□准测试

□目录包含使用 Benchee 的 SMS-C 系统性能基准测试。

可用□准测试

1. 原始 SMS □准测试 (`raw_sms_bench.exs`)

基准测试 `submit_message_raw` API 端点，使用真实的 SMS PDU。

特点:

- 使用真实的 SMS PDU（将您的 PDU 添加到文件中的 `@sample_pdus` 列表中）
- 通过在每次迭代前清除指纹来禁用重复检测
- 输出控制台和 HTML 报告

用法:

```
mix run benchmarks/raw_sms_bench.exs
```

输出: `benchmarks/output/raw_sms_benchmark.html`

2. 消息 API □准测试 (`message_api_bench.exs`)

基准测试各种消息 API 操作，包括插入、检索和路由。

特点:

- 测试 insert_message（简单和带路由）
- 测试 get_messages_for_smsc
- 测试 list_message_queues
- 使用测试数据预填充数据库，以实现真实场景

用法:

```
mix run benchmarks/message_api_bench.exs
```

输出: `benchmarks/output/message_api_benchmark.html`

配置

所有基准测试使用 Benchee，默认设置如下���

- 热身: 2 秒
- 时间: 10 秒
- 内存时间: 2 秒
- 启用扩展统计
- 自动生成 HTML 报告

输出

HTML 基准测试报告生成在 `benchmarks/output/` 中，包括:

- 详细的性能指标
- 比较图表
- 内存使用统计
- 统计分析

SMS-C 操作文档

← 返回主 **README**

欢迎来到 SMS-C 操作文档。本综合指南涵盖了配置、操作、监控和故障排除 SMS-C 系统的所有方面。

文档概述

入门

- 配置参考 - 完整的配置选项和示例

日常操作

- 操作指南 - 日常任务、监控和维护
- **SMS** 路由指南 - 路由管理和配置
- **API** 参考 - 完整的 API 文档及示例

性能与监控

- 性能调优 - 针对不同工作负载的优化
- 指标指南 - Prometheus 指标和监控

故障排除

- 故障排除指南 - 常见问题及解决方案

合规与监管

- **ANSSI R226** 拦截合规 - 法国合法拦截技术规范
  - 多协议前端集成 (IMS/SIP, SMPP, SS7/MAP)
  - ETSI X1/X2/X3 合法拦截接口
  - Mnesia + SQL 两层存储架构
  - 合法拦截查询的 CDR 模式

    - 加密和密码分析能力

快速链接

常见任务

- 提交消息
- 创建路由
- 检查消息状态
- 监控系统健康
- 处理投递失败

配置示例

- 消息存储与保留
- CDR 导出设置
- 隐私控制
- 高容量配置
- 地理路由
- 负载均衡
- ENUM/NAPTR 设置
- OCS 计费
- 号码翻译

监控与警报

- 关键指标
- 推荐警报
- 仪表板模板

系统架构概述

SMS-C 是一个分布式、高性能的消息路由平台，具有以下关键组件：

核心组件

- 消息存储 - 基于 Mnesia 的快速存储，具有可配置的保留和 CDR 导出
- 路由��擎 - 基于 Mnesia 的路由规则，具有前缀匹配和负载均衡
- 号码翻译 - 基于正则表达式的号码规范化，具有优先级排序
- 计费集成 - 基于路由的 OCS 在线计费
- **ENUM** 查找 - 基于 DNS 的号码路由，具有缓存
- 事件日志 - 消息生命周期跟踪
- **CDR** 导出 - 自动导出到 SQL 数据库以进行长期计费/分析

外部接口

- **REST API** - 消息提交和管理 (HTTPS)
- **Web UI** - 路由管理、消息浏览、监控
- **Prometheus** - 用于监控的指标暴露
- **OCS** - 计费/账单集成
- **DNS** - 用于路由的 ENUM/NAPTR 查找

分布与高可用性

- 多节点集群 - 分布式消息处理
- **Mnesia** 复制 - 节点间路由同步
- 自动故障转移 - 节点故障处理
- 负载均衡 - 加权路由分配

相关文档

- 性能□准 - 性能测试和结果
- **CDR** 模式参考 - 完整的 CDR 数据库模式及 SQL 示例

□□□□

□□□□

- **CPU**: 2 □□
- **RAM**: 4 GB
- □□: 50 GB (□□□□□□□□)
- □□□□: Linux (□□)□macOS (□□)
- **Erlang/OTP**: 26.x □□□
- **Elixir**: 1.15.x □□□
- **SQL** □□□: MySQL 8.0+□□MariaDB 10.5+ □ PostgreSQL 13+ (□□ CDR □□)

□□□□□□

- **CPU**: 8+ □□
- **RAM**: 16+ GB
- □□: 500+ GB SSD
- □□: 1 Gbps+
- **SQL** □□□: □□□□□□□□□□ (□□ CDR □□)

□□□□

- **80/443** - Web UI (HTTP/HTTPS)
- **8443** - API (HTTPS)
- **4369** - Erlang □□□□ (□□)
- **9100-9200** - Erlang □□ (□□)
- **9568** - Prometheus □□

□□□□□

□□

- □□□□□□: `/var/log/sms_c/` (□□) □□□□ (□□)

- **Web UI** 日志：实时日志查看器位于 `/logs`
- 事件日志：通过 API 进行逐消息事件跟踪

诊断

- 健康检查：`GET /api/status`
- 指标：`GET http://localhost:9568/metrics` (Prometheus 格式)
- 前端状态：Web UI 位于 `/frontend_status`
- 消息队列：Web UI 位于 `/message_queue`

寻求帮助

1. 查看 故障排除指南
2. 检查应用程序日志
3. 检查 Prometheus 指标以发现异常
4. 使用路由模拟器测试路由逻辑
5. 检查逐消息事件日志

版本信息

本文件的最新信息为：

- 最后更新：2025-10-30
- **SMS-C** 版本：最新开发版本
- 支持的 **Elixir**: 1.15.x - 1.17.x
- 支持的 **Erlang/OTP**: 26.x - 27.x

文档约定

在本文件中：

- 配置示例 显示典型值；请根据您的环境进行调整
- **API** 示例 使用 `curl` 命令行格式
- **IP** 地址和域名 仅为示例；请替换为您的实际值

- 指标名称遵循 Prometheus 命名约定
- 所有时间戳均为 UTC，除非另有说明

快速开始

1. 配置：通过 `config/runtime.exs` 进行配置 - 参见配置参考
2. 初始路由：通过 Web UI 或配置文件创建路由规则 - 参见 SMS 路由指南
3. 提交测试消息：使用 API 或 Web UI - 参见 API 参考
4. 监控：设置 Prometheus 抓取 - 参见指标指南

文档反馈

本文件与 SMS-C 代码库一起维护。如需更正或改进，请更新 `docs/` 目录中的 markdown 文件。

ANSSI R226 □□□□□□

□□□□□□ □□□□□□□ OmniMessage □□□□□□□ (SMSc) □□□□□□□□ R226-3 □ R226-7 □□□□□□ ANSSI R226 □□□□□□□□□

□□□ □□□□□□□

□□□□□□ □□□□□□□□□□□□ (ANSSI)

□□□ R226 - □□□□□□□□□□□□

---

1. □□□□□□

1.1 □□□□□□□□

□□□□□□ OmniMessage SMSc (□□□□□□□)
□□□□□□ □□□□□□□
□□□□□□ □□□□□□□□□□□□□□
□□□□□□ REST API (HTTPS)□□□□□□ (SMPP□IMS□SS7/MAP □□□□□□□)
□□□□□□ □□□□□□□□
□□□□ Elixir/Erlang□Phoenix □□□Mnesia□MySQL/PostgreSQL

□□□□

□□□□□□

- □□□□□□□□□□□□□□ REST API
- □□□□□□□□□□ SMPP□IMS□SS7/MAP □□
- □□□□□□□□□□□□
- □□□□□□□□□□
- □□□□□□□□□□□□
- □□□□□□□ (CDR) □□□□□□
- □□□□ 1,750 □□□/□□□□□□□□□□□□ 1.5 □□□□

□□□□□

- 活动消息队列： Mnesia 内存数据库，支持磁盘持久化
  - 主要存储：RAM 以实现超快访问（亚毫秒延迟）
  - 磁盘备份：`disc_copies` 模式写入磁盘以进行崩溃恢复
  - 自动恢复：消息在系统重启后仍然存在
  - 保留：可配置（默认 24 小时），然后自动清理
- 长期 **CDR** 归档： MySQL/PostgreSQL 数据库（与消息队列分开）
  - 消息交付、过期、失败或拒绝时写入 CDR
  - SQL 数据库仅用于 CDR 导出/归档，而不用于活动消息操作
  - 对消息路由没有性能影响（异步写入）
- 双层架构的好处：
  - 活动队列：快速（1,750 条消息/秒），没有 SQL 瓶颈
  - CDR 归档：长期保留（数月/数年），用于计费和合法拦截
  - 清晰分离：消息操作从不接触 SQL
- 支持集群以实现高可用性（Mnesia 在节点之间复制）

网络接口：

- **REST API**： HTTPS（端口 8443）用于外部前端通信
- 控制面板： HTTPS（端口 8086）用于基于 Web 的管理
- 前端协议： SMPP、IMS、SS7/MAP（通过外部网关应用）
- 数据库： MySQL/PostgreSQL 用于 CDR 存储

路由和处理：

- 动态短信路由，支持运行时配置更新
- 基于���缀的匹配（呼叫/被叫号码）
- 源 SMSC 和类型过滤
- 基于优先级和权重的负载均衡
- 号码转换和标准化
- ENUM (E.164 号码映射) DNS 查找支持
- 自动回复和消息丢弃能力
- 每条路由的收费控制（CGRates 集成）

 完整架构和功能记录在 **README.md**

1.2 □□□□

1.2.1 □□□□

□□□□□□□□

- OmniMessage SMSc □□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□
  - □ MSISDN□□□□□□□
  - □□ MSISDN□□□□□□□
  - □ IMSI□□□□□□□□□□□
  - □□ IMSI
  - □□□□□□□□□□□
  - □□ PDU□□□□□□□□□□□
  - TP-DCS□□□□□□□□□□□
  - □□□□□GSM7□UCS-2□8 □□Latin-1□
  - □□□□□□□□□□□□□□□
  - □□□□□□□UDH□□□

□□□□□□□□□

- □□□□□□□□□□□CDR□□□□□□□□□□□□□□
  - □□ ID□□□□□□□□
  - □□□□□□□ MSISDN□
  - □□□□□□□□ MSISDN□
  - □□□□□□□□□□□□□□□□□□□
  - □□□□□□□□□□□□□□□□□
  - □□□□□□□□□□□□□□□□□□□□□□□□□□□□
  - □□□□□□□□□□□□□□□□□□□□

- 死信标志（重试耗尽指示符）

 完整 **CDR** 模式记录在 **CDR_SCHEMA.md**

消息队列访问：

- 实时消息队列监控
- REST API 端点用于消息检索
- 数据库查询用于历史消息搜索
- 过滤能力：
  - 电话号码（源/目标）
  - SMSC 网关
  - 时间范围
  - 消息状态
  - 交付尝试

 完整 **API** 文档在 **API_REFERENCE.md**

1.2.2 数据处理能力

消息存储架构（双层系统）：

SMSc 使用复杂的双层存储架构，将操作消息处理与长期归档分开：

层 **1**：活动消息队列（**Mnesia**）

- 目的：实时消息路由和交付操作
- 技术： Erlang Mnesia 分布式数据库
- 存储模式：内存与 disc_copies 备份
  - 主要存储在 RAM 中以实现最大速度
  - 自动磁盘同步以进行崩溃恢复
  - 消息在系统重启后持久存在
- 性能：亚毫秒读/写操作
- 保留：短期（默认 24 小时），可配置
- 清理：自动归档到 CDR 数据库，然后从 Mnesia 中删除
- 操��：所有消息队列操作（插入、更新、交付状态、路由）
- 关键特性：在消息路由/交付期间绝不会查询 SQL 数据库

