）的配置详细信息，请参见 供应商配置。

流程概述：

步骤：

1. 实现处理器类，使用标准 `PaymentVendorInterface`（授权、捕获、收费、退款、释放）
2. 在 **VendorFactory** 中注册处理器，使用供应商名称
3. 在供应商的沙盒环境中测试集成
4. 部署 - 所有现有 API 自动支持新供应商

结果：一旦部署，新供应商即可无缝工作：

```
# 添加 Square 支付方式
curl -X POST /api/payments/methods \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "vendor": "square",
    "payment_token": "sq_xxxxx"
  }'
```

所有支付 API 自动支持新供应商，无需更改应用程序代码。

集成时间：通常需要 1-2 天来为新供应商处理器进行集成。

供应商配置

启用支付供应商

UI 控制哪些支付供应商可供客户使用。这通过 UI `.env` 文件中的 `VITE_PAYMENT_VENDORS` 变量设置的：

```
# 仅启用 Stripe（默认）
VITE_PAYMENT_VENDORS="stripe"

# 启用 Stripe 和 PayPal
VITE_PAYMENT_VENDORS="stripe,paypal"

# 仅启用 PayPal
VITE_PAYMENT_VENDORS="paypal"
```

重要：`VITE_PAYMENT_VENDORS` 一个 构建时 变量。更改它需要重新构建并重新启动 UI。

Stripe 配置

Stripe 主要的支付供应商，提供带有 3D Secure 支持的卡片处理。

1. 获取 Stripe API 密钥

- 在 https://stripe.com 注册
- 导航到开发者 → API 密钥
- 复制您的 可发布密钥（pk_live_... 或 pk_test_...）
- 复制您的 秘密密钥（sk_live_... 或 sk_test_...）

重要：

- 在开发中使用 测试密钥（pk_test_/sk_test_）
- 仅在生产中使用 实时密钥（pk_live_/sk_live_）
- 绝不要 将 API 密钥提交到版本控制中

2. 配置后端

添加到 `crm_config.yaml`（根级，而不嵌套在 `payment_vendors` 下）：

```yaml
stripe:
  secret_key: 'sk_live_YOUR_SECRET_KEY_HERE'
  publishable_key: 'pk_live_YOUR_PUBLISHABLE_KEY_HERE'
  currency: 'aud'
  statement_descriptor_suffix: 'YOUR_COMPANY'
```

API 在启动时从 `crm_config.yaml` 读取 Stripe 凭据。更改这些值需要重新启动 API。

3. 配置前端

添加到 UI `.env` 文件:

```
VITE_STRIPE_PUBLISHABLE_KEY=pk_live_YOUR_PUBLISHABLE_KEY_HERE
```

安全注意：仅可发布密钥放在前端。秘密密钥绝不能暴露给浏览器。

PayPal 配置

PayPal 提供卡��保管和 PayPal 账户支付。

1. 获取 **PayPal API** 凭据

- 在 https://developer.paypal.com/dashboard/ 创建一个 REST API 应用
- 复制您的 客户端 **ID** 和 客户端密钥
- 测试时使用 沙盒 凭据；生产时使用 实时 凭据

2. 配置后端

添加到 `crm_config.yaml`（根级）：

```yaml
paypal:
  client_id: 'YOUR_PAYPAL_CLIENT_ID'
  client_secret: 'YOUR_PAYPAL_CLIENT_SECRET'
  mode: 'sandbox'  # 'sandbox' 用于测试，'live' 用于生产
```

API 在启动时从 `crm_config.yaml` 读取 PayPal 凭据。更改这些值需要重新启动 API。

3. 配置前端

添加到 UI `.env` 文件:

```
VITE_PAYMENT_VENDORS="stripe,paypal"
VITE_PAYPAL_CLIENT_ID=YOUR_PAYPAL_CLIENT_ID
```

`VITE_PAYPAL_CLIENT_ID` 应与 `crm_config.yaml` 中的 `paypal.client_id` 值匹配。

沙盒与实时

	沙盒	实时
Stripe API	`sk_test_` / `pk_test_`	`sk_live_` / `pk_live_`
PayPal API	`api-m.sandbox.paypal.com`	`api-m.paypal.com`
PayPal `mode`	`sandbox`	`live`
资金	虚拟	真实

从沙盒切换到实时时，更新 两者 `crm_config.yaml`（凭据和模式）和 UI `.env`（可发布密钥 / 客户端 ID），然后重新构建 UI 并重新启动 API。

PCI 合��性

OmniCRM 如何维护 **PCI** 合规性:

- 卡数据直接输入到供应商托管的 iframe 中（Stripe Elements、PayPal Card Fields）
- OmniCRM 从不查看或存储 完整的卡号
- 仅在数据库中存储令牌化的支付方式
- 降低您业务的 PCI 合规性范围

支付处理指标

OmniCRM 提供全面的指标，以监控支付处理操作。有关 Stripe 支付指标的完整详细信息，包括 API 调用跟踪、支付量、失败率和响应时间，请参见 监控与指标。

数据库架构与供应商无关

数据库架构支持任何支付供应商，无需迁移：

示例 - 保存 Square 卡：

```
{
  "vendor": "square",
  "vendor_payment_method_id": "sq_card_abc123",
  "payment_type": "card"
}
```

无需代码更改。无需迁移。直接工作。

核心概念

数据模型

PaymentMethod

与供应商无关的客户支付方式存储。

```
{
  "payment_method_id": 789,
  "customer_id": 123,
  "vendor": "stripe",                    // 'stripe', 'paypal', 或任何添加的供应商
  "vendor_payment_method_id": "pm_xxx", // 供应商的内部 ID
  "payment_type": "card",                // 'card', 'paypal', 'ach', 等等
  "is_default": true,
  "card_brand": "visa",
  "card_last4": "4242",
  "card_exp_month": 12,
  "card_exp_year": 2025,
  "card_nickname": "My Visa Card",
  "status": "active"
}
```

PaymentAuthorization

两阶段提交授权记录（保留资金）。

```
{
  "authorization_id": 301,
  "customer_id": 123,
  "payment_method_id": 789,
  "vendor": "stripe",                    // 哪个供应商授权
  "vendor_authorization_id": "auth_xxx", // 供应商的授权 ID
  "amount": 200.00,
  "currency": "USD",
  "status": "authorized",                // 'authorized', 'captured', 'released'
  "authorized_at": "2025-12-27T10:00:00Z",
  "expires_at": "2026-01-03T10:00:00Z",
  "meta": {}
}
```

PaymentCapture

已捕获/完成的支付。

```
{
  "capture_id": 103,
  "authorization_id": 301,
  "customer_id": 123,
  "payment_method_id": 789,
  "vendor": "stripe",
  "vendor_transaction_id": "ch_xxx",      // 供应商的交易 ID
  "amount": 200.00,
  "currency": "USD",
  "status": "succeeded",                  // 'succeeded', 'failed',
'refunded'
  "captured_at": "2025-12-27T10:30:00Z",
  "vendor_response": {},                  // 完整的供应商响应
  "meta": {}
}
```

WalletAccount

客户钱包，带有余额跟踪（与客户一对一）。

```
{
  "wallet_account_id": 456,
  "customer_id": 123,
  "balance": 150.50,
  "currency": "USD",
  "auto_recharge_enabled": true,
  "auto_recharge_amount": 100.00,
  "auto_recharge_threshold": 10.00,
  "low_balance_warning_threshold": 10.00
}
```

WalletLedger

所有钱包交易的完整审计记录。

```
{
  "ledger_id": 501,
  "customer_id": 123,
  "wallet_account_id": 456,
  "transaction_type": "credit",          // 'credit', 'debit',
'refund', 'adjustment'
  "amount": 100.00,
  "balance_before": 150.50,
  "balance_after": 250.50,
  "currency": "USD",
  "description": "Card top-up",
  "reference_type": "payment_capture",    // 链接到相关对象
  "reference_id": 103,
  "meta": {},
  "created_at": "2025-12-27T10:35:00Z"
}
```

财务文档

有关发票模板和自定义的信息，请参见 客户发票。

发票

定义：发票包含 借记交易（收费）列表的文档。当调用 API 创建发票时，应提供借记交易 ID 的数组。这些交易通过设置其 `invoice_id` 字段与发票“链接”。

有关交易管理的详细信息，请参见 交易。

关键字段：

```
{
  "invoice_id": 12345,
  "invoice_number": "INV-2025-000001",    // 自动生成: INV-YYYY-NNNNNN
  "customer_id": 123,
  "title": "每月服务发票",
  "paid": true,                           // 支付状态
  "void": false,                          // 作废状态
  "payment_reference": "ch_xxxxx",        // 最后/主要支付 ID
  "payment_type": "stripe_capture",       // 最后支付类型
  "payment_time": "2025-12-27T10:30:00",
  "start_date": "2025-12-01",
  "end_date": "2025-12-31",
  "due_date": "2026-01-15",
  "retail_cost": 500.00,                  // 总发票金额
  "wholesale_cost": 250.00
}
```

发票生成：发票编号以 `INV-YYYY-NNNNNN` 格式自动生成，在日历年内顺序排列，每年 1 月 1 日重置。

发票上的支付

支付如何工作：一旦从借记交易创建发票，可以通过创建与发票链接的 信用交易（负的 `retail_cost`）来应用 支付。

支付应：

- 清楚地列在发票上，标记为“支付”
- 显示相关的支付日期（可能与发票创建日期不同）
- 支持每张发票的多次支付
- 净额与借记相抵消，以计算发票余额

发票状态：

- 已支付：总支付（信用）等于或超过总借记
- 部分支付：已应用部分支付，但余额仍然存在
- 超额支付：当前未处理 - 需要信用拆分（未来功能）

复式记账：系统实施适当的会计，每个收费都有一个抵消的支付：

```
// 1. 借记交易（收费）
{
  "transaction_id": 7001,
  "invoice_id": 12345,
  "retail_cost": 100.00,                    // 正数 = 客户欠款
  "title": "服务费"
}

// 2. 信用交易（支付）
{
  "transaction_id": 7002,
  "invoice_id": 12345,
  "retail_cost": -100.00,                   // 负数 = 收到的支付
  "title": "发票支付: 服务费 (12345)",
  "payment_type": "stripe_capture",
  "payment_reference": "ch_xxxxx"
}

// 净结果: $0 余额 → 发票标记为已支付
```

补充发票元数据：除了信用交易外，发票还存储摘要字段（`payment_reference`、`payment_type`、`payment_time`）以便快速查找。然而：

- 主要方法：通过 `invoice_id` 链接的信用交易（Hayden 的规范）
- 次要元数据：发票字段存储最后/主要支付的摘要
- 对于多次支付：查询信用交易以获取完整历史

声明

定义：声明显示客户在指定期间内的所有 借记和信用 交易。这□唯一一种类型的文档，其中借记和信用作为行项目一起显示，就像银行对账单一样。

信用票据

目的：仅适用于已应用支付的发票。必须作为发票作废过程的一部分进行。

流程：

1. 作废发票（连同其相关的借记交易）
2. 任何已应用的支付都链接到创建的信用票据

3. 客户的余额按等额进入信用
4. 信用票据余额可以：
    - 作为支付应用于另一张发票，或
    - 退款给客户

退款逻辑：如果选择退款，则根据相关的信用交易及其最初的支付方式进行调用（例如，如果信用交易类型 Stripe，则进行 Stripe 退款）。

支付方式 API

所有端点使用基本 URL: `/api/payments/`

有详细的支付方式管理和卡片处理，请参见 支付方式。

添加支付方式

为客户保存新的支付方式。

端点: `POST /api/payments/methods`

请求:

```
curl -X POST https://your-domain.com/api/payments/methods \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "vendor": "stripe",
    "payment_token": "pm_xxxxx",
    "is_default": false,
    "card_nickname": "My Visa Card"
  }'
```

请求体:

```
{
  "customer_id": 123,
  "vendor": "stripe",                    // 'stripe', 'paypal', 或任何添加的供应商
  "payment_token": "pm_xxxxx",           // 前端 SDK 的一次性令牌
  "is_default": false,                   // 设置为默认支付方式?
  "card_nickname": "My Visa Card"        // 可选的友好名称
}
```

响应（201 创建）：

```
{
  "success": true,
  "message": "支付方式添加成功",
  "data": {
    "payment_method_id": 789,
    "customer_id": 123,
    "vendor": "stripe",
    "payment_type": "card",
    "card_brand": "visa",
    "card_last4": "4242",
    "card_exp_month": 12,
    "card_exp_year": 2025,
    "card_nickname": "My Visa Card",
    "is_default": false,
    "status": "active"
  }
}
```

获取支付方式

检索客户的所有支付方式。

端点：`GET /api/payments/methods?customer_id={id}`

请求：

```
curl -X GET "https://your-domain.com/api/payments/methods?
customer_id=123" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

响应（200 OK）：

```
{
  "success": true,
  "data": [
    {
      "payment_method_id": 789,
      "customer_id": 123,
      "vendor": "stripe",
      "payment_type": "card",
      "card_brand": "visa",
      "card_last4": "4242",
      "is_default": true
    },
    {
      "payment_method_id": 790,
      "customer_id": 123,
      "vendor": "paypal",
      "payment_type": "paypal",
      "paypal_email": "user@example.com",
      "is_default": false
    }
  ]
}
```

获取默认支付方式

获取客户的默认支付方式。

端点：`GET /api/payments/methods/default?customer_id={id}`

请求：

```
curl -X GET "https://your-domain.com/api/payments/methods/default?
customer_id=123" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

响应（200 OK）：

```
{
  "success": true,
  "data": {
    "payment_method_id": 789,
    "vendor": "stripe",
    "payment_type": "card",
    "card_brand": "visa",
    "card_last4": "4242",
    "is_default": true
  }
}
```

设置默认支付方式

将支付方式设置为默认。

端点：`PUT /api/payments/methods/set-default`

请求：

```
curl -X PUT https://your-domain.com/api/payments/methods/set-
default \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "payment_method_id": 790
  }'
```

响应（200 OK）：

```
{
  "success": true,
  "message": "默认支付方式已更新",
  "data": {
    "payment_method_id": 790,
    "is_default": true
  }
}
```

删除支付方式

删除已保存的支付方式。

端点：`DELETE /api/payments/methods/{payment_method_id}?customer_id={id}`

请求：

```
curl -X DELETE "https://your-domain.com/api/payments/methods/789?
customer_id=123" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

响应（200 OK）：

```
{
  "success": true,
  "message": "支付方式删除成功"
}
```

支付流程 API

系统支持多种支付流程，具体取决于您的用例。

1. 直接支付（简单收费）

用例：简单的一步支付，无需服务提供。

端点: `POST /api/payments/charge`

请求:

```
curl -X POST https://your-domain.com/api/payments/charge \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 50.00,
    "currency": "USD",
    "payment_method_id": 789,
    "metadata": {
      "order_id": "12345"
    }
  }'
```

请求体:

```
{
  "customer_id": 123,
  "amount": 50.00,
  "currency": "USD",                    // 默认: USD
  "payment_method_id": 789,             // 可选 - 如果省略则使用默认
  "vendor": "stripe",                   // 如果使用 payment_token 则必需
  "payment_token": "pm_xxxxx",          // 可选 - 一次性令牌
  "save_method": false,                 // 保存支付方式以供将来使用?
  "metadata": {
    "order_id": "12345"
  }
}
```

响应 (200 OK):

```
{
  "success": true,
  "message": "支付成功",
  "data": {
    "transaction_id": "ch_xxxxx",
    "capture_id": 101,
    "amount": 50.00,
    "currency": "USD",
    "status": "succeeded"
  }
}
```

2. 发票支付（钱包优先路由）

用例：使用钱包余额支付发票，剩余部分使用卡支付。

端点：`POST /api/payments/invoice`

流程：

1. 检查钱包余额
2. 首先使用钱包资金
3. 对短缺部分收费（如果有）
4. 用卡金额信用钱包
5. 从钱包中借记全额发票金额

请求：

```
curl -X POST https://your-domain.com/api/payments/invoice \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 200.00,
    "payment_method_id": 789,
    "metadata": {
      "invoice_id": 12345,
      "description": "发票支付"
    }
  }'
```

响应（200 OK）：

```
{
  "success": true,
  "message": "发票支付处理成功",
  "data": {
    "customer_id": 123,
    "service_amount": 200.00,
    "routing_mode": "hybrid",
    "initial_balance": 150.00,
    "wallet_portion_used": 150.00,        // 钱包覆盖了这部分
    "card_portion_used": 50.00,           // 卡片收费剩余部分
    "charged_amount": 50.00,
    "wallet_credited": 50.00,
    "wallet_debited": 200.00,
    "final_balance": 0.00,
    "payment_method_used": true
  }
}
```

3. 授权保留（保留资金）

用例：保留资金以便稍后捕获（例如，酒店预订、租赁）。

> **Playbook** 实现：有关两阶段支付流程的 Ansible playbook 示例，请参见 两阶段提交模式

第 **1** 步：创建授权保留

端点：`POST /api/payments/authorize/hold`

钱包优先流程：

1. 检查钱包余额
2. 计算短缺（金额 - 钱包余额）
3. 仅为短缺部分授权卡
4. 钱包借记在捕获时发生，而不□在此时

请求：

```
curl -X POST https://your-domain.com/api/payments/authorize/hold \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 200.00,
    "payment_method_id": 789,
    "use_wallet": true,
    "metadata": {
      "reservation_id": "RES-001"
    }
  }'
```

请求体：

```
{
  "customer_id": 123,
  "amount": 200.00,
  "currency": "USD",
  "payment_method_id": 789,                // 可选 - 如果省略则使用默认
  "vendor": "stripe",                      // 如果使用 payment_token 则必需
  "payment_token": "pm_xxxxx",             // 可选 - 一次性令牌
  "save_method": false,
  "use_wallet": true,                      // 启用钱包优先路由（默认：true）
  "metadata": {
    "reservation_id": "RES-001"
  }
}
```

响应（200 OK）：

```
{
  "success": true,
  "message": "支付已授权（创建了保留）",
  "data": {
    "authorization_id": 301,
    "vendor_authorization_id": "auth_xxxxx",
    "amount": 200.00,
    "currency": "USD",
    "status": "authorized",
    "wallet_balance": 150.00,
    "wallet_to_use": 150.00,                    // 钱包将覆盖这部分
    "card_amount": 50.00,                       // 卡片授权这部分
    "message": "卡片授权 $50（钱包充值）。在捕获时将发生 $200 的钱包借记。"
  }
}
```

第 **2** 步：捕获授权

端点：`POST /api/payments/capture/{authorization_id}`

钱包优先捕获流程：

1. 捕获卡片（如果已授权） - 为钱包充值
2. 用捕获的卡金额信用钱包
3. 从钱包中借记全额服务金额
4. 创建发票/交易（如果请求）

请求：

```
curl -X POST https://your-domain.com/api/payments/capture/301 \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "amount": 200.00,
    "metadata": {
      "invoice": true,
      "title": "酒店预订",
      "description": "3晚住宿",
      "wholesale_cost": 100.00,
      "contract_days": 3,
      "send_email": true
    }
  }'
```

请求体:

```
{
  "amount": 200.00,                          // 可选 - 如果省略则捕□全额
  "metadata": {
    "invoice": true,                         // 创建发票和交易记录?
    "create_transaction": true,              // 创建交易记录?
    "title": "酒店预订",
    "description": "3晚住宿",
    "wholesale_cost": 100.00,
    "contract_days": 3,
    "send_email": true                       // 发送发票电子邮件?
  }
}
```

响应（200 OK）：

```
{
  "success": true,
  "message": "授权已捕获",
  "data": {
    "capture_id": 103,
    "transaction_id": "ch_xxxxx",
    "authorization_id": 301,
    "amount": 200.00,
    "currency": "USD",
    "status": "succeeded",
    "wallet_credit": {                    // 卡片为钱包充值
      "ledger_id": 401,
      "amount": 50.00
    },
    "wallet_debit": {                     // 从钱包中收取服务费
      "ledger_id": 402,
      "amount": 200.00
    },
    "transaction": {                      // 如果 invoice=true 则创建
      "transaction_id": 7001
    },
    "invoice": {                          // 如果 invoice=true 则创???
      "invoice_id": 12345,
      "invoice_number": "INV-2025-000001"
    }
  }
}
```

第 **3** 步：释放授权（取消）

端点：`POST /api/payments/release/{authorization_id}`

用例：取消预订、提供失败或客户改变主意。

请求：

