□置参考

所有□置参数的完整指南

架构概述

OmniMessage SMPP □关是一个 无状态协议前端，用于将 SMPP 消息转换为 OmniMessage 格式，反之亦然。所有业务逻辑、路由决策和消息存储均由 OmniMessage Core 处理 - □关仅仅：

1. 接收 来自运营商和客户端的 SMPP PDU
2. 通过 **REST API** 将其转换为 OmniMessage 格式
3. 轮询 OmniMessage 以获取要发送的消息
4. 发送 SMPP PDU 到运营商
5. 将 交付状态报告回 OmniMessage

这与其他 OmniMessage 前端（Diameter、MAP、IMS）的工作方式相同 - 它们都是无状态协议转换器，委托给 OmniMessage Core。

□置文件位置

`/opt/omnimessage-smpp/config/runtime.exs`

重要：更改配置后，请重启□关：

```
sudo systemctl restart omnimessage-smpp
```

□置结构

配置文件使用 Elixir 语法。基本结构：

```
import Config

# □局设置
config :omnimessage_smpp,
  setting_name: value

# SMPP 绑定
config :omnimessage_smpp, :binds, [
  %{
    name: "bind_name",
    # ... 绑定设置
  }
]
```

全局设置

API_BASE_URL

OmniMessage Core 平台 URL

```
config :omnimessage_smpp,
  api_base_url: "https://omnimessage-core.example.com:8443"
```

参数	类型	必需	默认
`api_base_url`	字符串 (URL)	是	-

目的: OmniMessage Core 平台的 URL。□关通过 REST API 与 OmniMessage 进行所有消息处理的通信:

- 提交消息: 将接收到的 SMPP 消息发送到 OmniMessage 进行处理
- 检索消息: 轮询目标为 SMPP 运营商的消息
- 报告交付状态: 将消息交付状态更新回 OmniMessage
- 系统健康: 定期健康检查

关键: 这是□关获取所有“智能”的地方。OmniMessage 处理:

- ✓ 消息验证和格式检查
- ✓ 路由决策（使用哪个运营商）
- ✓ 限速和节流
- ✓ 号码验证
- ✓ 消息存储和持久性
- ✓ 交付重试逻辑
- ✓ 状态跟踪

□关仅仅是将 SMPP ↔ OmniMessage 格式进行转换。

示例：

```
# 使用 IP 的 HTTPS
api_base_url: "https://192.168.1.100:8443"

# 使用主机名的 HTTPS
api_base_url: "https://omnimessage-core.company.com:8443"

# HTTP（不推荐用于生产）
api_base_url: "http://192.168.1.100:8080"
```

网络要求：

- □关必须能够访问 OmniMessage Core
- 在生产环境中使用 HTTPS（配置 `verify_ssl_peer`）
- 防火墙必须允许指定端口的出站 HTTPS

OAM/管理 API 端口（`api_ex`）

网关自身 **OAM/管理 API** 的端口

□关通过 `api_ex` 提供其自身 OAM/管理 API（`/status`，`/smpp/client_peers`，`/smpp/server_peers`，`/smpp/binds`，`/smpp/summary`），默认值为 `8443`（`config/config.exs`）。这是□关托管的服务器 - 与 `api_base_url` 不同，后者是□关在 OmniMessage Core 上调用的客户端端点。

不要混淆这两者：

设置	方向	默认	意义
`api_base_url`	关 → Core (客户端)	`...:8443`	关发送/轮询消息的地方
`api_ex` `port`	关 (服务器)	`8443`	关自身 OAM API 监听的地方

与 **OmniMessage Core** 的共址: OmniMessage Core 也在 `api_ex` 端口 `8443` 上托管其存储/消息 API。当 SMPP 关在与 Core 相同的主机上运行时，两个都默认为 `8443`，只能有一个绑定 - 另一个会因 `:eaddrinuse` 崩溃循环（而 `/api/messages` 请求会命中错误的服务，返回 404）。

要修复此问题，请编辑 `config/runtime.exs`（部署到 `/etc/omnimessage_smpp/runtime.exs`）并取消注释 `api_ex` 覆盖，设置一个空闲端口，以便关的 OAM API 移出 Core 的端口。`api_base_url` 仍然指向 Core 的 `8443`：

```
# Core 拥有 :8443 用于存储 API; 将网关的 OAM API 移至 :8444
config :api_ex,
  api:
    Application.get_env(:api_ex, :api, %{})
    |> Map.put(:port, 8444)
    |> Map.put(:listen_ip, "0.0.0.0")
```

如果覆盖保持注释状态，端口默认为 `8443`。独立关（Core 在不同主机上）无需更改。

SMPP_POLL_INTERVAL

队列检查频率（毫秒）

```
config :omnimessage_smpp,
  smpp_poll_interval: 100
```

参数	类型	必需	默认
`smpp_poll_interval`	整数	否	100

目的：每个客户端检查消息队列的频率（以毫秒为单位）。

指南:

- 高流量 **(>100 TPS)**: 100-500ms
- 中等流量 **(10-100 TPS)**: 500-1000ms
- 低流量 **(<10 TPS)**: 1000-2000ms

环境变量: `SMPP_POLL_INTERVAL`

VERIFY_SSL_PEER

SSL 证书验证

```
config :omnimessage_smpp,
  verify_ssl_peer: false
```

参数	类型	必需	默认
`verify_ssl_peer`	布尔值	否	false

目的: 在连接到后端 API 时是���验证 SSL 证书。

值:

- `true`: 验证证书（生产环境中使用有效证书）
- `false`: 跳过验证（自签名证书，测试）

环境变量: `VERIFY_SSL_PEER`

SMSC_NAME

网关注册标识符

```
config :omnimessage_smpp,
  smsc_name: "smpp_gateway"
```

参数	类型	必需	默认
`smsc_name`	字符串	否	"smpp_gateway"

目的：在消息队列后端中标识此□关实例。

环境变量：`SMSC_NAME`

SMPP 客户端绑定□置

客户端绑定是 出站连接，其中□关充当 **ESME**（客户端），连接到运营商 **SMSC** 服务器。在此模式下，□关发起连接以通过外部运营商发送和接收消息。

完整客户端绑定示例

```
config :omnimessage_smpp, :binds, [
  %{
    # 此连接的唯一标识符
    name: "vodafone_uk",

    # 连接模式
    mode: :client,

    # SMPP 绑定类型
    bind_type: :transceiver,

    # 运营商 SMPP 服务器地址
    host: "smpp.vodafone.co.uk",
    port: 2775,

    # 固定出站连接的本地源地址/端口（可选，
    # 如果运营商按源 IP/端口进行白名单设置，则设置此项）
    bind_source_ip: "10.20.30.40",
    bind_source_port: 5000,

    # ��份验证凭据
    system_id: "your_username",
    password: "your_password",

    # SMPP 协议字段（可选，如果运营商要求则设置）
    system_type: "",
    addr_ton: 0,
    addr_npi: 0,
    address_range: "",

    # 限速
    tps_limit: 100,

    # 共享队列名称（可选，默认为绑定名称）。在多个绑定上设置相同的
    # 值以使它们从一个公共队列中排出。
    queue: "vodafone_uk",

    # 队列检查频率
    queue_check_frequency: 1000,

    # 保持连接间隔（秒，0 表示禁用）
```

```
    enquire_link_interval: 60,

    # 消息缓存（可选）
    cache_enabled: false,
    cache_max_size: 10000,
    cache_retry_interval: 60
  }
]
```

客户端绑定参数

name

唯一连接标识符

类型	必需	示例
字符串	是	`"vodafone_uk"`

目的：唯一标识此 SMPP 连接。

- 用于日志和指标
- 在所有绑定中必须唯一
- 使用描述性名称（运营商、区域、目的）

命名约定：

- `carrier_region`：`"vodafone_uk"`，`"att_us"`
- `purpose_number`：`"marketing_1"`，`"alerts_primary"`

mode

连接类型

类型	必需	值
原子	是	`:client`

□□□

port

SMPP 服务器端口

类型	必需	默认	范围
整数	是	2775	1-65535

目的: SMPP 连接的 TCP 端口。

标准端口: 2775

示例:

```
port: 2775  # 标准
port: 3000  # 自定义
```

bind_source_ip

出站连接的本地源 IP

类型	必需	默认	示例
字符	否	(操作系统默认)	`"10.20.30.40"`

目的: 固定用于出站 TCP 连接到运营商的本地源 IP 地址。当运营商按源 IP 白名单设置连接且关主机有多个地址时，请设置此项。无效地址将被记录并忽略（操作系统选择源 IP）。

注意: 这是一个仅配置字段（从 `:binds` 应用程序配置中读取）；无法通过 Web UI / Mnesia 存储进行编辑。

bind_source_port

出站连接的本地源端口

类型	必需	默认	范围
整数	否	(临时)	1-65535

目的：固定用于出站 TCP 连接的本地源端口。当运营商按源 IP 和 端口进行白名单设置时，请设置此项。套接字使用 `SO_REUSEADDR` 打开，因此固定端口可以在重新连接时立即重新绑定（而不是在先前套接字处于 `TIME_WAIT` 状态时因 `:eaddrinuse` 失败）。

注意：与 `bind_source_ip` 类似，这是一个仅配置字段，无法通过 Web UI 进行编辑。

system_id

身份验证用户名

类型	必需	示例
字符串	是	`"company_user"`

目的：运营商提供的用于身份验证的用户名。

安全性：保护此凭据 - 存储在配置文件中。

password

身份验���密码

类型	必需	示例
字符串	是	`"secret_password"`

目的：运营商提供的用于身份验证的密码。

安全性：

- 保护此凭据
- 使用强密码
- 定期轮换

tps_limit

每秒事务限制

类型	必需	默认	范围
整数	是	100	1-10000

目的：通过此连接发送的最大消息数。

指南：

- 设置为运营商最大值的 70-80%
- 防止限速/断开连接
- 为交付回执留出余地

示例：

```
tps_limit: 10    # 低流量
tps_limit: 50    # 中等流量
tps_limit: 100   # 高流量（最常见）
tps_limit: 1000  # 非常高的流量
```

计算：

```
如果运营商最大 = 100 TPS
设置 tps_limit = 70-80
留出 20-30 TPS 的余地
```

queue

共享队列名称

类型	必需	默认	示例
字符串	否	绑定 `name`	`"sinch.smpp"`

目的：此绑定从后端排出的队列。默认为绑定自身的 `name`，因此每个绑定排出自己的队列。在多个绑定上设置相同的 `queue` 值以使它们都从一个共享队列中排出 - 例如，两个 Sinch 连接都排出 `"sinch.smpp"` 以进行负载共享/冗余。

示例：

```
# 两个绑定共享一个队列
%{name: "sinch_1", queue: "sinch.smpp", ...}
%{name: "sinch_2", queue: "sinch.smpp", ...}
```

queue_check_frequency

消息队列轮询间隔（毫秒）

类型	必需	默认	范围
整数	是	1000	100-10000

目的：多长时间检查后端以获取要发送的新消息。

指南：

- 高流量 **(>100 TPS)**: 500-1000ms
- 中等流量 **(10-100 TPS)**: 1000-2000ms
- 低流量 **(<10 TPS)**: 2000-5000ms

权衡：

- 较低的值 = 更快的消息提取，更多的 API 负载
- 较高的值 = 较慢的提取，较少的 API 负载

enquire_link_interval

SMPP 保持连接间隔（秒）

类型	必需	默认	范围
整数	否	60	0-3600

目的：多长时间（以秒为单位）发送 SMPP enquire_link PDU 以验证连接是否仍然有效。远程服务器以 enquire_link_resp 响应。

指南：

- 默认（**60**）：适合大多数运营商
- 较低值（**15-30**）：更快的故障检测，更多流量
- 较高值（**120-300**）：较少的开销，较慢的故障检测
- **0**：完全禁用 enquire_link（不推荐）

示例：

```
enquire_link_interval: 60    # 标准(1分钟)
enquire_link_interval: 30    # 激进的保持连接
enquire_link_interval: 0     # 禁用
```

system_type

SMPP 系统类型标识符

类型	必需	默认	示例
字符串	否	`""`	`"OTP"`

目的：在绑定期间发送的 SMPP 协议字段。一些运营商要求特定值。除非运营商指定，否则保持为空。

addr_ton

号码类型的地址

类型	必需	默认	范围
整数	否	0	0-6

目的：在绑定请求中指定的 SMPP 协议字段，指示使用的号码类型。

常见值：

- `0` - 未知
- `1` - 国际
- `2` - 国内
- `5` - 字母数字

根据运营商的要求进行设置。

addr_npi

地址编号计划指示符

类型	必需	默认	范围
整数	否	0	0-18

目的：在绑定请求中指定的 SMPP 协议字段，指示编号计划。

常见值：

- `0` - 未知
- `1` - ISDN/E.164
- `3` - 数据/X.121
- `9` - 私有

根据运营商的要求进行设置。

address_range

绑定的地址范围

类型	必需	默认	示例
字符串	否	`""`	`"614*"`

目的：在绑定中指定的 SMPP ��议字段，指示此绑定处理的地址范围。某些运营商使用此字段来过滤哪些消息被传递到此连接。除非运营商指定值，否则保持为空。

enabled

对等体启用状态

类型	必需	默认
布尔值	否	`true`

目的：控制此对等体是否处于活动状态。禁用的对等体保留在配置中，但不建立连接。对于暂时将连接下线而不删除其配置非常有用。

cache_enabled

启用本地消息缓存

类型	必需	默认
布尔值	否	`false`

目的：启用时，如果后端 API 无法访问，则本地缓存传入消息，随后在恢复连接时自动交付。有关详细信息，请参见 MESSAGE_CACHE.md。

cache_max_size

最大缓存消息数

类型	必需	默认	范围
整数	否	10000	1-1000000

目的：每个绑定的最大缓存消息数。当达到限制时，最旧的消息将被驱逐（FIFO）。仅在 `cache_enabled` 为 `true` 时适用。

cache_retry_interval

基本重试间隔（秒）

类型	必需	默认
整数	否	60

目的：在重试交付缓存消息之前���基本间隔（以秒为单位）。结合指数退避（重试 0：60s，重试 1：120s，重试 2：240s，等等）。仅在 `cache_enabled` 为 `true` 时适用。

Web UI 示例:

SMPP 服务器绑定□置

服务器绑定定义 入站连接，其中□关充当 **SMSC**（服务器），接受来自外部 **ESME**（客户端）的连接。在此模式下，合作伙伴系统连接到□关以发送和接收消息。

完整服务器绑定示例

```
config :omnimessage_smpp, :server_binds, [
  %{
    # 此客户端的唯一标识符
    name: "partner_acme",

    # 客户端预期的凭据
    system_id: "acme_corp",
    password: "acme_secret",

    # 允许的绑定类型
    allowed_bind_types: [:transmitter, :receiver, :transceiver],

    # IP 限制
    ip_whitelist: ["192.168.1.0/24", "10.50.1.100"],

    # 源地址限制（空 = 允许所有）
    source_address_whitelist: [],

    # 限速
    tps_limit: 50,

    # 队列检查频率
    queue_check_frequency: 1000,

    # 保持连接间隔（秒，0 表示禁用）
    enquire_link_interval: 60,

    # 消息缓存（可选）
    cache_enabled: false,
    cache_max_size: 10000,
    cache_retry_interval: 60,

    # 将交付回执（DLR）返回到此绑定（可选）
    deliver_receipts: false
  }
]
```