层 **2**: **CDR** 归档 (**MySQL/PostgreSQL**)

- 目的：用于计费、分析和合法拦截的长期存储
- 技术：传统 SQL 数据库 (MySQL 或 PostgreSQL)
- 写入触发器：仅在消息达到最终状态时写入 CDR:
  - 消息成功交付
  - 消息过期（超过有效期）
  - 消息永久失败
  - 消息被路由规则拒绝
- 写入模式：异步批量写入（对消息路由性能没有影响）
- 保留：长期（数月到数年），可根据监管要求配置
- 操作：历史查询、报告、合规、合法拦截
- 访问： SQL 查询、REST API (未来)、CSV/JSON 导出

关键架构优势:

1. 性能：活动路由操作从不接触 SQL (没有数据库瓶颈)
2. 可扩展性： Mnesia 每秒处理 1,750 条以上消息，无 SQL 开销
3. 可靠性： disc_copies 模式确保崩溃时没有消息丢失
4. 合规性： CDR 数据库提供永久审计跟踪
5. 关注点分离：操作数据与归档数据清晰分开

消息生命周期:

```
1. □息提交 → 存储在 Mnesia (RAM + 磁盘备份)
2. □息路由 → Mnesia 查询 (超快)
3. □息交��/过期 → CDR 写入 SQL (异步)
4. 24 小时后 → □息从 Mnesia 中删除 (清理工作)
5. CDR 保留在 SQL 中 → 可用于合法拦截查询 (数年)
```

数据保留和检索:

- 可配置的消息正文保留或删除以保护隐私
- 二进制数据保留（原始 PDU 存储在 Mnesia 和 CDR 中）
- 完整文本搜索能力（如果在 CDR 数据库上启用）
- 索引 CDR 字段以快速合法拦截查询

前端跟踪：

- 实时跟踪外部 SMSC 前端（SMPP、IMS、MAP 网关）
- 前端注册与心跳监控
- 健康状态跟踪（活动/过期）
- 正常运行/停机历史
- IP 地址和主机名跟踪
- 前端特定配置日志记录

1.2.3 分析能力

实时监控：

- Web UI 仪表板显示：
  - 活动消息队列
  - 消息提交和交付
  - 路由决策和网关选择
  - 前端网关状态
  - 系统资源利用率
- Prometheus 指标集成用于操作监控
- 性能指标（吞吐量、延迟、成功率）

 完整监控指南在 **OPERATIONS_GUIDE.md** 指标文档在 **METRICS.md**

历史分析：

- CDR 数据库可通过以下方式查询：
  - 时间范围
  - 呼叫/被叫方号码
  - 消息状态
  - SMSC 网关
  - 交付尝试
  - 消息内容（如果启用，则进行全文搜索）
- 统计分析能力：
  - 按小时/天/月的消息量
  - 按路由的成功/失败率

    - 平均交付时间
    - 多部分消息分析
    - 失败交付模式

订阅者跟踪：

- 按电话号码（MSISDN）的消息历史
- 基于 IMSI 的跟踪（当从 IMS/MAP 前端可用时）
- 通话模式分析
- 通信方关联
- 时间分析（消息频率、时间模式）

网络分析：

- 路由性能指标
- 网关可用性和健康状况
- 消息流可视化
- 集群节点分布（多节点部署）
- 交付尝试分析
- 重试模式分析

号码智能：

- E.164 号码标准化
- 从号码前缀识别国家/地区
- 号码转换和重写规则
- ENUM DNS 查找以获取路由智能
- 基于前缀的路由决策

 号码转换指南在 **number_translation_guide.md** 路由指南在 **sms_routing_guide.md**

1.3 反制能力

1.3.1 隐私保护机制

通信机密性：

- HTTPS/TLS □□ REST API □□
- □□□□□□□□□□
- □□□□□□□□□□□ TLS□
- □□□□□□□□□□□□□□□

□□□□□

- Web UI □□□□
- API □□□□□□
- □□□□□□□□
- □□□□□□□□□

□□□□□

- □□□□□□□□□□
- □□□□/□□□□
- □□□□□□□
- □□□□□□□
- □□□□□□□□□□□□□□□□

1.3.2 □□□□□□

□□□□□

- □□□□□□□□□□□□□□□
- □□□□□□□ UI □□□□□□□□□□
- □□□□□□□□□□□□□□□□□
- CDR □□□□□□□□□□□ NULL □□□□□

□□□□□□

- MySQL □□□□□□ENCRYPTION='Y'□
- PostgreSQL □□□□□□□□□
- □□□□□□□□□□
- □□□□□□□□□□
- □□□□□□□□□□

□□□□□

- □□□□□□□□□□
- TLS □□□□
- □□□□□□□
- □□□□□□□□□□
- □□ Erlang □□□□□□□□□□□

1.4 □□□□□Mnesia + SQL □□□□

□□

OmniMessage SMSc □□□□□□□□□□□

1. 消息通过 REST API 到达 → 插入到 Mnesia RAM + 磁盘备份
2. 路由引擎查询 Mnesia → 即时响应（内存访问）
3. 外部网关轮询消息 → Mnesia 查询（内存访问）
4. 网关更新交付状态 → Mnesia 更新（内存 + 磁盘）
5. 交付/过期后 → 消息标记为清理
6. 清理工作（默认 24 小时） → 消息从 Mnesia 中删除

关键性能特性：

- 在活动消息路由/交付期间绝不会进行 **SQL** 数据库查询
- SQL 完全绕过操作消息处理
- 这消除了传统 SMS-C 瓶颈（数据库 I/O）

层 **2**: **SQL** 数据库用于 **CDR** 导出/归档

什么是 **CDR**（通话详细记录）？

- 消息元数据和内容的永久审计记录
- 写入 MySQL 或 PostgreSQL 数据库
- 用于计费、分析、合规和合法拦截

何时写入 **CDR**： CDR 记录仅在消息达到最终状态时创建：

- □ □□□□□□
- □ □□□□□□□□□□□□交付）
- □ □□□□□□□□□□□□□□□□□□
- □ □□□□□□□□□□□□□□□□□

如何写入 **CDR**：

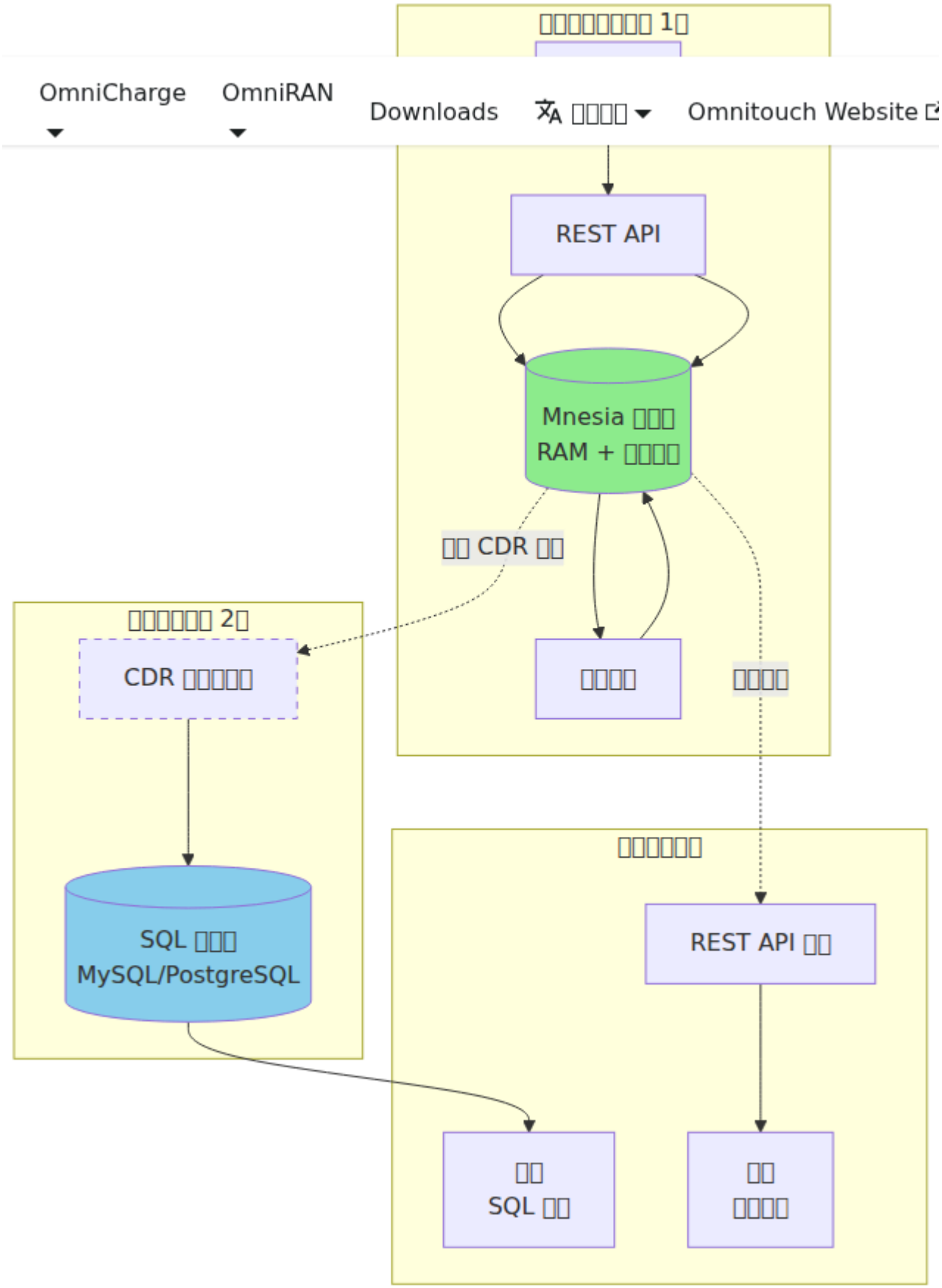
- 异步批量写入： CDR 在后台工作进程中写入
- 无阻塞：消息路由从不等待 SQL 写入
- 批量插入：多个 CDR 分组（默认 100）一起写入
- 刷新间隔：默认 100ms（可配置）
- 错误处理：失败的 CDR 写入记录，消息处理继续

```
# □ config/runtime.exs □□□□
config :sms_c,
  batch_insert_batch_size: 100,          # CDR □□□□□□□□
  batch_insert_flush_interval_ms: 100    # □□□□
```

SQL □□□□□□□□

- □ □□□□□□□□□□□□□□□
- □ □□□□□□□□□□□□
- □ □□□□□□□□□□□□
- □ □□□□□□□ CDR □□□□□□□□
- □ □□□□□□□□□□□□□□□□/□□□□□□
- □ □□□□□□□□□□□□□

□□□


OmniCharge
OmniRAN
Downloads
Omnitouch Website
REST API
Mnesia
RAM +
CDR
CDR
SQL
MySQL/PostgreSQL
REST API
SQL