```
curl -X POST https://your-domain.com/api/payments/release/301 \
  -H "Authorization: Bearer YOUR_API_KEY"
```

响应（200 OK）：

```
{
  "success": true,
  "message": "授权已释放",
  "data": {
    "authorization_id": 301,
    "vendor_authorization_id": "auth_xxxxx",
    "status": "released",
    "released_at": "2025-12-27T10:45:00Z"
  }
}
```

注意：在钱包优先流程中，由于钱包在捕获时才被借记，因此无需退款。

4. 充值支付（两阶段与提供）

用例：处理需要提供的服务充值支付（例如，热点/加密狗激活）。如果提供失败，则释放授权。

端点：`POST /api/payments/topup`

流程：授权 → 提供服务 → 捕获

请求：

```
curl -X POST https://your-domain.com/api/payments/topup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 30.00,
    "payment_method_id": 789,
    "service_uuid": "svc-uuid-123",
    "imsi": "123456789012345",
    "days": 30,
    "metadata": {
      "is_rental": false
    }
  }'
```

对于匿名租赁/热点（支付方式未保存）：

```
curl -X POST https://your-domain.com/api/payments/topup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 1,
    "amount": 5.00,
    "vendor": "stripe",
    "payment_token": "pm_xxxxx",
    "service_uuid": "hotspot-uuid",
    "imsi": "123456789012345",
    "days": 1,
    "metadata": {
      "is_rental": true,
      "billing_email": "user@example.com"
    }
  }'
```

响应（200 OK）：

```
{
  "success": true,
  "message": "充值支付处理成功",
  "data": {
    "transaction_id": "ch_xxxxx",
    "authorization_id": 302,
    "capture_id": 104,
    "amount": 30.00,
    "status": "succeeded",
    "provision_result": {
      "success": true,
      "topup_result": {...},
      "service_uuid": "svc-uuid-123",
      "imsi": "123456789012345",
      "days": 30
    },
    "payment_method_saved": false        // 匿名/租赁为假
  }
}
```

5. 租赁支付（第三方）

用例：向另一位客户的服务收费。

端点：`POST /api/payments/rental`

请求：

```
curl -X POST https://your-domain.com/api/payments/rental \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "beneficiary_customer_id": 456,
    "charge_customer_id": 123,
    "amount": 75.00,
    "payment_method_id": 789,
    "service_description": "租赁服务支付",
    "metadata": {
      "rental_agreement_id": "RA-001"
    }
  }'
```

对于匿名租赁（租赁者的卡未保存）：

```
curl -X POST https://your-domain.com/api/payments/rental \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "beneficiary_customer_id": 456,
    "amount": 75.00,
    "vendor": "stripe",
    "payment_token": "pm_xxxxx",
    "service_description": "匿名租赁支付",
    "metadata": {
      "billing_email": "renter@example.com"
    }
  }'
```

响应（200 OK）：

```
{
  "success": true,
  "message": "租赁支付处理成功",
  "data": {
    "transaction_id": "ch_xxxxx",
    "amount": 75.00,
    "payment_method_saved": false          // 匿名：未保存方法
  }
}
```

6. 退款支付

端点：`POST /api/payments/refund`

请求：

```
curl -X POST https://your-domain.com/api/payments/refund \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "transaction_id": "ch_xxxxx",
    "vendor": "stripe",
    "amount": 50.00,
    "reason": "customer_request"
  }'
```

请求体：

```
{
  "transaction_id": "ch_xxxxx",          // 供应商交易 ID
  "vendor": "stripe",                    // 'stripe', 'paypal' 等
  "amount": 50.00,                       // 可选 - 如果省略则全额退款
  "reason": "customer_request"           // 可选退款原因
}
```

响应（200 OK）：

```
{
  "success": true,
  "message": "退款处理成功",
  "data": {
    "refund_id": "re_xxxxx",
    "amount": 50.00,
    "status": "succeeded"
  }
}
```

钱包 API

所有钱包端点使用基本 URL: `/api/wallet/`

获取钱包余额

端点: `GET /api/wallet/balance?customer_id={id}`

请求:

```
curl -X GET "https://your-domain.com/api/wallet/balance?
customer_id=123" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

响应 (200 OK):

```
{
  "customer_id": 123,
  "balance": 150.50,
  "currency": "USD"
}
```

获取钱包信息

获取完整的钱包和信用信息，包括自动充值设置。

端点: `GET /api/wallet/info?customer_id={id}`

请求:

```
curl -X GET "https://your-domain.com/api/wallet/info?
customer_id=123" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

响应（200 OK）:

```
{
  "customer_id": 123,
  "wallet": {
    "wallet_account_id": 456,
    "balance": 150.50,
    "currency": "USD",
    "auto_recharge_enabled": true,
    "auto_recharge_amount": 100.00,
    "auto_recharge_threshold": 10.00,
    "low_balance_warning_threshold": 10.00
  }
}
```

钱包充值

通过收费支付方式为钱包充值。

端点: `POST /api/wallet/topup`

流程: 收费卡/PayPal → 信用钱包

请求:

```
curl -X POST https://your-domain.com/api/wallet/topup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 100.00,
    "payment_method_id": 789
  }'
```

请求体:

```
{
  "customer_id": 123,
  "amount": 100.00,
  "currency": "USD",                    // 默认: USD
  "payment_method_id": 789              // 可选 - 如果省略则使用默认
}
```

响应（200 OK）：

```
{
  "success": true,
  "payment": {
    "transaction_id": "ch_xxxxx",
    "amount": 100.00
  },
  "wallet": {
    "ledger_id": 503,
    "balance_after": 250.50
  },
  "message": "钱包充值 100.00 美元"
}
```

获取钱包交易

获取钱包交易历史（分类账）。

端点：`GET /api/wallet/transactions?customer_id={id}&limit={n}&offset={n}&type={type}`

请求：

```
curl -X GET "https://your-domain.com/api/wallet/transactions?customer_id=123&limit=50&offset=0&type=credit" \
  -H "Authorization: Bearer YOUR_API_KEY"
```

查询参数：

- `customer_id`（必需）：客户 ID
- `limit`（可选）：记录数（默认：50）
- `offset`（可选）：分页偏移量（默认：0）
- `type`（可选）：按交易类型筛选（'credit'、'debit'、'refund'、'adjustment'）

响应（200 OK）：

```
{
  "customer_id": 123,
  "count": 2,
  "transactions": [
    {
      "ledger_id": 501,
      "transaction_type": "credit",
      "amount": 100.00,
      "balance_before": 150.50,
      "balance_after": 250.50,
      "description": "□□□□",
      "reference_type": "payment_capture",
      "reference_id": 103,
      "created_at": "2025-12-27T10:35:00Z"
    },
    {
      "ledger_id": 502,
      "transaction_type": "debit",
      "amount": 50.00,
      "balance_before": 250.50,
      "balance_after": 200.50,
      "description": "□□□",
      "reference_type": "service_charge",
      "reference_id": 789,
      "created_at": "2025-12-27T11:00:00Z"
    }
  ]
}
```

API 参考摘要

支付方式端点

方法	端点	描述
POST	`/api/payments/methods`	添加支付方式
GET	`/api/payments/methods?customer_id={id}`	获取所有支付方式
GET	`/api/payments/methods/default?customer_id={id}`	获取默认支付方式
PUT	`/api/payments/methods/set-default`	设置默认支付方式
DELETE	`/api/payments/methods/{id}?customer_id={id}`	删除支付方式

支付流程端点

方法	端点	描述
POST	`/api/payments/charge`	直接支付（一步）
POST	`/api/payments/invoice`	发票支付（钱包优先）
POST	`/api/payments/topup`	充值与提供
POST	`/api/payments/authorize/hold`	创建授权保留
POST	`/api/payments/capture/{id}`	捕获授权
POST	`/api/payments/release/{id}`	释放授权
POST	`/api/payments/rental`	租赁/第三方支付
POST	`/api/payments/refund`	退款支付

钱包端点

方法	端点	描���
GET	`/api/wallet/balance?customer_id={id}`	获取钱包余额
GET	`/api/wallet/info?customer_id={id}`	获取钱包信息 + 设置
POST	`/api/wallet/topup`	通过支付方式为钱包充值
GET	`/api/wallet/transactions?customer_id={id}`	获取交易历史

PayPal 特定端点

方法	端点	描述
POST	`/api/payments/paypal/vault/setup-token`	为卡片字段 SDK 创建 PayPal 设置令牌
POST	`/api/payments/paypal/vault/finalize`	完成 PayPal 保管并保存支付方式
POST	`/api/payments/paypal/vault/update-setup-token`	更新 3DS/SCA 处理的设置令牌

常见用例

用例 **1**：向客户收费（预付）

场景：客户使用钱包优先路由支付 30 天的服务。

```
# 发票支付（钱包优先，卡片支付短缺部分）
curl -X POST https://your-domain.com/api/payments/invoice \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 99.99,
    "metadata": {
      "service_id": 456,
      "contract_days": 30,
      "description": "30天高级服务"
    }
  }'
```

响应：

```
{
  "success": true,
  "message": "票支付处理成功",
  "data": {
    "service_amount": 99.99,
    "wallet_portion_used": 50.00,          // 钱包有 $50
    "card_portion_used": 49.99,            // 卡片收费 $49.99
    "final_balance": 0.00
  }
}
```

用例 2：两阶段支付与服务提供

场景：仅在授权支付后提供服务。如果提供失败，则不收费。

```
# 充值支付与自动提供
curl -X POST https://your-domain.com/api/payments/topup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 149.99,
    "payment_method_id": 789,
    "service_uuid": "fiber-svc-uuid",
    "imsi": "123456789012345",
    "days": 30,
    "metadata": {
      "service_type": "fiber_internet"
    }
  }'
```

流程：

1. 在卡上授权 $149.99
2. 提供光纤互联网服务
3. 如果提供成功 → 捕获支付
4. 如果提供失败 → 释放授权（不收费）

用例 3：热点匿名支付

场景：匿名用户支付 WiFi 热点访问费。没有客户记录，没有保存的支付方式。

```
# 匿名热点支付
curl -X POST https://your-domain.com/api/payments/topup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 1,
    "amount": 5.00,
    "vendor": "stripe",
    "payment_token": "pm_xxxxx",
    "service_uuid": "hotspot-downtown",
    "imsi": "999999999999999",
    "days": 1,
    "metadata": {
      "is_rental": true,
      "billing_email": "user@example.com",
      "hotspot_location": "市中心咖啡馆"
    }
  }'
```

结果：

- □□□□□□ □
- □□□□□□ □
- □□□□□□□ □
- □□□□□□□ □

用例 4：酒店预订（授权保留 + 捕获）

场景：为酒店预订保留资金，入住时捕获，取消时释放。

```
# 第 1 步: 入住 - 保留资金
curl -X POST https://your-domain.com/api/payments/authorize/hold \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 500.00,
    "use_wallet": true,
    "metadata": {
      "reservation_id": "RES-2025-001"
    }
  }'

# 响应: {"authorization_id": 301, "status": "authorized"}

# 第 2a 步: □户入住 - 捕获
curl -X POST https://your-domain.com/api/payments/capture/301 \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "metadata": {
      "invoice": true,
      "title": "酒店���宿 - 3 晚",
      "description": "302 房间, 12 月 27 日至 30 日",
      "send_email": true
    }
  }'

# 第 2b 步: □户取消 - 释放保留
curl -X POST https://your-domain.com/api/payments/release/301 \
  -H "Authorization: Bearer YOUR_API_KEY"
```

用例 5: 添加支付方式 (Stripe)

场景: 客户将新的 Visa 卡添加到其帐户。

```
# 第 1 步: 前端通过 Stripe.js 创建 Stripe 令牌
# const {token} = await stripe.createToken(cardElement);

# 第 2 步: 后端保存支付方式
curl -X POST https://your-domain.com/api/payments/methods \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "vendor": "stripe",
    "payment_token": "pm_xxxxx",
    "is_default": true,
    "card_nickname": "工作 Visa"
  }'
```

用例 6: 钱包充值

场景: 客户使用保存的支付方式为其钱包充值 $100。

```
curl -X POST https://your-domain.com/api/wallet/topup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 100.00
  }'
```

响应:

```
{
  "success": true,
  "payment": {
    "transaction_id": "ch_xxxxx",
    "amount": 100.00
  },
  "wallet": {
    "balance_after": 250.50
  },
  "message": "钱包充值 100.00 美元"
}
```

错误处理

常见错误响应

资金不足（400 错误请求）：

```
{
  "error": "钱包余额不足。可用: 50.00, 所需: 200.00"
}
```

支付失败（400 错误请求）：

```
{
  "error": "支付失败。请重试。"
}
```

验证错误（400 错误请求）：

```
{
  "error": "customer_id 和 amount 是必需的"
}
```

未找到（404 未找到）：

```
{
  "error": "授权 999 未找到"
}
```

服务器错误（500 内部服务器错误）：

```
{
  "error": "处理您的支付时发生错误"
}
```

退款和钱包优先路由

退款选项

系统支持 两种类型的退款，具体取决于您的业务需求：

1. 退款到支付来源（Stripe/PayPal）

用例：客户请求因取消服务或缺陷产品而全额退款。

流程：资金返回到原始支付方式（卡、PayPal 账户等）。

端点：`POST /api/payments/refund`

示例：

```
curl -X POST https://your-domain.com/api/payments/refund \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "transaction_id": "ch_xxxxx",
    "vendor": "stripe",
    "amount": 100.00,
    "reason": "customer_request"
  }'
```

结果：

- Stripe/PayPal 将退款处理到原始支付方式
- 客户在 5-10 个工作日内在其卡/PayPal 账户上看到信用
- 资金不会进入钱包
- 在 `PaymentCapture` 表中维护完整的审计记录

何时使用：

- □ □□□□□□
- □ □□□□□
- □ □□□□
- □ □□□□□□□□□□

2. 退款到钱包

用例：部分退款、服务信用或保留资金以供未来购买。

流程：资金立即记入客户的钱包余额以供立即使用。

注意：钱包信用通常在错误场景中由系统内部处理。有关手动钱包信用的信息，请联系支持或使用管理员工具。

结果：

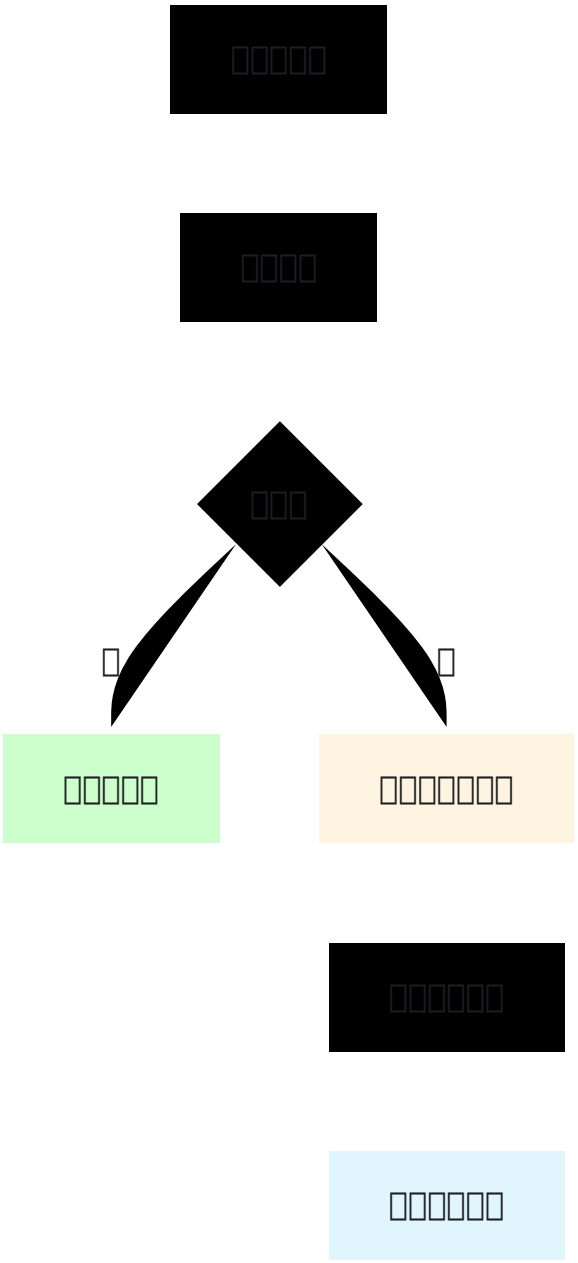
- 资金立即可用在钱包中
- 无等待期
- 可用于未来购买
- 在 `WalletLedger` 表中维护完整的审计记录

□□□□□

- □ □□□□□□□□□□□□□□□□□
- □ □□□□□□□□□□□□□
- □ □□□□□□□□□□□
- □ □□□□

□□□□□□

□□□□□□□□□□□□□□□□



□□□□□□

1. □□□□ $100 □□□□□□
2. □□□□□□□□
3. □□□□□
4. □□□□ $100 □□□□□□5-10 □ + □□□□□□□□
   - □□□ $100 □□□□□□
   - □□□□□□□□□□□□□
   - □□□□□□
   - □□□□□□□□□

钱包优先路由：优化的卡收费

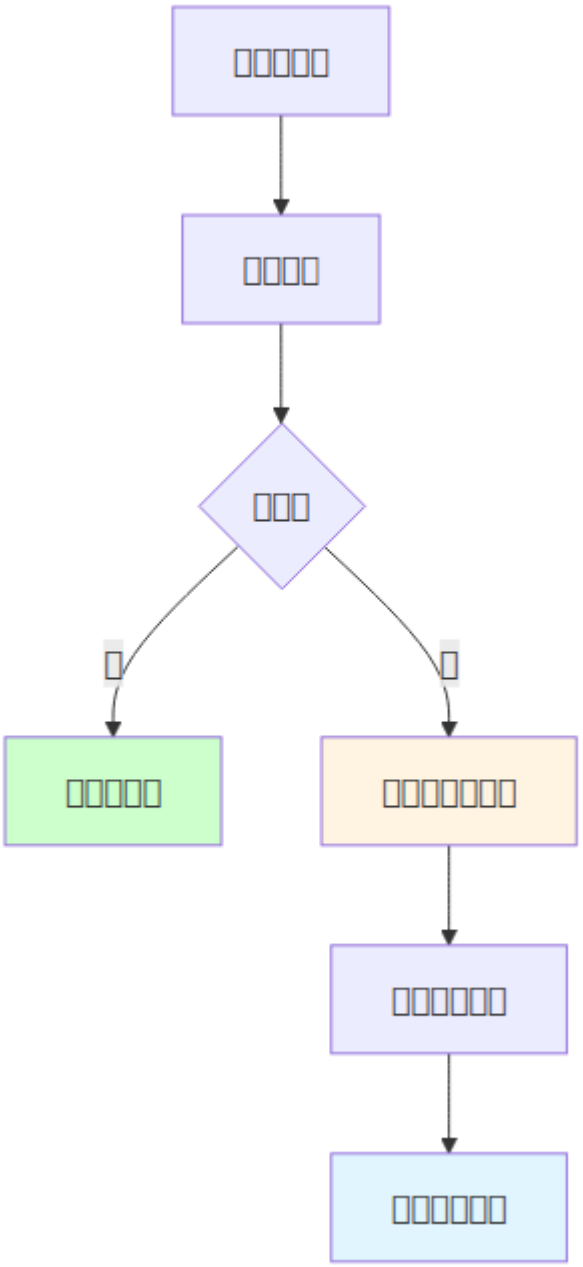
系统 仅对短缺部分收费，而不□全额，当您有钱包余额时。

示例 **1**: **$1** 钱包余额 **+ $10** 购买

场景：您在钱包中有 $1，想购买 $10 的附加服务。

传统支付流程（不□该系统的工作方式）：

OmniCRM 钱包优先流程：



API □□□

```
curl -X POST https://your-domain.com/api/payments/invoice \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 10.00,
    "metadata": {
      "addon_id": 456
    }
  }'
```

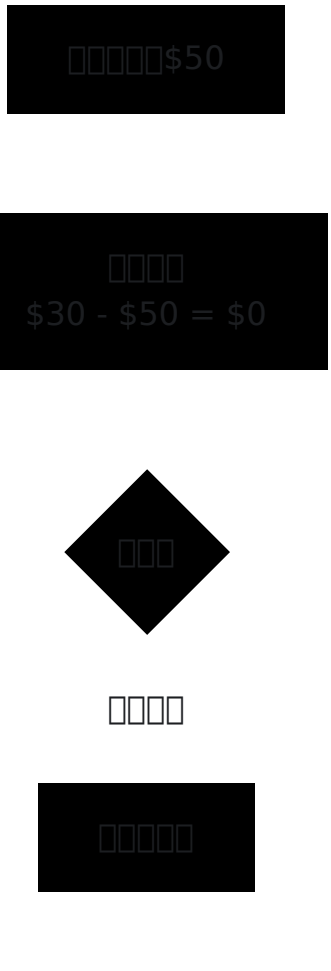
API 响应:

```
{
  "success": true,
  "message": "发票支付处理成功",
  "data": {
    "customer_id": 123,
    "service_amount": 10.00,
    "routing_mode": "hybrid",
    "initial_balance": 1.00,
    "wallet_portion_used": 1.00,      // 使用现有的 $1
    "card_portion_used": 9.00,        // 仅收费 $9 短缺
    "charged_amount": 9.00,           // ← 卡收费
    "wallet_credited": 9.00,          // ← 钱包充值
    "wallet_debited": 10.00,          // ← 服务费用
    "final_balance": 0.00
  }
}
```

示例 **2**: **$50** 钱包余额 **+ $30** 购买

场景: 您在钱包中有 $50, 购买费用为 $30。

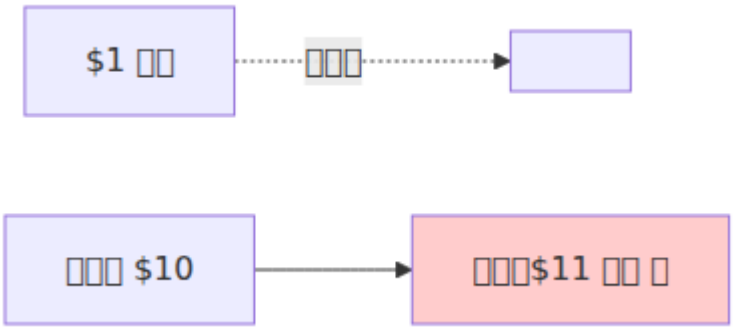
钱包优先流程:



没有短缺

最终余额: $20
卡收费: $0
钱包使用: $30 □

API 响应:



```
{
  "success": true,
  "data": {
    "service_amount": 30.00,
    "initial_balance": 50.00,
    "charged_amount": 0,              // ← 无卡收费
    "wallet_debited": 30.00,
    "final_balance": 20.00,
    "payment_method_used": false      // ← 卡未使用
  }
}
```

示例 **3**：授权保留与钱包余额

场景：酒店保留 $500 授权，钱包余额为 $150。

第 **1** 步：授权（`POST /api/payments/authorize/hold`）：

API 响应：

```
{
  "authorization_id": 301,
  "amount": 500.00,
  "status": "authorized",
  "wallet_balance": 150.00,
  "wallet_to_use": 150.00,
  "card_amount": 350.00,
  "message": "卡片授权 $350（钱包充值）。在捕获时将发生 $500 的钱包借记。"
}
```

第 **2** 步：捕获（`POST /api/payments/capture/301`）：

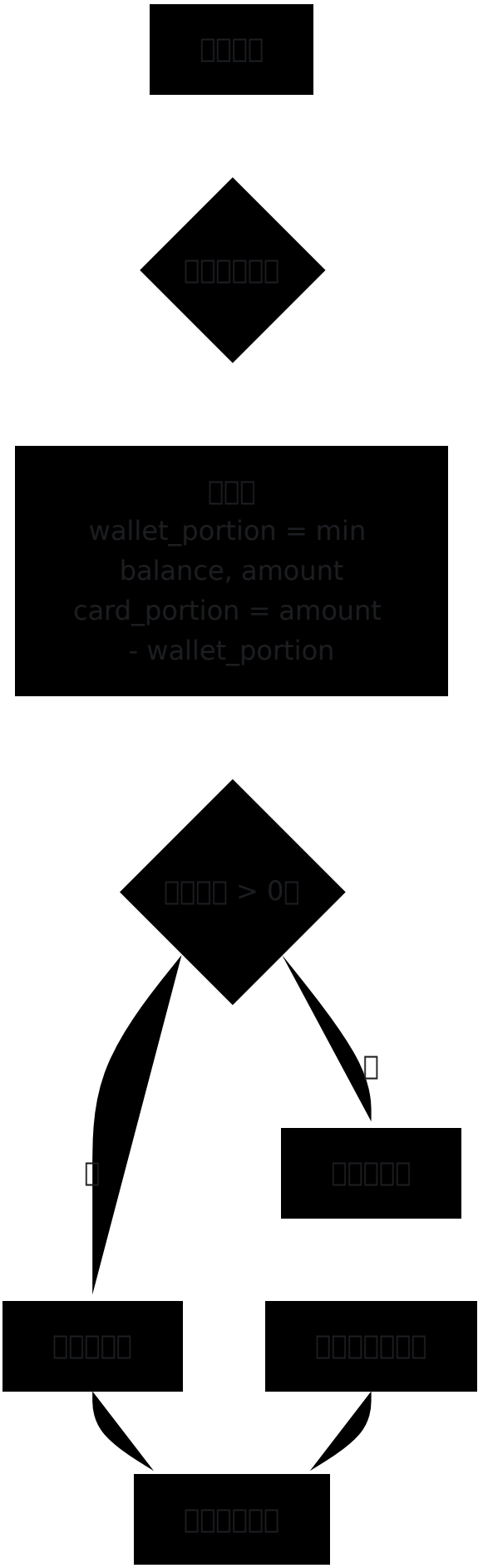
最终余额：$0
客户收费：$350

这很重要：

- □ □□□□ $350 “保留”在其卡上，而不□ $500
- □ □□□□□□□□□□□
- □ □□□□□□□□
- □ □□□□□□□

实施细节

钱包优先路由逻辑：


□□□□
□□□□□□
□□□
wallet_portion = min
balance, amount
card_portion = amount
- wallet_portion
□□□□ > 0□
□
□
□□□□□
□□□□□
□□□□□□□□
□□□□□□

工作原理：

1. 系统检查当前钱包余额
2. 计算短缺：`amount - wallet_balance`
3. 如果短缺 > 0，则仅对短缺部分收费
4. 用卡支付金额信用钱包
5. 借记钱包全额服务金额
6. 结果：客户仅对钱包无法覆盖的部分收费

路由模式覆盖

如果需要，您可以覆盖钱包优先行为：

绕过模式 - 即使钱包有资金也始终收费卡：

```
curl -X POST https://your-domain.com/api/payments/charge \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "customer_id": 123,
    "amount": 100.00,
    "metadata": {
      "routing_mode": "bypass"
    }
  }'
```

结果：卡收费 $100，钱包信用 $100，钱包借记 $100（净：$0）

用例：客户希望使用卡以获得奖励/积分，即使有钱包余额。

最佳实践

1. 始终使用钱包优先路由 进行客户支付，以启用预付功能
2. 使用两阶段支付（充值端点）时提供服务，以避免在提供失败时收费
3. 为所有支付设置元数据 以维护审计记录
4. 仅对真正的匿名用户使用匿名支付（热点、租赁）
5. 优雅地处理错误 并提供清晰的用户反馈
6. 测试支付流程 在成功和失败场景中
7. 监控授权过期（通常为卡片 7 天）
8. 实施自动充值 以改善客户体验

Playbook 特定最佳实践

在 Ansible playbooks 中实现支付流程时：

1. 始终使用块/恢复模式 - 将提供包装在 try/catch 中以实现自动回滚
2. 存储 **authorization_id** - 保存用于捕获/释放操作
3. 验证 **API** 响应 - 在继续之前断言成功
4. 四舍五入货币值 - 始终使用 2 位小数
5. 检查支付方式 - 在授权之前验证客户□否具有默认支付方式

有关完整详细信息和示例，请参见 Playbook 最佳实践。

��份验证

所有 API 端点都需要通过 API 密钥进行身份验证，密钥在 `Authorization` 头中：

```
Authorization: Bearer YOUR_API_KEY
```

请联系您的系统管理员以获取 API 密钥。

供应商支持

当前支持

- **Stripe** - 完全支持（卡、ACH）
- **PayPal** - 完全支持（PayPal 账户、通过卡片字段 SDK 的卡）

添加新供应商

模块化架构使添加新的支付供应商变得简单。有关详细信息，请参见 模块化架构 部分。

相关文档

实施指南

- 从 **Playbooks** 进行收费和支付 - Ansible playbook 实施
  - 两阶段提交模式
  - 授权/捕获示例
  - 直接交易创建
  - 按比例计算

特性特定指南

- 客户发票 - 发票生成、模板和自定义
- 支付方式 - 管理客户支付方式、卡和 PayPal 账户
- 交易 - 交易管理、手动收费和信用
- 支付处理 - 支付工作流程和��理
- 计费基础知识 - 客户计费基础

快速导航

- 添加供应商？ → 请参见 模块化架构
- 配置 **Stripe/PayPal**？ → 请参见 供应商配置

- 自定义发票？ → 请参见 客户发票
- 提供服务？ → 请参见 Playbook 集成

客户发票

> **API** 参考：有关发票支付 API 和工作流程，请参见 支付系统 **API** 指南

`Transactions </payments_transaction>` 被组合在一起以形成发票，并发送给客户进行支付。

发票具有开始和结束日期，表示发票所涵盖的期间，以及到期日期，即发票到期支付的日期。

发票可以由系统自动生成，例如，当服务计费时，会为零售成本创建发票，或者可以手动创建，例如，如果客户请求发票副本，或者如果客户被收取一次性费用。

客户发票完全使用 `Mailjet <integrations_mailjet>` 模板化，可以自定义以包括公司徽标、地址和支付详情，并可以通过电子邮件发送给客户，或下载为 PDF。

自定义发票模板

OmniCRM 使用 HTML 模板和 Jinja2 模板生成发票。您可以完全自定义发票设计、品牌、颜色和布局。

发票模板位置

发票模板存储在 `OmniCRM-API/invoice_templates/`

默认模板：

- `norfone_invoice_template.html` - 示例发票模板
- `cifi_invoice_template.html` - 替代模板示例

配置：

活动发票模板在 `OmniCRM-API/crm_config.yaml` 中指定：

```
invoice:
  template_filename: 'norfone_invoice_template.html'
```

可用模板变量

发票模板可以访问以下 Jinja2 变量：

发票信息：

- `{{ invoice_number }}` - 唯一发票 ID（例如，`INV-2025-001234`）
- `{{ date }}` - 发票发行日期（ISO 格式：`2025-01-10T12:00:00`）
- `{{ due_date }}` - 支付到期日期（例如，`2025-02-10`）
- `{{ start_date }}` - 计费期间开始日期
- `{{ end_date }}` - 计费期间结束日期
- `{{ total_amount }}` - 税前总发票金额（数字）
- `{{ total_tax }}` - 从所有交易计算的总税额（数字）

客户信息：

- `{{ client.name }}` - 客户的全名或公司名称
- `{{ client.address.address_line_1 }}` - 地址行 1
- `{{ client.address.address_line_2 }}` - 地址行 2
- `{{ client.address.city }}` - 城市
- `{{ client.address.state }}` - 州/省
- `{{ client.address.zip_code }}` - 邮政编码
- `{{ client.address.country }}` - 国家

交易项目：

使用以下代码循环遍历交易：

```
{% for sub_transaction in transactions %}
    <tr>
        <td>{{ sub_transaction.transaction_id }}</td>
        <td>{{ sub_transaction.created.split("T")[0] }}</td>
        <td>{{ sub_transaction.title }}</td>
        <td>{{ sub_transaction.description }}</td>
        <td>${{ "%.2f"|format(sub_transaction.retail_cost) }}</td>
    </tr>
{% endfor %}
```

交易字段：

- `sub_transaction.transaction_id` - 交易 ID

- `sub_transaction.created` - 交易日期/时间
- `sub_transaction.title` - 交易标题
- `sub_transaction.description` - 详细描述
- `sub_transaction.retail_cost` - 项目金额
- `sub_transaction.tax_percentage` - 应用的税率（例如，10 表示 10%）
- `sub_transaction.tax_amount` - 以美元计算的税额

在模板中显示税：

```
<td>
    {% if sub_transaction.tax_amount and
sub_transaction.tax_amount > 0 %}
        ${{ "%.2f"|format(sub_transaction.tax_amount) }} ({{
sub_transaction.tax_percentage }}%)
    {% else %}
        -
    {% endif %}
</td>
```

创建自定义发票模板

步骤 **1**：复制现有模板

```
cd OmniCRM-API/invoice_templates/
cp norfone_invoice_template.html
your_company_invoice_template.html
```

步骤 **2**：自定义 **HTML/CSS**

编辑 `your_company_invoice_template.html` 以匹配您的品牌：

关键自定义区域：

1. 公司徽标和品牌

```
<!-- 替换为您的徽标 URL -->


<!-- 更新公司名称 -->
<h1>您的公司名称</h1>
```

2. 配色方案

```
<style>
    /* 主要品牌颜色 */
    .navbar {
        background: linear-gradient(to bottom right, #your-color-1, #your-color-2);
    }

    /* 表头 */
    .table thead th {
        background-color: #your-brand-color !important;
        color: white !important;
    }

    /* 按钮和链接 */
    .btn-primary {
        background-color: #your-brand-color;
    }
</style>
```

3. 公司信息页脚

```
<footer>
    <p>您的公司名称</p>
    <p>123 商务街，城市，国家</p>
    <p>电话: +1-555-123-4567 | 电子邮件: billing@yourcompany.com</p>
    <p>ABN/税号: 12345678900</p>
</footer>
```

4. 支付说明

```
<div class="payment-info">
    <h3>支付方式</h3>
    <p><strong>在线: </strong> 在 https://yourcompany.com/pay 支付
</p>
    <p><strong>银行转账: </strong></p>
    <ul>
        <li>账户名称: Your Company Ltd</li>
        <li>BSB: 123-456</li>
        <li>账户号码: 987654321</li>
        <li>参考: {{ invoice_number }}</li>
    </ul>
</div>
```

5. 条款和条件

```
<div class="terms">
    <h4>付款条款</h4>
    <p>发票日期后 30 天内到期付款。</p>
    <p>逾期付款费用: 逾期余额每月 2%。</p>
    <p>有关账单查询: billing@yourcompany.com</p>
</div>
```

步骤 **3**: 更新配置

编辑 `OmniCRM-API/crm_config.yaml`:

```
invoice:
  template_filename: 'your_company_invoice_template.html'
```

步骤 **4**: 重启 **API**

```
cd OmniCRM-API
sudo systemctl restart omnicrm-api
```

步骤 **5**: 测试发票生成

1. 导航到有交易的客户
2. 生成测试发票

3. 下载 PDF 以验证格式
4. 将发票通过电子邮件发送给自己以测试电子邮件发送

高级自定义

条件内容:

使用 Jinja2 条件显示/隐藏内容:

```
{% if total_amount > 1000 %}
    <div class="high-value-notice">
        <p><strong>注意: </strong> 大额余额 - 可根据请求提供付款计划。</p>
    </div>
{% endif %}

{% if client.address.country == "Australia" %}
    <p>包含 GST: ${{ "%.2f"|format(total_amount * 0.10) }}</p>
{% endif %}
```

多语言支持:

创建特定语言的模板:

```
invoice_template_en.html
invoice_template_es.html
invoice_template_fr.html
```

根据客户的语言偏好进行配置。

自定义计算:

```
<!-- 显示小计和税务明细 -->
<tr>
    <td colspan="4" class="text-right"><strong>小计: </strong></td>
    <td>${{ "%.2f"|format(total_amount) }}</td>
</tr>
<tr>
    <td colspan="4" class="text-right"><strong>税: </strong></td>
    <td>${{ "%.2f"|format(total_tax) }}</td>
</tr>
<tr>
    <td colspan="4" class="text-right"><strong>总计: </strong></td>
    <td>${{ "%.2f"|format(total_amount + total_tax) }}</td>
</tr>
```

注意：`total_tax` 变量□通过对发票中所有交易的 `tax_amount` 求和自动计算的。每个交易的税□根据其 `tax_percentage` 字段计算的，如果未指定，则默认为产品的 `tax_percentage` 或 0%。

支付的二维码：

生成用于移动支付的二维码：

```
<div class="qr-payment">
    ![扫描支付](https://api.qrserver.com/v1/create-qr-code/?
size=150x150&data={{ payment_url }})
    <p>用手机扫描以立即支付</p>
</div>
```

PDF 样式最佳实践

OmniCRM 使用 **WeasyPrint** 将 HTML 转换为 PDF。请遵循以下指南：

支持的 **CSS**：

- 大多数 CSS 2.1 属性
- 有限的 CSS3（flexbox，一些转换）
- 通过 `@font-face` 使���网页字体

不支持：

- JavaScript
- CSS 网格（请使用表格）
- 复杂动画
- 一些现代 CSS 属性

页面大小和边距：

```
@page {
    size: A4;
    margin: 1cm;
}

body {
    font-family: Arial, sans-serif;
    font-size: 10pt;
}
```

打印特定样式：

```
@media print {
    .no-print {
        display: none;
    }

    .page-break {
        page-break-after: always;
    }
}
```

表格布局：

```
.table {
    table-layout: fixed;
    width: 100%;
}

.table th, .table td {
    word-wrap: break-word;
    padding: 4px;
}
```

字体嵌入：

对于自定义字体，使用安全的网页字体或嵌入：

```
@font-face {
    font-family: 'YourFont';
    src: url('https://yourcompany.com/fonts/yourfont.woff2')
format('woff2');
}

body {
    font-family: 'YourFont', Arial, sans-serif;
}
```

测试发票模板

测试清单：

1. 视觉检查：
    - 徽标正确显示
    - 颜色符合品牌指南
    - 文本可读（不太小）
    - 表格对齐正确
    - 所有部分存在
2. 数据准确性：
    - 客户详细信息正确
    - 交易金额正确相加

- 日期格式正确
- 所有变量正确替换

3. **PDF** 质量:
   - 文件大小合理（<5MB）
   - 图像清晰锐利
   - 无文本截断或溢出
   - 页面在适当位置换行
4. 多页发票:
   - 每页重复标题
   - 显示页码
   - 长交易列表正确分页
5. 电子邮件发送:
   - PDF 附加到电子邮件
   - 文件大小低于 Mailjet 限制（15MB）
   - 在 Gmail、Outlook、Apple Mail 中呈现

测试命令（手动生成）:

您可以通过 API 测试发票生成:

```
curl -X GET "http://localhost:5000/crm/invoice/{invoice_id}/pdf" \
     -H "Authorization: Bearer YOUR_TOKEN" \
     --output test_invoice.pdf
```

常见模板问题

变量未替换:

- 原因：变量名称拼写错误或缺少数据
- 修复：检查拼写□否完全正确（区分大小写），验证数据库中□否存在数据

PDF 样式损坏:

- 原因：不支持的 CSS 属性
- 修复：使用 CSS 2.1 属性，测试与 WeasyPrint 兼容的 CSS

图像未显示：

- 原因：相对 URL 或被阻止的外部资源
- 修复：使用绝对 HTTPS URL，确保图像可以公开访问

表格溢出页面：

- 原因：固定列宽度过宽
- 修复：使用百分比宽度，`table-layout: fixed`

字体未呈现：

- 原因：字体未嵌入或不可用
- 修复：使用安全的网页字体（Arial、Times New Roman 等）或正确嵌入自定义字体

PDF 生成失败：

- 原因： HTML 语法错误或 WeasyPrint 崩溃
- 修复：验证 HTML，检查 WeasyPrint 日志，简化复杂布局

发票 **PDF** 缓存

为了提高性能并减少冗余 PDF 生成，OmniCRM 包含发票 PDF 缓存系统。当首次生成发票 PDF 时，它会被缓存到数据库中以供后续请求使用。

PDF 缓存的工作原理：

1. 首次请求 - 当请求发票 PDF（下载或电子邮件）时，系统：
    - 从发票模板生成 PDF
    - 将 PDF 编码为 Base64
    - 计算 PDF 内容的 SHA256 哈希
    - 存储在 `Invoice_PDF_Cache` 表中，包含：
        - 发票 ID 参考
        - PDF 数据（Base64 编码）
        - 文件名
        - 内容哈希（用于完整性验证）
        - 创建时间戳

2. 后续请求 - 当再次请求相同发票时：
    - 系统根据 invoice_id 检查缓存 PDF
    - 如果缓存存在且有效，则立即返回缓存的 PDF
    - 更新 `last_accessed` 时间戳以跟踪缓存使用情况
3. 缓存失效 - 当以下情况发生时，缓存的 PDF 会失效：
    - 发票被修改（添加/删除交易，详细信息更改）
    - 发票模板被更新
    - 手动触发缓存清除

好处：

- 性能 - 对于重复请求，立即提供 PDF（无重新生成延迟）
- 一致性 - 所有发票下载的 PDF 相同（除非发票被修改）
- 服务器负载 - 减少 PDF 生成的 CPU 使用
- 用户体验 - 在初始生成期间显示加载指示器，后续请求立即响应

缓存管理：

发票 PDF 缓存由系统自动管理。可以根据以下条件定期清除旧的或未使用的缓存条目：

- 年龄（例如，删除超过 90 天的缓存条目）
- 访问模式（删除 30 天未访问的条目）
- 存储限制（如有必要，实施缓存大小限制）

API 行为：

通过 API 或 UI 下载发票时：

- 首次请求：在生成 PDF 时显示加载指示器，然后缓存
- 后续请求：从缓存立即下载
- 缓存命中/未命中对用户透明

重要提示：当您更新发票模板时，请清除缓存以确保新发票使用更新的设计：