服务器绑定参数

name

客户端标识符

类型	必需	示例
字符串	是	`"partner_acme"`

目的：标识连接到您的外部客户端。

命名约定：使用合作伙伴/客户端名称以便于识别。

system_id

客户端预期的用户名

类型	必需	示例
字符串	是	`"acme_corp"`

目的：外部客户端必须提供的用户名以进行身份验证。

提供给客户端：与您的合作伙伴共享此凭据。

password

客户端预期的密码

类型	必需	示例
字符串	是	`"secure_password"`

目的：外部客户端必须提供的密码以进行身份验证。

安全性：

- 使用强密码
- 每个客户端唯一
- 安全地与合作伙伴共享

allowed_bind_types

允许的会话类型

类型	必需	默认
原子列表	是	-

目的：限制客户端可以使用的绑定类型。

选项：

```
allowed_bind_types: [:transceiver]  # 仅限双向
allowed_bind_types: [:transmitter, :receiver]  # 发送或接收
allowed_bind_types: [:transmitter, :receiver, :transceiver]  # 任何
```

建议：除非需要限制，否则允许所有三种类型。

ip_whitelist

允许的客户端 IP 地址

类型	必需	默认	格式
字符串列表	是	`[]`	IP 或 CIDR 表示法

目的：安全性 - 仅允许来自已知 IP 的连接。

格式：

- 单个 IP: `"192.168.1.100"` (自动 `/32`)
- CIDR 子网: `"192.168.1.0/24"`, `"10.0.0.0/8"`
- 混合: `["192.168.1.0/24", "10.50.1.100"]`

示例：

```
# 允许任何 IP（不推荐）
ip_whitelist: []

# 单个 IP
ip_whitelist: ["203.0.113.50"]

# 多个 IP
ip_whitelist: ["203.0.113.50", "203.0.113.51"]

# 子网
ip_whitelist: ["192.168.1.0/24"]

# 混合
ip_whitelist: ["192.168.1.0/24", "10.50.1.100", "10.60.0.0/16"]
```

常见子网：

- `/32` - 单个 IP（自动适用于没有掩码的 IP）
- `/24` - 256 个地址（例如，192.168.1.0-255）
- `/16` - 65,536 个地址（例如，10.50.0.0-255.255）
- `/8` - 16,777,216 个地址（例如，10.0.0.0-255.255.255.255）

source_address_whitelist

允许的来源地址

类型	必需	默认	格式
字符串列表	否	`[]`	正则表达式模式

目的：限制连接的客户端在提交消息时可以使用的来源地址（发送者 ID）。空列表允许所有地址。每个条目是一个正则表达式模式。

示例：

```
# 允许任何来源地址
source_address_whitelist: []

# 精确匹配（锚定）
source_address_whitelist: ["^MyBrand$", "^AlertService$"]

# 前缀匹配
source_address_whitelist: ["^614", "^\+61"]

# 带字符类的数字范围
source_address_whitelist: ["^61[45]\d{8}$"]

# 混合
source_address_whitelist: ["^MyBrand$", "^614", "^\+61400000001$"]
```

具有不允许的来源地址的消息将被拒绝，返回 `ESME_RINVSRCADR`。有关详细信息，请参见 SOURCE_ADDRESS_WHITELIST.md。

tps_limit

每秒消息限制

与客户端绑定的 `tps_limit` 相同 - 控制向连接客户端的 outbound deliver_sm 速率。

queue_check_frequency

队列轮询间隔

与客户端绑定的 `queue_check_frequency` 相同 - 检查要交付给此客户端的消息的频率。

enquire_link_interval

SMPP 保持活动间隔（秒）

与客户端绑定的 `enquire_link_interval` 相同。控制服务器多长时间向连接的客户端发送 enquire_link PDU，以验证它们是否仍然在线。

enabled

对等体启用状态

与客户端绑定的 `enabled` 相同。禁用的服务器对等体不接受入站连接。

cache_enabled

启用本地消息缓存

与客户端绑定的 `cache_enabled` 相同。请参见 MESSAGE_CACHE.md。

cache_max_size

最大缓存消息

与客户端绑定的 `cache_max_size` 相同。

cache_retry_interval

基本重试间隔（秒）

与客户端绑定的 `cache_retry_interval` 相同。

deliver_receipts

将 **SMPP** 投递回执（**DLRs**）发送到此绑定

类型	必需	默认
布尔值	否	`false`

目的：□关为每条提交的消息向此绑定发送一个 SMPP 投递回执。□关跟踪每条提交的消息，并在消息达到终端状态时发送回执。回执是一个 `deliver_sm` PDU，`esm_class` 设置为 `0x04`（SMSC 投递回执），根据 SMPP v3.4 section 5.2.12。

何时使用：对于需要每条消息的投递确认的专用互联绑定，设置 `deliver_receipts: true`。对于不需要回执的绑定，保持默认的 `false`。

要求：绑定必须以 `cache_enabled: false` 运行。回执使用□关在 `submit_sm_resp` 中返回的消息 ID。缓存返回一个临时 ID，与核心 ID 不匹配，因此缓存绑定会破坏回执关联。如果绑定设置了 `deliver_receipts: true` 和 `cache_enabled: true`，□关在启动时会写入警告，并且回执 ID 不匹配。

工作原理：提交绑定发送一个 `submit_sm` °□关接受消息并返回一个带有消息 ID 的 `submit_sm_resp` °□关跟踪消息并轮询 OmniMessage 核心以获取其状态。当消息被投递、过期或被拒绝时，□关向绑定发送一个 `deliver_sm` 回执。回执携带相同的消息 ID 和状态字段（`stat:DELIVRD`、`stat:EXPIRED`、`stat:UNDELIV` 或 `stat:REJECTD`）。

跟踪器仅在内存中保持状态。如果□关重启，当前正在传输的消息的回执将丢失。这是一种尽力而为的机制。

状态与回执映射：

核心状态	回执 `stat`
`delivered`	`DELIVRD`
`expired`	`EXPIRED`
`dropped`	`UNDELIV`
`rejected`	`REJECTD`
`balance_rejected`	`REJECTD`

有关端到端消息流的详细信息，请参见 USAGE.md，有关相关指标的信息，请参见 MONITORING.md。

Web UI 示例：

服务器监听□置

配置服务器绑定后，□关监听传入连接。

完整监听示例

```
config :omnimessage_smpp, :listen, %{
  host: "0.0.0.0",
  port: 2775,
  max_connections: 100
}
```

监听参数

host

要绑定的 **IP** 地址

类型	必需	默认	常见值
字符串	否	`"0.0.0.0"`	`"0.0.0.0"`，`"127.0.0.1"`

目的：监听哪个□络接口。

值：

- `"0.0.0.0"` - 在所有接口上监听（推荐）
- `"127.0.0.1"` - 仅在本地主机上监听（测试）
- `"192.168.1.10"` - 在特定 IP 上监听

port

要监听的 **TCP** 端口

类型	必需	默认	范围
整数	否	2775	1-65535

目的：用于传入 SMPP 连接的端口。

标准：2775

max_connections

最大并发连接数

类型	必需	默认	范围
整数	否	100	1-10000

目的：限制同时客户端连接的总数。

指南：

- 根据预期的客户端设置
- 更高的值使用更多内存
- 典型：10-100 个连接

完整配置示例

示例 1：单个运营商连接

```
import Config

config :omnimessage_smpp,
  api_base_url: "https://smsc.company.com:8443",
  verify_ssl_peer: true,
  smsc_name: "smpp_prod"

config :omnimessage_smpp, :binds, [
  %{
    name: "att_primary",
    mode: :client,
    bind_type: :transceiver,
    host: "smpp.att.com",
    port: 2775,
    system_id: "company_user",
    password: "secure_pass_123",
    tps_limit: 100,
    queue_check_frequency: 1000
  }
]
```

□□ 2□□□□□□

```
import Config

config :omnimessage_smpp,
  api_base_url: "https://smsc.company.com:8443"

config :omnimessage_smpp, :binds, [
  # □□
  %{
    name: "att_us",
    mode: :client,
    bind_type: :transceiver,
    host: "smpp.att.com",
    port: 2775,
    system_id: "att_username",
    password: "att_password",
    tps_limit: 100,
    queue_check_frequency: 1000
  },

  # □□
  %{
    name: "vodafone_uk",
    mode: :client,
    bind_type: :transceiver,
    host: "smpp.vodafone.co.uk",
    port: 2775,
    system_id: "voda_username",
    password: "voda_password",
    tps_limit: 50,
    queue_check_frequency: 1000
  }
]
```

示例 **3**：带服务器绑定的网关

```
import Config

config :omnimessage_smpp,
  api_base_url: "https://smsc.company.com:8443"

# 出站连接
config :omnimessage_smpp, :binds, [
  %{
    name: "upstream_carrier",
    mode: :client,
    bind_type: :transceiver,
    host: "smpp.carrier.com",
    port: 2775,
    system_id: "my_username",
    password: "my_password",
    tps_limit: 100,
    queue_check_frequency: 1000
  }
]

# 入站客户端定义
config :omnimessage_smpp, :server_binds, [
  %{
    name: "partner_alpha",
    system_id: "alpha_corp",
    password: "alpha_secret",
    allowed_bind_types: [:transmitter, :receiver, :transceiver],
    ip_whitelist: ["203.0.113.0/24"],
    tps_limit: 50,
    queue_check_frequency: 1000
  },
  %{
    name: "partner_beta",
    system_id: "beta_inc",
    password: "beta_password",
    allowed_bind_types: [:transceiver],
    ip_whitelist: ["198.51.100.50"],
    tps_limit: 25,
    queue_check_frequency: 2000
  }
]
```

```
# 服务器监听
config :omnimessage_smpp, :listen, %{
  host: "0.0.0.0",
  port: 2775,
  max_connections: 100
}
```

示例 4：带源绑定的共享队列

两个连接到同一运营商，两个连接都从一个共享队列（`sinch.smpp`）中排出，以实现负载共享和冗余。运营商通过源 IP 和 端口进行白名单，因此每个绑定在同一源 IP 上固定一个不同的本地源端口。

```
import Config

config :omnimessage_smpp,
  api_base_url: "https://smsc.company.com:8443"

config :omnimessage_smpp, :binds, [
  %{
    name: "sinch_1",
    mode: :client,
    bind_type: :transceiver,
    host: "smpp.sinch.com",
    port: 2775,
    system_id: "sinch_user",
    password: "sinch_pass",
    tps_limit: 100,
    queue_check_frequency: 1000,

    # 两个绑定都从同一后端队列中排出
    queue: "sinch.smpp",

    # 固定的源地址/端口（运营商通过两者进行白名单）
    bind_source_ip: "10.20.30.40",
    bind_source_port: 5001
  },
  %{
    name: "sinch_2",
    mode: :client,
    bind_type: :transceiver,
    host: "smpp.sinch.com",
    port: 2775,
    system_id: "sinch_user",
    password: "sinch_pass",
    tps_limit: 100,
    queue_check_frequency: 1000,

    # 与 sinch_1 相同的共享队列
    queue: "sinch.smpp",

    # 相同的源 IP，不同的源端口
    bind_source_ip: "10.20.30.40",
    bind_source_port: 5002
```

```
  }
]
```

工作原理：两个绑定注册一个名为 `sinch.smpp` 的前端（它们共享的 路由键）。具有 `dest_smsc: "sinch.smpp"` 的消息由第一个获取它们的绑定排出，从而在两个连接之间分散负载。如果一个连接掉线，另一个连接将继续排出队列。每个连接向运营商提供固定的源 IP 和端口以进行白名单；`SO_REUSEADDR` 允许固定端口在重新连接时立即重新绑定。

有关参数详细信息，请参见 `queue`、`bind_source_ip` 和 `bind_source_port`。

□置验证

编辑配置后，请在重启之前验证：

语法检查

```
# 检查 Elixir 语法
/opt/omnimessage-smpp/bin/omnimessage-smpp eval "File.read!
('config/runtime.exs')"
```

如果语法无效，您将看到错误。在重启之前修复。

测试□置

```
# 在前台重启以查看错误
sudo -u omnimessage-smpp /opt/omnimessage-smpp/bin/omnimessage-
smpp console
```

按 `Ctrl+C` 两次退出。

环境变量

所有全局设置都可以通过环境变量覆盖。在启动□关之前，在您的 systemd 单元文件或 shell 环境中设置这些。

环境变量	□���键	默
`API_BASE_URL`	`api_base_url`	`"https://10.17`
`SMSC_NAME`	`smsc_name`	`"smpp_gateway"`
`SMPP_POLL_INTERVAL`	`smpp_poll_interval`	`100`
`VERIFY_SSL_PEER`	`verify_ssl_peer`	`false`
`CACHE_FLUSH_INTERVAL`	`cache_flush_interval`	`10000`
`CACHE_MAX_RETRY_ATTEMPTS`	`cache_max_retry_attempts`	`10`
`CACHE_BACKOFF_MULTIPLIER`	`cache_backoff_multiplier`	`2`
`MNESIA_STORAGE_TYPE`	`mnesia_storage_type`	`disc_copies`

示例 **systemd** 覆盖:

```
sudo systemctl edit omnimessage-smpp
```

```
[Service]
Environment="API_BASE_URL=https://omnimessage-core.company.com:8443"
Environment="SMSC_NAME=smpp_prod_01"
Environment="VERIFY_SSL_PEER=true"
```

安全最佳实践

1. 保护□置文件:

```
sudo chmod 600 /opt/omnimessage-smpp/config/runtime.exs
sudo chown omnimessage-smpp:omnimessage-smpp /opt/omnimessage-
smpp/config/runtime.exs
```

2. 使用强密码:
   - 至少 12 个字符
   - 混合字母、数字、符号
   - 每个连接唯一
3. 使用 **IP** 白名单:
   - 始终为服务器绑定配置 `ip_whitelist`
   - 在生产中绝不要使用空列表 `[]`
4. 启用 **SSL** 验证:
   - 设置 `verify_ssl_peer: true` 并使用有效证书
5. 定期更换凭据:
   - 每季度更改密码
   - 与运营商/合作伙伴协调

下一步

- 查看 MONITORING.md 以获取指标配置
- 阅读 USAGE.md 以管理连接
- 查看 TROUBLESHOOTING.md 以获取常见问题
- 返回 README.md 以获取概述

术语表

术语和定义

A

API（应用程序编程接口**）**用于与消息队列后端系统进行通信的接口。

自动滚动 Web UI 日志选项卡中的功能，自动滚动以显示最新的日志条目。

B

后端 SMPP □关连接以检索和存储消息的消息队列系统。

绑定 两个系统之间的 SMPP 连接。可以是发射器、接收器或收发器。

绑定类型 SMPP 会话的类型：

- 发射器：仅发送消息
- 接收器：仅接收消息
- 收发器：发送和接收消息

绑定失败 SMPP 身份验证尝试失败，通常是由于凭据不正确或 IP 限制。

C

CIDR（无类域间路由**）**指定 IP 地址范围的表示法（例如，`192.168.1.0/24` 表示 256 个 IP 地址）。

客户端绑定 出站 SMPP 连接，其中□关充当 **ESME** 连接到外部 **SMSC**（通常是运营商的 SMPP 服务器）。在此模式下，□关是客户端。

连接状态 SMPP 绑定的当前状态：

- 已连接：活动且正常运行
- 未连接：未连接

- 重新连接：正在尝试建立连接

计数器 仅增加的指标（在服务重启时重置），用于发送消息等总数。

D

数据编码 SMPP 字段，指定消息字符编码（GSM-7、UCS-2 等）。

Deliver_SM 由 SMSC（服务器）发送的 SMPP PDU，用于将消息传递给连接的 ESME（客户端）。由服务器绑定用于将消息推送到连接的合作伙伴。

投递失败 当消息无法投递时，由运营商的错误响应指示。

投递回执 **(DLR)** 运营商关于消息投递状态的确认。

deliver_time 消息队列中的字段，保存消息投递的时间戳。由 OmniMessage Core 设置，当□关报告成功投递时。该字段名称是后端在标记消息为已投递时所期望的。

dest_smsc 消息队列中的字段，指示哪个 SMPP 连接应处理该消息。其值必须等于目标对等方注册的 路由键（见 前端注册）。

断开连接 当活动的 SMPP 连接被终止时，无论是故意还是由于错误。

E

查询链接 定期发送的 SMPP 保持活动消息，以验证连接是否处于活动状态。

ESM 类 SMPP 字段，指示消息类型和特性。

ESME (外部短消息实体**)** 在 SMPP 术语中，连接到 SMSC 以发送或接收消息的客户端应用程序。当□关以 客户端模式 运行时，它充当连接到运营商 SMSC 的 ESME。当它以 服务器模式 运行时，它接受来自外部 ESME 的连接。