-

- □□□□□□□□□□□□□
- □□□□□□□□□□□□□
- □□□□□□□□□□□ SQL□

□□□□□□□□

□□□□□□< **24** □□□□

- □□ Mnesia □□□REST API □□□
- □□□□
- □□□□□□□□□
- □□□□□□□□

□□□□□□> **24** □□□□

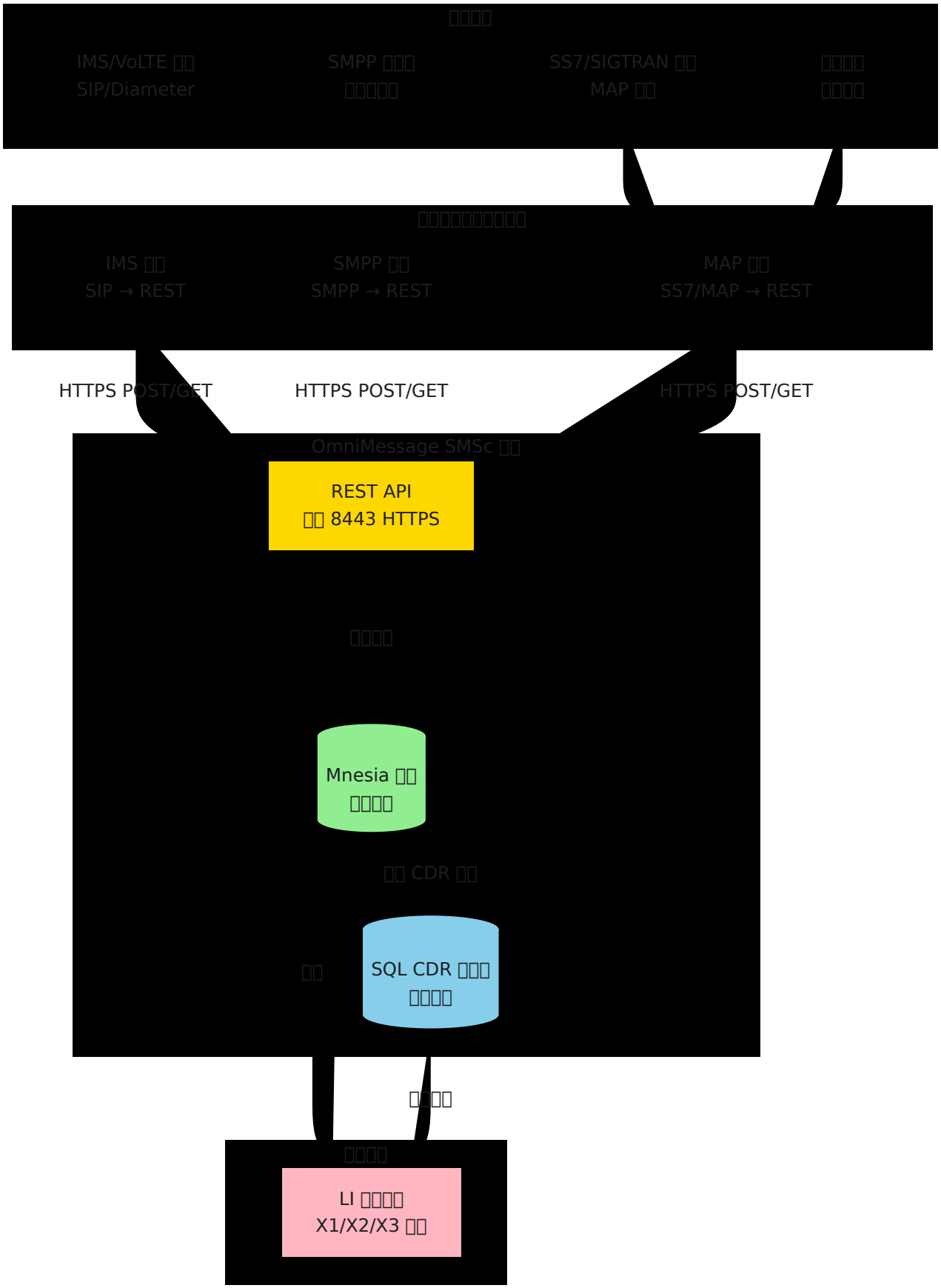
- □□ SQL □□□□□□□CDR □□
- □□ SQL □□□□
- □□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□

□□□□□□□

1. □□□□□□□ disc_copies □□□□□□□□□□□□□
2. □□□□□□□□ CDR □ SQL □□□□□□□□□
3. □□□ □□□□□□□□□□□□□□□□
4. □□□□ □□□□□□Mnesia□+ □□□□□□SQL□□□□□

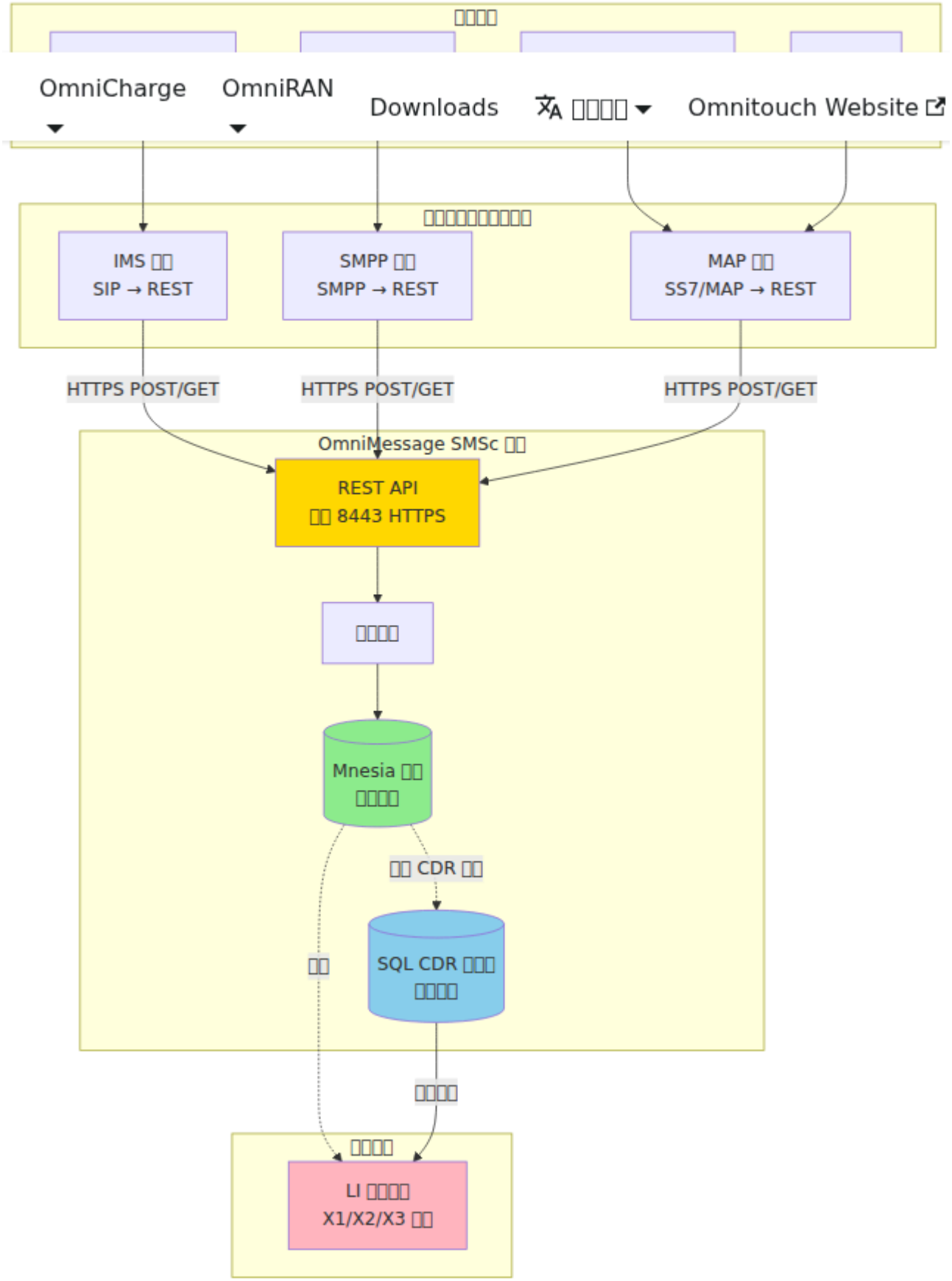
1.5 □□□□□□□□□

OmniMessage SMSc □□□□□□□□□□□□□□□□□□□□ REST API □□□□□□□□□□□□□□□□□□□□


IMS/VoLTE
SIP/Diameter
SMPP
SS7/SIGTRAN
MAP
IMS
SIP → REST
SMPP
SMPP → REST
MAP
SS7/MAP → REST
HTTPS POST/GET
HTTPS POST/GET
HTTPS POST/GET
OmniMessage SMSc
REST API
8443 HTTPS
Mnesia
SQL CDR
X1/X2/X3

1. IMS/SIP 前端集成

IMS 网络使用 SIP 协议进行 SMS-over-IP 消息传递。IMS 网关在 SIP 和 SMSc REST API 之间进行转换。


OmniCharge
OmniRAN
Downloads
Omnitouch Website
IMS
SIP → REST
SMPP
SMPP → REST
MAP
SS7/MAP → REST
HTTPS POST/GET
HTTPS POST/GET
HTTPS POST/GET
OmniMessage SMSc
REST API
8443 HTTPS
Mnesia
CDR
SQL CDR
LI
X1/X2/X3

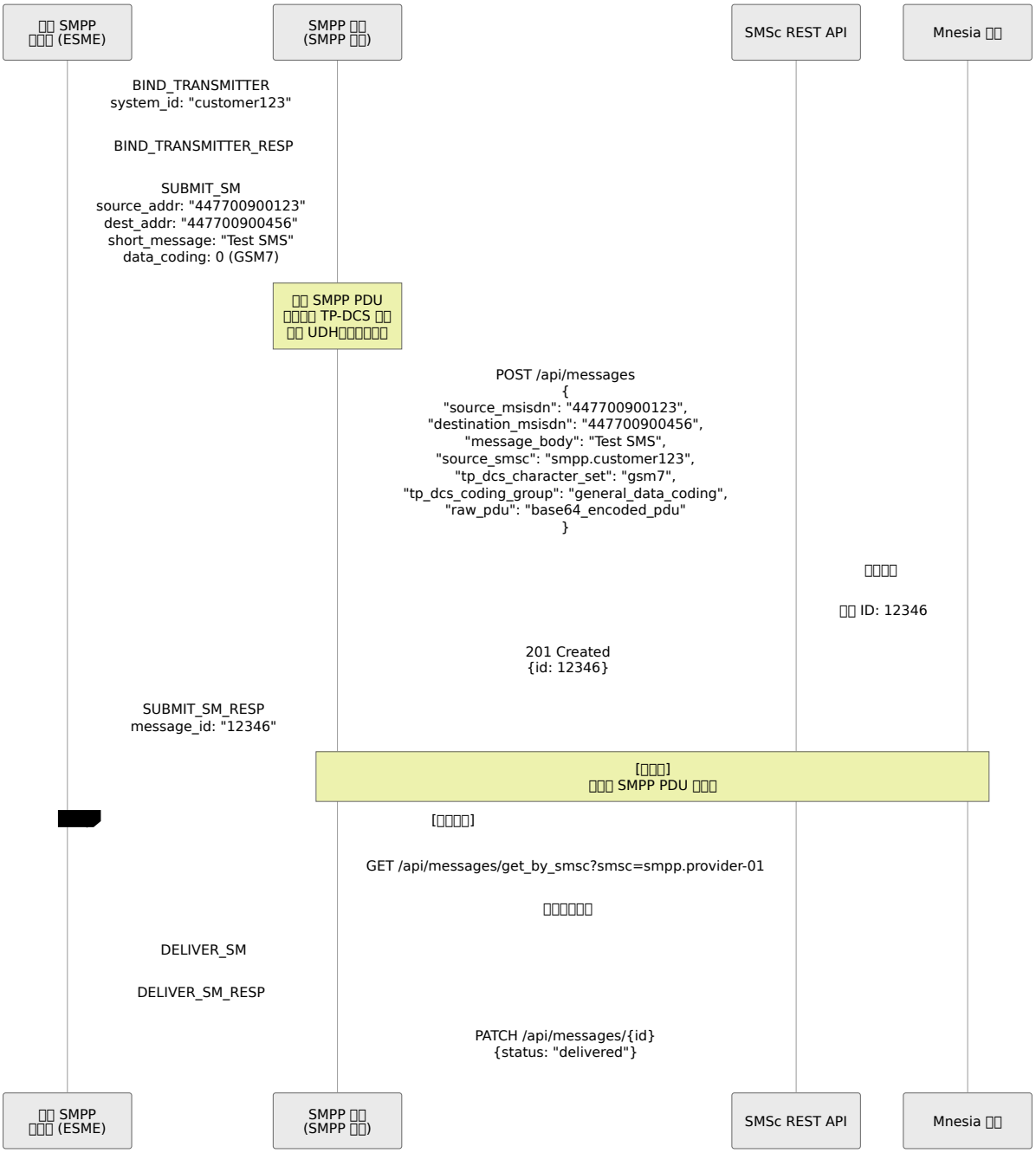
IMS

- /IMSI IMS

- P-Asserted-Identity SIP □
- SIP Call-ID □□□□
- IMS □□□□□P-Access-Network-Info□
- IMS HSS □□□□□□□□□□