```
-- 清除所有缓存的发票 PDF（在 MySQL 中运行）
DELETE FROM Invoice_PDF_Cache;
```

或更新 `crm_config.yaml` 以在模板更改时自动使缓存失效。

访问发票

可以在系统级别或按客户查看发票：

按客户查看：

1. 导航到 客户 → 【选择客户】
2. 点击 账单 选项卡
3. 在第三张卡中查看发票列表

系统范围内查看：

1. 导航到 账单 → 发票（从主菜单）
2. 查看所有客户的所有发票

发票统计小部件

在发票页面顶部，四个统计卡显示财务摘要。

小部件描述：

- 总发票 - 所有发票零售成本的总和（所有时间）和发送的发票数量

- 未支付发票 - 尚未支付的发票总和和未支付发票数量
- 本月发票 - 本日历月创建的发票总和及数量
- 上月发票 - 上日历月创建的发票总和及数量

值格式化：

- 超过 1,000 的值：显示为 "k" 后缀（例如，$1.5k）
- 超过 1,000,000 的值：显示为 "M" 后缀（例如，$2.3M）
- 超过 1,000,000,000 的值：显示为 "B" 后缀（例如，$1.1B）

趋势指标：

- “本月”和“上月”的小部件显示百分比变化
- 绿色向上箭头：与前一时期相比增加
- 红色向下箭头：与前一时期相比减少
- 灰色向右箭头：没有变化

发票列表

发票表显示所有发票，包含以下列：

列描述：

- **ID** - 唯一发票 ID
- 标题 - 发票标题/描述
- 期间 - 计费期间（开始日期 - 结束日期）或一次性发票的 "N/A"
- 到期日期 - 支付到期日期
- 创建 - 发票创建日期
- 金额 - 总发票金额（零售成本）
- 状态 - 已支付、未支付或已退款
- 操作 - 可用操作（根据状态而异）

操作图标：

- ⬇（下载）- 下载发票 PDF
- 🗑（删除）- 作废发票（仅在未支付时）
- 💳（支付）- 在线支付发票（仅在未支付时）
- ✉（电子邮件）- 将发票电子邮件发送给客户
- ↩（退款）- 退款 Stripe 支付（仅针对已支付的 Stripe 发票）

生成发票

点击 "+ 生成形式发票" 创建新发票。

字段描述：

- 搜索客户 - 选择客户（仅在系统范围视图中显示，在客户视图中预填）
- 标题 - 发票标题/名称（可选，默认为 "发票 [期间]"）
- 开始日期 - 计费期间开始（默认为 14 天前）
- 结束日期 - 计费期间结束（默认为今天）
- 到期日期 - 支付截止日期（默认为今天）
- 交易预览 - 显示日期范围内所有未开票交易，并能够包含/排除特定交易

交易选择：

- ✓（绿色加号）- 点击以排除发票中的交易
- ×（红色 X）- 点击以包含先前排除的交易
- 全选 - 包含所有显示的交易
- 清除所有 - 排除所有交易
- 被排除的交易以灰色显示并带有删除线文本
- 实时总计在选择/取消选择交易时更新

发生了什么：

1. 系统查找客户在日期范围内的所有未开票交易
2. 显示交易预览，能够包含/排除单个交易
3. 显示基于所选交易的实时小计、税和总计计算
4. 仅将所选（包含的）交易添加到发票中
5. 生成发票 PDF 并缓存
6. 将所选交易标记为已开票（`invoice_id` 字段填充）
7. 被排除的交易保持未开票状态，并可用于将来的发票
8. 发票在列表中以 "未支付" 状态出现

示例用例：

每月计费：将开始日期设置为每月的第一天，结束日期设置为每月的最后一天，预览显示该期间内所有未开票交易。选择所有或手动排除特定交易。

服务特定发票：使用相同的日期范围，然后手动排除不需要的交易（例如，排除非移动交易以创建仅移动的发票）。

一次性发票：将开始和结束日期设置为同一天，预览仅显示该日期的交易。排除与该特定发票无关的任何费用。

查看发票详情

点击表格中的任何发票行以查看完整的发票详情，包括所有交易、总计和可用操作。

发票详情模态框：

- 发票信息 - 显示发票 ID、标题、日期、支付状态和作废状态
- 交易列表 - 显示发票中包含的所有交易，包含：
  - 交易日期
  - 标题和描述
  - 零售成本
  - 税额和百分比（格式为 `$10.00 (10%)`）
  - 免税交易在税列中显示 "-"
- 总计摘要 - 实时计算显示：
  - 交易数量
  - 小计（所有零售成本的总和）
  - 税（所有税额的总和）
  - 发票总计（小计 + 税）
- 作按钮 - 与表格中可用的操作相同：
  - 下载 **PDF** - 下载发票 PDF（始终可用）
  - 发送电子邮件 - 将发票通过电子邮件发送给客户（未作废的发票）
  - 支付发票 - 处理支付（仅未支付、未作废的发票）
  - 退款 - 退款 Stripe 支付（仅已支付的 Stripe 发票）

- 删除 - 作废发票（仅未支付、未作废的发票）

下载发票 PDF

点击表格中的 下载图标（⬇）或发票详情模态框中的 "下载 PDF" 按钮以下载发票作为 PDF。

下载过程：

1. 点击发票旁边的下载图标
2. 生成期间显示加载旋转图标（仅首次）
3. 浏览器提示保存文件：`Invoice_01234.pdf`
4. PDF 打开或保存到下载文件夹

PDF 缓存行为：

- 首次下载 - 从模板生成 PDF，缓存到数据库中（可能需要 2-3 秒）
- 后续下载 - 从缓存立即下载
- 缓存失效 - 如果发票被修改或模板更新，则清除缓存

故障排除下载问题：

- 旋转图标永不停止 - 检查浏览器控制台，API 可能已关闭
- **PDF** 空白或损坏 - 检查发票模板□否存在语法错误
- 下载失败 - 检查弹出窗口阻止程序设置，尝试不同的浏览器

支付发票

点击 支付图标（□）在线支付发票。

支付过程：

1. 点击未支付发票上的支付图标
2. 支付模态框打开，显示发票详情
3. 选择支付方式：
    - **Stripe** 交易 - 收取保存的信用卡（所有用户可用）
    - 现金 - 手动现金支付（仅限工作人员）
    - 退款 - 将退款作为支付（仅限工作人员）
    - **POS** 交易 - 销售终端（仅限工作人员）
    - 银行转账 - 手动银行转账（仅限工作人员）
4. 如果选择了 Stripe：
    - 从保存的支付方式中选择卡
    - 默认卡预先选择
    - 点击选择不同的卡
5. 如果选择了其他方法：
    - 输入参考号码（可选）
6. 点击 "支付发票" 进行处理
7. 系统处理支付：
    - **Stripe** - 通过 Stripe API 收取卡费
    - 其他方法 - 为支付金额创建负交易
8. 发票状态更改为 "已支付"
9. 显示成功通知

自助服务与工作人员支付：

:doc:`自助服务门户 <self_care_portal>`（客户）：

- 仅提供 Stripe 支付
- 必须有保存的支付方式
- 如果没有支付方式，则显示警告
- 提供添加支付方式的链接

工作人员门户（管理员）：

- 所有支付方式可用
- 可以手动标记发票为已支付（现金、POS、银行转账）
- 可以输入参考号码以便跟踪

支付方式警告：

如果客户没有保存的支付方式，则会显示警告，提示他们在支付发票之前添加支付方式。

发送发票电子邮件

点击 电子邮件图标（✉）将发票发送给客户。

发生了什么：

1. 点击发票旁边的电子邮件图标

2. 系统从缓存中检索发票 PDF（如果未缓存则生成）
3. 通过 `Mailjet <integrations_mailjet>` 使用 `api_crmCommunicationCustomerInvoice` 模板发送电子邮件
4. 电子邮件包括：
    - 附件的发票 PDF
    - 客户姓名
    - 发票编号和到期日期
    - 应付总额
    - 在线支付发票的链接
    - 查看/下载发票的链接
5. 成功通知：“发票电子邮件成功发送”

电子邮件收件人：

电子邮件发送给所有类型为 "账单" 的客户联系人，或者如果没有账单联系人，则发送给主要联系人。

电子邮件模板变量：

- `{{ var:customer_name }}` - 客户的全名
- `{{ var:invoice_number }}` - 发票 ID
- `{{ var:invoice_date }}` - 发票发行日期
- `{{ var:due_date }}` - 支付到期日期
- `{{ var:total_amount }}` - 应付总额
- `{{ var:invoice_url }}` - 查看/下载 PDF 的链接
- `{{ var:pay_url }}` - 在线支付发票的链接

故障排除电子邮件问题：

- 电子邮件未发送 - 检查 `crm_config.yaml` 中的 Mailjet API 凭据
- 客户未收到 - 验证客户联系电子邮件地址
- **PDF** 未附加 - 检查 PDF 生成□否成功（先尝试下载）

作废发票

点击 删除图标（□）作废发票。

要求：

- 发票必须□ 未支付
- 已支付的发票不能作废（必须退款）

如何作废：

1. 在列表中找到未支付的发票
2. 点击删除图标 (□)
3. 在模态框中确认：

发生了什么：

- 发票标记为 `void = true`
- 所有交易与发票解除链接（`invoice_id` 设置为 null）
- 交易再次变为 "未开票"
- 交易可以包含在新发票中
- 发票在列表中以 "作废：" 前缀显示标题
- 发票操作禁用（无法下载、支付或发送电子邮件）
- 可以通过筛选 "作废" 发票进行查看

重要说明：

- 作废与退款不同
- 作废 = "该发票不应存在"（账单错误、重复）

- 退款 = "撤销有效的已支付发票"（将钱退还给客户）

退款发票

点击 退款图标 (□) 退款已支付的发票。

要求：

- 发票必须□ 已支付
- 发票必须通过 **Stripe** 支付
- 发票必须有有效的 `payment_reference`（Stripe 支付意图 ID）
- 仅限工作人员用户可用（不适用于自助服务）

如何退款：

1. 找到已支付的 Stripe 发票
2. 点击退款图标 (□)
3. 退款确认模态框打开：

4. 点击 "确认退款"
5. 系统处理 Stripe 退款：
   - 调用 Stripe API 进行退款
   - 在 Stripe 中创建退款交易
   - 更新发票以包含 `refund_reference`
6. 发票状态更改为 "已退款"

7. 显示成功通知

退款后的发生情况：

- 发票仍保留在系统中（未作废）
- 状态显示 "已退款"
- 交易仍与发票关联
- 客户收到退款到原支付方式（3-7 个工作日）
- Stripe 仪表板显示退款交易

退款限制：

- 不能退款通过现金、POS 或银行转账支付的发票（需要手动撤销）
- 不能部分退款（仅限全额发票金额）
- 不能退款两次

搜索和筛选发票

搜索

使用搜索栏查找发票。搜索内容包括：

- 发票 ID
- 发票标题
- 客户姓名（仅系统范围视图）

筛选

应用筛选以缩小发票列表：

可用筛选：

- 作废状态 - 所有、作废、未作废
- 支付状态 - 所有、已支付、尚未支付

筛选作：

- 应用筛选 - 将所选筛选应用于列表
- 重置筛选 - 清除所有筛选并显示所有发票

排序

点击任何列标题进行排序：

- **ID** - 按发票 ID 排序（最新/最旧）
- 标题 - 按字母顺序排序
- 到期日期 - 按到期日期排序
- 创建 - 按创建日期排序
- 金额 - 按零售成本排序（最高/最低）
- 状态 - 按支付状态排序（已支付优先或未支付优先）

再次点击以反转排序方向（升序 ↔ 降序）。

分页

通过页面控件在大型发票列表中导航，显示当前页面、总页面和每页项目选择器（10、25、50 或 100 项）。

常见发票工作流程

工作流程 1：带交易预览的每月计费

1. 月末到来（例如，1 月 31 日）
2. 导航到 账单 → 发票
3. 点击 "+ 生成形式发票"
4. 选择客户（或如果计费多个客户则按客户操作）
5. 设置日期：
   - 开始日期：2025-01-01
   - 结束日期：2025-01-31
   - 到期日期：2025-02-15（从现在起 15 天）
   - 标题：“2025 年 1 月服务”（可选）
6. 交易预览 部分显示 1 月份所有未开票交易

7. 审查预览：
    - 默认包含所有交易
    - 检查总计：小计、税和发票总计
    - 验证所有费用□否正确
8. 点击 "生成发票"（按钮显示交易数量，例如，“生成发票（15）”）
9. 发票创建，包含所有选定交易
10. 点击发票行查看详情并验证
11. 点击详情模态框中的 "发送电子邮件" 按钮或表格中的电子邮件图标
12. 客户收到带有 PDF 和支付链接的发票电子邮件

工作流程 2：选择性交易开票

1. 客户有多项服务（移动 + 网络）和杂项费用
2. 希望为每项服务开具单独的发票
3. 生成第一张发票（移动服���）：
    - 点击 "+ 生成形式发票"
    - 标题：“移动服务 - 2025 年 1 月”
    - 开始/结束：1 月 1 日至 31 日
    - 到期日期：2 月 15 日
    - 在交易预览中，排除所有非移动交易：
        - 点击互联网交易旁的 X 按钮
        - 点击杂项费用旁的 X 按钮
        - 仅移动服务交易保持选中
    - 验证总计仅反映移动服务
    - 点击 "生成发票"（显示移动交易的数量）
4. 生成第二张发票（互联网服务）：
    - 再次点击 "+ 生成形式发票"
    - 标题：“互联网服务 - 2025 年 1 月”
    - 开始/结束：1 月 1 日至 31 日（相同期间）
    - 在交易预览中：
        - 移动交易已开票（不再出现）
        - 使用 X 按钮排除杂项费用
        - 仅互联网服务交易保持
    - 点击 "生成发票"

5. 生成第三张发票（额外费用）：
    - 再次点击 "+ 生成形式发票"
    - 标题：“额外费用 - 2025 年 1 月”
    - 预览中仅显示未开票的杂项费用
    - 点击 "全选" 以包括所有
    - 点击 "生成发票"
6. 将所有三张发票通过电子邮件发送给客户

工作流程 3：排除争议或待处理交易

1. 计费周期结束
2. 导航到客户 ��单 选项卡
3. 点击 "+ 生成形式发票"
4. 设置计费周期日期
5. 交易预览显示 20 笔交易
6. 客户对一项费用提出争议，另一项正在调查中
7. 在交易预览中：
    - 找到争议交易（例如，“数据超额费用”）
    - 点击 X 按钮将其排除
    - 找到待处理交易（例如，“安装费用”）
    - 点击 X 按钮将其排除
    - 交易数量更新：“选择了 18 笔交易”
    - 总计自动重新计算
8. 审查更新后的总计（不包括争议金额）
9. 点击 "生成发票（**18**）"
10. 发票生成，仅包含已批准的交易
11. 争议/待处理交易保持未开票状态，待下一个计费周期处理

工作流程 4：快速发票审核和调整

1. 工作人员生成每月发票
2. 交易预览显示意外的高总额
3. 审查预览中的每笔交易：
    - 注意到同一服务的重复收费

- 点击 **X** 以排除重复项
- 注意到不应计费的测试交易
- 点击 **X** 以排除测试交易

4. 总计实时更新
5. 验证新总额与预期金额匹配
6. 点击 **"**生成发票**"**，包含已更正的交易
7. 如果需要，返回并作废/删除排除的交易
8. 将发票电子邮件发送给客户，充满信心

工作流程 **5**：一次性安装发票

1. 现场技术人员完成安装
2. 工作人员手动添加安装交易
3. 导航到客户 账单 选项卡
4. 点击 **"+** 生成形式发票**"**
5. 设置日期：
   - 开始日期：今天
   - 结束日期：今天
   - 到期日期：今天 + 7 天
   - 标题：“安装服务”
6. 交易预览仅显示今天的交易
7. 验证安装费用出现
8. 使用 **X** 按钮排除任何重复费用（如果存在）
9. 点击 **"**生成发票**"**
10. 立即将发票通过电子邮件发送给客户
11. 客户通过 Stripe 在线支付

工作流程 **6**：在客户联系之前审核发票

1. 客户打电话询问账单问题
2. 工作人员导航到客户的发票列表
3. 点击发票行 打开发票详情模态框
4. 审查发票信息：
   - 发票 ID、日期、状态

    - 包含的所有交易及描述
    - 每笔交易的税务明细
    - 小计、税和总金额
5. 用确切的细节回答客户的问题
6. 如果客户请求 PDF，点击模态框中的 "下载 **PDF**" 按钮
7. 如果客户请求重新发送电子邮件，点击 "发送电子邮件" 按钮
8. 完成后关闭模态框

工作流程 **7**：纠正账单错误

1. 客户报告错误收费
2. 工作人员点击发票行查看详情
3. 在发票详情模态框中审查交易列表
4. 确定错误交易
5. 发票未支付，因此可以作废
6. 点击模态框底部的 "删除" 按钮
7. 确认作废
8. 交易再次变为未开票状态
9. 工作人员修改或删除交易列表中的错误交易
10. 生成包含已更正交易的新发票：
    - 使用交易预览在需要时排除已更正的交易
    - 仅包含有效费用
11. 将更正后的发票通过电子邮件发送给客户

工作流程 **8**：处理多笔支付

1. 客户带现金支付多张发票
2. 导航到客户 账单 选项卡
3. 查看未支付的发票
4. 点击第一张发票行查看详情
5. 验证金额和交易
6. 点击模态框底部的 "支付发票" 按钮
7. 选择 "现金" 支付方式
8. 输入参考："现金支付 2025-01-15"

9. 点击 "支付发票"
10. 模态框关闭，发票标记为 "已支付"
11. 对剩余发票重复此过程
12. 所有发票现在标记为 "已支付"

工作流程 9：处理退款请求

1. 客户请求退款以补偿多付
2. 工作人员验证发票通过 Stripe 支付
3. 导航到发票列表
4. 点击发票行查看详情
5. 验证支付信息和金额
6. 点击模态框底部的 "退款" 按钮（仅对 Stripe 发票可见）
7. 确认退款
8. 系统处理 Stripe 退款
9. 发票状态更改为 "已退款"
10. 客户在 3-7 个工作日内收到退款
11. 工作人员跟进客户以确认收到

故障排除

无法生成发票 - 找不到交易

- 原因：指定日期范围内没有未开票交易
- 修复：检查交易列表，验证交易存在且未开票。调整日期范围或移除筛选。

发票 PDF 生成失败

- 原因：模板语法错误、WeasyPrint 崩溃或缺少客户数据
- 修复：检查发票模板 HTML 是否存在错误，验证客户地址字段是否填充，查看 API 日志。

支付失败，出现 Stripe 错误

- 原因：卡被拒绝、资金不足、卡过期或 Stripe API 问题
- 修复：尝试不同的支付方式，验证卡是否有效，检查 Stripe 仪表板以获取拒绝原因。

无法作废发票

- 原因：发票已支付
- 修复：已支付的发票不能作废。如果需要退款，请使用 Stripe 发票的退款功能或手动创建信用交易。

发票电子邮件未发送

- 原因： Mailjet API 凭据无效、客户没有账单联系人或电子邮件模板缺失
- 修复：验证 `crm_config.yaml` 中的 Mailjet 配置，检查客户联系人，确保发票电子邮件模板存在。

退款按钮未出现

- 原因：发票通过现金/POS/银行转账支付（不 Stripe），或发票未支付
- 修复：退款按钮仅在 Stripe 支付时出现。对于其他支付方式，请手动创建信用交易。

下载 **PDF** 显示旧模板设计

- 原因： PDF 在模板更新之前缓存
- 修复：清除发票 PDF 缓存： `DELETE FROM Invoice_PDF_Cache WHERE invoice_id = X;`

客户无法支付发票（没有支付方式）

- 原因：自助服务门户中没有保存的支付方式
- 修复：客户必须在 支付方式 页面添加信用卡，然后才能支付发票。

为同一期间生成多张发票

- 原因：工作人员生成发票两次，或日期范围重叠
- 修复：作废重复发票。调整日期范围以防止重叠。使用交易预览确保唯一的交易集。

交易预览显示没有交易

- 原因：日期范围内的所有交易均已开票或没有交易存在
- 修复：验证日期范围否正确。检查交易列表以确认交易存在。筛选发票以查看包含交易的发票。

无法从发票生成中排除交易

- 原因：交易已开票或浏览器问题
- 修复：验证交易在预览中否显示并带有复选标记。刷新页面并重试。如果问题仍然存在，请清除浏览器缓存。

发票总额与预期金额不匹配

- 原因：包含意外交易、税未计算或排除的交易仍被计算
- 修复：仔细审查交易预览。检查每笔交易的零售成本和税。验证排除的交易否显示为灰色。检查生成发票按钮上的交易计数徽章。

生成发票按钮被禁用

- 原因：没有选择交易或日期范围无效
- 修复：确保至少包含一笔交易（未排除）。验证开始日期在结束日期之前。检查到期日期否设置。

发票详情模态框未打开

- 原因：JavaScript 错误或页面未完全加载
- 修复：刷新页面。检查浏览器控制台否有错误。尝试不同的浏览器。验证互联网连接。

交易税未在发票详情中显示

- 原因：交易税率为 0% 或 tax_amount 为 null
- 修复：验证交易否设置了 tax_percentage。检查交易创建时否计算了 tax_amount。如有需要，更新交易。

发票详情模态框中缺少作按钮

- 原因：发票已作废或用户权限不足
- 修复：作废的发票仅显示下载 PDF 按钮。验证发票状态。检查用户角色和权限。

相关文档

- `integrations_mailjet` - 电子邮件发票交付和模板
- `administration_configuration` - 发票模板配置
- `payments_transaction` - 创建出现在发票的交易
- `payments_process` - 处理发票支付
- `basics_payment` - 支付方式管理
- `payment_system_guide` - 支付 API 参考和供应商配置

□□□□

□□□

客户交易

系统中任何花费金钱的事项都被记录为客户的交易。

每个交易都有批发成本和零售成本的货币金额，以及交易的描述。

交易可以由系统自动生成，例如，当服务被提供时，会为设置费用创建一个交易；当服务被计费时，会为零售成本创建一个交易。

交易也可以手动创建，例如，如果客户获得了信用，则会为信用金额创建一个交易；或者如果收取了安装费用，则会为安装费用创建一个交易。

交易被组合在一起形成发票 `<payments_invoices>`，并发送给客户进行支付。

访问交易

交易可以在系统级别或按客户查看：

按客户查看：

1. 导航到 客户 → 【选择客户】
2. 点击 计费 标签
3. 在第一个卡片中查看交易列表

系统范围查看：

1. 导航到 计费 → 交易（从主菜单）
2. 查看所有客户的所有交易

交易统计小部件

在交易页面顶部，四个统计卡片显示财务摘要：

小部件描述：

- 总交易 - 所有交易零售成本的总和（所有时间）
- 总未开票交易 - 尚未包含在发票中的交易总和
- 本月总交易 - 本日历月内创建的交易总和
- 上月总交易 - 上日历月内创建的交易总和

值格式化：

- 超过1,000的值：显示为“k”后缀（例如，$1.5k）
- 超过1,000,000的值：显示为“M”后缀（例如，$2.3M）
- 超过1,000,000,000的值：显示为“B”后缀（例如，$1.1B）

交易列表

交易表显示所有交易，包含以下列：

列描述：

- **ID** - 唯一交易ID
- 日期 - 交易创建日期
- 标题 - 短交易名称
- 描述 - 交易用途的详细描述
- 金额 - 零售成本（收费为正，信用为负）
- 发票 - 如果交易已开票，则为发票ID（可点击链接）
- 状态 - 如果已开票则为勾选标记，尚未开票则为破折号

每行□作：

每行都有一个操作菜单（⋮），选项包括：

- 查看详情 - 打开交易详情模态框
- 下载发票**PDF** - 下载PDF（仅在已开票时）
- 作废交易 - 将交易标记为作废（仅在未开票时）

交易类型

交��分为两大类：

借记交易（收费）

正金额，增加客户应付余额：

- 服务设置费用 - 服务提供时的一次性收费
- 月度服务费用 - 服务的定期收费
- 安装费用 - 现场技术人员访问的收费
- 设备费用 - 调制解调器、路由器、SIM卡的收费
- 逾期付款费用 - 对逾期发票的罚款
- 手动收费 - 员工添加的自定义收费

信用交易（支付/退款）

负金额，减少客户应付余额：

- 现金支付 - 客户以现金支付
- 卡支付 - 客户以信用卡/借记卡支付
- 银行转账支付 - 客户通过银行转账支付
- 账户信用 - 善意信用，赔偿
- 退款 - 退还给客户的钱
- 折扣 - 促销或忠诚折扣

手动添加交易

点击 "+ 添加交易" 打开添加交易模态框。

借记交易（收费）：

信用交易（支付/退款）：

字段描述：

- 交易类型 - 选择借记（收费）或信用（支付/退款）
- 信用类型 - 如果选择了信用，选择支付方式（现金、卡、银行转账）
- 标题 - 交易的短名称（必填）
- 描述 - 详细说明（可选）
- 零售成本 - 客户支付的金额（必填，正数）
- 批发成本 - 你的成本（可选，用于利润跟踪）
- 税率 - 适用于此交易的税率（可选，默认为产品税或0%）
- 服务 - 将交易链接到特定服务（可选）
- 站点 - 将交易链接到特定站点（可选）
- 交易日期 - 交易日期（默认为今天）

验证：

- 标题和零售成本为必填项
- 零售成本必须为正数
- 如果选择了信用类型，必须选择一个信用类型

发生的事情：

1. 交易在数据库中创建
2. 出现在客户的交易列表中

3. 包含在“未开票交易”计数中
4. 可用于下次发票生成
5. 创建活动日志条目

搜索和过滤交易

搜索

使用搜索框查找交易。搜索内容包括：

- 交易ID
- 标题
- 描述
- 发票ID

过滤器

应用过滤器以缩小交易列表：

可用过滤器：

- 作废状态 - 所有、作废、未作废
- 发票状态 - 所有、已开票、未开票

过滤器作：

- 应用过滤器 - 将所选过滤器应用于列表
- 重置过滤器 - 清除所有过滤器并显示所有交易

排序

点击任何列标题进行排序：

- **ID** - 按交易ID排序（最新/旧）
- 日期 - 按交易日期排序
- 标题 - 按字母顺序排序

- 金额 - 按零售成本排序（最高/最低）
- 发票 - 按发票ID排序

再次点击以反转排序方向（升序 ↔ 降序）。

作废交易

错误添加的交易可以被 作废（标记为删除）。

要求：

- 交易必须未开票
- 一旦开票，交易不能被作废（必须退款）

如何作废：

1. 在列表中找到交易
2. 点击操作菜单（⋮）
3. 选择 "作废交易"
4. 在模态框中确认

发生的事情：

- 交易标记为 `void = true`
- 不再出现在默认交易列表中

- 从发票生成中排除
- 可以通过过滤“作废”交易查看
- 从“未开票交易”总数中扣除

注意：作废与退款不同。作废意味着“此交易本不应存在。”退款意味着“撤销有效交易。”

交易的税收

交易可以包括税收，税收□根据产品的税收配置自动计算的，或根据每个交易手动指定。

税收行为：

- 借记交易（收费） - 税收适用于收费，基于：
  - 产品税率 - 如果交易链接到产品，则自动应用该产品的税率
  - 手动覆盖 - 员工在创建交易时可以覆盖税率
  - 税额 - 计算为：`retail_cost × (tax_percentage / 100)`
  - 显示格式 - 在交易列表中显示为：$10.00 (10%)
- 信用交易（支付/退款） - 不对信用应用税收
  - 信用交易的税率字段被隐藏
  - 所有支付和退款的税收自动设为0%
  - 信用在没有税收影响的情况下减少客户的未付款余额

税收计算示例：

- 产品：移动计划，税率10%，零售成本$50.00
- 自动税收计算：$50.00 × 0.10 = $5.00
- 显示：$5.00 (10%)

零税（**NIL/**免税）：

- 产品可以通过将税率设置为0来免税
- 如果未指定，税收默认为0%
- 免税交易在税收列中显示为“-”

交易详情视图

点击交易以查看完整详情：

已开票与未开票交易

未开票交易：

- 尚未包含在任何发票中
- 可用于下次发票生成
- 可以作废
- 计入“未开票交易”总数

- 状态显示破折号 (-)

已开票交易：

- 包含在发票中
- 不能作废（如有需要必须退款）
- 发票ID可点击（链接到发票详情）
- 状态显示勾选标记 (✓)
- 不能修改

发票生成：

当您为客户生成发票时：

1. 系统查找该客户的所有未开票交易
2. 可选择按日期范围过滤
3. 交易包含在新发票中
4. 交易的 `invoice_id` 字段被填充
5. 交易现在标记为“已开票”

请参见 `payments_invoices` 以获取发票生成的详细信息。

常见工作流程

工作流程 1：因服务中断手动信用

1. 客户来电：“服务中断了2天”
2. 员工决定给予£10的信用
3. 导航到客户 计费 标签
4. 点击 "+ 添加交易"
5. 选择 信用 交易类型
6. 选择 现金支付 信用类型
7. 输入标题：“服务中断信用”
8. 输入描述：“因1月8-9日的中断赔偿”
9. 输入零售成本：10.00

10. 从下拉列表中选择受影响的服务
11. 点击 "添加交易"
12. 交易以-£10.00的金额出现
13. 将作为信用包含在下次发票中

工作流程 2：手动安装费用

1. 现场技术人员安装服务
2. 员工需要收取£75的安装费用
3. 导航到客户 计费 标签
4. 点击 "+ 添加交易"
5. 选择 借记 交易类型
6. 输入标题：“安装费用”
7. 输入描述：“现场技术人员进行光纤安装”
8. 输入零售成本：75.00
9. 输入批发成本：45.00（可选，用于利润跟踪）
10. 选择已安装的服务
11. 选择安装地点
12. 点击 "添加交易"
13. 交易出现在未开票列表中
14. 将包含在下次发票中

工作流程 3：作废重复交易

1. 员工注意到重复交易
2. 验证交易尚未开票
3. 点击重复交易上的操作菜单（⋮）
4. 选择 "作废交易"
5. 在模态框中确认
6. 交易从列表中移除
7. 未开票总数相应减少

工作流程 4：查找发票的交易

1. 需要生成月度发票
2. 点击 发票过滤器：“未开票”
3. 点击 应用过滤器
4. 查看所有未开票交易
5. 从小部件中记录总金额
6. 导航到生成发票
7. 选择日期范围（例如，1-31日）
8. 范围内的交易包含在发票中

故障排除

无法作废交易

- 原因：交易已开票
- 解决：交易□发票历史的一部分。如果需要退款，请创建信用交易。

重复交易出现

- 原因：服务多次收费或提供错误
- 解决：如果未开票，请作废重复交易。如果已开票，请发放信用。

交易未出现在列表中

- 原因：应用了过滤器或交易已作废
- 解决：点击“重置过滤器”以显示所有交易。要查看作废交易，请按“作废：作废”过滤。

未开票总数与预期不符

- 原因：一些交易已开票，或作废交易被排除
- 解决：应用过滤器“发票：未开票”以仅查看未开票。单独检查作废交易。

无法添加交易（客户字段被禁用）

- 原因：正在查看客户特定的交易页面
- 解决：客户已预选。如果需要为不同客户添加交易，请转到系统范围的交易页面。

相关文档

- `payments_invoices` - 发票生成和管理
- `payments_process` - 处理发票的支付
- `basics_payment` - 支付方式概述
- `csa_activity_log` - 在活动日志中查看交易历史

从 Playbooks 进行收费和支付

本指南解释了如何在 Ansible playbooks 中实现收费和支付处理，以便于 OmniCRM 供应工作流。

> 完整 **API** 参考：有关支付 API、钱包路由、退款和供应商无关架构的完整详细信息，请参见 支付系统 **API** 指南

概述

OmniCRM Ansible Playbooks 可以以多种不同方式处理支付，但所有这些方式都只调用 CRM 的 API 来收费。

1. 两阶段提交支付流程 - 对于需要立即支付授权和捕获的付费服务（即预付费服务）
2. 直接交易创建 - 对于添加费用/信用而不需要立即支付处理（例如，设置费用、手动信用），这些费用可以在后续处理（即后付费服务）

这些方法确保原子交易，只有在供应成功时才会向客户收费，如果任何步骤失败，则会自动回滚。

定价灵活性：将 Playbooks 作为脚本引擎

重要：在产品和服务中定义的美元价值（例如，`retail_cost`、`wholesale_cost`、`retail_setup_cost`）仅仅存储在数据库中 默认值。它们并不决定您必须收费的金额——playbook 完全控制最终定价。

Ansible playbook 本质上一个 脚本引擎，您可以根据任何业务逻辑需求进行定制。您可以：

- 按原样使用存储的价格 - 在您的授权中简单引用 `api_response_product.json.retail_cost`
- 完全覆盖价格 - 无论产品定义中什么，都收取不同的金额
- 应用动态折扣 - 根据客户等级、促销或忠诚度计算折扣百分比
- 实施分层定价 - 根据数量、使用水平或合同条款收取不同的费率
- 捆绑定价 - 将多个产品与自定义捆绑折扣组合
- 基于时间的定价 - 根据一天中的时间、季节或促销期调整价格
- 客户特定定价 - 从客户合同中查找谈判的费率

示例：覆盖产品价格