指数退避 重试策略，在每次失败后等待时间加倍（1分钟、2分钟、4分钟、8分钟……）。

F

防火墙 控制进出□络流量的□络安全系统。

前端注册 SMPP 关向 OmniMessage Core 注册的过程。每 60 秒发送一次心跳以保持注册有效。如果关停止，注册将在 90 秒后过期，OmniMessage 将停止向其路由消息。每个对等方都在其 路由键 下注册，因此注册名称等于后端路由的 `dest_smsc`：客户端对等方在其 `queue`（或裸对等方名称）下，服务器绑定在其绑定名称下注册。

G

网关 连接消息队列和移动络的 SMPP 关应用程序。

仪表 可以增加或减少的指标，表示当前值（例如，连接状态）。

Grafana 用于在仪表板中显示 Prometheus 指标的流行可视化工具。

GSM-7 用于 SMS 的标准 7 位字符编码，每条消息支持最多 160 个字符。

H

HTTP/HTTPS 用于 Web 通信的协议。HTTPS 是加密版本。

I

IP 白名单 可以连接到关的允许 IP 地址列表（安全功能）。

ISDN（综合业务数字网**）** 通常用于电话号码的编号计划。

J

（□□□）

K

保持活动 定期发送的消息（查询链接），以维护连接并检测故障。

KPI（关键绩效指标**）** 指示系统性能的可测量值（例如，投递成功率）。

L

标签 在 Prometheus 中，附加到指标的键值对，用于标识（例如，`bind_name="vodafone_uk"`）。

实时视图 用于实时 Web UI 更新的 Phoenix 框架技术。

M

消息队列 存储等待发送或接收的消息的后端系统。

指标 系统性能的定量测量，以 Prometheus 格式公开。

MO（移动发起） 从移动电话发送到□关的消息（入站）。

MT（移动终止） 从□关发送到移动电话的消息（出站）。

MSISDN（移动台国际用户目录号码） 移动电话号码的标准格式。

N

NPI（编号计划指示符） SMPP 字段，指定编号方案（例如，ISDN）。

O

出站 从□关流向移动□络的消息。

入站 从移动□络流向□关的消息。

P

PDU（协议数据单元） 单个 SMPP 消息数据包（例如，submit_sm、deliver_sm）。

Prometheus 开源监控系统，收集和存储时间序列指标。

Q

队列 等待处理或发送的消息列表。

队列检查频率 □关多长时间（以毫秒为单位）轮询后端以获取新消息。

队列工作者 从队列中检索消息并通过 SMPP 发送的组件。

R

速率限制 控制消息吞吐量以符合运营商限制。见 TPS。

接收器 仅接收消息的 SMPP 绑定类型（deliver_sm）。

重新连接 重新建立断开的 SMPP 连接。

重试 再次尝试发送失败的消息，通常采用指数退避。

路由键 对等方在 OmniMessage Core 中注册的名称，因此 `dest_smsc` 值路由消息到它。对于客户端对等方，这是其配置的 `queue`（或在未设置 `queue` 时的裸对等方名称）；对于服务器绑定，它是绑定名称。

S

序列号 分配给每个 SMPP PDU 的唯一数字标识符。用于将请求与其响应匹配（例如，将 submit_sm 与其 submit_sm_resp 匹配）。

服务器绑定 允许外部 **ESMEs**（客户端）连接到□关的配置。在此模式下，□关充当接受来自合作伙伴系统的入站连接的 **SMSC**（服务器）。

会话 两个系统之间的活动 SMPP 连接。

共享队列 由多个客户端绑定排空的单个后端队列。绑定通过在其配置中设��相同的 `queue` 值来选择加入（例如，两个运营商连接都排空 `"sinch.smpp"`），允许跨连接的负载共享和冗余。默认为绑定自己的名称，在这种情况下，每个绑定都有自己的队列。

短代码 用于应用程序与个人消息传递的缩短 SMS 发起/目的地址（通常为 5-6 位数字）。五位数的 `deliver_sm` 目的地以 TON 未知 发送，以便运营商不会将其误解为国际号码。

源绑定 通过 `bind_source_ip` 和 `bind_source_port` 参数固定出站（客户端）连接的本地源 IP 地址和/或 TCP 端口。当运营商通过源 IP 和端口将连接列入白名单时使用。见 CONFIGURATION.md。

source_smsc 消息队列中的字段，指示哪个服务器绑定应通过 deliver_sm 将消息传递给其连接的客户端。

SMPP（短消息对等网络**）**用于在系统之间交换 SMS 消息的行业标准协议。

SMSC（短消息服务中心**）**在 SMPP 术语中，接受来自 ESME（客户端）的连接并处理 SMS 消息路由和投递的服务器组件。当 关以 服务器模式 运行时，它充当接受来自外部 ESME 的连接的 SMSC。

SSL/TLS 用于安全通信的加密协议。

Submit_SM 用于��交消息进行投递的 SMPP PDU。

Submit_SM_Resp 对 submit_sm 的 SMPP 响应，指示成功或失败。

系统 **ID** 用于 SMPP 身份验证的用户名。

T

遥测 系统指标的自动收集和传输。

TON（号码类型**）**SMPP 字段，指定号码格式（例如，国际、国家）。

TPS（每秒事务数**）**通过连接的最大每秒消息速率限制。

收发器 可以同时发送和接收消息的 SMPP 绑定类型（最常见）。

发射器 仅发送消息的 SMPP 绑定类型（submit_sm）。

吞吐量 消息处理速率，通常以每秒消息数来衡量。

U

UCS-2 用于 SMS 的 16 位 Unicode 字符编码，每条消息支持最多 70 个字符。

正常运行时间 连接或服务持续运行的时间。

V

有效期 在过期之前尝试消息投递的时间限制。

W

Web 仪表板 用于监控和管理□关的基于浏览器的用户界面。

白名单 见 IP 白名单和源地址白名单。

X

(□□□)

Y

(□□□)

Z

(□□□)

缩略语快速参考

缩略语	全称
API	应用程序编程接口
CIDR	无类域间路由
DLR	投递回执
ESME	外部短消息实体
GSM	全球移动通信系统
HTTP	超文本传输协议
HTTPS	安全超文本传输协议
IP	互联□协议
ISDN	综合业务数字□
KPI	关键绩效指标
MO	移动发起
MSISDN	移动台国际用户目录号码
MT	移动终止
NPI	编号计划指示符
PDU	协议数据单元
SMPP	短消息对等□络
SMSC	短消息服务中心

缩略语	全称
SMS	短消息服务
SSL	安全套接层
TLS	传输层安全
TON	号码类型
TPS	每秒事务数
UCS	通用编码字符集
UI	用户界面
URL	统一资源定位符

相关文档

- **README.md** - 系统概述和入门
- **CONFIGURATION.md** - 配置参数说明
- **USAGE.md** - 日常操作
- **MONITORING.md** - 指标和监控
- **TROUBLESHOOTING.md** - 问题解决

SMPP 消息缓存

概述

SMPP 消息缓存是一个本地持久性层，允许 SMPP 网关在后端 API 不可用时继续接受传入消息。消息会在 Mnesia 中本地缓存，并在连接恢复时自动交付给 API，使用智能重试逻辑和指数退避。

特性

- 弹性的消息接受 - 在 API 中断期间继续接受 SMPP 消息
- 持久存储 - 使用 Mnesia 和 `:disc_copies` 以确保重启后的耐久性
- 自动重试 - 后台工作者自动尝试以指数退避进行交付
- 每个绑定的配置 - 独立启用/禁用每个 SMPP 绑定的缓存
- 溢出保护 - 当缓存达到配置的大小限制时，采用 FIFO 驱逐
- 失败消息保留 - 永久失败的消息保留以供手动审核
- 实时监控 - 带有缓存统计和指标的 LiveView 仪表板
- **Prometheus** 指标 - 完整的指标导出用于监控和警报

架构

消息流

启用缓存时

```
SMPP Client → submit_sm → Gateway Server
                                ↓
                        Cache in Mnesia (immediate response)
                                ↓
                        CacheFlushWorker (background)
                                ↓
                          Backend API
```

禁用缓存时

```
SMPP Client → submit_sm → Gateway Server
                                ↓
                          Backend API (direct)
```

组件

1. **MessageCache** 模块
   - 核心缓存逻辑
   - 溢出处理
   - 用于 LiveView 和工作者的查询函数
2. **CacheFlushWorker**
   - 每个启用缓存的绑定的 GenServer
   - 轮询准备重试的消息
   - 实现指数退避
   - 标记永久失败的消息
3. **Mnesia** 表 (`:smpp_message_cache`)
   - 使用 `:disc_copies` 的持久存储
   - 按 bind_name、next_retry_at 和 status 索引
   - 在应用重启后仍然存在

□置

全局设置

编辑 `config/runtime.exs`:

```
config :omnimessage_smpp,
  # 刷新工作者轮询消息的频率（毫秒）
  cache_flush_interval: 10_000,

  # 标记为失败的最大重试次数
  cache_max_retry_attempts: 10,

  # 指数退避乘数
  cache_backoff_multiplier: 2,

  # Mnesia 存储类型 (:disc_copies 或 :ram_copies)
  mnesia_storage_type: :disc_copies
```

每个绑定的□置

每个 SMPP 绑定（客户端或服务器）可以独立配置：

```
config :omnimessage_smpp, :binds, [
  %{
    name: "my_smpp_bind",
    mode: :server,
    system_id: "username",
    password: "password",

    # 缓存配置
    cache_enabled: true,          # 启用缓存（默认：false）
    cache_max_size: 10_000,       # 最大缓存消息数（默认：10,000）
    cache_retry_interval: 60      # 基础重试间隔（秒）（默认：60）
  }
]
```

环境变量

```
# □局缓存设置
CACHE_FLUSH_INTERVAL=10000                  # 刷新轮询间隔（毫秒）
CACHE_MAX_RETRY_ATTEMPTS=10                 # 永久失败前的最大重试次数
CACHE_BACKOFF_MULTIPLIER=2                  # 指数退避乘数
MNESIA_STORAGE_TYPE=disc_copies             # Mnesia 存储类型
```

重试行为

指数退避

当消息交付失败时，重试间隔在每次尝试时加倍：

```
基础间隔：60 秒
退避乘数：2

重试 0：60s   (1 分钟)
重试 1：120s  (2 分钟)
重试 2：240s  (4 分钟)
重试 3：480s  (8 分钟)
重试 4：960s  (16 分钟)
重试 5：1920s (32 分钟)
...
重试 9：30,720s (8.5 小时)
```

永久失败

在 10 次失败尝试后（默认），消息被标记为 `failed_permanent` 并且：

- 保留在缓存中以供手动审核
- 停止自动重试
- 出现在缓存仪表板的“永久失败”部分
- 可以���动重试或清除

状态转换

```
:pending → :delivering → SUCCESS (从缓存中删除)
                       → FAILURE → :pending (重试并退避)
                                 → :failed_permanent (在最大重试后)
```

监控

LiveView 仪表板

访问缓存仪表板 `http://your-server:4000/smpp` → “消息缓存”选项卡

摘要卡片：

- 总缓存 - 当前缓存中的所有消息
- 待交付 - 等待重试的消息
- 永久失败 - 超过最大重试次数的消息

每个绑定表格：

- 绑定名称
- 缓存消息计数
- 待处理 / 失败细分
- 最大缓存大小
- 利用率百分比（带有可视进度条）

Prometheus 指标

```
# 每个绑定的当前缓存大小
smpp_cache_size{bind_name="my_bind",mode="server"} 42

# 成功交付的消息总数
smpp_cache_delivered_total{bind_name="my_bind"} 1234

# 总重试尝试次数
smpp_cache_retry_total{bind_name="my_bind"} 56

# 总永久失败次数
smpp_cache_permanent_failures_total{bind_name="my_bind"} 2

# 总溢出事件（消息丢弃）
smpp_cache_overflow_total{bind_name="my_bind"} 0
```

日志消息

```
INFO  Message -123456789 cached for my_smpp_bind
INFO  Successfully delivered cached message -123456789, API ID:
99999
WARN  Failed to deliver message -123456789 (retry 3/10), next
retry at 2026-02-01 12:34:56Z
ERROR Message -123456789 exceeded max retries (10), marking as
failed_permanent
WARN  Cache overflow for my_smpp_bind: deleted oldest message
```

操作

启用/禁用缓存

通过 **LiveView UI**

1. 导航到 `http://your-server:4000/smpp`
2. 转到“客户端对等”或“服务器对等”选项卡
3. 编辑绑定
4. 切换“启用缓存”复选框
5. 保存更改

通过 **IEx** 控制台

```
# 为绑定启用缓存
SmsC.SMPPConfig.update_server_peer("my_bind", "username",
"password",
  cache_enabled: true,
  cache_max_size: 10_000,
  cache_retry_interval: 60
)

# 禁用缓存
SmsC.SMPPConfig.update_server_peer("my_bind", "username",
"password",
  cache_enabled: false
)
```

监控缓存状态

```
# 获取□局摘要
SmsC.SMPP.MessageCache.get_cache_summary()
# => %{total_cached: 42, pending: 40, failed: 2}

# 获取每个绑定的细分
SmsC.SMPP.MessageCache.get_cache_by_bind()
# => [
#   %{bind_name: "bind1", total: 30, pending: 28, failed: 2,
max_size: 10_000},
#   %{bind_name: "bind2", total: 12, pending: 12, failed: 0,
max_size: 10_000}
# ]

# 计算特定绑定的消息
SmsC.SMPP.MessageCache.count_cached_messages("my_bind")
# => 42
```

手动干预

清除���败的消息

```
# 获取绑定的所有失败消息
{:atomic, failed_messages} = :mnesia.transaction(fn ->
  :mnesia.match_object({:smpp_message_cache, :_, "my_bind", :_,
:_, :_, :_, :_, :_, :failed_permanent})
end)

# 删除它们
Enum.each(failed_messages, fn {_, {bind_name, msg_id}, _, _, _, _,
_, _, _, _} ->
  SmsC.SMPP.MessageCache.delete_cache_record(bind_name, msg_id)
end)
```

强制重试失败的消息

```
# 将一个 failed_permanent 消息重置为 pending
SmsC.SMPP.MessageCache.update_cache_record("my_bind", -123456, %{
  status: :pending,
  retry_count: 0,
  next_retry_at: DateTime.utc_now(),
  last_error: nil
})
```

故障排除

缓存已满 / 溢出事件

症状：`cache_overflow_total` 指标增加，最旧的消息被丢弃

原因：达到缓存大小限制

解决方案：

1. 增加绑定的 `cache_max_size`
2. 调查 API 交付失败的原因（检查 API 日志、□络）
3. 手动清除旧的失败消息
4. 检查刷新间隔是否过慢

消息未被交付

症状：消息卡在 `:pending` 状态

可能原因：

1. **API** 已关闭
   - 检查 API 可用性
   - 验证后端 API 日志
   - 检查网络连接
2. **next_retry_at** 在未来
   - 消息将在达到 `next_retry_at` ��重试
   - 检查指数退避计划
3. 刷新工作者未运行

```
# 检查工作者是否在运行
Supervisor.which_children(SmsC.SMPP.Supervisor)
```