2. SMPP □□□□

SMPP □□□□□□□□□□□□□□□□□□□□□□□□□□SMPP □□□□□□ PDU □ SMPP □□□□□□ REST API □□□

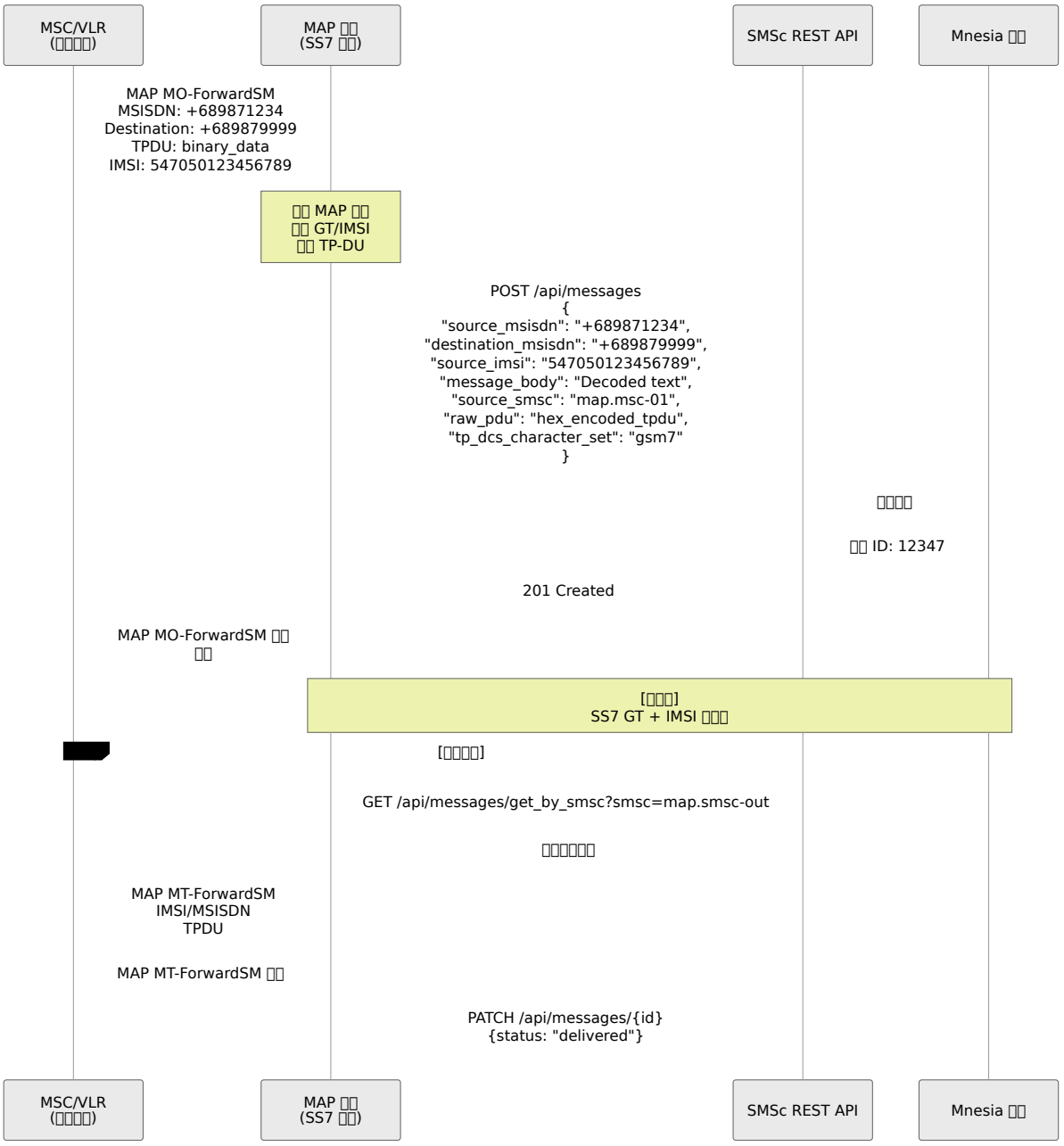


SMPP □□□□□□□

- □□□ SMPP PDU□□□□□□□□□□
- □□□□□□□□DCS□□□□□□
- □□□□□□□UDH□□□□□□□
- ESME system_id□□□□□□□
- TON/NPI □□□□□□□
- □□□□□□□

3. SS7/MAP □□□□

□□□□□□□□□□□ SS7 MAP □□□□□□□□MAP □□□ SS7 □□□ REST API □□□□□□□□



SS7/MAP □□□□□□□□

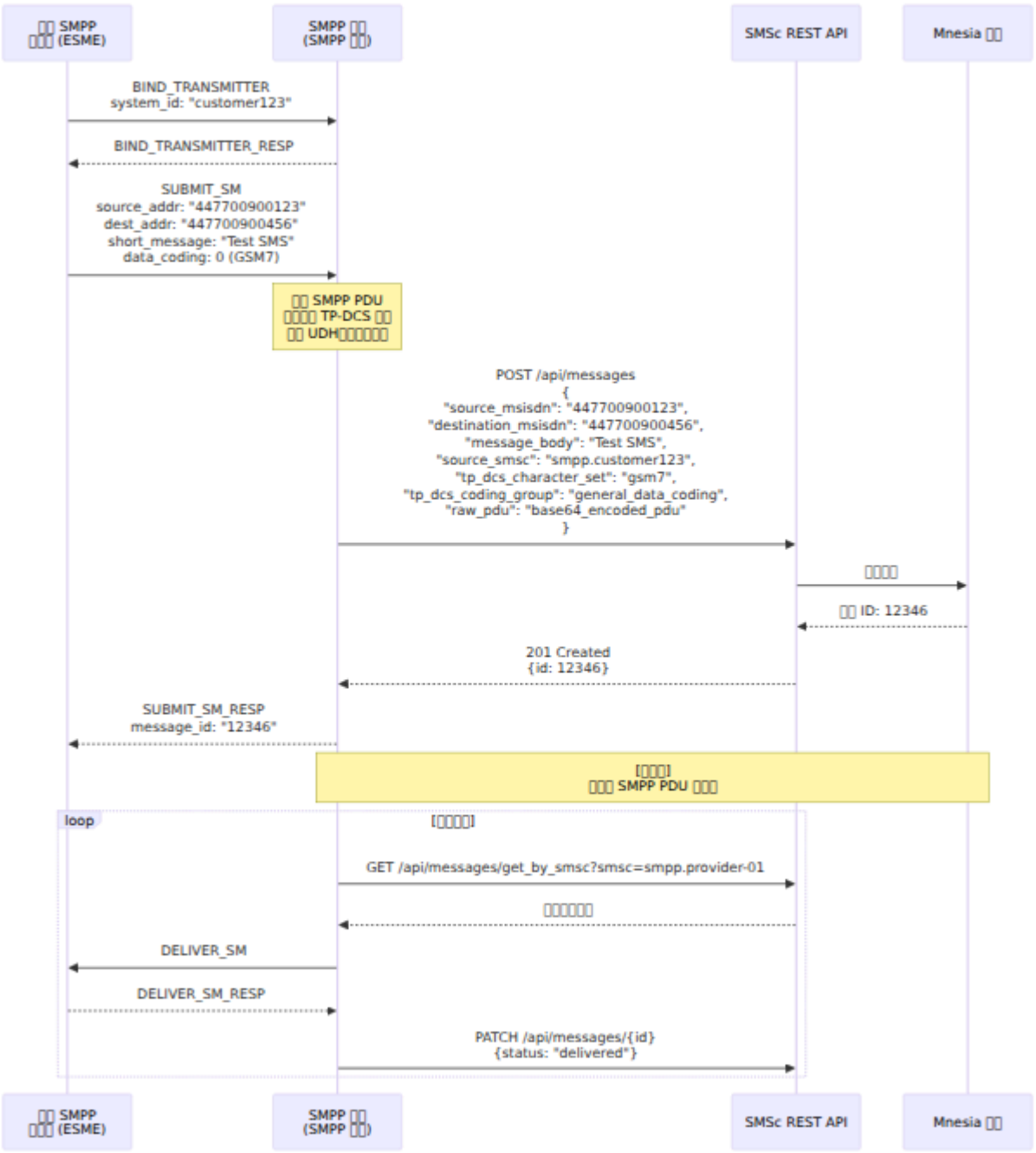
- MAP 消息中的 IMSI
- 全局标题 (GT) 地址
- MSC/VLR 地址（网络元素识别）
- SCCP 呼叫/被叫方地址
- MAP 操作码
- TP-用户数据的二进制格式

所有协议的统一拦截

合法拦截的关键好处：无论使用何种协议（IMS/SIP、SMPP 或 SS7/MAP），所有消息都在 SMSc 核心中汇聚，形成标准化的数据结构，使得：

1. 协议无关监控：单一拦截点捕获所有消息类型
2. 统一 **CDR** 格式：所有协议写入相同的 CDR 模式
3. 跨协议关联：跟踪跨协议边界的消息
4. 完整元数据保留：协议特定字段在 CDR 中保留

数据流摘要：



CDR 中的协议识别:

- `source_smsc` 字段指示前端协议（例如，“ims.gateway-01”、“smpp.customer123”、“map.msc-01”）
- 允许按协议类型进行过滤和分析
- 合法拦截查询可以针对特定协议或所有协议