```
# 产品 retail_cost 为 $99.00，但我们想以 $79.00 收费以进行促销
- name: 设置促销价格（忽略产品的 retail_cost）
  set_fact:
    charge_amount: 79.00

- name: 授权促销支付
  uri:
    url: "http://localhost:5000/crm/payments/authorize/hold"
    method: POST
    body_format: json
    body:
      customer_id: "{{ customer_id | int }}"
      amount: "{{ charge_amount | float }}"  # 使用我们的覆盖，而不是
retail_cost
      # ...
```

示例：客户等级折扣

```
# 根据客户等级应用折扣
- name: 获取客户等级
  uri:
    url: "http://localhost:5000/crm/customer/{{ customer_id }}"
    # ...
  register: customer_info

- name: 计算等级折扣
  set_fact:
    base_price: "{{ api_response_product.json.retail_cost | float
}}"
    discount_percent: >-
      {% if customer_info.json.tier == 'gold' %}20
      {% elif customer_info.json.tier == 'silver' %}10
      {% else %}0{% endif %}

- name: 将折扣应用于最终价格
  set_fact:
    final_price: "{{ (base_price | float) * (1 - (discount_percent
| float / 100)) | round(2) }}"
```

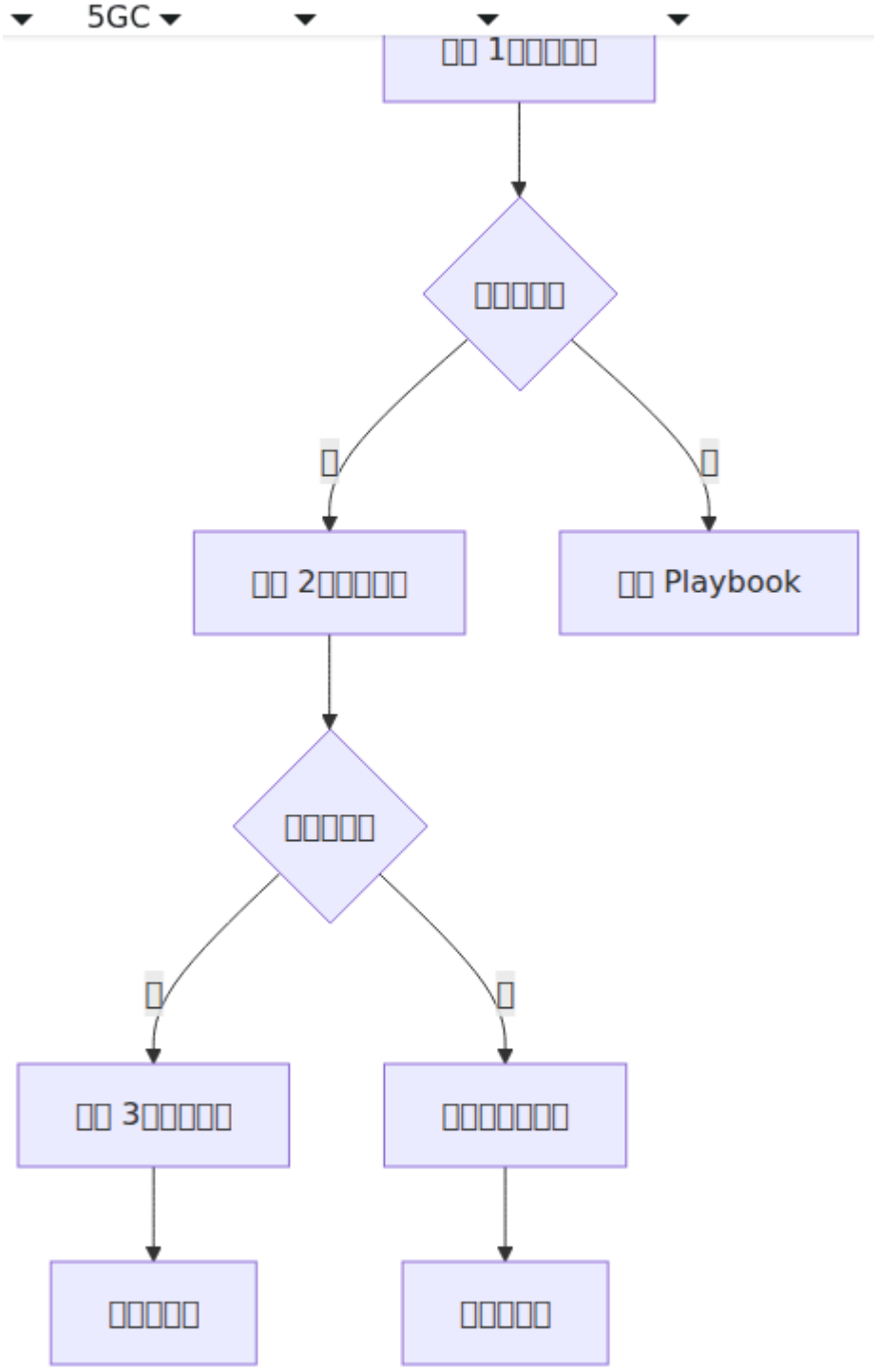
示例：批发合作伙伴定价

```
# 批发合作伙伴支付 wholesale_cost 而不是 retail_cost
- name: 根据□户类型确定价格
  set_fact:
    charge_amount: >-
      {% if customer_info.json.is_wholesale_partner %}
      {{ api_response_product.json.wholesale_cost | float }}
      {% else %}
      {{ api_response_product.json.retail_cost | float }}
      {% endif %}
```

关键点□ 您并不受限于任何定价模型。playbook 赋予您完全的编程控制，以实施您的组织所需的任何业务规则。CRM 中的产品/服务价格只□方便的默认值，playbooks 可以根据需要使用或忽略这些值。

两阶段提交支���流程

当您需要在供应过程中向客户的支付方式收费时，使用两阶段提交模式。这确保只有在供应成功完成时，客户才会被收费。



流程概述

实现模式

该模式遵循以下阶段：

阶段 **1**：授权 - 在支付方式上保留资金

阶段 **2**：供应 - 执行 OCS 余额/操作更新

阶段 **3**：捕获 **-** 在成功供应后完成支付

回滚 **-** 如果供应失败则释放授权

完整示例

以下来自 `play_topup_charge_then_action.yaml` 的完整示例：