4. 缓存已禁用
   - 验证绑定配置中的 `cache_enabled: true`

高重试计数

症状：许多消息具有高 `retry_count` 值

调查：

```
# 查找具有高重试计数的消息
{:atomic, messages} = :mnesia.transaction(fn ->
  :mnesia.match_object({:smpp_message_cache, :_, "my_bind", :_,
:_, :_, :_, :_, :_, :_})
end)

high_retry = Enum.filter(messages, fn {_, _, _, _, _, _,
retry_count, _, _, _} ->
  retry_count >= 5
end)

# 检查每个消息的 last_error
Enum.each(high_retry, fn {_, _, _, msg_id, _, _, retry_count, _,
last_error, _} ->
  IO.puts("Message #{msg_id}: #{retry_count} retries, error: #
{inspect(last_error)}")
end)
```

Mnesia 磁盘空间

症状：磁盘空间填满

检查 **Mnesia** 目录：

```
ls -lh Mnesia.*
du -sh Mnesia.*
```

清理：

1. 清除旧的失败消息（见手动干预部分）
2. 减少每个绑定的 `cache_max_size`
3. 启用缓存溢出（确保正确的 FIFO 驱逐）

性能考虑

内存使用

- 每个缓存的消息大约使用 500-1000 字节（取决于消息大小）
- 10,000 条消息 ≈ 5-10 MB 内存
- 使`:disc_copies`时，数据也会写入磁盘

CPU 使用

- 刷新工作者默认每 10 秒轮询一次（可配置）
- 批处理（每个周期 100 条消息）减少开销
- 并发交付（每个工作者最多 10 个并发 API 调用）

磁盘 I/O

- `:disc_copies`在每次事务时写入磁盘
- 对于非常高的吞吐量（>1000 条消息/秒），考虑:
  - 使用`:ram_copies`（失去持久性）
  - 增加刷新间隔
  - 横向扩展

推荐限制

场景	cache_max_size	cache_flush_interval
低流量 (<100 条消息/秒)	10,000	10,000ms
中等流量 (100-500 条消息/秒)	50,000	5,000ms
高流量 (>500 条消息/秒)	100,000	3,000ms

恢复场景

应用重启

1. Mnesia 自动从磁盘加载 `:disc_copies` 表
2. 缓存的消息保持不变
3. 刷新工作者重新启动并继续处理

数据库迁移

从不支持缓存的版本升级时：

1. 迁移会自动将缓存字段添加到现有绑定
2. 默认值：`cache_enabled: false`，`cache_max_size: 10,000`，`cache_retry_interval: 60`
3. 无数据丢失
4. 第一次运行时创建缓存表

API 中断恢复

1. 消息在中断期间累积在缓存中
2. 当 API 恢复时，刷新工作者会自动交付
3. 最旧的消息优先交付（FIFO）
4. 指数退避防止恢复期间 API 过载

最佳实践

1. 默认启用缓存 - 防止在中断期间消息丢失
2. 监控指标 - 设置对 `cache_permanent_failures_total` 和 `cache_overflow_total` 的警报
3. 适当设置大小 - 根据预期的中断持续时间设置 `cache_max_size`
4. 审核失败消息 - 定期检查失败的永久消息以发现模式
5. 测试故障转移 - 模拟 API 中断以验证缓存行为
6. 调整重试间隔 - 根据 API 恢复时间模式进行调整
7. 使用持久存储 - 在生产中保持 `mnesia_storage_type: disc_copies`

另见

- 配置参考
- 监控和指标
- 故障排除

监控和指标指南

SMPP网关监控的完整参考

概述

SMPP□关以Prometheus格式公开指标，以监控连接健康、消息吞吐量和系统性能。

关键：由于□关是无状态的，并依赖于OmniMessage Core，**OmniMessage**连接性是最重要的监控指标。监控以下两个方面：

1. **SMPP**网关指标 - 协议级健康
2. **OmniMessage API**指标 - 后端连接性和健康

指标端点

URL: `http://your-server:4000/metrics`

格式：Prometheus文本格式

访问：默认开放给localhost（配置防火墙以进行远程访问）

快速测试

```
curl http://localhost:4000/metrics
```

可用指标

所有指标以`smpp_`为前缀，并包括用于识别的标签。

许可证指标

omnimessage_smpp_license_status

类型: Gauge

描述: 当前许可证状态

值:

- `1` = 有效许可证
- `0` = 无效/过期许可证

标签: 无

示例:

```
omnimessage_smpp_license_status 1
```

用途:

- 当值为0（无效许可证）时发出警报
- 当许可证无效时，出站队列处理停止，但SMPP绑定保持连接
- Web UI保持可访问以进行故障排除

产品名称: `omnimessage_smpp`

注意:

- 当许可证无效（`license_status == 0`）时，□关停止处理出站队列
- SMPP绑定（客户端和服务器）保持连接并接受绑定请求
- 仍然接收入站消息，但不进行处理
- 无论许可证状态如何，UI和监控保持可访问

警报示例:

```
- alert: SMPP_License_Invalid
  expr: omnimessage_smpp_license_status == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "SMPP网关许可证无效或过期"
    description: "许可证状态无效 - 出站消息处理被阻止"
```

连接状态指标

smpp_connection_status

类型: Gauge

描述: 当前SMPP绑定的连接状态

值:

- `1` = 已连接。对于服务器绑定，至少有一个ESME会话已绑定。
- `0` = 未连接。对于服务器绑定，没有ESME会话已绑定。

标签:

- `bind_name` - 连接名称（例如，“vodafone_uk”）
- `mode` - 连接类型（“client”或“server”）
- `host` - 远程主机（仅客户端模式）
- `port` - 远程端口（仅客户端模式）
- `bind_type` - SMPP绑定类型（仅客户端模式）
- `system_id` - 使用的系统ID

示例:

```
smpp_connection_status{bind_name="vodafone_uk",mode="client",host="sm
1
```

用途:

- 当值为0（未连接）时发出警报
- 跟踪连接正常运行时间百分比
- 监控重新连接频率

消息计数器

smpp_messages_sent_total

类型: Counter
描述: 通过SMPP绑定发送的消息总数：在服务器绑定中为`deliver_sm`到ESME，在客户端绑定中为`submit_sm`到运营商
单位: SMPP PDUs。每个多部分消息的每个部分都算一次。

标签: 与connection_status相同

示例:

```
smpp_messages_sent_total{bind_name="vodafone_uk",mode="client",...}
150234
```

用途:

- 计算消息速率（消息/秒）
- 跟踪每日/月度量
- 比较实际吞吐量与预期吞吐量

smpp_messages_received_total

类型: Counter
描述: 通过SMPP绑定接收的消息总数：在服务器绑定中为来自ESME的`submit_sm`，在客户端绑定中为来自运营商的移动发起的`deliver_sm`。不计算交付回执。
单位: SMPP PDUs

标签: 与connection_status相同

示例:

```
smpp_messages_received_total{bind_name="partner_acme",mode="server",..
45123
```

用途：

- 监控入站消息量
- 跟踪移动发起（MO）流量
- 对意外的量变化发出警报

交付指标

smpp_delivery_failures_total

类型：Counter

描述：消息交付失败的总数：无法发送的`deliver_sm`或`submit_sm`，或以非零状态响应的消息

单位：SMPP PDUs

标签：与connection_status相同

示例：

```
smpp_delivery_failures_total{bind_name="vodafone_uk",mode="client",..
234
```

用途：

- 计算交付成功率
- 对高失败率发出警报
- 识别问题连接

成功率计算：

```
success_rate = (messages_sent - delivery_failures) / messages_sent
* 100
```

绑定操作指标

smpp_bind_success_total

类型: Counter

描述: 成功绑定操作的总数

单位: 绑定尝试

示例:

```
smpp_bind_success_total{bind_name="vodafone_uk",...} 45
```

用途:

- 跟踪绑定稳定性
- 监控身份验证成功

smpp_bind_failures_total

类型: Counter

描述: 失败绑定操作的总数

单位: 绑定尝试

示例:

```
smpp_bind_failures_total{bind_name="vodafone_uk",...} 3
```

用途:

- 对身份验证失败发出警报
- 识别凭证问题
- 跟踪运营商连接问题

连接事件指标

smpp_connection_attempts_total

类型：Counter

描述：连接尝试的总数。在服务器绑定中，已绑定的ESME会话数量。

单位：尝试

示例：

```
smpp_connection_attempts_total{bind_name="vodafone_uk",...} 48
```

用途：

- 跟踪连接流失
- 监控重新连接频率

smpp_disconnection_total

类型：Counter

描述：断开连接的总数。在服务器绑定中，已绑定的ESME会话结束的数量。

单位：断开连接

示例：

```
smpp_disconnection_total{bind_name="vodafone_uk",...} 3
```

用途：

- 对频繁断开连接发出警报
- 识别□络问题
- 跟踪连接稳定性

smpp_reconnect_attempts_total

类型：Counter

描述：在连接失败、绑定被拒绝或会话中断后，为客户端绑定计划的重新连接总数

单位：重新连接尝试

示例：

```
smpp_reconnect_attempts_total{bind_name="vodafone_uk",...} 7
```

用途：

- 发现接受连接后立即断开的运营商（在`smpp_connection_status`保持为`0`时快速上升的计数器）
- 测量绑定在中断后尝试恢复的时间

查询链接指标

smpp_enquire_link_sent_total

类型：Counter
描述：发送的查询链接PDU的总数，以��证连接的活动性
单位：PDUs

标签：与connection_status相同

示例：

```
smpp_enquire_link_sent_total{bind_name="vodafone_uk",mode="client",..
1440
```

用途：

- 跟踪保持活动的活动
- 与接收的比较以检测单向故障

smpp_enquire_link_received_total

类型：Counter
描述：从远程对等方接收的查询链接响应PDU的总数
单位：PDUs

标签：与connection_status相同

示例：

```
smpp_enquire_link_received_total{bind_name="vodafone_uk",mode="client
1438
```

用途：

- 检测无响应的对等方（发送量 >> 接收量）
- 监控连接健康状况，超出简单状态

正常运行时间指标

smpp_uptime_seconds

类型：Gauge
描述：SMPP绑定的当前正常运行时间（以秒为单位）
单位：秒

示例：

```
smpp_uptime_seconds{bind_name="vodafone_uk",...} 86400
```

用途：

- 跟踪连接稳定性
- 计算正常运行时间百分比
- 对最近的重启发出警报

消息缓存指标

这些指标在一个或多个绑定上启用消息缓存时可用。有关缓存配置的详细信息，请参见MESSAGE_CACHE.md。

smpp_cache_size

类型：Gauge
描述：每个绑定的本地缓存中当前���息的数量
单位：消息

标签：

- `bind_name` - 连接名称
- `mode` - 连接类型（“client”或“server”）

示例：

```
smpp_cache_size{bind_name="partner_acme",mode="server"} 42
```

用途：

- 监控缓存利用率
- 当接近`cache_max_size`时发出警报

smpp_cache_delivered_total

类型：Counter

描述：成功交付到后端API的缓存消息总数

单位：消息

示例：

```
smpp_cache_delivered_total{bind_name="partner_acme"} 1234
```

smpp_cache_retry_total

类型：Counter

描述：对缓存消息的重试尝试总数

单位：尝试

示例：

```
smpp_cache_retry_total{bind_name="partner_acme"} 56
```

smpp_cache_permanent_failures_total

类型：Counter

描述：超过最大重试次数并被标记为永久失败的消息总数

单位：消息

示例：

```
smpp_cache_permanent_failures_total{bind_name="partner_acme"} 2
```

用途：

- 当> 0时发出警报（需要手动审核）

smpp_cache_overflow_total

类型：Counter

描述：缓存溢出事件的总数，其中最旧的消息被驱逐以腾出空间

单位：事件

示例：

```
smpp_cache_overflow_total{bind_name="partner_acme"} 0
```

用途：

- 当增加时发出警报（缓存太小或API中断时间过长）

交付回执（DLR）指标

这些指标适用于设置了`deliver_receipts: true`的服务器绑定。有关设置的详细信息，请参见CONFIGURATION.md以及USAGE.md中的消息流。

smpp_dlr_tracked_total

类型：Counter

描述：注册以进行交付回执跟踪的提交消息总数

单位：消息

标签：

- `bind_name` - 连接名称
- `mode` - 连接类型（“client”或“server”）

示例：

```
smpp_dlr_tracked_total{bind_name="masorange",mode="server"} 10234
```

smpp_dlr_sent_total

类型：Counter

描述：发送回提交绑定的交付回执的总数

单位：回执

示例：

```
smpp_dlr_sent_total{bind_name="masorange",mode="server"} 10180
```

用途：

- 与`smpp_dlr_tracked_total`进行比较，以查看回执完成率

smpp_dlr_dropped_total

类型：Counter

描述：在没有回执的情况下丢弃的跟踪消息总数。当核心不再拥有消息，或消息在24小时内未达到终端状态时，□关将丢弃消息。

单位：消息

示例：

```
smpp_dlr_dropped_total{bind_name="masorange",mode="server"} 12
```

用途：

- 当增加时发出警报（消息从未达到终端状态，或核心��失它们）

smpp_dlr_pending

类型：Gauge

描述：当前等待终端状态和回执的在途消息数量

单位：消息

示例：

```
smpp_dlr_pending{bind_name="masorange",mode="server"} 37
```

用途：

- 监控跟踪器积压
- 当值保持高时发出警报（核心到达终端状态缓慢，或对等会话已关闭）

示例查询：

```
# 5分钟内的回执完成率
sum(rate(smpp_dlr_sent_total[5m]))
  / sum(rate(smpp_dlr_tracked_total[5m]))

# 每秒丢弃的回执
rate(smpp_dlr_dropped_total[5m])

# 每个绑定的当前跟踪器积压
smpp_dlr_pending
```

OmniMessage API健康指标

虽然□关本身公开SMPP相关指标，但**OmniMessage API**健康至关重要。您还应监控：

来自**OmniMessage**的指标（如果可用）

- `omnimessage_api_requests_total` - 来自□关的API请求总数
- `omnimessage_api_request_duration_seconds` - API响应时间
- `omnimessage_queue_depth` - OmniMessage队列中待处理的消息

来自网关日志（如果未公开指标）

查找以下模式以检测API问题:

- "api.*connection refused" - 无法连接到OmniMessage
- "api.*timeout" - OmniMessage未响应
- "api.*http 503" - OmniMessage暂时不可用
- "api.*parse error" - 响应格式问题

Prometheus□置

基本抓取□置

添加到`/etc/prometheus/prometheus.yml`:

```
scrape_configs:
  - job_name: 'omnimessage-smpp'
    scrape_interval: 15s
    static_configs:
      - targets: ['your-server:4000']
        labels:
          environment: 'production'
          service: 'omnimessage-smpp'
```

多个网关

```
scrape_configs:
  - job_name: 'omnimessage-smpp-instances'
    scrape_interval: 15s
    static_configs:
      - targets:
        - 'smpp-gw-1:4000'
        - 'smpp-gw-2:4000'
        - 'smpp-gw-3:4000'
        labels:
          environment: 'production'
```

服务发现

使用基于文件的发现：

```yaml
scrape_configs:
  - job_name: 'omnimessage-smpp-instances'
    file_sd_configs:
      - files:
        - '/etc/prometheus/targets/smpp-*.json'
```

文件`/etc/prometheus/targets/smpp-production.json`：

```
[
  {
    "targets": ["smpp-gw-1:4000", "smpp-gw-2:4000"],
    "labels": {
      "environment": "production",
      "datacenter": "us-east"
    }
  }
]
```