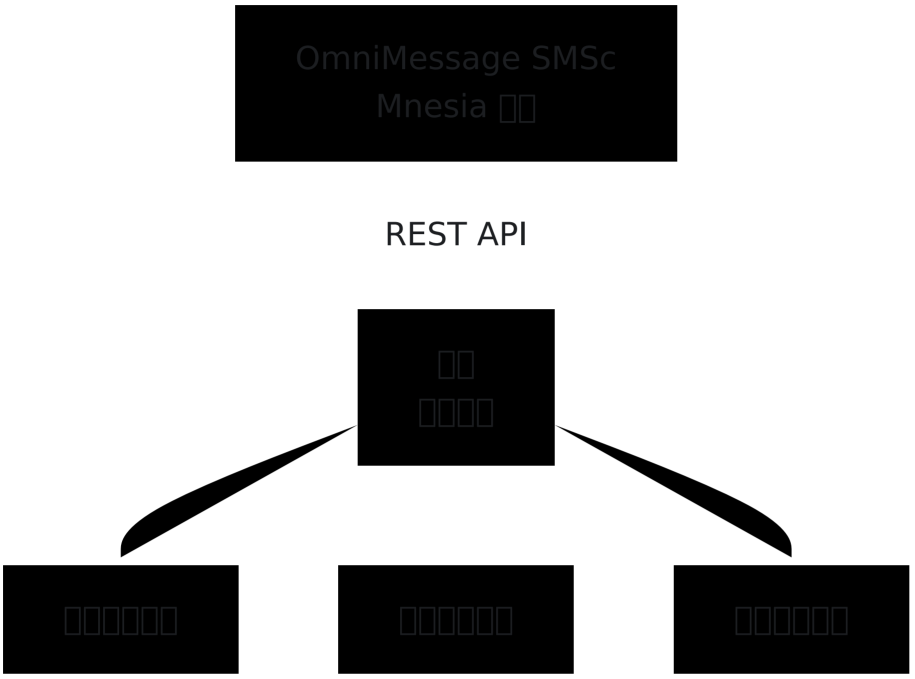
1.6 合法拦截的技术架构

合法拦截集成点

双层存储架构为合法拦截提供多个访问点，优化了实时监控（Mnesia）和历史分析（SQL）。

1. 最近消息的 REST API 访问（Mnesia）：

访问 Mnesia 队列中的活动消息（通常为最近 24 小时）：



实时拦截的 API 端点：

- `GET /api/messages` - 列出活动消息并进行过滤
- `GET /api/messages/{id}` - 获取特定消息详细信息（来自 Mnesia）
- `GET /api/messages/get_by_smsc?smsc=X` - 按网关获取消息
- 所有查询都命中 Mnesia（内存）以获得即时响应

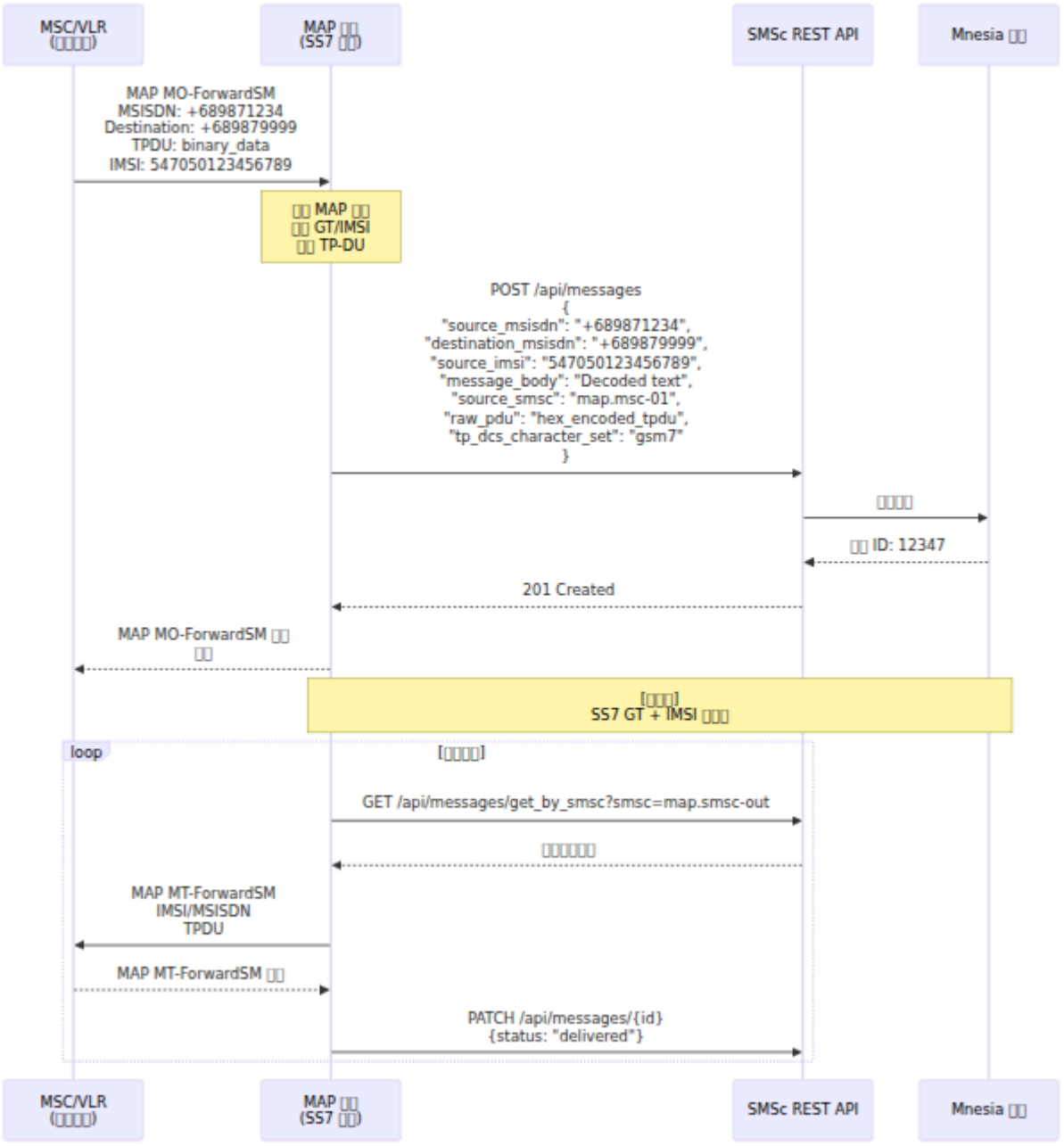
注意：这些端点查询活动 Mnesia 消息队列，提供���当前正在处理或最近交付的消息（在保留期内）的访问。

查询参数：

- 按源/目标 MSISDN 过滤
- 按时间范围过滤
- 按 SMSC 网关过滤
- 按消息状态过滤
- 支持排序和分页

2. CDR 数据库直接访问历史消息（SQL）：

访问 SQL 数据库中的归档消息（所有已交付/过期/失败的消息）：



```
{
"source_msisdn": "+689871234",
"destination_msisdn": "+689879999",
"source_imsi": "547050123456789",
"message_body": "Decoded text",
"source_smsc": "map.msc-01",
"raw_pdu": "hex_encoded_tpdu",
"tp_dcs_character_set": "gsm7"
}
```

直接 **SQL** 访问:

- 授权系统的只读数据库凭据
- 对 `cdrs` 表的 SQL 查询（永久审计跟踪）
- 访问方法：标准 SQL 客户端（mysql、psql、DBeaver 等）
- 数据源：仅归档消息（不包括活动队列）
- 索引字段以提高搜索效率：
  - `calling_number`（索引） - 源电话号码
  - `called_number`（索引） - 目标电话号码

    - `message_id`（索引） - 唯一消息标识符
    - `submission_time`（索引） - 消息进入系统的时间
    - `status`（索引） - 最终交付状态
    - `dest_smsc`（索引） - 用于交付的网关

注意：CDR 数据库包含所有处理消息的永久记录。这是历史合法拦截查询的主要数据源（数月/年的数据）。

3. 实时消息馈送（PubSub）：

- Phoenix PubSub 集成用于实时事件
- 消息提交通知
- 消息交付通知
- 消息状态更改事件
- 可配置的事件过滤标准
- WebSocket 支持以进行实时监控

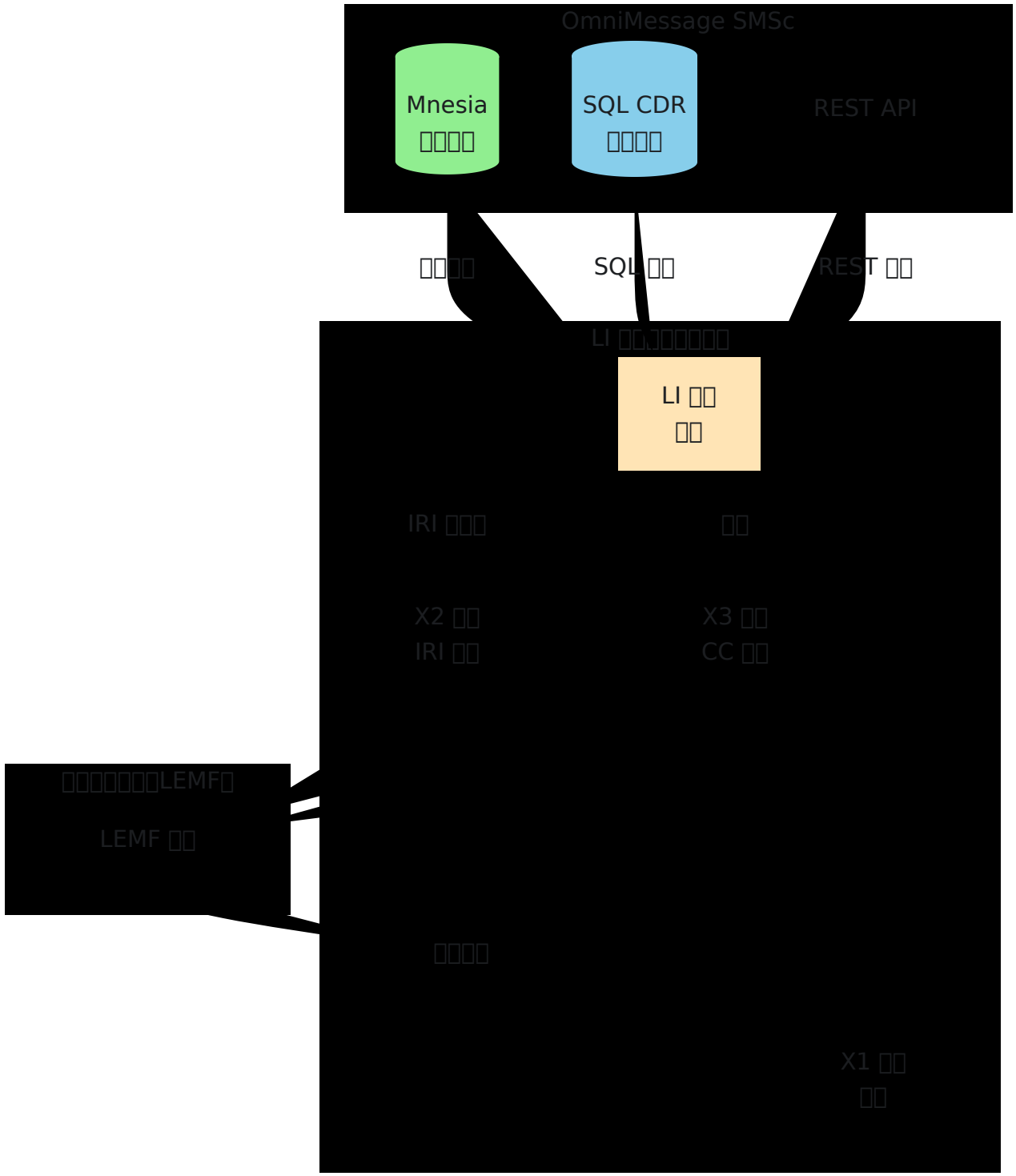
4. 批量导出接口：

- CDR 记录的 CSV 导出
- 程序访问的 JSON 导出
- 可配置的导出字段
- 基于时间范围的导出
- 保护隐私的导出（可选消息正文排除）

ETSI 合法拦截标准接口

OmniMessage SMSc 为实施符合 ETSI 的合法拦截接口提供基础。虽然 SMSc 核心不原生实现 X1/X2/X3 接口，但它提供所有必要的数据访问点，可以与外部合法拦截中介功能 (LIMF) 系统集成。