```yaml
- name: Play Topup - Charge card then action the Topup
  hosts: localhost
  gather_facts: no
  become: False

  tasks:
    - name: 包含 crm_config 的变量
      ansible.builtin.include_vars:
        file: "../../crm_config.yaml"
        name: crm_config

    # 取产品和服务信息
    - name: 从 CRM API 获取产品信息
      uri:
        url: "http://localhost:5000/crm/product/product_id//{{
product_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
        validate_certs: no
      register: api_response_product

    - name: 从 CRM API 获取服务信息
      uri:
        url: "http://localhost:5000/crm/service/service_id/{{
service_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
        validate_certs: no
      register: api_response_service

    - name: 设置服务和套餐事实
      set_fact:
        service_uuid: "{{ api_response_service.json.service_uuid
}}"
        customer_id: "{{ api_response_service.json.customer_id }}"
        package_name: "{{ api_response_product.json.product_name
}}"
        monthly_cost: "{{ api_response_product.json.retail_cost
}}"
```

```
        wholesale_cost: "{{
api_response_product.json.wholesale_cost }}"

    # □取客户的默认支付方式
    - name: 从支付控制器获取□户支付方式
      uri:
        url: "http://localhost:5000/crm/payments/methods?
customer_id={{ customer_id }}"
        method: GET
        headers:
          Authorization: "Bearer {{ access_token }}"
        return_content: yes
        validate_certs: no
      register: api_response_payment_methods

    - name: 从响应中获取默认 payment_method_id
      set_fact:
        payment_method_id: "{{ api_response_payment_methods.json |
json_query(query) }}"
      vars:
        query: "data[?is_default==`true`].payment_method_id | [0]"

    # ============================================================
    # 两阶段提交支付流程
    # ============================================================

    - name: "阶段 1: 授权支付（保留资金）"
      uri:
        url: "http://localhost:5000/crm/payments/authorize/hold"
        method: POST
        headers:
          Content-Type: "application/json"
          Authorization: "Bearer {{ access_token }}"
        body_format: json
        body:
          {
            "customer_id": "{{ customer_id | int }}",
            "amount": "{{ monthly_cost | float }}",
            "currency": "{{ crm_config.currency | default('AUD')
}}",
            "payment_method_id": "{{ payment_method_id | int }}",
            "metadata": {
              "description": "{{ package_name }} on {{
api_response_service.json.service_name }}",
```

```
              "service_id": "{{
api_response_service.json.service_id | int }}",
              "site_id": "{{ api_response_service.json.site_id |
int }}",
              "product_id": "{{ product_id }}",
              "user_id": "{{ (initiating_user | int) if
(initiating_user is defined and initiating_user is not none) else
omit }}",
              "title": "{{ package_name }}",
              "wholesale_cost": "{{ wholesale_cost | float }}",
              "invoice": true,
              "contract_days": "{{ days | int }}",
              "send_email": true
            }
          }
        return_content: yes
      register: api_response_authorization

    - name: 确保授权成功
      assert:
        that:
          - api_response_authorization.status == 200
          - api_response_authorization.json.success == true

    - name: 存储 authorization_id 以供捕获/释放
      set_fact:
        authorization_id: "{{
api_response_authorization.json.data.authorization_id }}"

    # 阶段 2: OCS 供应（包装在块中以便于事务回滚）
    - block:
        - name: "阶段 2: 执行 CGRateS 操作"
          uri:
            url: "http://{{ crm_config.ocs.cgrates }}/jsonrpc"
            method: POST
            body_format: json
            headers:
              Content-Type: "application/json"
              Authorization: "Bearer {{ access_token }}"
            body:
              {
                "id": "{{ 999999999 | random }}",
                "method": "APIerSv1.ExecuteAction",
                "params": [{
```

```
                  "Tenant": "{{ crm_config.ocs.ocsTenant }}",
                  "Account": "{{ service_uuid }}",
                  "ActionsId": "{{ cgr_action_name }}"
                }]
              }
            status_code: 200
          register: action_execute_response

        - name: 确保操作的响应是 OK
          assert:
            that:
              - action_execute_response.status == 200
              - action_execute_response.json.result == "OK"

        # 阶段 3: 支付捕 - 在成功供应后完成交易
        - name: "阶段 3: 捕获授权支付"
          uri:
            url: "http://localhost:5000/crm/payments/capture/{{
authorization_id }}"
            method: POST
            headers:
              Content-Type: "application/json"
              Authorization: "Bearer {{ access_token }}"
            body_format: json
            body:
              {
                "metadata": {
                  "provisioning_status": "success",
                  "cgr_action": "{{ cgr_action_name }}"
                }
              }
            return_content: yes
          register: api_response_capture

        - name: 确保捕获成功
          assert:
            that:
              - api_response_capture.status == 200
              - api_response_capture.json.success == true

      rescue:
        # 事务回滚: 作废授权以释放保留的资金
        - name: "回滚: 释放支付授权"
          uri:
```

```
            url: "http://localhost:5000/crm/payments/release/{{
authorization_id }}"
            method: POST
            headers:
              Content-Type: "application/json"
              Authorization: "Bearer {{ access_token }}"
            body_format: json
            body:
              {
                "metadata": {
                  "release_reason": "provisioning_failed",
                  "cgr_action": "{{ cgr_action_name }}"
                }
              }
            return_content: yes
          register: api_response_release

        - name: 在回滚后终止 playbook 执行
          fail:
            msg: "OCS 供应失败。支付授权 {{ authorization_id }} 已作废。□户未收费。"
```

支付 API 端点

> □ 钱包优先路由：支付系统通过优先使用钱包余额自动优化卡收费。如果客户钱包中有 $1 并且购买了 $10 的附加服务，则仅向其卡收费 $9。有关详细信息，请参见 钱包优先路由部分。

1. 授权/保留支付

端点： `POST /crm/payments/authorize/hold`

此端点对客户的支付方式进行指定金额的保留。资金被保留但尚未捕获。

钱包优先行为：授权在检查钱包余额后自动计算短缺。如果钱包余额为 $150，而授权为 $500，则仅在卡上授权 $350。钱包在捕获时扣款。

请求体：

```
{
  "customer_id": 123,
  "amount": 49.99,
  "currency": "AUD",
  "payment_method_id": 456,
  "metadata": {
    "description": "套餐名称和描述",
    "service_id": 789,
    "site_id": 12,
    "product_id": "34",
    "user_id": 5,
    "title": "套餐名称",
    "wholesale_cost": 25.00,
    "invoice": true,
    "contract_days": 30,
    "send_email": true
  }
}
```

字段描述：

- **customer_id**（必需） - 被收费的客户 ID
- **amount**（必需） - 要授权的金额，采用小数格式
- **currency**（可选） - 货币代码（默认为系统默认值）
- **payment_method_id**（必需） - 要收费的支付方式
- **metadata**（可选） - 交易创建的附加数据：
  - **description** - 交易描述
  - **service_id** - 被收费的服务
  - **site_id** - 与服务相关联的网站
  - **product_id** - 正在供应的产品
  - **user_id** - 发起收费的用户（可选）
  - **title** - 交易标题
  - **wholesale_cost** - 您的成本（用于利润跟踪）
  - **invoice** - 如果为 `true`，在捕获时自动创建交易
  - **contract_days** - 合同期限
  - **send_email** - 如果为 `true`，发送电子邮件通知

响应：

```
{
  "success": true,
  "message": "支付已授权（创建了保留）",
  "data": {
    "authorization_id": 301,
    "vendor_authorization_id": "auth_xxxxx",
    "amount": 500.00,
    "currency": "USD",
    "status": "authorized",
    "wallet_balance": 150.00,
    "wallet_to_use": 150.00,
    "card_amount": 350.00,
    "message": "卡已授权 $350（钱包充值）。在捕获时将发生 $500 的钱包扣款。"
  }
}
```

重要：

- 保存 `authorization_id` 以便在捕获或释放调用中使用
- 注意 `card_amount` 仅显示短缺已被授权
- 钱包余额已检查，但在捕获之前不会扣款

2. 捕获支付

端点：`POST /crm/payments/capture/{authorization_id}`

此端点完成支付授权并向客户收费。如果在授权元数据中设置了 `invoice: true`，则会自动创建交易。

请求体：

```
{
  "metadata": {
    "provisioning_status": "success",
    "cgr_action": "Action_Topup_Standard",
    "additional_info": "任何其他相关数据"
  }
}
```

响应：

```
{
  "success": true,
  "data": {
    "payment_id": "pay_xyz789",
    "transaction_id": 1234
  }
}
```

发生的事情：

1. 卡捕获短缺金额（如果卡已被授权）
2. **钱包记���**捕获的卡金额
3. 钱包扣款全额服务金额
4. 如果在授权元数据中为 `invoice: true`：
   - 借记交易创建（正金额 = 收费）
   - 发票创建并链接到借记交易
   - 贷记交易创建（负金额 = 收到的付款）
   - 发票标记为已支付（借记和贷记净额为零）
   - 交易出现在客户账单中
5. 数据库中创建支付记录
6. 如果 `send_email: true`，客户将收到发票电子邮件

示例：$500 授权与 $150 钱包余额：

- 卡捕获：$350
- 钱包记入：+$350（钱包现在为 $500）
- 钱包扣款：-$500（服务费用）
- 最终钱包：$0

有关详细流程，请参见 钱包优先路由示例。

3. 释放支付授权

端点：`POST /crm/payments/release/{authorization_id}`

此端点取消支付授权并释放保留的资金。在供应失败时使用此方法进行恢复/回滚场景。

请求体：

```
{
  "metadata": {
    "release_reason": "provisioning_failed",
    "error_details": "OCS 账户创建失败"
  }
}
```

响应：

```
{
  "success": true,
  "message": "授权已释放"
}
```

发生的事情：

1. 卡授权已释放（如果卡已被授权）
2. ���留的资金返回到客户的可用信用中
3. 钱包未扣款（因为扣款仅在捕获时发生）
4. 客户未收费
5. 不会创建任何交易

注意：使用钱包优先路由时，无需退款，因为钱包在捕获时才会扣款。

直接交易创建

对于不需要支付处理的费用（设置费用、手动信用、调整），您可以通过 API 直接创建交易。

通过 **API** 添加交易

端点：`PUT /crm/transaction/`

来自 **`play_simple_service.yaml`** 的示例：

```
- name: 通过 API 添加设置费用交易
  uri:
    url: "http://localhost:5000/crm/transaction/"
    method: PUT
    headers:
      Content-Type: "application/json"
      Authorization: "Bearer {{ access_token }}"
    body_format: json
    body:
      {
        "customer_id": {{ customer_id | int }},
        "service_id": {{ service_creation_response.json.service_id
| int }},
        "title": "{{ package_name }} - 设置费用",
        "description": "为 {{ package_comment }} 的设置费用",
        "invoice_id": null,
        "wholesale_cost": {{
api_response_product.json.wholesale_setup_cost | float }},
        "retail_cost": "{{
api_response_product.json.retail_setup_cost | float }}"
      }
    return_content: yes
  register: api_response_transaction

- name: 确保交易响应正常
  assert:
    that:
      - api_response_transaction.status == 200
```

请求体字段：

- **customer_id** (必需) - 客户 ID
- **service_id** (可选) - 用于链接交易的服务 ID
- **title** (必需) - 短交易名称
- **description** (可选) - 详细描述
- **invoice_id** (可选) - 如果已开票，则为发票 ID (通常为 null)
- **wholesale_cost** (可选) - 您的成本
- **retail_cost** (必需) - 面向客户的成本
- **site_id** (可选) - 用于链接交易的网站 ID

- **tax_percentage**（可选） - 税率百分比

用例：

- 服务供应期间的设置费用
- 安装费用
- 手动信用或调整
- 设备费用
- 行政费用

注意：直接交易创建不会处理支付 - 它仅创建账单记录。该交易将显示为未开票，并可以包含在未来的发票中。

计算按比例收费

按比例收费允许您根据部分计费周期按比例向客户收费。这在以下情况下很常见：

- 客户在中旬注册
- 服务在周期中升级/降级
- 根据使用天数计算账单

按比例计算公式

```
pro_rata_charge = (monthly_cost × days_remaining) / days_in_month
```

实现示例

以下在 playbook 中计算按比例收费的方法：

```yaml
# 计算??分月份的按比例收费
# 如果客户在 15 号注册，而账单在 1 号，
# 按剩余天数按比例收费

- name: 获取当前月份的天数
  command: "date +%d"
  register: current_day

- name: 获取当前月份的总天数
  command: "date -d 'last day of this month' +%d"
  register: days_in_month

- name: 获取月份的最后一天
  command: "date -d 'last day of this month' +%Y-%m-%d"
  register: last_day_of_month

- name: 计算本月剩余天数
  set_fact:
    days_remaining: "{{ (days_in_month.stdout | int) -
(current_day.stdout | int) + 1 }}"

- name: 计算按比例费用
  set_fact:
    pro_rata_cost: "{{ ((monthly_cost | float) * (days_remaining |
float) / (days_in_month.stdout | float)) | round(2) }}"

- name: 显示计算细节
  debug:
    msg:
      - "每月费用: ${{ monthly_cost }}"
      - "本月天数: {{ days_in_month.stdout }}"
      - "剩余天数: {{ days_remaining }}"
      - "按比例收费: ${{ pro_rata_cost }}"

# 在授权或交易创建中使用 pro_rata_cost
- name: "授权按比例支付"
  uri:
    url: "http://localhost:5000/crm/payments/authorize/hold"
    method: POST
    headers:
      Content-Type: "application/json"
      Authorization: "Bearer {{ access_token }}"
    body_format: json
```

```
    body:
      {
        "customer_id": "{{ customer_id | int }}",
        "amount": "{{ pro_rata_cost | float }}",
        "currency": "{{ crm_config.currency | default('AUD') }}",
        "payment_method_id": "{{ payment_method_id | int }}",
        "metadata": {
          "title": "{{ package_name }} (按比例 {{ days_remaining }}
天)",
          "description": "为 {{ days_remaining }}/{{
days_in_month.stdout }} 天的 {{ package_name }} 收取按比例费用",
          "service_id": "{{ service_id | int }}",
          "invoice": true
        }
      }
```

从自定义开始日期计算按比例

如果您需要从特定开始日期到账单日期计算按比例：

```yaml
- name: 设置自定义开始日期和账单日期
  set_fact:
    service_start_date: "2024-01-15"
    next_billing_date: "2024-02-01"

- name: 计算日期之间的天数
  shell: |
    echo $(( ( $(date -d "{{ next_billing_date }}" +%s) - $(date -d "{{ service_start_date }}" +%s) ) / 86400 ))
  register: days_until_billing

- name: 获取计费周期的天数（通常为 30）
  set_fact:
    billing_period_days: 30

- name: 计算按比例费用
  set_fact:
    pro_rata_cost: "{{ ((monthly_cost | float) * (days_until_billing.stdout | float) / (billing_period_days | float)) | round(2) }}"

- name: 显示计算
  debug:
    msg:
      - "开始日期: {{ service_start_date }}"
      - "下一个账单: {{ next_billing_date }}"
      - "距离账单的天数: {{ days_until_billing.stdout }}"
      - "按比例收费: ${{ pro_rata_cost }}"
```

按比例示例场景

场景 **1**：中旬注册

- 客户在 1 月 15 日注册
- 每月费用：$60.00
- 1 月的天数：31
- 剩余天数：17（包括 15 日到 31 日）
- 按比例收费：`$60.00 × 17 ÷ 31 = $32.90`

场景 **2**：服务升级

- 客户在计费周期的第 10 天升级
- 旧计划：$30/月
- 新计划：$50/月
- 周期中的天数：30
- 剩余天数：21
- 差额：$20/月
- 按比例收费：`$20.00 × 21 ÷ 30 = $14.00`

最佳实践

1. 始终使用块/恢复模式

将支付捕获和供应包装在块/恢复中，以确保回滚：

```
- block:
    # 供应任务
    - name: 供应服务
      uri: ...

    # 仅在成功后捕获支付
    - name: 捕获支付
      uri: ...

  rescue:
    # 如果任何操作失败，则释放授权
    - name: 释放支付授权
      uri: ...

    - name: 失败 playbook
      fail:
        msg: "供应失败，客户未收费"
```

2. 验证所有 API 响应

始终确保关键操作成功：

```
- name: 授权支付
  uri: ...
  register: api_response_authorization

- name: 确保授权成功
  assert:
    that:
      - api_response_authorization.status == 200
      - api_response_authorization.json.success == true
    fail_msg: "支付授权失败: {{ api_response_authorization.json }}"
```

3. 存储授权 ID

始终保存 authorization_id 以供捕获/释放使用:

```
- name: 存储 authorization_id 以供捕获/释放
  set_fact:
    authorization_id: "{{
api_response_authorization.json.data.authorization_id }}"
```

4. 有效使用元数据

在授权请求中包含全面的元数据:

```
metadata:
  description: "清晰描述□户收费内容"
  service_id: "{{ service_id | int }}"
  product_id: "{{ product_id }}"
  user_id: "{{ initiating_user | int }}"
  title: "交易的简短标题"
  wholesale_cost: "{{ wholesale_cost | float }}"
  invoice: true  # 在捕□时自动创建交易
  send_email: true  # 发送客户通知
```

5. 四舍五入货币值

始终将货币计算四舍五入到小数点后两位:

```yaml
- name: 计算带四舍五入的费用
  set_fact:
    final_cost: "{{ (base_cost | float * multiplier | float) |
round(2) }}"
```

6. 处理缺失的支付方式

在尝试授权之前检查客户□否具有默认支付方式：

```yaml
- name: 获取默认支��方式
  set_fact:
    payment_method_id: "{{ api_response_payment_methods.json |
json_query(query) }}"
  vars:
    query: "data[?is_default==`true`].payment_method_id | [0]"

- name: 验证支付方式是否存在
  assert:
    that:
      - payment_method_id is defined
      - payment_method_id != ""
      - payment_method_id != None
    fail_msg: "未找到□户 {{ customer_id }} 的默认支付方式"
```

常见模式

模式 1：付费服务供应

对于需要立即支付的服务：

1. 获取客户的支付方式
2. 授权支付
3. 在 OCS/CGRateS 中供应服务
4. 在 CRM 中创建服务记录
5. 捕获支付
6. 失败时：释放授权并回滚

请参阅 `play_topup_charge_then_action.yaml` 获取完整示例。

模式 **2**：免费服务与设置费用

对于免费的服务但有一次性设置费用：

1. 供应服务
2. 创建服务记录
3. 直接添加设置费用交易（无需支付处理）
4. 设置费用出现在下一个发票中

请参阅 `play_simple_service.yaml` 的第 202-232 行获取完整示例。

模式 **3**：免费充值/附加服务

对于不需要支付的免费充值：

1. 获取服务信息
2. 执行 CGRateS 操作
3. 更新服务日期
4. 无需支付或交易创建

模式 **4**：通过 **ActionPlan** 进行定期收费

对于自动定期收费：

1. 创建带有 `*http_post` 的操作，指向供应端点
2. 创建带有 `*monthly` 定时的 ActionPlan
3. 将 ActionPlan 分配给账户
4. CGRateS 将每月自动调用该端点
5. 端点 playbook 处理支付处理

故障排除

授权失败

症状：授权端点返回错误

常见原因：

- 支付方式不存在或无效
- 资金不足
- 支付方式过期
- 客户 ID 不匹配

解决方案：检查支付方式状态和客户余额。

在成功供应后捕获失败

症状：服务已供应但支付捕获失败

问题：这□一个关键失败状态 - 服务处于活动状态但客户未收费

解决方案：

- 授权可能已过期（通常为 7 天）
- 在尝试捕获之前检查授权□否仍然有效
- 实施失败捕获的监控
- 可能需要手动干预

捕获后未创建交易

症状：支付已捕获但账单中没有交易

原因：在授权元数据中未设置 `invoice: true`

解决方案：或者：

- 在授权元数据中设置 `invoice: true`，或者

- 在成功捕获后手动创建交易

按比例计算不正确

症状：按比例收费与预期值不匹配

常见问题：

- 在天数计算中出现错误（包括/排除开始/结束日期）
- 使用了错误��月份进行天数计算
- 四舍五入错误

解决方案：

- 使用包含的日期范围（包括开始和结束天数）
- 始终四舍五入到小数点后两位
- 使用已知值测试计算
- 记录计算中包含的日期

退款和错误处理

退款选项

支付系统支持两种类型的退款：

1. 退款到支付来源 - 资金退回到原始卡/PayPal