Grafana仪表板

示例仪表板面板

连接状态面板

查询：

```
smpp_connection_status{job="omnimessage-smpp"}
```

可视化：Stat

阈值：

- 红色：值 < 1（未连接）
- 绿色：值 == 1（已连接）

消息速率面板

查询：

```
rate(smpp_messages_sent_total{job="omnimessage-smpp"}[5m])
```

可视化：Graph
单位：消息/秒
图例：`{{bind_name}}`

交付成功率面板

查询：

```
100 * (1 - (
  rate(smpp_delivery_failures_total{job="omnimessage-smpp"}[5m])
  /
  rate(smpp_messages_sent_total{job="omnimessage-smpp"}[5m])
))
```

可视化：Gauge
单位：百分比（0-100）
阈值：

- 红色：< 95%
- 黄色：95-98%
- 绿色：> 98%

连接正常运行时间面板

查询：

```
smpp_uptime_seconds{job="omnimessage-smpp"} / 3600
```

可视化: Stat

单位: 小时

警报规则

Prometheus警报规则

保存到`/etc/prometheus/rules/smpp-alerts.yml`:

```yaml
groups:
  - name: smpp_gateway
    interval: 30s
    rules:
      # 接断开
      - alert: SMPPConnectionDown
        expr: smpp_connection_status == 0
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "SMPP连接{{ $labels.bind_name }}已断开"
          description: "连接{{ $labels.bind_name }}已断开超过2分钟。"

      # 高失败率
      - alert: SMPPHighFailureRate
        expr: |
          (
            rate(smpp_delivery_failures_total[5m])
            /
            rate(smpp_messages_sent_total[5m])
          ) > 0.05
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "在{{ $labels.bind_name }}上高交付失败率"
          description: "交付失败率在{{ $labels.bind_name }}上为{{ $value |
humanizePercentage }}。"

      # 绑定失败
      - alert: SMPPBindFailures
        expr: increase(smpp_bind_failures_total[10m]) > 3
        labels:
          severity: warning
        annotations:
          summary: "在{{ $labels.bind_name }}上多次绑定失败"
          description: "{{ $labels.bind_name }}在过去10分钟内绑定失败{{
$value }}次。"

      # 没有发送消息（当预期时）
      - alert: SMPPNoTraffic
        expr: rate(smpp_messages_sent_total[10m]) == 0
```

```
        for: 30m
        labels:
          severity: warning
        annotations:
          summary: "在{{ $labels.bind_name }}上没有发送消息"
          description: "{{ $labels.bind_name }}在30分钟内没有发送任何消息。"

      # 频繁断开□接
      - alert: SMPPFrequentDisconnections
        expr: increase(smpp_disconnection_total[1h]) > 5
        labels:
          severity: warning
        annotations:
          summary: "在{{ $labels.bind_name }}上频繁断开连接"
          description: "{{ $labels.bind_name }}在过去一小时内断开连接{{
$value }}次。"

      # OmniMessage API无法访问
      - alert: OmniMessageAPIUnreachable
        expr: |
          count(count_over_time({job="omnimessage-smpp"} |=
"api.*connection refused"[5m])) > 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "OmniMessage API无法访问"
          description: "SMPP网关无法访问OmniMessage API。检查API_BASE_URL配置
和网络连接。"

      # OmniMessage API超时
      - alert: OmniMessageAPITimeout
        expr: |
          count(count_over_time({job="omnimessage-smpp"} |=
"api.*timeout"[5m])) > 5
        for: 2m
        labels:
          severity: warning
        annotations:
          summary: "OmniMessage API超时"
          description: "检测到多个API超时。OmniMessage可能很慢或超负荷。"

      # 没有消息流（API问题）
      - alert: NoMessageFlow
```

```
        expr: rate(smpp_messages_sent_total[10m]) == 0 and
rate(smpp_messages_received_total[10m]) == 0
        for: 30m
        labels:
          severity: warning
        annotations:
          summary: "未检测到消息流 - 检查OmniMessage连接性"
          description: "在30分钟内没有发送或接收消息。检查OmniMessage API连接性和队列状
态。"
```

在`prometheus.yml`中加载规则:

```
rule_files:
  - '/etc/prometheus/rules/smpp-alerts.yml'
```

Web仪表板监控

内置的Web UI提供实时监控，无需Prometheus。

访问

URL: `https://your-server:8087`

实时状态页面

导航: SMPP → 实时状态

功能:

- 实时连接状态
- 消息计数器
- 连接正常运行时间
- 手动重新连接/断开控制
- 每5秒自动刷新

用途:

- 快速状态检查
- 手动干预
- 实时故障排除

仪表板显示：

- 总绑定：所有客户端和服务器连接的总计数
- 客户端绑定：到运营商的出站连接（显示连接/未连接计数）
- 服务器绑定：来自合作伙伴的入站连接（显示活动/等待计数）
- 服务器监听：入站服务器套接字的配置（主机、端口、最大连接数）

日志监控

系统日志

查看日志：

```
# 实时跟踪日志
sudo journalctl -u omnimessage-smpp -f

# 最近100行
sudo journalctl -u omnimessage-smpp -n 100

# 自特定时间以来
sudo journalctl -u omnimessage-smpp --since "1 hour ago"

# 按级别过滤
sudo journalctl -u omnimessage-smpp -p err
```

Web UI日志

导航: Web UI中的日志标签

功能:

- 实时日志流
- 按级别过滤（调试、信息、警告、错误）
- 搜索日志
- 暂停/恢复
- 清除日志

日志视图允许您:

- 级别过滤：选择日志级别（所有、调试、信息、警告、错误）
- 搜索：按文本内容查找特定日志条目
- 自动滚动：启用/禁用新日志到达时的自动滚动
- 暂停/恢复：暂停日志更新以查看特定条目
- 清除：清除所有显示的日志

关键绩效指标（KPI）

连接健康

指标：连接正常运行时间百分比

```
avg_over_time(smpp_connection_status[24h]) * 100
```

目标：> 99.9%

消息交付率

指标：每秒交付的消息

```
rate(smpp_messages_sent_total[5m])
```

目标：符合预期量

交付成功率

指标：成功交付的百分比

```
100 * (1 - rate(smpp_delivery_failures_total[5m]) /
rate(smpp_messages_sent_total[5m]))
```

目标：> 98%

绑定稳定性

指标：每小时绑定尝试次数

```
rate(smpp_bind_success_total[1h]) * 3600
```

目标：每小时 < 10（表示连接稳定）

监控最佳实践

1. 设置警报

- 配置 Prometheus 警报以监控关键指标
- 使用 PagerDuty/OpsGenie 进行 24/7 警报
- 定期测试警报

2. 创建仪表板

- 为每个□关构建 Grafana 仪表板
- 在一个仪表板上包含所有连接
- 添加容量规划面板

3. 定期审查

- 每周审查指标
- 识别趋势和模式
- 计划容量调整

4. 记录基线

- 记录正常消息量
- 记录预期的 TPS 速率
- 注意高峰时段/天数

5. 与后端关联

- 监控后端 API 指标
- 跟踪端到端消息流
- 识别瓶颈

使用指标进行故障排除

连接问题

检查：`smpp_connection_status`

- 值为 0 = 检查日志，检查□络，验证凭据
- 频繁变化 = □络不稳定

低投递率

检查：`smpp_delivery_failures_total`

- 高速率 = 检查运营商状态，审查消息格式
- ���连接之间比较 = 识别问题运营商

低吞吐量

检查：`smpp_messages_sent_total` 速率

- 低于预期 = 检查 TPS 限制，队列可用性
- 检查后端 API 指标

绑定问题

检查：`smpp_bind_failures_total`

- 增加 = 身份验证问题，凭据问题
- 检查配置中的 system_id 和密码

相关文档

- **CONFIGURATION.md** - 配置监控设置
- **USAGE.md** - 操作程序
- **TROUBLESHOOTING.md** - 解决问题
- **README.md** - 概述和快速入门

源地址白名单

对每个服务器对等体控制**SMPP**客户端在提交消息时可以使用的源地址（**source_addr**）。

概述

当外部ESME（客户端）通过SMPP□关发送`submit_sm` PDU时，PDU包含一个表示源地址（CLI / 发送者ID）的`source_addr`字段。默认情况下，经过身份验证的客户端可以使用任何源地址。源地址白名单功能允许操作员限制每个服务器对等体可以使用的源地址。

这遵循现有IP白名单的相同模式：当白名单为空时，允许所有值。当填充时，仅接受匹配的源地址。

匹□规则

每个白名单条目是一个正则表达式模式。源地址会与每个模式进行测试。如果任何模式匹配，则允许该地址。

精确匹□

使用`^`和`$`锚定模式以要求精确匹配。匹配默认是区分大小写的。

白名单条目	源地址	结果
`^MyBrand$`	`MyBrand`	允许
`^MyBrand$`	`mybrand`	拒绝
`^MyBrand$`	`MyBrands`	拒绝
`^\+61400000001$`	`+61400000001`	允许

前缀匹

使用`^`而不带尾部的`$`来匹配以前缀开头的地址。

白名单条目	源地址	结果
`^614`	`61400000001`	允许
`^614`	`61412345678`	允许
`^614`	`61500000001`	拒绝
`^\+614`	`+61400000001`	允许

高级模式

支持完整的正则表达式语法，使强大的匹配规则成为可能。

白名单条目	描述	示例
`^61[45]\d{8}$`	澳大利亚614/615手机号码（11位数字）	`61400000001`, `61512345678`
`^(MyBrand|AlertSvc)$`	精确匹配任一名称	`MyBrand`, `AlertSvc`
`(?i)^mybrand$`	不区分大小写的精确匹配	`MyBrand`, `mybrand`, `MYBRAND`
`^(?!Test).+`	任何不以“Test”开头的非空地址	`MyBrand`（允许），`TestBrand`（拒绝）

多个条目

当配置多个条目时，如果源��址匹配白名单中的任何条目，则允许该地址。

示例白名单：`^MyBrand$, ^614, ^\+61400000001$`

源地址	匹□	结果
`MyBrand`	`^MyBrand$`	允许
`61412345678`	`^614`	允许
`+61400000001`	`^\+61400000001$`	允许
`OtherBrand`	无	拒绝
`61500000001`	无	拒绝

错误处理

当由于源地址白名单违规而拒绝`submit_sm`时，□关会响应：

字段	值
PDU	`submit_sm_resp`
命令状态	`0x0000000A`
错误名称	`ESME_RINVSRCADR`（无效源地址）
消息**ID**	空

会记录一条警告，包含被拒绝的源地址和对等体名称：

```
SMPP Server: Rejected submit_sm from partner_acme - source_addr
'UnauthorisedBrand' not in whitelist
```

无效的正则表达式模式被视为不匹配（地址被拒绝），并记录一条警告。

□置

通过**Web UI**

1. 导航到 **SMPP > Server Peers**
2. 点击目标对等体的 **Edit**（或 **Add New Server Peer**）
3. 找到 **Source Address Whitelist** 字段（在IP白名单下方）
4. 输入以逗号分隔的正则表达式模式：

   ```
   ^MyBrand$,^614,^\+61400000001$
   ```

5. 点击 **Save**

更改会立即生效，适用于现有连接上的新`submit_sm` PDU。

��过□置文件

将`source_address_whitelist`添加到`runtime.exs`中的服务器绑定配置：

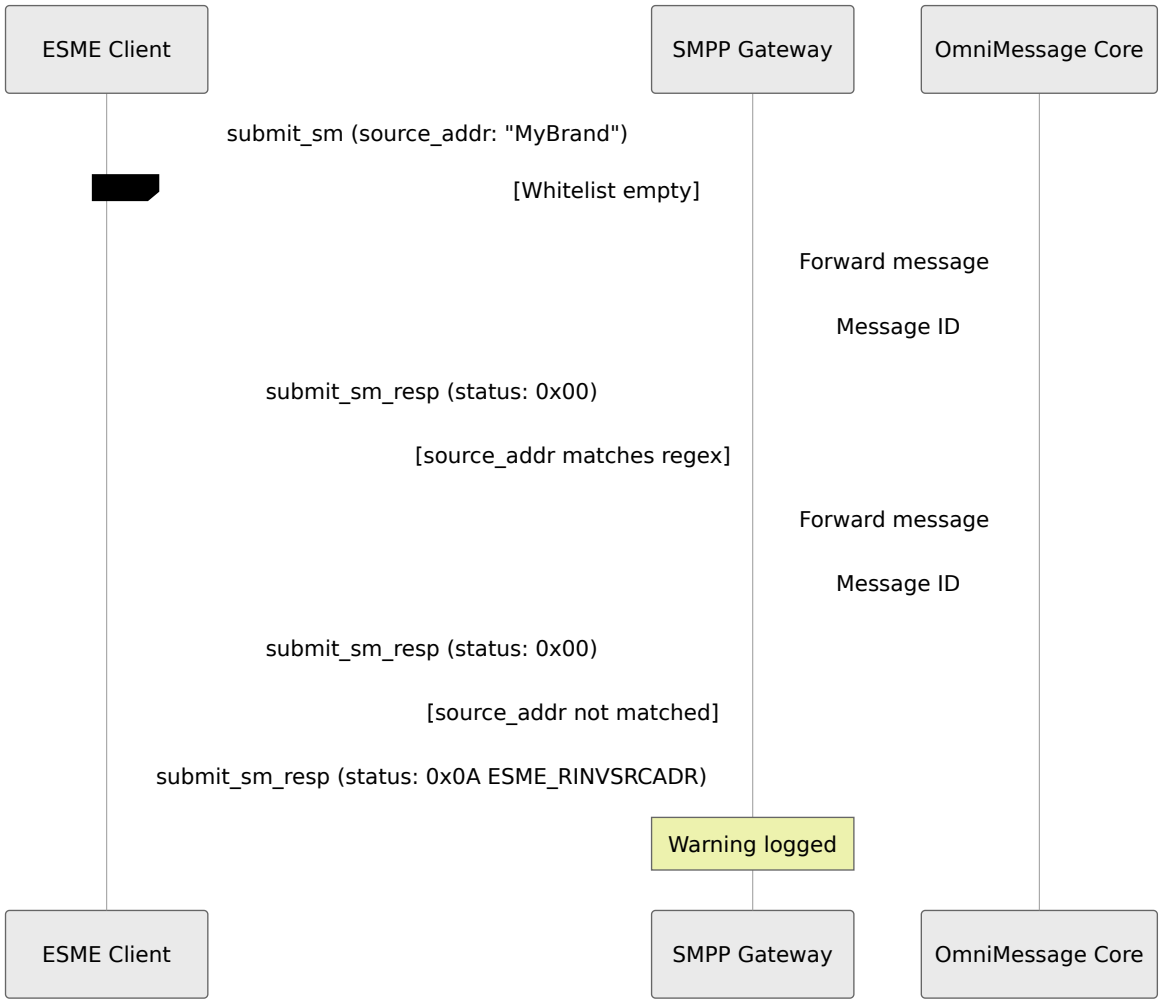
```
config :omnimessage_smpp, :server_binds, [
  %{
    name: "partner_acme",
    system_id: "acme_corp",
    password: "secure_password",
    allowed_bind_types: [:transmitter, :receiver, :transceiver],
    ip_whitelist: ["203.0.113.0/24"],
    source_address_whitelist: ["^MyBrand$", "^614",
"^\+61400000001$"],
    tps_limit: 50,
    queue_check_frequency: 1000
  }
]
```

参数

参数	类型	必需	默认	描述
`source_address_whitelist`	字符串列表	否	`[]`（允许所有）	允许的源地址正则表达式模式列表。空列表允许所有源地址。