标准 ETSI LI 接口：



接口概述

X1 接口 - 管理接口

- 用于 管理和配置拦截目标和拦截参数
- 用于 LEMF → LIMF的连接
- 用于
  - 添加/删除拦截目标（通过MSISDN或IMSI）

    - □□□□□□□□□□□□□□
    - □□□□□□□□□□□□□□□□□□□□
    - □□□□□□□
- □ **SMSc** □□□□
    - LIMF □□□□□□□□□□□□□□□
    - LIMF □□ SMSc CDR/API □□□□□□□□
    - LIMF □□ X1 □□□□□□□□□□

X2 □□ - IRI□□□□□□□□□□□□

- □□□ □□□□□□□□□□□□□
- □□□ LIMF → LEMF□□□□
- □□□□□ □□ ETSI TS 102 232-x □ XML/ASN.1
- □□ **SMSc CDR** □□□□
    - □□ ID
    - □□□□□□ MSISDN□
    - □□□□□□□ MSISDN□
    - IMSI□□□□□□□□□□□□
    - □□□□□
    - □□□□□
    - □□□□□□□□/□□/□□□
    - □□□□
    - SMSC □□□□□□/□□□
    - □□□□□□□□□□
- □ **SMSc** □□□□
    - LIMF □□ CDR □□□□□□□□□□□□□□□
    - LIMF □ CDR □□□□□ ETSI IRI □□
    - LIMF □□ X2 □ IRI □□□ LEMF

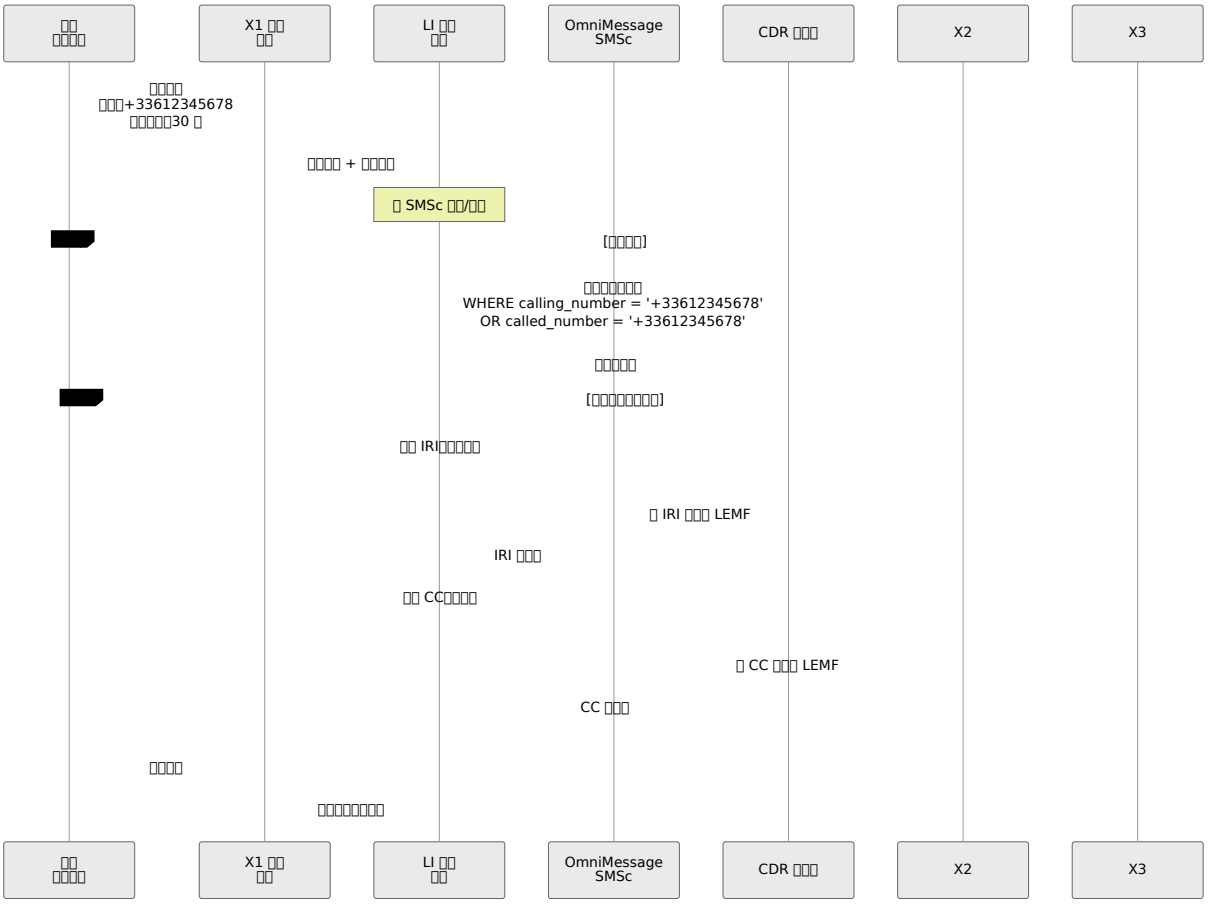
X3 □□ - CC□□□□□□□□□

- □□□ □□□□□□□□□□□□□□
- □□□ LIMF → LEMF□□□□
- □□□□□ □□ ETSI TS 102 232-x □□□
- □□ **SMSc** □□□□

- □□□□□□□□□□□□□
- □□ PDU□□□□□ SMS □□□
- □□□□□
- □□□□□□□
- TP-DCS □□
- □□□□□□□UDH□

- □ **SMSc** □□□□□
  - LIMF □ CDR `message_body` □□□□□□□□□□
  - LIMF □□□□□ PDU □□□□□□□□□□
  - LIMF □ ETSI CC □□□□□□□
  - LIMF □□ X3 □ CC □□□ LEMF

□□□□□



SMSc □□□□□□ LI □□□

SMSc □□□□	X2 (IRI)	X3 (CC)	CDR □□
□□ ID	□ □□ ID	□ □□	`message_id`
□□□□	□ A □	-	`calling_number`
□□□□	□ B □	-	`called_number`
□□□□	□ □□□	-	`submission_time`
□□□□	□ □□	-	`delivery_time`
□□	□ □□	-	`status`
□□□□	-	□ □□	`message_body`
□□ PDU	-	□ □□□	(Mnesia/CDR)
□ SMSC	□ □□□□	-	`source_smsc`
□□ SMSC	□ □□□□	-	`dest_smsc`
IMSI	□ □□□ ID	-	(□□□□)

LIMF □□□□□

□□ **1**□□□□□

- LIMF □□□□ CDR □□□□□ 1-60 □□
- SQL □□□ X1 □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□
- □□□□□ LI □□□□□□□□□□

□□ **2**□□□□□□□

- SMSc PubSub □□□□□□
- LIMF □□□□□□□

- LIMF 根据目标列表进行过滤
- 合法拦截的近乎零延迟
- 需要自定义集成开发

选项 **3**：混合架构

- 最近消息：实时 PubSub 馈送（< 24 小时）
- 历史消息：CDR 数据库轮询
- 最优平衡延迟和可靠性

拦截触发机制

于目标的拦截：

- 电话号码匹配（MSISDN）
- 基于 IMSI 的目标（当可用时）
- 可配置的观察列表
- 目标隔离的数据库视图
- 按目标标识符过滤的 API

于事件的拦截：

- 所有与特定号码的消息
- 通过特定 SMSC 网关的消息
- 具有特定特征的消息（多部分、交付失败等）
- 地理路由（通过 ENUM 或前缀匹配）

于时间的拦截：

- CDR 查询中的日期/时间范围过滤
- 保留期强制执行
- 自动归档旧消息
- 可配置的数据保留策略

合法拦截的示例 **SQL** 查询：

```sql
-- 取目标号码的所有消息
SELECT * FROM cdrs
WHERE calling_number = '+33612345678'
   OR called_number = '+33612345678'
ORDER BY submission_time DESC;

-- 取特定时间窗口内的消息
SELECT * FROM cdrs
WHERE (calling_number = '+33612345678' OR called_number =
'+33612345678')
  AND submission_time BETWEEN '2025-11-01 00:00:00' AND '2025-11-
30 23:59:59'
ORDER BY submission_time;

-- 取两方之间的对话
SELECT * FROM cdrs
WHERE (calling_number = '+33612345678' AND called_number =
'+33687654321')
   OR (calling_number = '+33687654321' AND called_number =
'+33612345678')
ORDER BY submission_time;
```

---

2. 加密和密码分析能力

2.1 加密能力概述

OmniMessage SMSc 实施了加密机制，以保护通信和敏感数据。部分记录了所有符合 ANSSI 要求的加密能力。

2.2 传输层加密

2.2.1 TLS/SSL 实现

支持的协议：

- TLS 1.2 (RFC 5246)
- TLS 1.3 (RFC 8446) - 推荐
- SSL 2.0/3.0: 不支持（已知漏洞）

- TLS 1.0/1.1□□□□□□□□□□

□□□

- Erlang/OTP SSL/TLS □□□□□□□□□□
- Cowboy Web □□□□□ TLS
- Phoenix □□ HTTPS □□

□□□□□

□□□□ Erlang/OTP □□□□□□□□□□□□□□□□

□□ - TLS 1.3□

- TLS_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256
- TLS_CHACHA20_POLY1305_SHA256

□□ - TLS 1.2□

- ECDHE-RSA-AES256-GCM-SHA384
- ECDHE-RSA-AES128-GCM-SHA256
- DHE-RSA-AES256-GCM-SHA384
- DHE-RSA-AES128-GCM-SHA256

□□□□□

- □□ ECDHE/DHE □□□□□□□□□□□□□□PFS□
- □□□□ Diffie-Hellman □□2048 □□□□
- □□□□□□□□□
- □□□□□□□□□□SNI□

□□□□□

- □□ X.509 □□
- RSA □□□□□2048 □□□□4096 □□□
- □□ ECDSA
- □□□□□

- 自签名证书（仅用于开发）
- 外部 CA 集成

TLS 配置位置：

```
# config/runtime.exs
config :api_ex,
  api: %{
    enable_tls: true,
    tls_cert_path: "priv/cert/omnitouch.crt",
    tls_key_path: "priv/cert/omnitouch.pem"
  }
```

 完整配置参考在 **CONFIGURATION.md**

应用：

- HTTPS 用于 REST API（端口 8443）
- HTTPS 用于 Web 控制面板（端口 8086）
- 数据库连接（MySQL/PostgreSQL 通过 TLS）

2.3 数据加密静态

2.3.1 数据库加密

MySQL/MariaDB 加密：

- 表级加密支持
- AES-256 加密算法
- 透明数据加密（TDE）

```
-- 为 CDR 表启用加密
ALTER TABLE cdrs ENCRYPTION='Y';
```

PostgreSQL 加密：

- 透明数据加密支持

- □□□□□□□□
- □□□□□pgcrypto □□□

2.3.2 Mnesia □□□□

Mnesia □□□□

- □□□□□□□□□□□□□□□□□
- □□□□□□□□□□□LUKS□dm-crypt□
- □□ Erlang VM □□□□□□□□□

2.3.3 □□□□□□□

□□□□□□□

- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□0600□+ □□□□□□□
- □□□□□□□□□□□□□□□□□□□□
- CDR □□□□□□□□□□□□□□□□