```
- name: 退款到□户的卡
  uri:
    url: "http://localhost:5000/crm/payments/refund"
    method: POST
    headers:
      Content-Type: "application/json"
      Authorization: "Bearer {{ access_token }}"
    body_format: json
    body:
      {
        "transaction_id": "{{ vendor_transaction_id }}",
        "vendor": "stripe",
        "amount": "{{ refund_amount | float }}",
        "reason": "customer_request"
      }
```

2. 退款到钱包 - 立即为未来的购买提供余额（在错误场景中自动处理）

对于供应失败，系统自动将钱包记入，而不□退款到卡，以：

- 避免退款费用
- 提供立即可用性以进行重试
- 改善客户体验

有关完整详细信息，请参见 退款选项。

供应商支持

支付系统□ 与供应商无关的，目前支持：

- □ Stripe（卡，ACH）
- □ PayPal（PayPal 账户，卡）

可以在不更改 playbooks 的情况下添加新的支付供应商（Square、Adyen、Braintree 等）。

另请参见：

- 模块化架构 - 供应商抽象的工作原理
- 供应商配置 - Stripe/PayPal 设置和 API 密钥

相关文档

特定于 **Playbook** 的指南

- `concepts_ansible.md` - 一般 playbook 模式和结构
- `concepts_provisioning.md` - 供应系统概述

支付系统文档

- 支付系统 **API** 指南 - 完整的 API 参考，钱包路由，退款
  - 支付方式 API
  - 支付流程 API
  - 钱包优先路由
  - 退款选项
  - 模块化架构
- `payments_transaction.md` - 交易管理和手动收费
- `payments_process.md` - 处理支付和发票
- `basics_payment.md` - 支付方式和客户账单

基于角色的访问控制

OmniCRM中的角色、权限和用户

OmniCRM使用基于角色的访问控制（**RBAC**）：人（用户）被分配一个或多个角色，每个角色□权限的集合。权限□访问的最小单位（例如，view_customer，create_inventory）。用户的有效访问□所有分配角色的权限的并集。

目的

RBAC使得:

1. 数据保护 — 用户只能看到和做他们被允许的事情。
2. □作适应性 — 角色反映工作职能（管理员、支持、财务、客户管理员）。
3. 简单管理 — 通过分配角色授予访问权限；避免逐个用户的微观管理。
4. 租户隔离 — “查看自己的…”权限限制可见性到用户自己的客户/租户数据。

用户、角色和权限的关系

- 用户 — 实际登录OmniCRM的人。
- 权限 — 原子能力（例如，view_customer，delete_product）。

- ��色 — 权限的命名集合（例如，*Admin*, *Support*, *Finance*）。
- 分配 — 用户接收一个或多个角色；权限聚合。

身份验证证明□□□（JWT、API密钥或白名单IP）。授权（角色/权限）决定□□□□□。

管理用户

OmniCRM用户管理系统允许管理员创建和管理员工用户（管理员、客户服务代理），查看和修改用户角色，重置密码，管理双因素身份验证，并控制用户访问。

用户类型

客户用户 - 通过自助注册或管理员创建。自动分配“客户”角色。这些用户访问自助服务门户以管理他们的服务、查看使用情况、支付发票等。

员工用户 - 由具有适当权限的管理员创建。可以分配如Admin、Support、Finance等角色。这些用户访问CRM界面以管理客户、提供服务、处理账单等。

管理员用户 - 拥有`admin`权限的用户。对系统拥有完全访问权限，包括用户管理、角色管理和所有受保护的端点。

初始管理员用户由Omnitouch团队在系统部署时创建。

添加新用户（管理员和客户服务代理）

管理员可以通过Web UI或API创建新的员工用户。

通过**Web UI**

1. 导航到用户和角色 - 从管理菜单访问用户管理界面
2. 点击“添加用户” - 打开用户创建表单

3. 填写用户详细信息：
    - 用户名 - 登录的唯一用户名（必填）
    - 电子邮件 - 用户的电子邮件地址（必填，必须唯一）
    - 密码 - 临时密码（必填，用户应在首次登录时更改）
    - 名字 - 用户的名字（必填）
    - 中间名 - 用户的中间名（可选）
    - 姓氏 - 用户的姓氏（必填）
    - 电话号码 - 联系电话（可选）
    - 角色 - 选择一个或多个要分配的角色（必填）
    - 客户联系人 - 可选地将用户链接到客户联系人记录（针对客户用户）
4. 点击“创建用户” - 用户被创建并可以立即使用提供的凭据登录
5. 用户收到通知 - 可选地发送欢迎电子邮件，包含登录说明

最佳实践：

- 使用像`TempP@ssw0rd!`这样的临时密码，并要求用户在首次登录时更改
- 根据工作职能分配适当的角色（见下文的典型角色设计）
- 为所有管理员和支持人员启用双因素身份验证
- 将客户用户链接到他们的客户联系人记录，以便正确的数据范围

通过API

以编程方式创建用户：

端点：`POST /auth/users`

所需权限：`admin`

请求体：

```
{
  "username": "john.smith",
  "email": "john.smith@company.com",
  "password": "TempP@ssw0rd!",
  "first_name": "John",
  "middle_name": "D",
  "last_name": "Smith",
  "phone_number": "+61412345678",
  "role": "Support"
}
```

响应：

```
{
  "id": 123,
  "username": "john.smith",
  "email": "john.smith@company.com",
  "first_name": "John",
  "last_name": "Smith",
  "roles": ["Support"],
  "created": "2025-01-04T10:30:00Z"
}
```

分配多个角色：

□□□

获取特定用户：

`GET /auth/users/{user_id}`

返回完整的用户详细信息，包括：

- 个人信息
- 分配的角色和有效权限
- 双因素身份验证状态
- 上次登录和会话信息
- 关联的客户联系人（如适用）

创建和管理角色

角色可以分配给用户的权限集合。与其单独将权限分配给每个用户，不如创建角色，将相关权限打包并将这些角色分配给用户。

创建新角色

通过**Web UI**：

1. 导航到用户和角色 → 角色选项卡
2. 点击**“创建角色”**
3. 输入角色详细信息：
    - 名称 - 简短的描述性名称（例如，“Tier2_Support”）
    - 描述 - 解释角色的目的和责任
4. 点击**“创建”**
5. 角色被创建但没有权限；在下一步中添加权限

通过**API**：

端点：`POST /auth/roles`

所需权限：`admin`

请求：

```
{
  "name": "Tier2_Support",
  "description": "Level 2 support team with elevated provisioning
access"
}
```

响应:

```
{
  "id": 45,
  "name": "Tier2_Support",
  "description": "Level 2 support team with elevated provisioning
access",
  "permissions": [],
  "users": []
}
```

向角色添加权限

一旦角色被创建，分配权限以定义具有该角色的用户可以做什么。

通过**Web UI**:

1. 导航到用户和角色 → 角色选项卡
2. 点击角色名称以查看详细信息
3. 在权限部分，点击**“添加权限”**
4. 从列表中选择一个或多个权限
5. 点击**“添加”** - 权限立即分配

□□API□

□□□ `POST /auth/roles/{role_id}/permissions`

□□□

```
{
  "permission_id": 123
}
```

或添加多个权限：

```
{
  "permission_ids": [123, 124, 125]
}
```

示例：创建“供应专员”角色

该角色可以查看客户、管理服务和提供：

1. 创建角色：

```
POST /auth/roles
{
  "name": "Provisioning_Specialist",
  "description": "Can provision services and manage customer
services"
}
```

2. 添加权限：

```
POST /auth/roles/45/permissions
{
  "permission_ids": [
    1,   # view_customer
    20,  # view_customer_service
    21,  # create_customer_service
    22,  # update_customer_service
    30,  # view_provision
    31,  # create_provision
    40,  # view_inventory
    50,  # view_product
  ]
}
```

从角色中移除权限

通过**Web UI**:

1. 导航到角色详细信息
2. 在权限列表中，点击权限旁边的**“X”或“移除”**按钮
3. 确认移除

通过**API**:

端点: `DELETE /auth/roles/{role_id}/permissions/{permission_id}`

示例:

```
DELETE /auth/roles/45/permissions/31
```

这将从角色中移除`create_provision`权限。

编辑角色详细信息

更新角色名称或描述:

通过**Web UI**:

1. 导航到用户和角色 → 角色选项卡
2. 点击要编辑的角色
3. 修改角色名称或描述
4. 点击**“保存”**

通过**API**:

端点: `PUT /auth/roles/{role_id}`

```
{
  "name": "Senior_Support",
  "description": "Senior support team with full customer access"
}
```

删除角色

警告：删除角色将从所有分配的用户中移除该角色。确保用户有替代角色，否则他们将失去访问权限。

通过**API**:

`DELETE /auth/roles/{role_id}`

最佳实践：而不□删除，考虑归档或重命名不再需要的角色。

将角色分配给用户

在用户创建时:

在用户创建请求中包含角色（见“添加新用户”上文）。

对于现有用户：

通过**Web UI**：

1. 导航到用户和角色 → 用户选项卡
2. 点击要编辑的用户
3. 在角色部分，选择/取消选择角色
4. 点击**“保存”**

通过**API**：

更新用户的角色：

端点：`PUT /auth/users/{user_id}`

```
{
  "role": "Support,Finance"
}
```

或通过角色端点将单个角色分配给用户：

端点：`POST /auth/roles/{role_id}/users/{user_id}`

查看角色分配

角色中的所有用户：

`GET /auth/roles/{role_id}/users`

返回分配给该角色的所有用户的列表。

用户的所有角色：

`GET /auth/users/{user_id}`

响应包括`roles`数组，包含所有分配的角色。

管理用户密码

OmniCRM根据上下文提供多种密码管理方法。

用户自助密码重置

忘记密码的用户可以通过登录页面自行重置密码：

1. 点击登录页面上的“忘记密码”
2. 输入电子邮件地址 - 系统发送码重置电子邮件
3. 检查电子邮件 - 电子邮件包含一个安全的重置链接和令牌（有效期为1小时）
4. 点击链接 - 打开密码重置表单
5. 输入新密码 - 必须满足密码复杂性要求：
    - 最少8个字符
    - 至少一个大写字母
    - 至少一个小写字母
    - 至少一个数字
    - 至少一个特殊字符
6. 提交 - 密码立即更新；用户可以使用新密码登录

API流程：

1. 请求重置：

端点: `POST /auth/forgot_password`

```
{
  "email": "user@example.com"
}
```

系统生成重置令牌并发送电子邮件。

2. 使用令牌重置:

端点: `POST /auth/reset_password`

```
{
  "token": "abc123...",
  "new_password": "NewSecureP@ssw0rd!"
}
```

管理员密码重置

管理员可以在不需要电子邮件验证的情况下重置用户的密码。这将设置一个临时密码，用户应在下次登录时更改。

通过**Web UI**:

1. 导航到用户和角色 → 用户
2. 找到用户并点击**“重置密码”**按钮
3. 输入临时密码
4. 点击**“重置”**
5. 通知用户他们的临时密码（通过安全渠道）
6. 用户应在下次登录时更改密码

通过**API**:

端点: `POST /auth/users/{user_id}/admin_reset_password`

所需权限: `admin`

请求:

```
{
  "new_password": "TempP@ssw0rd!",
  "force_change": true
}
```

参数：

- `new_password` - 要设置的临时密码
- `force_change`（可选） - 如果为true，用户必须在下次登录时更改密码

用户更改自己的密码

经过身份验证的用户可以从其个人资料中更改自己的密码：

端点：`POST /auth/change_password`

请求：

```
{
  "current_password": "OldP@ssw0rd!",
  "new_password": "NewSecureP@ssw0rd!"
}
```

系统在允许更改之前验证当前密码。

密码安全

- 密码使用bcrypt（werkzeug安全性）进行哈希处理
- 从不以明文存储
- 重置令牌在1小时后过期
- 登录失败尝试可能触发账户锁定（可配置）
- 密码历史记录跟踪防止重用（可配置）
- 强制执行复杂性要求

管理双因素身份验证（2FA）

OmniCRM支持基于TOTP的双因素身份验证以增强安全性。管理员可以为用户启用、禁用和重置2FA。

为用户启用2FA

通过**Web UI**:

1. 导航到用户和角色 → 用户
2. ��击用户以查看详细信息
3. 在安全性部分，点击**“启用2FA”**
4. 系统生成:
   - TOTP密钥（显示QR码）
   - 10个备份代码（一次性使用）
5. 用户用身份验证器应用程序（Google Authenticator、Authy等）扫描QR码
6. 用户输入应用程序中的验证码以确认设置
7. 用户将备份代码保存在安全位置
8. 2FA现在已启用；所有未来的登录都需要

□□API□

1. **□□TOTP□□□**

   □□□ `POST /2fa/enable/user/{user_id}`

   □□□

```
{
  "totp_secret": "JBSWY3DPEHPK3PXP",
  "qr_code_url": "otpauth://totp/OmniCRM:user@example.com?
secret=JBSWY3DPEHPK3PXP&issuer=OmniCRM",
  "backup_codes": [
    "12345678",
    "23456789",
    "34567890",
    ...
  ]
}
```

2. 验证设置：

   端点：`POST /2fa/verify-setup/user/{user_id}`

   ```
   {
     "code": "123456"
   }
   ```

   成功时返回`{"verified": true}`。

2FA登录流程

一旦启用2FA，登录过程将发生变化：

1. 用户输入用户名和密码
2. 系统验证凭据
3. 如果有效，提示输入2FA代码
4. 用户输入来自身份验证器应用程序的代码或备份代码
5. 系统验证代码
6. 成功后，用户登录

备份代码：

- 在2FA设置期间生成的10个代码
- 仅限一次使用（使用后消耗）
- 如果身份验证器应用程序不可用，则使用
- 用户或管理员可以重新生成

验证2FA代码

端点：`POST /2fa/verify/user/{user_id}`

```
{
  "code": "123456"
}
```

接受以下两种：

- **TOTP**代码（来自身份验证器应用程序的6位数字）
- 备份代码（来自备份代码列表的8位数字）

重新生成备份代码

如果用户用尽备份代码或丢失它们，可以生成新的：

通过**Web UI**：

1. 导航到用户详细信息
2. 点击**“重新生成备份代码”**
3. 显示或发送新代码给用户
4. 旧代码失效

通过**API**：

端点：`POST /2fa/backup-codes/regenerate/user/{user_id}`

响应：

```
{
  "backup_codes": [
    "98765432",
    "87654321",
    "76543210",
    ...
  ]
}
```

管理员**2FA**重置

如果用户失去对其身份验证器应用程序和所有备份代码的访问权限，管理员可以禁用并重新启用2FA。

通过**Web UI**：

1. 导航到用户和角色 → 用户
2. 点击用户
3. 点击**“重置2FA”**按钮
4. 确认重置
5. 2FA被禁用；用户可以仅使用密码登录
6. 指导用户使用新密钥重新设置2FA

通过**API**:

端点: `POST /2fa/admin/disable/user/{user_id}`

所需权限: `admin`

这将完全禁用用户的2FA:

- 清除TOTP密钥
- 清除备份代码
- 将`is_2fa_enabled`标志设置为false

用户可以重新启用2FA以获取新密钥和备份代码。

用户自助服务**2FA**重置（新设备）

如果用户获得新设备但仍然有备份代码的访问权限:

端点: `POST /2fa/reset-for-new-device/user/{user_id}`

```
{
  "backup_code": "12345678"
}
```

系统验证备份代码，然后生成新的TOTP密钥和备份代码。用户可以在新设备上设置身份验证器应用程序。

2FA最佳实践

- 要求所有管理员和支持人员启用**2FA**
- 安全存储备份代码（密码管理器或安全笔记）
- 在使用多个后重新生成备份代码
- 使用信誉良好的身份验证器应用程序（Google Authenticator、Authy、Microsoft Authenticator）
- 为支持人员记录**2FA**重置程序

- 审计**2FA**使用情况 - 监控哪些用户启用了2FA

更新用户信息

管理员可以随时���新用户详细信息。

通过**Web UI**:

1. 导航到用户和角色 → 用户
2. 点击要编辑的用户
3. 修改任何可编辑字段:
    - 名字、中间名、姓氏
    - 电子邮件地址（需要验证）
    - 电话号码
    - 角色
    - 客户联系人链接
4. 点击**“保存”**

通过**API**:

端点: `PUT /auth/users/{user_id}`

```
{
  "first_name": "Jane",
  "last_name": "Doe-Smith",
  "email": "jane.doesmith@newcompany.com",
  "phone_number": "+61498765432",
  "role": "Support,Finance"
}
```

电子邮件更改:

当电子邮件更改时，新电子邮件将标记为待处理，直到验证:

- `pending_email`字段存储新电子邮件
- 向新地址发送验证电子邮件
- 用户点击链接进行验证

- `email`字段更新为新值
- `email_verified`标志设置为true

删除用户

OmniCRM使用软删除用户 - 它们被标记为已删除但不会从数据库中移除。这保留了审计跟踪和历史数据。

删除用户

通过**Web UI**:

1. 导航到用户和角色 → 用户
2. 找到要删除的用户
3. 点击**“删除”**按钮
4. 确认删除
5. 用户立即被注销，无法再次登录

通过**API**:

端点: `DELETE /auth/users/{user_id}`

所需权限: `admin`

发生的事情:

- `deleted`标志设置为`True`
- 记录`deleted_at`时间戳
- 用户无法登录
- 所有活动会话失效
- 用户仍然出现在审计日志和历史记录中
- 关联数据（客户联系人、活动）被保留

查看已删除用户

过滤已删除用户:

```
GET /auth/users/search?filters={"deleted":[true]}
```

恢复已删除用户

如果用户被错误删除，管理员可以恢复他们：

端点：`PUT /auth/users/{user_id}`

```
{
  "deleted": false
}
```

这将清除`deleted`标志，并允许用户再次登录。

注意：用户的密码保持不变，因此他们可以使用之前的密码。

永久删除用户

警告：这□不可逆的，并会从数据库中删除所有用户数据。

未通过UI公开。仅通过直接数据库访问可用，出于合规原因（例如，GDPR数据删除请求）。

用户删除的最佳实践

- 默认软删除 - 保留审计跟踪
- 记录删除原因 - 在删除之前在活动日志中添加注释
- 转移所有权 - 在删除之前重新分配用户的未完成工单、任务
- 审查访问 - 确保没有关键流程依赖于该用户
- 归档数据 - 如有需要，导出用户的工作历史
- 通知相关团队 - 通知经理/同事删除情况

权限目录

权限通常遵循CRUD模式：

- view_* — 读取/浏览
- create_* — 创建/添加
- update_* — 编辑/修改

- delete_* — 删除/移除

某些实体还包括**“查看自己的...”**变体，限制可见性到当前用户的客户/租户。

全局/管理

- `admin` — 完全管理访问权限（管理用户、角色和权限；访问所有受保护的端点）。
- `can_impersonate` — 临时充当其他用户（经过审计；用于支持/故障排除）。
- `grafana_access` — 访问Grafana分析仪表板以创建和查看自定义报告和可视化。有关详细信息，请参见下面的Grafana分析部分。

客户及相关记录

- 客户
  - view_customer，create_customer，update_customer，delete_customer
  - 自有范围：□□□□□□□
- 客户站点
  - view_customer_site，create_customer_site□□□ update_customer_site，delete_customer_site
  - 自有范围：□□□□□□□□□
- 客户联系人

    - view_customer_contact，create_customer_contact，update_customer_contact，delete_customer_contact
    - 自有范围：*[illegible]*
- 客户属性（见`[illegible]户属性 <administration_attributes>`）
    - view_customer_attribute，create_customer_attribute，update_customer_attribute，delete_customer_attribute
    - 自有范围：*[illegible]*
- 客户标签（见`[illegible]户标签 <administration_tags>`）
    - view_customer_tag，create_customer_tag，update_customer_tag，delete_customer_tag
    - 自有范围：*[illegible]*
- 客户服务
    - view_customer_service，create_customer_service，update_customer_service，delete_customer_service
    - 自有范围：*[illegible]*
- 客户活动
    - view_customer_activity，create_customer_activity，update_customer_activity，delete_customer_activity
    - 自有范围：*[illegible]*

账单

- **Stripe**卡
    - `view_customer_stripe_card`，`create_customer_stripe_card`，`update_customer_stripe_card`，`delete_customer_stripe_card`
    - 自有范围：*[illegible]Stripe[illegible]*
- 交易
    - `view_customer_transaction`，`create_customer_transaction`，`update_customer_transaction`，`delete_customer_transaction`
    - 自有范围：*[illegible]*
- 发票
    - `view_customer_invoice`，`create_customer_invoice`，`update_customer_invoice`，`delete_customer_invoice`
    - 自有范围：*[illegible]*

通信

- view_communication，create_communication，update_communication，delete_communication
- 自有范围：*[illegible]*

库存和模板

- 库存
  - `view_inventory`，`create_inventory`，`update_inventory`，`delete_inventory`
  - 自有范围：*[illegible]*
- 库存模板
  - `view_inventory_template`，`create_inventory_template`，`update_inventory_template`，`delete_inventory_template`
  - 自有范围：*[illegible]*

产品

- `view_product`，`create_product`，`update_product`，`delete_product`

单元广播（**CBC**）

- `view_cbc_message`，`create_cbc_message`，`update_cbc_message`，`delete_cbc_message`

提供

- 提供
  - `view_provision`，`create_provision`，`update_provision`，`delete_provision`
  - 自有范围：*[illegible]*
- 提供事件
  - `view_provision_event`，`create_provision_event`，`update_provision_event`，`delete_provision_event`

“查看自己的”访问

“查看自己的...”权限将读取（并在实现的情况下可选地编辑）范围限制为与用户的自己客户/租户相关的数据。例如，□□□□□角色可以管理□□租户的联系人、站点、发票和服务，但无法查看其他租户。

典型角色设计

角色	典型权限	备注
系统管理员	`admin`，可选`can_impersonate`；加上所需的广泛CRUD权限	对用户/角色/权限的完全控制
支持	`view_customer`，`view_customer_service`，`view_communication`，`view_provision`；可选更新	如果允许，添加`can_impersonate`
财务	`view_customer_invoice`，`view_customer_transaction`，`view_product`；可选`create_customer_invoice`	以读取为主；写入有限
客户管理员（租户）	“查看自己的...”跨越客户、站点、联系人、服务、库存、发票、交易、通信、提供	租户范围内的管理
只读审计员	仅广泛的`view_*`	无创建/更新/删除

示例□□色及包含的权限（摘要）

通过API管理角色和权限

所有端点都需要`admin`权限。

列出权限

端点: `GET /auth/permissions`

创建权限（少见）

端点: `POST /auth/permissions`

请求体:

```
{
    "name": "view_example",
    "description": "Read-only access to example objects"
}
```

列出角色

端点: `GET /auth/roles`

创建角色

端点: `POST /auth/roles`

请求体:

```
{
"name": "Support",
"description": "Tier-1 support team"
}
```

向角色添加权限

端点: `POST /auth/roles/{role_id}/permissions`

请求体:

```
{
  "permission_id": 123
}
```

从角色中移除权限

端点: `DELETE /auth/roles/{role_id}/permissions/{permission_id}`

将角色分配给用户

创建带角色的用户

端点: `POST /auth/users`

请求体:

```
{
  "username": "sara",
  "email": "sara@example.com",
  "password": "TempP@ssw0rd!",
  "first_name": "Sara",
  "last_name": "Ng",
  "phone_number": "+61...",
  "role": "Support"
}
```

更新用户的角色

端点: `PUT /auth/users/{user_id}`

请求体:

```
{
  "role": "Finance"
}
```

列出用户（仅管理员）

端点: `GET /auth/users`

伪装（受控）

- 要求: `can_impersonate` 或 `admin`

开始伪装

端点：`POST /auth/impersonate`

请求体：

```
{ "user_id": 42 }
```

停止伪装

端点：`POST /auth/stop_impersonation`

最佳实践

- 最小权限优先。从最小角色开始；根据需要添加权限。
- 优先使用“查看自己的...”。对于面向客户的角色，使用租户范围的权限。
- 保持角色稳定。当功能发生变化时更新角色权限——不要编辑每个用户。
- 定期审计。审查谁拥有`admin`或`can_impersonate`。

Grafana分析和仪表板

OmniCRM与Grafana集成，以提供强大的分析和可视化功能。拥有`grafana_access`权限的用户可以访问Grafana以创建自定义仪表板、报告和可视化。

访问Grafana

如果您的用户帐户已获得`grafana_access`权限，您可以通过在浏览器中导航到`/grafana`来访问Grafana。您将使用您的OmniCRM凭据自动进行身份验证 - 无需单独的Grafana登录。

在Grafana界面中，您可以：

- 创建带有图表、图形和表格的自定义仪表板
- 构建复杂查询以分析客户数据、收入趋势和服务指标
- 根据关���绩效指标设置警报
- 与其他团队成员共享仪表板

在**OmniCRM**中嵌入仪表板

一旦您在Grafana中创建了仪表板，您可以将它们直接嵌入到OmniCRM界面中。这使您能够在不离开CRM的情况下查看最重要的指标，同时保持OmniCRM导航的完整性。

要嵌入仪表板：

1. 在**Grafana**中创建仪表板 - 导航到`/grafana`并设计您需要的可视化仪表板。
2. 记下仪表板**ID** - 在Grafana中查看仪表板时，检查URL。它将类似于`/grafana/d/abc123/dashboard-name`。ID`/d/`后面的部分（在这个例子中`abc123`）。
3. 在您的环境中配置仪表板 - 联系您的系统管理员，将仪表板ID和显示名称添加到系统配置中。仪表板通过环境变量进行配置。
4. 从侧边栏访问 - 一旦配置，您的仪表板将自动出现在OmniCRM侧边栏的“仪表板”菜单下。您可以单击它以查看完整的仪表板，同时保持OmniCRM导航可见。

仪表板配置

仪表板由您的系统管理员使用环境变量进行配置。每个仪表板需要：

- 仪表板**ID** - 来自Grafana URL的唯一标识符
- 仪表板标签 - 在OmniCRM侧边栏中显示的友好名称

可以配置多个仪表板，它们将按定义的顺序作为菜单项出现。有关环境变量配置详细信息，请参见管理配置文档。

谁可以访问**Grafana**？

只有拥有`grafana_access`权限的用户才能访问Grafana并查看嵌入的仪表板。此权限通常授予：

- 需要全面了解系统指标的管理员
- 创建报告和可视化的业务分析师
- 跟踪收入和账单指标的财务团队
- 监控服务性能的运营团队

常见问题

用户可以有多个角色吗？可以。权限累加的。

我需要自定义权限吗？通常不需要。内置目录涵盖大多数需求。

“查看自己的**...”**规则如何知道我的是什么？它们评估您的用户/联系人与您的客户（租户）之间的链接。

我可以创建自己的**Grafana**仪表板吗？可以，如果您拥有`grafana_access`权限。导航到`/grafana`以访问完整的Grafana界面。

自助服务门户

自助服务门户□一个面向客户的界面，允许最终用户管理自己的账户、查看使用情况、支付发票和修改服务，而无需客户服务人员的协助。

访问方式：

- 通过客户凭证直接登录
- 员工模拟 `<customer_care>` 进行故障排除（从客户 → 联系人 → “以用户身份登录”）

另见：□户服务 `<customer_care>` 以获取模拟详细信息，身份验证流程 `<authentication_flows>` 以获取登录过程。

目的

自助服务门户为客户提供：

1. 账户管理 - 查看和更新个人信息、联系人和地址
2. 服务概览 - 查看所有活动服务、使用情况和到期日期
3. 使用跟踪 - 监控数据、语音、短信和货币余额
4. 账单访问 - 查看和支付发票，管理支付方式
5. 服务修改 - 添加充值、购买附加功能、修改服务
6. **24/7** 可用性 - 随时访问账户信息，无需拨打支持电话

门户概览

当客户登录自助服务门户时，他们会看到一个个性化��仪表板，包含：

导航部分：

- 仪表板 - 服务和最近活动的快速概览
- 账户 - 个人详细信息、联系人、站点
- 服务 - 所有服务的状态和详细信息列表
- 使用情况 - 余额消耗和到期信息
- 账单 - 发票、交易、支付方式

- 充值 - 购买数据、语音、短信或货币信用

账户详细信息

账户部分显示客户信息，并允许有限的自助更新。

可编辑字段：

客户可以更新：

- 电子邮件地址（需要验证）
- 电话号码
- 密码
- 通知偏好

只读信息：

- 客户ID
- 账户创建日期
- 客户类型（个人/企业）
- 站点（地址）

- 关联联系人

更新账户信息：

1. 导航到 账户 → 详细信息
2. 点击要更新的字段旁边的“编辑”
3. 输入新信息
4. 点击“保存更改”
5. 对于电子邮件更改，通过发送到新地址的链接进行验证

安全功能：

- 密码更改需要当前密码
- 电子邮件更改需要验证
- 活动记录以供审计
- 2FA 设置（如果启用）

服务概览

服务部分显示客户的所有活动和非活动服务。

服务卡显示：

每项服务显示：

- 服务名称 - 可读的标识符（例如，“移动 - +44 7700 900123”）
- 产品 - 计划或产品名称
- 状态 - 活动、暂停、过期、取消
- 创建日期 - 服务激活日期
- 到期日期 - 服务到期时间（如适用）
- 月费 - 定期收费
- 自动续订 - 启用/禁用指示

服务□作：

- 查看使用情况 - 查看余额消耗（数据、语音、短信、货币）
- 充值 - 添加信用或数据
- 附加功能 - 购买额外功能
- 修改 - 更改服务参数（如果允许）
- 查看详细信息 - 查看完整的服务配置

服务状态指示器：

- □ 活动 - 服务正常运行
- □ 即将到期 - 在 <7 天内续订或到期
- □ 暂停 - 服务暂时禁用（付款问题，手动暂停）
- ● 过期 - 服务不再活动

使用跟踪

客户可以实时监控所有余额类型的使用情况。

数据使用

查看数据消耗，并按桶和到期进行详细划分。

数据使用显示：

- 总余额 - 所有数据桶的总和
- 本期使用 - 自上次续订以来的消耗
- 进度条 - 消耗的可视化表示
- 到期信息 - 每个桶的到期时间
- 桶划分 - 多个数据桶按优先级顺序排列

桶优先级：

当存在多个数据桶时（例如，基础计划 + 充值），它们按权重顺序消耗：

- 权重 **10** - 首先消耗（通常为促销/奖金数据）
- 权重 **20** - 第二消耗（通常为充值数据）
- 权重 **30** - 最后消耗（通常为基础计划数据）

语音使用

跟踪所有语音桶中剩余的通话分钟。

语音使用显示：

- 剩余分钟 - 总语音余额
- 使用分钟 - 本期消耗
- 通话记录 - 最近通话（如果启用）
- 到期日期 - 语音桶的到期时间
- 国际分钟 - 单独跟踪（如适用）

使用划分：

- 网络内通话 - 同一网络内的通话
- 网络外通话 - 拨打其他网络的通话
- 国际通话 - 拨打国外的通话
- 特殊号码 - 特殊费率号码

短信使用

监控短信消息的配额和消耗。

短信使用显示：

- 剩余消息 - 短信余额
- 本月使用 - 已发送的消息
- 包含**MMS** - MMS□否计入余额
- 国际短信 - 单独跟踪（如适用）

货币余额

查看按需服务的预付信用余额。

货币显示：

- 当前余额 - 可用信用
- 上次充值 - 最近的充值金额和日期
- 到期日期 - 余额到期时间（如适用）
- 自动充值 - 启用/禁用状态

账单管理

客户可以查看发票、交易并管理支付方式。

发票

直接从门户查看和支付未付款的发票。

发票列表显示:

- 发票编号 - 唯一标识符
- 日期 - 发票创建日期
- 到期日期 - 付款截止日期
- 金额 - 发票总金额
- 状态 - 已支付、未支付、逾期
- □作 - 下载PDF、在线支付

支付发票:

1. 导航到 账单 → 发票
2. 在列表中找到未支付的发票
3. 点击“立即支付”按钮
4. 选择支付方式（保存的卡或新卡）
5. 确认支付
6. 收到确认电子邮件

下载发票:

1. 点击发票旁的“下载”图标

2. PDF 下载包含完整的发票详细信息
3. 保存用于税务记录和文档

交易

查看完整的交易历史，包括收费、信用和付款。

交易显示：

- 日期 - 交易创建日期
- 描述 - 收费/信用的原因
- 金额 - 收费（正数）或信用（负数）
- 发票 - 包含此交易��发票
- 状态 - 已开票或未开票

支付方式

管理保存的信用卡以便自动计费和在线支付。

支付方式管理：

- 添加卡 - 通过Stripe安全地添加新信用/借记卡
- 设为默认 - 选择主要支付方式
- 删除卡 - 删除过期或未使用的卡
- 卡详细信息 - 查看最后4位数字、到期、卡品牌

请参见 支付方式 `<payment_methods>` 以获取详细的支付管理文档。

充值 / 充值

立即购买额外的数据、语音、短信或货币信用。

充值流程：

1. 导航到 服务 → 选择服务 → “充值”
2. 从目录中选择充值产品

3. 选择金额（预设选项或自定义）
4. 查看费用和到期信息
5. 选择支付方式
6. 确认购买
7. 余额立即更新

可用充值：

- 数据充值 - 1GB、5GB、10GB、20GB、50GB 选项
- 语音充值 - 额外通话分钟
- 短信包 - 消息包
- 货币信用 - 预付余额（£5、£10、£20、£50、£100）

请参见 充值与充值 `<features_topup_recharge>` 以获取详细的充值工作流程。

服务附加功能

购买现有服务的额外功能和增强功能。

可用附加功能：

- 国际漫游 - 启用海外服务
- 静态IP地址 - 家庭互联网的固定IP
- 优质内容 - IPTV频道、流媒体服务
- 硬件升级 - 调制解调器租赁、机顶盒
- 速度提升 - 临时带宽增加

购买附加功能：

1. 导航到 服务 → 选择服务 →“附加功能”
2. 浏览此服务类型的可用附加功能
3. 点击所需附加功能上的“添加到服务”
4. 查看费用（一次性 + 定期）
5. 确认购买
6. 附加功能自动提供

请参见 `修改服务 - 附加功能 <csa_modify>` 以获取附加功能管理的详细信息。

通知与警报

客户会收到重要事件的自动通知：

电子邮件通知：

- 发票生成并准备付款
- 收到付款确认
- 服务到期警告（7天、3天、1天）
- 余额不足警报（数据、语音、货币）
- 服务激活/停用
- 密码重置请求
- 安全警报（新设备登录）

门户内警报：

- 未支付的发票
- 即将到期的服务
- 数据不足警告（剩余10%）
- 支付方式即将到期
- 需要的操作（验证电子邮件，更新支付方式）

通知偏好：

客户可以配置：

- 电子邮件通知频率
- 短信警报（如果启用）
- 警报阈值（例如，当剩余数据<20%时通知）
- 通知类别（账单、使用、服务）

自助服务限制

某些操作需要员工协助��

需要客户服务：

- 更改客户类型（个人 ↔ 企业）
- 在客户之间转移服务
- 取消服务（如果启用，可能□自助服务）
- 争议发票
- 请求退款
- ���改主要联系人
- 复杂的配置问题

安全限制：

- 不能查看或修改其他用户的账户
- 限于自己的客户数据（租户隔离）
- 不能访问管理功能
- 不能作废发票或交易
- 不能修改服务配置（仅限附加功能/充值）

员工模拟访问

支持人员可以以客户身份访问自助服务门户进行故障排除。

模拟流程：

1. 导航到 客户 → 选择客户
2. 转到 联系人 选项卡
3. 找到与用户账户关联的联系人
4. 点击 “以用户身份登录” 按钮
5. 新标签页打开客户的自助服务门户视图
6. 所有操作都被记录并归因于模拟用户
7. 员工看到的内容与客户看到的完全相同

用例：

- 故障排除 - 复制客户报告的问题
- 验证 - 确认服务配置□否正确显示
- 培训 - 演示门户功能
- 支持 - 在查看客户屏幕的同时指导客户使用门户

安全与审计：

- 需要 `can_impersonate` 或 `admin` 权限
- 所有操作都记录到审计日志中
- 客户在活动日志中看到模拟记录
- 会话在不活动后超时
- 模拟时无法更改客户密码

请参见 `□户服务 - 用户模拟 <customer_care>` 以获取完整的��拟文档。

移动响应性

自助服务门户完全响应并优化为移动设备使用。

移动功能：

- 触摸优化的导航
- 小屏幕的简化布局
- 轮播导航的滑动手势
- 移动友好的表单和输入
- 用于eSIM配置的二维码扫描
- 一键拨打电话
- GPS集成以实现地址自动完成功能

渐进式**Web**应用程序**（PWA）**:

- 安装为主屏幕上的应用
- 离线查看最近数据
- 推送通知（如果启用）
- 使用服务工作者快速加载

密码重置与账户恢复

客户可以在不拨打支持电话的情况下重置忘记的密码。

自助密码重置:

1. 点击登录页面上的**“**忘记密码**”**
2. 输入电子邮件地址
3. 接收密码重置电子邮件（有效期1小时）
4. 点击电子邮件中的链接
5. 输入新密码（必须满足复杂性要求）
6. 提交并使用新密码登录

密码要求:

- 最少8个字符
- 至少一个大写字母
- 至少一个小写字母
- 至少一个数字
- 至少一个特殊字符 (!@#$%^&*)

账户锁定：

在5次登录失败后：

- 账户锁定30分钟
- 自动发送密码重置电子邮件
- 向注册电子邮件发送安全通知

客户最佳实践

安全建议：

1. 启用2FA以增强安全性
2. 使用独特且强大的密码
3. 保持电子邮件地址最新以接收通知
4. 设置默认支付方式以便自动续订
5. 定期监控使用情况以避免超额收费
6. 将备份代码保存在安全位置（如果启用2FA）
7. 使用共享/公共计算机后注销

使用管理：

1. 启用低余额警报（剩余10-20%）
2. 在余额到期前充值以避免服务中断
3. 每月查看发票以查找意外费用
4. 在卡片到期前更新支付方式
5. 在整个月份内监控数据使用情况以避免限速

支持升级：

如果自助服务无法解决问题：

1. 检查知识库/帮助文章（如果可用）
2. 查看活动日志以获取最近更改
3. 通过电话、电子邮件或聊天联系客户服务
4. 提供客户ID和服务详细信息以便更快解决

API访问

具有技术需求的客户可以直接使用API。

API密钥生成:

可供企业客户或根据请求提供:

1. 导航到 账户 → **API**访问
2. 点击 **“**生成**API**密钥**”**
3. 设置权限（只读或读写）
4. 设置到期日期
5. 安全保存API密钥（仅显示一次）

API用例:

- 自动化使用监控
- 与内部计费系统集成
- 程序化充值
- 通过脚本进行服务配置
- 数据导出以进行分析

请参见 `API文档 <concepts_api>` 以��取端点详细信息和示例。

常见问题解答

问：为什么我看不到���所有的服务?

答：确保您使用正确的账户登录。如果您有多个客户账户，每个账户都有单独的服务。如有需要，请联系支持以合并账户。

问：我的付款失败，但我被收费了。我该怎么办?

答：检查您的银行对账单以查看待处理的费用。如果费用出现但发票仍显示未支付，请联系支持并提供交易参考编号。

问：我如何取消服务?

答：导航到服务详细信息并点击“取消服务”（如果启用自助服务）。否则，请联系客户服务处理取消。

问：我可以将服务转移给其他人吗？

答：不可以，服务转移需要客户服务的协助，以确保安全和合规。

问：为什么我的数据余额下降得比预期快？

答：检查后台应用更新、视频流质量和自动云备份。查看门户中的使用划分以获取详细消耗信息。

问：我丢失了我的**2FA**设备。我该如何恢复访问？

答：使用备份代码登录，然后禁用并重新启用2FA。如果没有备份代码，请联系支持以重置2FA（需要身份验证）。

问：我可以在不登录的情况下支付发票吗？

答：通过直接发票链接可能可以进行访客发票支付。否则，出于安全原因需要登录。

问：我如何一次性下载所有发票？

答：目前需要单独下载每张发票。如需批量下载，请联系支持或使用API（如果可用）。

相关文档

- □户服务 - 用户模拟 `<customer_care>` - 员工故障排除访问
- 支付方式 `<payment_methods>` - 管理卡片和支付
- 充值与充值 `<features_topup_recharge>` - 购买额外信用
- 服务使用 `<csa_service_usage>` - 理解余额跟踪
- 账单概述 `<billing_overview>` - 账单概念和发票
- 身份验证流程 `<authentication_flows>` - 登录和安全
- `2FA <2fa>` - 双因素身份验证设置
- 服务修改 `<csa_modify>` - 添加功能和附加功能