迁移

现有的服务器对等体在□关启动时会自动迁移。在此功能添加之前创建的对等体将接收一个空白名单（允许所有源地址），以保持现有行为。



验证流程

示例

限制为单一品牌

仅允许发送者ID为`AcmeCorp`的消息：

```
^AcmeCorp$
```

允许澳大利亚号码范围

允许任何澳大利亚手机号码（以`614`开头）：

```
^614
```

允许特定号码范围

允许澳大利亚614和615手机号码（正好11位数字）：

```
^61[45]\d{8}$
```

混合字母数字和数字

允许一个品牌名称和一系列数字：

```
^AcmeCorp$,^614,^\+61290000001$
```

不区分大小写的品牌匹

允许品牌名称而不考虑大小写：

```
(?i)^acmecorp$
```

允许所有（默认）

将字段留空以允许任何源地址。这是默认行为。

故障排除

消息被拒绝，错误代码为**ESME_RINVSRCADR**

症状：合作伙伴报告`submit_sm_resp`的命令状态为`0x0A`。

可能原因：

- 源地址与白名单中的任何正则表达式模式不匹配
- 正则表达式模式存在语法错误（被视为不匹配）
- 缺少锚点（`^`/`$`）导致意外的部分匹配或拒绝
- 大小写不匹配（匹配默认是区分大小写的；使用`(?i)`进行不区分大小写的匹配）

解决方案：

1. 检查Web UI中服务器对等体的源地址白名单
2. 将被拒绝的源地址与每个正则表达式模式进行比较
3. 使用Elixir正则表达式测试工具测试模式：`Regex.match?(~r/^pattern$/, "address")`
4. 添加缺失的模式或调整现有模式
5. 记得转义特殊的正则表达式字符（例如，`+`的字面量为`\+`）

白名单未生效

症状：尽管源地址未匹配白名单，消息仍被接受。

可能原因：

- 白名单为空（默认允许所有）
- 模式未锚定并作为子字符串匹配（例如，`614`匹配`X614Y`）
- ESME连接到不同的服务器对等体
- 配置文件更改尚未应用（需要重启）

解决方案：

1. 确保Web UI中的白名单已填充（非空）
2. 确保模式在需要精确匹配的地方使用`^`和`$`锚点
3. 检查ESME绑定到哪个服务器对等体，查看实时状态
4. 如果使用配置文件，请重启服务

相关文档

- 置参考 - 完整的服务器对等体参数文档
- 使用指南 - 管理SMPP连接

- 故障排除 - 一般故障排除程序

故障排除指南

常见问题及解决方案

服务无法启动 / 端口冲突

网关或 **OmniMessage Core** 在端口 **8443** 上因 `:eaddrinuse` 崩溃循环

症状：

- 其中一个服务持续重启（`systemctl show <unit> -p NRestarts` 快速上升）。
- 日志显示：

```
Running ApiEx.Endpoint with Bandit ... at https failed, port
8443 already in use
... failed to start child: ApiEx.Supervisor ... ** (EXIT)
:eaddrinuse
```

- `curl -sk https://<host>:8443/api/messages` 返回 `404`，尽管商店应该有响应。

原因：SMPP 网关和 OmniMessage Core 位于同一主机上，并且都在默认端口 `8443` 上绑定其 `api_ex` API。先启动的服务占用端口；另一个则崩溃循环。如果网关胜出，`/api/messages` 将由网关提供（网关没有商店）并返回 404。

解决方案：将网关自己的 OAM/管理 API 移出 Core 的端口（Core 保持 `8443`）。编辑 `/etc/omnimessage_smpp/runtime.exs` 并取消注释 `api_ex` 覆盖，设置一个空闲端口。网关通过 `api_base_url` 仍然是 Core 的商店 API 的客户端。

```
# /etc/omnimessage_smpp/runtime.exs
config :api_ex,
  api:
    Application.get_env(:api_ex, :api, %{})
    |> Map.put(:port, 8444)
    |> Map.put(:listen_ip, "0.0.0.0")
```

然后重启两个服务：

```
sudo systemctl restart omnimessage-smpp
sudo systemctl restart omnimessage        # 让 Core 重新占用 :8443
```

重启后，验证 Core 拥有 8443，□关拥有其新端口：

```
ss -lntp | grep -E ':8443|:8444'
curl -sk -o /dev/null -w '%{http_code}\n'
https://127.0.0.1:8443/api/messages    # 商店有响应（不是 404）
```

有关详细信息，请参见 CONFIGURATION.md → OAM/管理 API 端口。

OmniMessage 连接问题

由于 SMPP □关是无状态的，并且完全依赖于 OmniMessage Core，因此与 OmniMessage 的连接问题是最关键的问题。

OmniMessage 断开连接的症状

- 没有出站消息：队列积压，消息未发送
- 没有入站消息：合作伙伴无法提交消息
- 超时：API 调用超时或挂起
- 日志显示：“连接被拒绝”，“超时”，“HTTP 503”，“连接重置”

诊断

1. 检查 **OmniMessage** 可用性：

```
# 测试连接
curl -k -v https://omnimessage-
core.example.com:8443/api/system/health

# 从网关主机专门测试
ssh gateway-server 'curl -k https://omnimessage-
core.example.com:8443/api/system/health'
```

2. 检查□置的 **API URL**:

```
# 查看配置
grep -A1 'api_base_url' /opt/omnimessage-smpp/config/runtime.exs

# 检查网络连接
ping omnimessage-core.example.com
nc -zv omnimessage-core.example.com 8443
```

3. 检查网关日志中的 **API** 错误:

```
# 查找与 API 相关的错误
sudo journalctl -u omnimessage-smpp -f | grep -i
'api\|omnimessage\|connect'

# 搜索日志中的最近错误
sudo journalctl -u omnimessage-smpp -n 200 | grep -i error
```

解决方案

如果 **OmniMessage** 关闭:

1. 联系 OmniMessage 操作团队
2. 待处理消息将积累在队列中
3. □关将持续重试（参见 `SMPP_POLL_INTERVAL`）
4. 检查 OmniMessage 状态页面或监控

如果 **OmniMessage** 正常但网关无法访问:

1. 检查防火墙规则允许出站 HTTPS
2. 检查 DNS 解析: `nslookup omnimessage-core.example.com`
3. 检查□络路由: `traceroute omnimessage-core.example.com`
4. 验证 SSL 证书（如果使用 HTTPS）

如果 **API URL** □置错误:

1. 编辑 `/opt/omnimessage-smpp/config/runtime.exs`
2. 验证 `api_base_url` 是否正确（生产环境必须为 HTTPS）

3. 重启□关: `sudo systemctl restart omnimessage-smpp`

连接问题

连接无法建立

症状:

- 状态显示“已断开连接”(红色)
- 日志中没有成功绑定
- 重复的连接尝试

可能的原因和解决方案:

1. 网络连接问题

检查:

```
# 测试 DNS 解析
nslookup smpp.carrier.com

# 测试连接
ping -c 3 smpp.carrier.com

# 测试端口
telnet smpp.carrier.com 2775
# 或
nc -zv smpp.carrier.com 2775
```

解决方案:

- 如果 DNS 失败:在配置中使用 IP 地址而不是主机名
- 如果 ping 失败:检查防火墙规则,联系运营商
- 如果端口失败:验证正确的端口号,检查防火墙

2. 凭证不正确

检查:

- 日志显示“绑定失败”或“身份验证错误”
- Web UI: SMPP → 客户端对等体 → 验证 system_id 和密码

解决方案:

- 与运营商确认凭证
- 检查拼写错误（区分大小写）
- 更新配置并重新连接

3. IP 未列入白名单

检查:

- 连接立即被拒绝
- 运营商日志显示未授权 IP

解决方案:

- 确认您的□关公共 IP:

```
curl ifconfig.me
```

- 请求运营商将 IP 添加到白名单
- 验证 IP 是否已更改（动态 IP）

4. 防火墙阻塞

检查:

```
# 检查端口是否开放
sudo iptables -L -n | grep 2775

# 检查 UFW（Ubuntu/Debian）
sudo ufw status | grep 2775

# 检查 firewalld（RHEL/CentOS）
sudo firewall-cmd --list-ports | grep 2775
```

解决方案：

```
# Ubuntu/Debian
sudo ufw allow out 2775/tcp

# RHEL/CentOS
sudo firewall-cmd --permanent --add-port=2775/tcp
sudo firewall-cmd --reload
```

5. 固定源地址/端口不可用

仅在客户端绑定设置 `bind_source_ip` 和/或 `bind_source_port` 时适用（参见 CONFIGURATION.md）。

检查：

- 日志显示连接无法打开本地套接字
- `bind_source_ip` 不是此主机持有的地址
- 另一个进程已经占用了配置的 `bind_source_port`

解决方案：

- 确认 `bind_source_ip` 是本地地址：`ip addr show`
- 无效的 `bind_source_ip` 被记录并忽略——操作系统随后选择源 IP，这可能与运营商的白名单不匹配。如果交付失败，请更正地址。
- 固定端口使用 `SO_REUSEADDR` 打开，因此重新连接会立即重新绑定，即使旧套接字处于 `TIME_WAIT` 状态。如果端口仍不可用，请检查是否有其他服务在使用它：`sudo ss -ltnp | grep <port>`

连接持续掉线

症状：

- 连接已建立但频繁断开
- `smpp_disconnection_total` 指标增加
- 日志显示重复重连

可能的原因和解决方案：

1. 网络不稳定

检查:

```
# 监控丢包
ping -c 100 smpp.carrier.com | grep loss

# 检查网络错误
netstat -s | grep -i error
```

解决方案:

- 联系运营商解决□络问题
- 如果在您这边，检查 ISP
- 考虑备份连接/路由

2. 查询链路超时

检查:

- 日志显示“enquire_link timeout”
- 连接在不活动期间掉线

解决方案:

- 默认超时为 30 秒
- 验证□络允许保持活动数据包
- 检查是否有激进的防火墙超时空闲连接

3. 超过 TPS 限制

检查:

- 断开连接时消息速率高
- 运营商限制消息

解决方案:

- 查看 `tps_limit` 设置

- 将 TPS 降低到运营商最大值的 70-80%
- 在多个绑定之间分散流量

4. 运营商服务器问题

检查：

- 检查运营商服务状态
- 联系运营商支持

解决方案：

- 等待运营商解决
- 如果可用，配置备份运营商

消息传递问题

消息未发送

症状：

- 消息卡在队列中
- `smpp_messages_sent_total` 不增加
- 连接显示已连接

可能的原因和解决方案：

1. 错误的 dest_smsc ���由

检查：

- Web UI → 队列 → 检查消息 `dest_smsc` 字段
- 与 SMPP → 实时状态中的连接名称进行比较

解决方案：

- 消息根据 `dest_smsc` 字段路由

- 验证后端是否设置正确的 `dest_smsc`
- 如果 `dest_smsc` 为 NULL，请检查默认路由

2. 消息计划在未来发送

检查：

- Web UI → 队列 → 检查 `deliver_after` 字段
- 带有未来时间戳的消息尚未发送

说明：

- 重试系统为失败的消息设置 `deliver_after`
- 消息将在该时间之前等待重试

解决方案：

- 等待计划时间
- 如果紧急，请联系后端团队重置时间戳

3. TPS 限制过低

检查：

- 大量队列积压
- 消息发送非常缓慢

解决方案：

- 在配置中增加 `tps_limit`
- 验证运营商是否能够处理更高的速率
- 参见 CONFIGURATION.md

4. 队列工作者未运行

检查：

- 服务状态
- 日志中的错误

解决方案：

```
# 重启服务
sudo systemctl restart omnimessage-smpp

# 检查日志
sudo journalctl -u omnimessage-smpp -f
```

5. 前端未在预期名称下注册

OmniMessage Core 仅在对等体以与消息的 `dest_smsc` 相等的前端名称注册时，将消息路由到此□关。每个对等体 ���据其 路由键 注册（参见 前端注册）：

- 客户端对等体：配置的 `queue`，或者如果未设置 `queue` 则为裸对等体名称。
- 服务器绑定：绑定名称。

检查：

- 日志中是否有 `Successfully registered SMPP peer frontend '<name>'` 条目——确认 `<name>` 与卡住消息的 `dest_smsc` 匹配
- 对于共享 `queue` 的绑定，期望在共享队列名称下有单个注册，而不是每个对等体一个

解决方案：

- 确保后端将 `dest_smsc` 设置为路由键（对等体的 `queue`，否则为其名称），而不是 `{hostname}_{peer}`（早期命名方案不再适用）
- 如果使用共享 `queue`，将消息路由到队列名称而不是单个对等体名称
- 确认对等体已启用——禁用的对等体未注册

高送达失败率

症状：

- `smpp_delivery_failures_total` 增加
- 日志显示“submit_sm_resp”带有错误状态
- 消息未到达接收者

可能的原因和解决方案：

1. 无效的目的地号码

检查：

- 日志中的特定错误代码
- 查看消息目的地格式

常见错误代码：

- `0x0000000B` - 无效目的地
- `0x00000001` - 无效消息长度
- `0x00000003` - 无效命令

解决方案：

- 验证号码格式（��荐 E.164）
- 检查号码是否包含国家代码
- 验证运营商要求

2. 无效的消息内容

检查：

- 消息长度
- 特殊字符
- 编码

解决方案：

- GSM-7：最大 160 个字符
- UCS-2：最大 70 个字符
- 删除不支持的字符
- 检查编码设置

3. 运营商拒绝

检查：

- 运营商的特定错误代码
- 被拒绝消息的模式

解决方案：

- 联系运营商了解拒绝原因
- 可能需要内容过滤
- 检查是否存在垃圾邮件/滥用模式

4. 消息过期

检查：

- 消息 `expires` 时间戳
- 交付尝试时间

解决方案：

- 增加消息有效期
- 减少时间敏感消息的重试延迟

Web UI 问题

无法访问 Web 仪表板

症状：

- 浏览器无法连接到 https://your-server:8087
- 超时或连接被拒绝

可能的原因和解决方案：

1. 服务未运行

检查：

```
sudo systemctl status omnimessage-smpp
```

解决方案:

```
# 如果停止，则启动它
sudo systemctl start omnimessage-smpp

# 检查日志中的错误
sudo journalctl -u omnimessage-smpp -n 50
```

2. 防火墙阻止端口 8087

检查:

```
sudo ufw status | grep 8087
# 或
sudo firewall-cmd --list-ports | grep 8087
```

解��方案:

```
# Ubuntu/Debian
sudo ufw allow 8087/tcp

# RHEL/CentOS
sudo firewall-cmd --permanent --add-port=8087/tcp
sudo firewall-cmd --reload
```

3. SSL 证书问题

检查:

- 浏览器显示安全警告
- 证书过期或无效

解决方案:

- 接受安全例外（如果是自签名）

- 安装有效的 SSL 证书
- 检查证书文件是否存在:

```
ls -l /opt/omnimessage-smpp/priv/cert/
```

4. URL 错误

检查:

- 验证使用 HTTPS（而不是 HTTP）
- 验证正确的服务器 IP/主机名
- 验证端口 8087

Web UI 显示错误

症状:

- 页面加载但显示错误
- 功能无法正常工作
- 数据未显示

解决方案:

1. 清除浏览器缓存:
    - Ctrl+F5（强制刷新）
    - 清除浏览器缓存和 Cookie
2. 检查浏览器控制台:
    - 按 F12
    - 检查控制台标签中的 JavaScript 错误
    - 如果发现错误，请报告支持
3. 尝试不同的浏览器:
    - 在 Chrome、Firefox、Edge 中测试
    - 隔离浏览器特定问题

4. 检查服务日志:

```
sudo journalctl -u omnimessage-smpp -f
```

指标问题

Prometheus 指标不可用

症状:

- `curl http://localhost:4000/metrics` 失败
- Prometheus 无法抓取指标
- 空响应或错误响应

可能的原因和解决方案:

1. 服务未运行

检查:

```
sudo systemctl status omnimessage-smpp
```

解决方案:

```
sudo systemctl start omnimessage-smpp
```

2. 端口不可访问

检查:

```
# 本地测试
curl http://localhost:4000/metrics

# 远程测试
curl http://your-server-ip:4000/metrics
```