□□□□□

- TLS □□□□□□□□□ `priv/cert/`
- □□□□□□□□□□□□□□□□□
- □□□□□□□□□

2.4 □□□□□□□□□

2.4.1 API □□□□

REST API □□□□

- HTTPS/TLS □□□□□□□□
- □□□□□□□□□SMSc □□□□□□□□
- □□ IP □□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□

□□□□□

- □□□□□□□□□□□□□□□□IP□□□□□
- □□□□□□□□□□
- □□□□□□□□□□□90 □□□□
- □□□□□□□□

2.4.2 □□□□□□□□

□□□□□□□□□

- □□□/□□□□□□□
- □□ TLS/SSL □□
- □□ IP □□□□□
- □□□□□□□□□□□RBAC□

□□□

```
# config/runtime.exs
config :sms_c, SmsC.Repo,
  username: "omnitouch",
  password: "omnitouch2024",  # □□□□□□□□□□□
  hostname: "localhost",
  ssl: true  # □□ TLS □□□□□□□
```

□□□□□□□

```
-- □□□□□□□□□□□□□□□
CREATE USER 'li_readonly'@'%' IDENTIFIED BY 'secure_password';
GRANT SELECT ON sms_c.cdrs TO 'li_readonly'@'%';

-- □□□□□□□□□□□□□□□□
CREATE USER 'analytics'@'%' IDENTIFIED BY 'secure_password';
GRANT SELECT (id, message_id, calling_number, called_number,
              source_smsc, dest_smsc, submission_time,
delivery_time,
              status, delivery_attempts)
ON sms_c.cdrs TO 'analytics'@'%';
```

2.5 加密算法详细信息

2.5.1 哈希算法

在 **Erlang/OTP** 中可用：

- SHA-256、SHA-384、SHA-512（推荐）
- SHA-1（已弃用，仅用于遗留兼容性）
- MD5（已弃用，不用于安全）
- BLAKE2（在现代 OTP 版本中可用）

用途：

- 消息指纹（重复检测）
- 数据完整性验证
- 审计日志完整性

2.5.2 对称加密

可用算法：

- AES（高级加密标准）
  - AES-128-GCM
  - AES-256-GCM
  - AES-128-CBC
  - AES-256-CBC
- ChaCha20-Poly1305

密钥大小：

- 128 位（最小）
- 256 位（推荐）

用途：

- TLS 会话加密
- 数据库静态加密
- 可选消息正文加密

2.5.3 非对称加密

支持的算法：

- RSA（2048 位最小，4096 位推荐）
- ECDSA（椭圆曲线数字签名算法）
  - P-256、P-384、P-521 曲线
- Ed25519（EdDSA）

用途：

- TLS 证书身份验证
- 数字签名
- 密钥交换

2.6 短信协议安全

2.6.1 短信消息编码

字符编码支持：

- GSM 7 位（标准短信编码）
- UCS-2（Unicode，16 位）
- 8 位二进制数据
- Latin-1

TP-DCS（数据编码方案）：

- 消息类别指示
- 压缩标志
- 编码组规范
- 字符集识别

没有原生短信加密：

- 短信协议不提供端到端加密
- 消息内容在 SMSc 层可访问
- 允许合法拦截所需

2.6.2 协议安全考虑

SMPP 协议（外部前端）：

- SMPP 级别的用户名/密码身份验证
- 支持 TLS（SMPP 通过 TLS）
- 绑定身份验证

IMS 协议（外部前端）：

- 基于 SIP 的消息
- SIP 身份验证机制
- 与 IMS 核心网络安全的集成

SS7/MAP 协议（外部前端）：

- SS7 网络安全
- MAP 协议身份验证
- SCCP/TCAP 层安全

注意：协议特定的安全性在外部前端网关中实现，而不是在 SMSc 核心中。

2.7 密码分析和安全评估能力

2.7.1 协议分析工具

内置调试能力：

- 综合日志系统
- 消息流跟踪
- API 请求/响应日志
- 数据库查询日志
- 错误和异常跟踪

外部集成：

- 标准日志输出（stdout/文件）
- 支持网络分析的 PCAP 捕获

- 用于取证的数据库查询日志
- Prometheus 指标导出

2.7.2 漏洞评估考虑

已知限制：

- 短信协议本质上未加密（设计使然，允许合法拦截）
- 配置文件中的数据库凭据（应使用秘密管理）
- 支持自签名证书（仅用于开发/测试）

安全加固建议：

- 使用强 TLS 密码套件
- 实施数据库连接加密
- 使用外部秘密管理（Vault、AWS Secrets Manager）
- 定期对 Erlang/OTP 和依赖项进行安全更新
- 对 API 端口进行防火墙限制
- 对前端访问进行 IP 白名单管理

安全测试：

- 定期进行依赖项漏洞扫描
- 支持渗透测试
- TLS 配置验证
- 数据库安全审计
- 访问控制审查

2.8 密钥管理□础设施

2.8.1 密钥生成

TLS 证书生成：

```
# 生成私钥（RSA 4096 位）
openssl genrsa -out omnitouch.pem 4096

# 生成证书签名请求
openssl req -new -key omnitouch.pem -out omnitouch.csr

# 自签名证书（开发）
openssl x509 -req -days 365 -in omnitouch.csr -signkey
omnitouch.pem -out omnitouch.crt

# 生产：从受信任的 CA 获取证书
```

随机数生成：

- Erlang/OTP CSPRNG（密码学安全伪随机数生成器）
- 系统熵池（/dev/urandom）
- 用于会话密钥、ID、令牌的强随机性

2.8.2 密钥存储和保护

私钥存储：

- 文件系统具有限制权限（0600）
- 存储在 `priv/cert/` 目录中
- PEM 格式（可选加密）
- 安全备份程序

密钥轮换：

- TLS 证书续订（建议每年）
- 数据库凭据轮换
- API 令牌轮换（如果实现）

2.8.3 密钥分发

证书分发：

- 手动安装在 `priv/cert/`
- 配置文件引用

- 可能支持 ACME 协议（Let's Encrypt）

对称密钥分发：

- 通过外部数据库凭据进行带外密钥交换
- TLS 中的 Diffie-Hellman 密钥协商
- 不进行明文密钥传输

2.9 合规性和标准

本节记录了适用于所有支持协议的 SMS 处理的国际电信标准、监管框架和安全规范的合规性。

2.9.1 □于 SS7/MAP 协议的 SMS 合规性

3GPP 和 **ETSI** 标准：

- **3GPP TS 23.040**：短消息服务（SMS）的技术实现 - 核心 SMS 协议规范
- **3GPP TS 23.038**：字母表和语言特定信息 - 字符编码（GSM7、UCS-2）
- **3GPP TS 29.002**：移动应用部分（MAP）规范 - SS7 信令用于 SMS
- **3GPP TS 23.003**：编号、寻址和识别 - MSISDN、IMSI 格式
- **ETSI TS 100 901**：点对点短消息服务规范
- **ETSI TS 100 902**：小区广播短消息服务规范

SS7 信令标准：

- **ITU-T Q.711-Q.716**：信令连接控制部分（SCCP）
- **ITU-T Q.771-Q.775**：事务能力应用部分（TCAP）
- **ITU-T Q.701-Q.710**：消息传输部分（MTP）第 1-3 层
- **ETSI EN 300 356**：信令系统 No.7 - ISDN 用户部分（ISUP）

SS7/MAP 的安全标准：

- **GSMA FS.07**：SS7 和直径信令安全 - 基线安全控制
- **GSMA FS.11**：SS7 安全监控指南
- **3GPP TS 33.117**：一般安全保证要求目录
- **ETSI TS 133 210**：网络域安全 - IP 网络层安全

SS7/MAP 的合法拦截：

- **ETSI TS 101 671**：合法拦截（LI）；���信流量合法拦截的交接接口
- **ETSI TS 102 232-1**：合法拦截（LI）；交接规范 - 第 1 部分：LI 管理的交接接口
- **3GPP TS 33.107**：3G 网络的合法拦截架构和功能

2.9.2 □于 IMS 协议的 SMS 合规性

3GPP IMS 标准：

- **3GPP TS 23.228**：IP 多媒体子系统（IMS） - 第二阶段架构
- **3GPP TS 24.229**：IP 多媒体呼叫控制协议 - SIP 和 SDP 程序
- **3GPP TS 24.341**：支持基于 IP 网络的 SMS - 第三阶段规范
- **3GPP TS 23.204**：支持通过通用 3GPP IP 接入的短消息服务（SMS） - 第二阶段
- **3GPP TS 29.228**：IP 多媒体（IM）子系统 Cx 和 Dx 接口

IMS 安全标准：

- **3GPP TS 33.203**：3G 安全；基于 IP 的服务的接入安全（IMS AKA）
- **3GPP TS 33.210**：3G 安全；网络域安全（NDS）；IP 网络层安全（IPsec）
- **3GPP TS 33.310**：网络域安全（NDS）；身份验证框架（AF）
- **ETSI TS 133 203**：基于 IP 的服务的接入安全

SIP 协议标准：

- **RFC 3261**：SIP：会话发起协议 - 核心规范
- **RFC 3428**：SIP 即时消息扩展 - MESSAGE 方法
- **RFC 3325**：SIP 的私有扩展以进行声明的身份
- **RFC 5765**：对等系统中的安全问题和解决方案

IMS 的合法拦截：

- **ETSI TS 102 232-5**：合���拦截（LI）；交接规范 - 第 5 部分：针对 IP 多媒体子系统服务的服务无关 LI
- **3GPP TS 33.107**：合法拦截要求和架构
- **3GPP TS 33.108**：合法拦截（LI）的交接接口

2.9.3 SMPP 协议合规性

SMPP 规范：

- **SMPP v3.4**：短消息对等协议规范 - 行业标准
- **SMPP v5.0**：扩展的 SMPP 协议，具有增强功能

SMPP 安全指南：

- **TLS** 通过 **SMPP**：SMPP 连接的传输层安全（SMPP 通过 TLS）
- **SMPP** 绑定身份验证：系统 ID 和密码身份验证
- □于 **IP** 的访问控制：网络级限制用于 SMPP 连接

互操作性标准：

- **GSM 03.40 (ETSI TS 100 901)**：短消息点对点（PP）的技术实现
- **GSM 03.38 (ETSI TS 100 900)**：字母表和语言特定信息
- **GSM 04.11 (ETSI TS 100 942)**：移动无线接口上的点对点 SMS 支持

消息编码合规性：

- **ITU-T T.50**：国际字母表第 5 号（IA5）
- **ISO/IEC 8859-1**：Latin-1 字符编码
- **ISO/IEC 10646**：通用字符集（UCS-2/UTF-16）

2.9.4 加密标准合规性

TLS 和网络安全：

- **NIST SP 800-52**：TLS 实现的选择、配置和使用指南
- **NIST SP 800-131A**：加密算法和密钥长度的过渡
- **RFC 7525**：TLS 和 DTLS 的安全使用建议
- **RFC 8446**：传��层安全 (TLS) 协议版本 1.3

加密算法标准：

- **FIPS 197**：高级加密标准 (AES)
- **FIPS 180-4**：安全哈希标准 (SHA-2 家族)
- **NIST SP 800-38D**：块密码操作模式的建议：GCM 模式
- **RFC 7539**：IETF 协议的 ChaCha20 和 Poly1305

密钥管理：

- **NIST SP 800-57**：密钥管理建议
- **RFC 5280**：互联网 X.509 公钥基础设施证书和 CRL 配置文件

2.10 密码分析抵抗力

2.10.1 设计原则

抵御密码分析：

- 不使用自定义/专有的加密算法
- 仅使用行业标准、经过同行评审的算法
- 定期更新加密库的安全性
- 弃用弱算法
- 使用认证加密（GCM、Poly1305）