解决方案：

- 如果本地有效但远程无效：检查防火墙
- 为 Prometheus 服务器在防火墙中打开端口 4000

3. 错误的端点

验证：

- 端点是 `/metrics`（而不是 `/prometheus` 或 `/stats`）
- 端口是 4000（而不是 8087）

指标显示意外值

症状：

- 计数器重置为零
- 指标显示错误值
- 某些绑定缺少指标

解决方案：

1. 服务重启重置计数器：
    - 服务重启时计数器重置
    - 这是正常行为
    - 在 Prometheus 查询中使用 `increase()` 或 `rate()`
2. 新绑定未显示：
    - 指标仅在第一次事件后出现

- 发送测试消息以填充指标
- 检查绑定是否已启用并连接

3. 过时的指标：

- 旧绑定可能仍显示在指标中
- 重启服务以清除过时条目
- 或使用 Prometheus 重标记进行过滤

性能问题

CPU 使用率高

检查：

```
top -p $(pgrep -f omnimessage-smpp)
```

可能的原因：

- 消息量非常大
- 连接过多
- 配置问题

解决方案：

- 检查消息速率是否在容量范围内
- 查看 TPS 限制
- 如果持续高 CPU，请联系支持

内存使用率高

检查：

```
ps aux | grep omnimessage-smpp
```

可能的原因:

- 内存中队列消息过大
- 内存泄漏（罕见）

解决方案:

- 重启服务以清除内存
- 检查消息队列大小
- 如果内存持续增长，请联系支持

消息处理缓慢

症状:

- 消息发送时间长
- 队列积压
- 消息速率低

检查:

1. TPS 限制 - 可能过于严格
2. `queue_check_frequency` - 可能过高
3. 后端 API 响应时间 - 可能较慢
4. 到运营商的网络延迟

解决方案:

- 如果运营商允许，增加 TPS
- 减少 `queue_check_frequency` 以加快轮询
- 优化后端 API
- 检查网络延迟

□置问题

□置文件语法错误

症状：

- 配置更改后服务无法启动
- 日志显示“语法错误”或“解析错误”

检查：

```
# 验证 Elixir 语法
/opt/omnimessage-smpp/bin/omnimessage-smpp eval "File.read!
('config/runtime.exs')"
```

常见错误：

- 地图条目之间缺少逗号
- 引号不匹配（" vs '）
- 括号或大括号不匹配
- 顶部缺少 `import Config`

解决方案：

- 从备份恢复
- 仔细检查语法
- 使用具有 Elixir 语法高亮的文本编辑器

更改未生效

症状：

- 修改了配置但行为没有变化
- 旧设置仍然有效

解决方案：

```
# 配置更改需要重启
sudo systemctl restart omnimessage-smpp

# 验证重启成功
sudo systemctl status omnimessage-smpp

# 检查日志中的错误
sudo journalctl -u omnimessage-smpp -n 50
```

紧急恢复

完全系统故障

步骤:

1. 检查基本系统健康:

   ```
   # 磁盘空间
   df -h

   # 内存
   free -h

   # CPU 负载
   uptime
   ```

2. 检查服务状态:

   ```
   sudo systemctl status omnimessage-smpp
   ```

3. 查看最近的日志:

   ```
   sudo journalctl -u omnimessage-smpp -n 200
   ```

4. 尝试重启服务：

```
sudo systemctl restart omnimessage-smpp
```

5. 如果重启失败：
   - 检查配置语法
   - 验证 SSL 证书是否存在
   - 检查文件权限
   - 查看日志以获取具体错误
6. 从备份恢复（如有必要）：

```
# 恢复配置
sudo cp /opt/omnimessage-smpp/config/runtime.exs.backup \
  /opt/omnimessage-smpp/config/runtime.exs

# 重启
sudo systemctl restart omnimessage-smpp
```

7. 如果未解决，请联系支持

获取帮助

收集的信息

在联系支持之前，请收集：

1. 版本：`cat /opt/omnimessage-smpp/VERSION`
2. 最近的日志：

```
sudo journalctl -u omnimessage-smpp -n 200 > /tmp/smpp-logs.txt
```

3. □置（清理密码）：

```
sudo cp /opt/omnimessage-smpp/config/runtime.exs
/tmp/config.exs
# 编辑 /tmp/config.exs 以在发送之前删除密码
```

4. 指标输出：

```
curl http://localhost:4000/metrics > /tmp/metrics.txt
```

5. 系统信息：

```
uname -a > /tmp/system-info.txt
free -h >> /tmp/system-info.txt
df -h >> /tmp/system-info.txt
```

联系支持

- 电子邮件：support@omnitouch.com
- 电话：+61 XXXX XXXX (24/7)
- 包括：上述所有信息

相关文档

- **USAGE.md** - 正常操作程序
- **CONFIGURATION.md** - 配置参考
- **MONITORING.md** - 监控和指标
- **README.md** - 系统概述

操作指南

日常操作程序

关键依赖: OmniMessage Core

重要: OmniMessage SMPP 网关在没有访问 OmniMessage Core 的情况下无法运行。所有消息处理都发生在 OmniMessage 中 - 网关只是一个协议转换器。

如果 OmniMessage 无法使用:

- 无 短信功能
- 无 语音通话功能
- 无 数据服务
- 无 计费记录

检查 **OmniMessage** 健康状况:

```
# 测试 API 连接性
curl -k https://omnimessage-
core.example.com:8443/api/system/health

# 检查日志中的配置 API URL
grep api_base_url /opt/omnimessage-smpp/config/runtime.exs
```

日常操作

早晨健康检查

在每一天开始时执行以下检查:

1. 访问 **Web** 仪表板
    - URL: `https://your-server:8087`

- 检查仪表板是否加载正常

2. 检查连接状态

   - 导航到：SMPP → 实时状态
   - 验证所有连接显示“已连接”（绿色）
   - 注意任何断开的绑定

3. 查看消息指标

   - 导航到：队列选项卡
   - 检��消息计数是否合理
   - 验证没有意外的队列积压

4. 检查系统日志

   - 导航到：日志选项卡
   - 查找错误消息（红色）
   - 注意任何警告模式

5. 查看 **Prometheus** 指标

   - `curl http://localhost:4000/metrics`
   - 或检查 Grafana 仪表板
   - 验证消息速率是否正常

持续监控

设置警报以监控：

- 连接失败（> 2 分钟宕机）
- 高交付失败率（> 5%）
- 长时间没有流量
- 频繁断开连接

请参见 MONITORING.md 以获取警报配置。

理解消息路由

关使用两个关键字段在 OmniMessage Core 和 SMPP 连接之间路由消息：

- **`dest_smsc`** - 将出站消息路由到客户端绑定。当 OmniMessage 将消息排队时，`dest_smsc: "vodafone_uk"`，关的名为 `vodafone_uk` 的客户端绑定会接收并通过 SMPP `submit_sm` 发送。
- **`source_smsc`** - 将入站消息路由到服务器绑定。当 OmniMessage 将消息排队时，`source_smsc: "partner_acme"`，关会通过 SMPP `deliver_sm` 将其交付给连接到名为 `partner_acme` 的服务器绑定的客户端。

关键区别：客户端绑定发送 `submit_sm` PDU（关是向运营商提交的 ESME）。服务器绑定发送 `deliver_sm` PDU（关是将消息交付给连接的 ESME 的 SMSC）。

前端注册

关会自动向 OmniMessage Core 注册，以便后端知道哪些 SMPP 连接可用于消息路由。

- 注册名称：由 `smsc_name` 配置控制（默认值："smpp_gateway"，环境变量：`SMSC_NAME`）
- 心跳：每 60 秒发送一次以保持注册状态
- 过期：在没有心跳的情况下，注册在后端 90 秒后过期
- 每个绑定注册：每个启用的对等体在其 路由键 下单独注册，因此注册的前端名称等于后端路由的 `dest_smsc`：
  - 客户端对等体：根据其配置的 `queue` 注册（如果设置），否则为裸对等体名称。因此，共享一个 `queue` 的多个客户端对等体会注册在该单一共享名称下。

- 服务器绑定：根据其绑定名称注册。

如果□关停止或失去与 OmniMessage Core 的连接，其注册将过期，后端将停止向其路由消息。

故障排除：如果消息未被路由到□关，请检查：

1. 日志中是否有 "frontend_register" 条目
2. `smsc_name` 是否与 OmniMessage 期望的匹配
3. 与 OmniMessage Core 的□络连接（`api_base_url`）

管理 SMPP 连接

如何□置 SMPP 对等体

SMPP 连接（对等体）可以使用 两种方法 进行配置：

方法 1: Web UI 或 OAM API（推荐）

- 优点：更改立即生效，无需重启
- 位置：SMPP → 客户端对等体 / 服务器对等体选项卡，或 OAM API 上的 `/api/smpp/client_peers` 和 `/api/smpp/server_peers`
- 操作：添加、编辑、删除对等体
- 保存的内容：
  - 服务器对等体：其队列工作者启动或使用新设置重新启动。如果这是第一个服务器对等体，则 SMPP 监听器启动。已经绑定的会话继续运行。
  - 客户端对等体：连接被解除绑定并使用新设置重新建立。
  - 禁用或删除对等体：其工作者被停止。客户端对等体首先解除绑定。服务器对等体的绑定会话发送 `unbind`（如果 ESME 不响应，则在 2 秒后关闭），并拒绝新的绑定。
- 持久性：存储在 Mnesia 数据库中
- 最佳用途：日常操作、测试、快速更改

方法 2: □置文件

- 优点：代码配置，版本控制
- 位置：`/opt/omnimessage-smpp/config/runtime.exs`

- 操作：在 Elixir 配置中定义对等体
- 持久性：基于文件，重启后仍然有效
- 要求：更改后需要重启服务
- 最佳用途：初始设置、基础设施即代码

注意：Web UI 的更改单独存储并覆盖配置文件设置。

请参见 CONFIGURATION.md 以获取配置文件参考。

添加新的客户端连接

目的：配置□关作为连接到运营商的 **ESME**（客户端）

准备：收集运营商的信息：

- SMPP 服务器主机名/IP
- 端口号（通常为 2775）
- 系统 ID（用户名）
- 密码
- 绑定类型（通常为收发器）
- TPS 限制

选择以下方法之一：

选项 **A**：通过 **Web UI**（推荐）

优点：立即生效，无需重启

步骤：

1. 导航到客户端对等体：
    - 打开 Web UI：`https://your-server:8087`
    - 导航到：SMPP → 客户端对等体
2. 添加新对等体：
    - 点击“添加新客户端对等体”
    - 填写表单：

        - 名称：`vodafone_uk`（唯一标识符）
        - 主机：`smpp.vodafone.co.uk`
        - 端口：`2775`
        - 系统 **ID**：`your_username`
        - 密码：`your_password`
        - 绑定类型：`Transceiver`
        - **TPS** 限制：`100`
        - 队列检查频率：`1000`
    - 点击“保存”

3. 连接自动建立：
    - 关立���尝试连接
    - 导航到：SMPP → 实时状态
    - 状态应在 10-30 秒内更改为“已连接”（绿色）
    - 检查日志选项卡以获取成功绑定消息
4. 测试消息流：
    - 导航到：队列选项卡
    - 提交测试消息，`dest_smsc` 与绑定名称匹配
    - 在实时状态中监控传输
    - 验证交付确认

选项 **B**：通过置文件

优点：基础设施即代码，版本控制

步骤：

1. 编辑□置文件：

```
sudo nano /opt/omnimessage-smpp/config/runtime.exs
```

2. 将新绑定添加到□置中：

```
config :omnimessage_smpp, :binds, [
  # 现有绑定...

  # 添加新绑定
  %{
    name: "vodafone_uk",
    mode: :client,
    bind_type: :transceiver,
    host: "smpp.vodafone.co.uk",
    port: 2775,
    system_id: "your_username",
    password: "your_password",
    tps_limit: 100,
    queue_check_frequency: 1000,

    # 可选：如果运营商通过本地源 IP/端口进行白名单，则固定
    bind_source_ip: "10.20.30.40",
    bind_source_port: 5000,

    # 可选：在多个绑定之间共享一个后端队列（默认为名称）
    queue: "vodafone_uk"
  }
]
```

请参见 CONFIGURATION.md ��获取客户端绑定参数的完整列表，包括 `bind_source_ip`、`bind_source_port` 和 `queue`。

3. 保存并重启服务：

```
# 保存文件（在 nano 中按 Ctrl+X, Y, Enter）

# 重启服务
sudo systemctl restart omnimessage-smpp
```

4. 验证连接：
    - 导航到：SMPP → 实时状态
    - 找到新连接
    - 状态应为“已连接”（绿色）
    - 检查日志以获取成功绑定
5. 测试消息流：
    - 导航到：队列选项卡
    - 提交测试消息，`dest_smsc` 与新绑定名称匹配
    - 在实时状态中监控传输
    - 验证交付确认

添加服务器绑定

目的：配置□关作为 **SMSC**（服务器），接受来自外部 **ESME**（合作伙伴客户端）的连接

准备：

1. 生成凭据：
    - 创建唯一的系统 ID：`partner_name`
    - 创建强密码
    - 安全地记录并与合作伙伴共享
2. 获取合作伙伴信息：
    - 合作伙伴的源 IP 地址
    - 预期消息量（用于 TPS 限制）
    - 所需的绑定类型

选择以下方法之一：

选项 **A**：通过 **Web UI**（推荐）

优点：立即生效，无需重启

步骤：

1. 导航到服务器对等体：
    - 打开 Web UI：`https://your-server:8087`
    - 导航到：SMPP → 服务���对等体
2. 添加新服务器对等体：
    - 点击“添加新服务器对等体”
    - 填写表单：
        - 名称：`partner_acme`（唯一标识符）
        - 系统 **ID**：`acme_corp`
        - 密码：`secure_password_123`
        - 允许的绑定类型：选择所有（发射器、接收器、收发器）
        - **IP** 白名单：`203.0.113.0/24`（多个用逗号分隔）
        - **TPS** 限制：`50`
        - 队列检查频率：`1000`
    - 点击“保存”
3. 网关准备好连接：

- 服务器对等体现在处于活动状态，等待合作伙伴连接
- 无需重启

4. 与合作伙伴共享信息：

   - □关 IP 地址
   - 端口：`2775`
   - 系统 ID：`acme_corp`
   - 密码：`secure_password_123`
   - 绑定类型：按配置

5. 等待合作伙伴连接：

   - 导航到：SMPP → 实时状态
   - 观察传入连接
   - 验证身份验证成功
   - 检查 IP 是否匹配白名单

选项 **B**：通过□置文件

优点：基础设施即代码，版本控制

步骤：

1. 编辑□置文件：

   ```
   sudo nano /opt/omnimessage-smpp/config/runtime.exs
   ```

2. 添加服务器绑定和监听□置：

```
# 添加到 server_binds 列表
config :omnimessage_smpp, :server_binds, [
  # 现有服务器绑定...

  # 添加新服务器绑定
  %{
    name: "partner_acme",
    system_id: "acme_corp",
    password: "secure_password_123",
    allowed_bind_types: [:transmitter, :receiver,
:transceiver],
    ip_whitelist: ["203.0.113.0/24"],
    tps_limit: 50,
    queue_check_frequency: 1000
  }
]

# 确保存在监听配置（只需一次）
config :omnimessage_smpp, :listen, %{
  host: "0.0.0.0",
  port: 2775,
  max_connections: 100
}
```

3. 保存并重启服务：

```
sudo systemctl restart omnimessage-smpp
```

4. 与合作伙伴共享信息：

   - □关 IP 地址
   - 端口：`2775`
   - 系统 ID：`acme_corp`
   - 密码：`secure_password_123`
   - 绑定类型：按配置

5. 等待合作伙伴连接：

   - 导航到：SMPP → 实时状态

- 观察传入连接
- 验证身份验证成功
- 检查 IP 是否匹配白名单

修改现有连接

目的：更新连接参数（TPS 限制、密码、IP 白名单等）

选择以下方法之一：

选项 **A**：通过 **Web UI**（推荐）

优点：立即生效，无需重启

步骤：

1. 导航到对等体：
   - 打开 Web UI：`https://your-server:8087`
   - 对于客户端连接：SMPP → 客户端对等体
   - 对于服务器连接：SMPP → 服务器对等体
2. 编辑对等体：
   - 找到要修改的对等体
   - 点击“编辑”按钮
   - 更新所需参数：
     - 常见更改：TPS 限制、密码、IP 白名单、主机/端口
   - 点击“保存”
3. 更改立即生效：
   - 连接会自动使用新设置重新连接
   - 无需重启服务
   - 导航到：SMPP → 实时状态以验证
4. 验证更改：
   - 检查连接是否成功建立

- 监控日志选项卡以查找错误
- 如适用，测试消息流

选项 **B**：通过□置文件

优点：基础设施即代码，版本控制

步骤：

1. 编辑□置文件：

```
sudo nano /opt/omnimessage-smpp/config/runtime.exs
```

2. 修改绑定参数：

   - 在 `:binds` 或 `:server_binds` 列表中找到绑定
   - 更新所需参数：
     - 常见更改：TPS 限制、密码、IP 白名单、主机/端口
   - 示例：

```
%{
  name: "vodafone_uk",
  # ... 其他参数
  tps_limit: 150,  # 从 100 更改
  password: "new_password"  # 更新密码
}
```

3. 保存并重启服务：

```
sudo systemctl restart omnimessage-smpp
```

4. 验证更改：

   - 导航到：SMPP → 实时状态
   - 检查连接是否成功建立
   - 监控日志以查找错误
   - 测试消息流

移除连接

目的：停用一个 SMPP 连接

步骤：

1. 通知相关方：
    - 通知运营商/合作伙伴
    - 协调停机窗口
2. 通过 **Web UI** 断开连接：
    - 导航到：SMPP → 实时状态
    - 找到连接
    - 点击“断开连接”
    - 确认操作
3. 移除□置：
    - 导航到：SMPP → 客户端/服务器对等体
    - 找到连接
    - 点击“删除”
    - 确认移除
4. 验证移除：
    - 检查实时状态 - 连接应消失
    - 查看日志以确保干净关闭

启用和禁用连接

目的：在不删除其配置的情况下，暂时将连接下线

对等体具有一个 `enabled` 字段，用于控制它们是否处于活动状态。禁用的对等体保留所有配置，但不会建立或接受连接。

通过 **Web UI**：

1. 导航到：SMPP → 客户端对等体或服务器对等体

2. 找到要禁用的对等体
3. 点击“编辑”
4. 取消选中“启用”复选框
5. 点击“保存”

连接将立即断开。要重新启用，请重复步骤并再次选中复选框。

使用案例：

- 计划中的运营商维护窗���
- 在调查期间暂时暂停合作伙伴连接
- 在等待新凭据时禁用连接

连接行为

重新连接逻辑

当客户端绑定无法连接、绑定被拒绝或因任何原因失去已建立的会话时，□关会自动重新连接：

- 退避：第一次重试在 1 秒后，然后每次连续失败时等待时间加倍（2 秒、4 秒、8 秒...）最多到 30 秒
- 退避重置：一旦会话保持绑定 30 秒，等待时间会回到 1 秒。因此，接受连接但立即断开的运营商会以较慢的速度重试，而不是每秒重试
- 连接意味着绑定：只有在运营商接受绑定后，绑定才会显示为“已连接”，并且仅在发送流量。一旦 TCP 连接正常但绑定响应尚未到达，绑定会显示为连接中
- 连接超时：未收到响应的 TCP 连接在 3 秒后被放弃，并使用上述退避重新尝试
- 隔离：一个绑定的重新连接不会重新启动其他绑定或服务，无论运营商多么频繁地断开连接
- 错开启动：当多个绑定同时启动（例如，在服务重启后），连接会错开，每个绑定之间有 500 毫秒的延迟，以避免压倒□络
- 弹性启动：如果在□关启动时运营商无法访问，□关会成功启动并在后台重试连接

通过 `smpp_reconnect_attempts_total` 指标跟踪重新连接（请参��� MONITORING.md）。

干净断开

当客户端绑定通过“断开连接”或通过停止或重启服务被关闭时，□关会向运营商发送 SMPP `unbind`，并等待最多 2 秒以获取 `unbind_resp`，然后关闭 TCP 连接。运营商会立即释放绑定，而不是等待其自己的保持活动时间到期。这对每个账户只允许一个绑定的运营商很重要，否则在重启后会拒绝重新连接。

仍在连接中的绑定（TCP 正常，绑定尚未被接受）会在没有 `unbind` 的情况下关闭。

查询链接（保持活动）

关定期发送 SMPP `enquire_link` PDU 以验证连接是否正常：

- 默认间隔：60 秒（可通过 `enquire_link_interval` 为每个绑定配置）
- 禁用：设置 `enquire_link_interval: 0`（不推荐）
- 故障检测：如果远程对等体停止响应 `enquire_link`，则连接被视为已死亡，并开始重新连接

通过 Prometheus 指标 `smpp_enquire_link_sent_total` 和 `smpp_enquire_link_received_total` 监控查询链接健康状况。发送和接收之间的差距不断扩大表示连接问题。

TPS 速率限制

每个绑定使用每秒滑动窗口强制执行其 `tps_limit`：

- 消息在每个 1 秒窗口计数
- 当达到限制时，队列工作者暂停，直到下一个秒
- 每个绑定最多可以有 100 条消息在飞行中（等待响应）
- 窗口在每个新秒的开始时自动重置

如果您看到吞吐量缓慢，请检查：

1. `tps_limit` 是否设置得足够高以满足您的流量
2. `queue_check_frequency` 是否足够低以保持管道供给
3. 运营商是否及时响应消息（缓慢的响应会降低有效吞吐量）

管理消息流

检查消息队列

目的：监控待处理消息

步骤：

1. 访问队列：
    - 导航到：队列选项卡
    - 查看待处理消息列表

2. 检查消息详细信息：
    - 点击消息行
    - 查看：
        - 目标号码
        - 消息正文
        - 目标 SMSC（`dest_smsc`）
        - 交付尝试
        - 状态

3. 搜索特定消息：
    - 使用搜索过滤器
    - 按目标、内容或 SMSC 过滤

故障排除卡住的消息

症状：消息未被交付

步骤：

1. 检查连接状态：
    - 导航到：SMPP → 实时状态
    - 验证目标连接是否已连接
    - 如果断开，请参见 重新连接
2. 检查消息详细信息：
    - 导航到：队列选项卡
    - 找到卡住的消息
    - 检查 `dest_smsc` 字段是否与连接名称匹配
    - 检查 `deliver_after` 时间戳（重试调度）
3. 检查交付尝试：
    - 高尝试 = 重复失败
    - 检查日志中的错误消息
    - 可能表示格式无效或运营商拒绝
4. 手动干预（如有必要）：
    - 联系运营商以验证问题
    - 可能需要取消并重新提交消息
    - 与后端团队检查队列问题

交付回执（DLR）透传

交付回执（DLR）是报告已提交消息最终状态的 SMPP 消息。当绑定设置 `deliver_receipts: true` 时，关会将 DLR 发送回提交绑定。使用此功能用于需要每条消息交付确认的互连绑定。

关将回执作为 `deliver_sm` PDU 发送，`esm_class` 设置为 `0x04`（SMSC 交付回执）。回执携带关在 `submit_sm_resp` 中返回的相同消息 ID，以便运营商可以将回执与原始提交关联。

需求

- 在服务器绑定上设置 `deliver_receipts: true`。请参见 CONFIGURATION.md。

- 在同一绑定上设置 `cache_enabled: false`。缓存返回一个临时消息 ID，该 ID 与核心 ID 不匹配，因此缓存绑定会破坏收据关联。如果绑定同时设置了这两个选项，□关在启动时会写入警告。

消息流

收据状态值

收据 `stat` 字段报告消息的最终状态。

核心状态	收据 `stat`	意义
`delivered`	`DELIVRD`	收件人已收到消息。
`expired`	`EXPIRED`	消息有效期在交付前已结束。
`dropped`	`UNDELIV`	□关无法交付消息。
`rejected`	`REJECTD`	核心拒绝了消息。
`balance_rejected`	`REJECTD`	核心因账户余额原因拒绝了消息。

操作说明

- □关在内存中跟踪正在传输的消息。如果服务重启，当前正在传输的消息的收据将丢失。这是一种尽力而为的机制。
- 如果在消息达到终态时绑定会话已关闭，□关会保留收据，并在会话重新连接时发送。□关在 24 小时后会丢弃收据。
- 监控 DLR 指标以确认功能正常。请参见 MONITORING.md。

连接故障排除

重新连接绑定

症状：连接显示“已断开”（红色）

步骤：

1. 检查网络连接：

   ```
   ping -c 3 carrier-smpp-server.com
   telnet carrier-smpp-server.com 2775
   ```

2. 检查日志中的错误：

   - 导航至：日志标签

- 过滤：错误级别
- 查找身份验证失败、□络超时

3. 验证凭据：
   - 导航至：SMPP → 客户端/��务器对等体
   - 检查 system_id 和密码是否正确
   - 如果不确定，请联系运营商
4. 手动重新连接：
   - 导航至：SMPP → 实时状态
   - 找到断开的绑定
   - 点击“重新连接”按钮
   - 等待 10-30 秒
   - 检查状态是否更改为“已连接”
   - 重新连接也会恢复被“丢弃”的绑定
5. 如果重新连接失败：
   - 检查防火墙规则
   - 验证运营商服务器是否正常运行
   - 联系运营商支持
   - 请参见 TROUBLESHOOTING.md

处理身份验证失败

症状：日志中重复绑定失败

原因：

- 用户名/密码不正确
- IP 未在运营商处列入白名单
- 账户被暂停/过期

步骤：

1. 验证凭据：
    - 导航至：SMPP → 客户端对等体
    - 仔细检查 system_id 和密码
    - 与运营商确认
2. 检查 **IP** 白名单：
    - 确认您的□关 IP 与运营商
    - 请求运营商验证 IP 白名单
3. 检查账户状态：
    - 验证账户是否处于活动状态
    - 检查是否有过期的合同
    - 联系运营商账单部门
4. 更新□置：
    - 如果凭据已更改，请在 Web UI 中更新
    - 点击“重新连接”以使用新凭据重试

监控和警报

检查 **Prometheus** 指标

快速检查：

```
curl http://localhost:4000/metrics | grep smpp_connection_status
```

预期输出：

```
smpp_connection_status{bind_name="vodafone_uk",...} 1
smpp_connection_status{bind_name="att_us",...} 1
```

所有值应为 `1`（已连接）。

响应警报

连接中断警报：

1. 检查 Web UI → SMPP → 实时状态
2. 尝试手动重新连接
3. 检查日志中的错误
4. 如果长时间中断，请联系运营商
5. 请参见 TROUBLESHOOTING.md

高失败率警报：

1. 检查日志中的错误模式
2. 回顾最近的配置更改
3. 联系运营商关于拒绝的情况
4. 检查消息格式合规性

无流量警报：

1. 检查后端队列是否有消息
2. 验证 `dest_smsc` 路由是否正确

3. 检查 TPS 限制是否过于严格
4. 回顾 `queue_check_frequency` 设置

维护程序

定期维护

每月执行：

1. 审查指标：
    - 分析消息量趋势
    - 检查交付成功率
    - 识别优化机会
2. 更新文档：
    - 记录任何配置更改
    - 更新联系信息
    - 记录运营商维护窗口
3. 凭据审计：
    - 审查所有 SMPP 密码
    - 计划凭据轮换
    - 验证 IP 白名单是否最新
4. 容量规划：
    - 审查高峰消息速率
    - 检查与 TPS 限制对比
    - 计划增长

服务重启

需要时：

- 在配置文件更改后
- 在系统更新后
- 在故障排除期间

步骤：

```
# 检查当前状态
sudo systemctl status omnimessage-smpp

# 重启服务
sudo systemctl restart omnimessage-smpp

# 验证重启
sudo systemctl status omnimessage-smpp

# 检查日志
sudo journalctl -u omnimessage-smpp -n 50
```

通过 **Web UI** 验证：

1. 访问仪表板（可能需要 30-60 秒才能上线）
2. 导航至：SMPP → 实时状态
3. 等待所有连接建立（1-2 分钟）
4. 检查日志中的错误

□置备份

在更改之前备份关键文件：

```
# 备份配置
sudo cp /opt/omnimessage-smpp/config/runtime.exs \
  /opt/omnimessage-smpp/config/runtime.exs.backup.$(date +%Y%m%d)

# 备份证书
sudo tar -czf /tmp/smpp-certs-$(date +%Y%m%d).tar.gz \
  /opt/omnimessage-smpp/priv/cert/
```

如有需要，恢复：

```
# 恢复配置
sudo cp /opt/omnimessage-smpp/config/runtime.exs.backup.YYYYMMDD \
  /opt/omnimessage-smpp/config/runtime.exs

# 重启服务
sudo systemctl restart omnimessage-smpp
```

紧急程序

完全服务中断

步骤：

1. 检查服务状态：

   ```
   sudo systemctl status omnimessage-smpp
   ```

2. 如果服务已停止，启动它：

   ```
   sudo systemctl start omnimessage-smpp
   ```

3. 检查日志以了解崩溃原因：

   ```
   sudo journalctl -u omnimessage-smpp -n 100
   ```

4. 如果无法启动：

   - 检查配置语法错误
   - 验证 SSL 证书是否存在
   - 检查磁盘空间：`df -h`
   - 检查内存：`free -h`

5. 如果未解决，请联系支持

运营商请求紧急断开连接

步骤：

1. 立即断开连接：
    - 导航至：SMPP → 实时状态
    - 找到受影响的连接
    - 点击“断开连接”
2. 记录原因：
    - 记录运营商名称
    - 记录时间和原因
    - 保存通信记录
3. 调查问题：
    - 检查最近的消息模式
    - 回顾日志中的错误
    - 确定根本原因
4. 协调解决方案：
    - 与运营商合作
    - 实施修复
    - 在重新连接之前进行测试

高流量峰值

症状：意外的高消息流量

步骤：

1. 检查 **TPS** 限制：
    - 导航至：SMPP → 实时状态
    - 验证连接没有被限流
    - 可能需要暂时增加 TPS 限制

2. 监控运营商稳定性：
    - 观察断开连接的情况
    - 检查交付成功率
3. 与后端协调：
    - 验证消息源是否合法
    - 可能需要在上游实施速率限制
4. 如有需要，扩展：
    - 可能需要额外的关实例
    - 联系支持以获取扩展建议

最佳实践

每日检查清单

- [ ] 检查所有 SMPP 连接是否已连接
- [ ] 审查错误日志以查找任何问题
- [ ] 监控消息队列是否有积压
- [ ] 检查 Prometheus/Grafana 仪表板
- [ ] 验证交付成功率 > 98%

每周任务

- [ ] 审查指标趋势
- [ ] 检查模式异常
- [ ] 测试灾难恢复程序
- [ ] 根据需要更新文档
- [ ] 审查和确认警报

每月任务

- [ ] 凭据审计
- [ ] 容量规划审查
- [ ] 更新运营商联系人
- [ ] 审查和优化 TPS 设置
- [ ] 备份配置文件

相关文档

- **CONFIGURATION.md** - 配置连接和设置
- **SOURCE_ADDRESS_WHITELIST.md** - 限制每个服务器对等体的来源地址
- **MONITORING.md** - 设置 Prometheus 警报
- **TROUBLESHOOTING.md** - 解决常见问题
- **README.md** - 系统概述