2.10.2 操作安全

密钥轮换：

- TLS 证书续订程序
- 会话密钥轮换（每个会话用于 TLS）
- 数据库凭据轮换政策

监控和检测：

- 失败的身份验证日志
- 证书过期监控
- TLS 握手失败日志
- 加密失败的异常检测
- 安全事件警报

---

3. 拦截控制和授权

3.1 合法拦截的访问控制

行政授权：

- 配置需要系统管理员访问
- CDR 查询的数据库级访问控制
- API 访问通过 IP/身份验证限制
- 所有访问的审计日志记录

法律框架集成：

- 拦截令跟踪（外部系统集成）
- 目标标识符授权列表（数据库视图）
- 有时间限制的查询（SQL WHERE 子句）
- 通过访问策略自动强制执行

3.2 数据保留和隐私

保留政策：

- 活动消息保留：可配置（默认 24 小时在 Mnesia 中）
- CDR 保留：可配置（典型 6 个月到 2 年）
- 从 Mnesia 自动归档到 SQL
- 自动清除旧 CDR（基于 cron）

隐私保护：

- 消息正文交付后删除选项
- 消息正文从 UI/导出中排除
- 数据库静态加密
- 访问日志记录和监控
- 最小数据收集原则

配置：

```
# config/runtime.exs
config :sms_c,
  # Mnesia 息保留在归档之前
  message_retention_hours: 24,

  # 为隐私在交付后删除息正文
  delete_message_body_after_delivery: false,  # 设置为 true 以启用隐私模式

  # CDR 写入控制
  cdr_enabled: true,

  # 批量归档设置
  batch_insert_batch_size: 100,
  batch_insert_flush_interval_ms: 100
```

 查看所有保留设置的 **CONFIGURATION.md**

3.3 执法的交接接口

标准接口:

1. REST API 访问:

- HTTPS 端点用于消息检索
- JSON 格式数据交换
- 身份验证和授权
- 按目标标准过滤查询

2. 直接数据库访问:

- 只读 SQL 凭据
- 标准 SQL 查询
- CDR 表��问
- 索引搜索能力

3. 批量导出:

- CSV 导出格式
- JSON 导出格式

- 基于时间范围的导出
- 可配置字段选择

交付格式：

IRI（拦截相关信息）：

- CDR 元数据字段：
    - 消息 ID
    - 呼叫/被叫号码
    - 时间戳（提交、交付、过期）
    - 状态
    - 交付尝试
    - SMSC 路由信息
    - 节点信息（集群跟踪）

CC（通信内容）：

- 消息正文（文本内容）
- 原始 PDU 数据
- 编码信息
- 多部分消息组装

导出示例：

```
# 为执法导出 CSV
mysql -u li_readonly -p -D sms_c -e "
SELECT
  message_id,
  calling_number,
  called_number,
  message_body,
  submission_time,
  delivery_time,
  status
FROM cdrs
WHERE (calling_number = '+33612345678' OR called_number =
'+33612345678')
  AND submission_time BETWEEN '2025-11-01' AND '2025-11-30'
ORDER BY submission_time
" --batch --silent | sed 's/\t/,/g' > interception_report.csv
```

---

4. 系统安全与完整性

4.1 应用安全

Elixir/Erlang 安全性:

- Erlang VM 隔离和沙箱
- 进程隔离和监督
- 崩溃恢复和容错
- 无缓冲区溢出漏洞（受管理的运行时）

依赖管理:

- 依赖版本锁定（mix.lock）
- 安全漏洞扫描
- 定期更新依赖
- 最小依赖足迹

4.2 网络安全

网络暴露：

- 最小暴露的端口：
  - 8443（HTTPS REST API）
  - 8086（HTTPS 控制面板）
- 建议进行防火墙配置
- 对前端访问进行 IP 白名单管理
- DMZ 部署用于面向互联网的服务

网络分段：

- 单独的管理网络
- 隔离的数据库网络
- 前端网关网络分离
- 集群通信网络（Erlang 分布）

4.3 监控和入侵检测

日志能力：

- 结构化应用日志
- 可配置的日志级别
- 日志轮换和归档
- Syslog 集成支持
- 集中式日志记录（兼容 ELK 堆栈）

安全事件监控：

- 失败的身份验证尝试
- 不寻常的消息模式
- 数据库连接失败
- TLS 握手失败
- 系统资源异常

指标和警报：

- Prometheus 指标导出
- 消息吞吐量监控
- 错误率跟踪
- 系统资源利用率
- 自定义警报规则

 完整监控文档在 **OPERATIONS_GUIDE.md** 和 **METRICS.md**

4.4 高可用性和灾难恢复

集群支持：

- Erlang 分布式集群能力
- Mnesia 在节点之间复制
- 自动故障转移
- 节点发现和加入

数据冗余：

- Mnesia 在所有集群节点上的 disc_copies
- SQL 数据库复制（MySQL/PostgreSQL 原生）
- CDR 备份程序
- 配置备份

恢复程序：

- 数据库备份和恢复
- Mnesia 表恢复
- 配置恢复
- 节点替换程序

---

5. □□□□

5.1 □□□□

□□□□□□□□□□□□□□□□

- **README.md** - □□□□□□□□□□□
- **CONFIGURATION.md** - □□□□□□□
- **API_REFERENCE.md** - REST API □□
- **OPERATIONS_GUIDE.md** - □□□□□□□□
- **CDR_SCHEMA.md** - □□□□□□□□□□□□
- **sms_routing_guide.md** - □□□□□□□
- **number_translation_guide.md** - □□□□□□
- **METRICS.md** - Prometheus □□□□□□
- **PERFORMANCE_TUNING.md** - □□□□□
- **TROUBLESHOOTING.md** - □□□□□□□□□□

5.2 □□□□

- □□□□□□□□ [□□□□□□□□]
- □□□□□□□□ [□□□□□□□□]
- □□□□□□ [□□□□□□□□]
- **Erlang/OTP** □□□□□□ □□□□□□□□

5.3 □□□□

- **ANSSI R226** □□□□□□ □□□□
- □□□□□□□□□□ □□□□□□□□□□□□□□
- □□□□□□□□□□ □□□□□□□□ GDPR □□

---

6. □□□□

□□□□□/□□□□□□□□

- 公司名称: Omnitouch Network Services Pty Ltd
- 地址: PO BOX 296, QUINNS ROCKS WA 6030, AUSTRALIA
- 联系人：合规团队
- 电子邮件: compliance@omnitouch.com.au

技术安全联系人：

- 姓名：合规团队
- 电子邮件: compliance@omnitouch.com.au

法律/合规联系人：

- 姓名：合规团队
- 电子邮件: compliance@omnitouch.com.au

附录

附录 A：带有拦截点的短信消息流

A.1 出站短信流（移动终止）

Parse error on line 11: ...提交短信) Note over ----------^ Expecting 'ACTOR', got 'NEWLINE'

试

SMS-C API 参考

← 返回文档索引 | 主 README

完整的 SMS-C REST API 端点参考，包括请求/响应示例。

目录

API 概述

SMS-C REST API 提供了对消息提交、路由和管理功能的编程访问。

□础 URL

```
https://api.example.com:8443/api
```

□□□□: 8443 (□□□)
□□: HTTPS (□□□□□□□ TLS)

□□□□

□□□□□□□□□□ JSON□

```
Content-Type: application/json
```

API □□□□

□□ API □□□ 1□□□□□□□□□□□□□ URL □□□□□□□

```
https://api.example.com:8443/api/v2/...
```

□□

TLS □□□□□□□□□

□□□□□□□ TLS □□□□□□□□□

```
curl --cert client.crt --key client.key \
  https://api.example.com:8443/api/status
```

API □□□□

□□ `X-API-Key` □□□□□□ API □□□□□

```
curl -H "X-API-Key: your_api_key_here" \
  https://api.example.com:8443/api/status
```

IP 白名单

在防火墙级别限制 API 访问到受信任的 IP 地址。

常见响应格式

成功响应

```
{
  "data": {
    ...
  }
}
```

错误响应

```
{
  "errors": {
    "detail": "错误信息，描述发生了什么问题"
  }
}
```

列表响应

```
{
  "data": [
    {...},
    {...}
  ]
}
```

状态端点

用于监控和负载均衡的健康检查端点。

获取 API 状态

请求:

```
GET /api/status
```

响应 (200 OK):

```
{
  "status": "ok",
  "application": "OmniMessage",
  "timestamp": "2025-10-30T12:34:56Z"
}
```

示例:

```
curl https://api.example.com:8443/api/status
```

用例:

- 负载均衡器健康检查
- 监控系统连接性
- 服务可用性验证

消息队列 API

核心消息提交和管理端点。

列出消息

从队列中检索消息。

请求:

```
GET /api/messages
```

可选头：

- `smsc: frontend_name` - 按目标 SMSC 过滤
- `include-unrouted: true|false|1|0` - 包括没有位置注册的消息（默认：false）
  - `false`（默认）：仅返回具有明确路由或位置注册的消息
  - `true`：包括没有位置注册的消息（向后兼容模式）

查询参数：

- `status` - 按状态过滤：`pending`，`delivered`，`expired`，`dropped`
- `source_smsc` - 按源 SMSC 过滤
- `dest_smsc` - 按目标 SMSC 过滤
- `limit` - 限制结果（默认：100，最大：1000）
- `offset` - 分页偏移量

响应 (200 OK)：

```
{
  "data": [
    {
      "id": 12345,
      "source_msisdn": "+15551234567",
      "destination_msisdn": "+447700900000",
      "message_body": "Hello World",
      "source_smsc": "api_client",
      "dest_smsc": "uk_gateway",
      "status": "pending",
      "send_time": "2025-10-30T12:00:00Z",
      "deliver_time": null,
      "delivery_attempts": 0,
      "inserted_at": "2025-10-30T12:00:00Z"
    }
  ]
}
```

示例：

获取特定 SMSC 的待处理消息（仅具有明确路由或位置）：

```
curl -H "smsc: uk_gateway" \
  https://api.example.com:8443/api/messages
```

获取包括未路由消息的待处理消息（向后兼容）：

```
curl -H "smsc: uk_gateway" \
  -H "include-unrouted: true" \
  https://api.example.com:8443/api/messages
```

获取所有已发送的消息:

```
curl "https://api.example.com:8443/api/messages?status=delivered&limit=50"
```

获取单条消息

检索特定消息的详细信息。

请求:

```
GET /api/messages/:id
```

响应 (200 OK):

```
{
  "data": {
    "id": 12345,
    "source_msisdn": "+15551234567",
    "destination_msisdn": "+447700900000",
    "message_body": "Hello World",
    "source_smsc": "api_client",
    "dest_smsc": "uk_gateway",
    "source_imsi": null,
    "dest_imsi": null,
    "message_parts": 1,
    "message_part_number": 1,
    "tp_data_coding_scheme": "00",
    "tp_user_data_header": null,
    "status": "pending",
    "send_time": "2025-10-30T12:00:00Z",
    "deliver_time": null,
    "expires": "2025-10-31T12:00:00Z",
    "deadletter": false,
    "delivery_attempts": 0,
    "charge_failed": false,
    "deliver_after": "2025-10-30T12:00:00Z",
    "raw_data_flag": false,
    "raw_sip_flag": false,
    "raw_pdu": null,
    "inserted_at": "2025-10-30T12:00:00Z",
    "updated_at": "2025-10-30T12:00:00Z"
  }
}
```


示例:

```
curl https://api.example.com:8443/api/messages/12345
```

提交消息（同步）

提交一条消息并立即接收消息 ID。

请求:

```
POST /api/messages
Content-Type: application/json
```

主体:

```
{
  "source_msisdn": "+15551234567",
  "destination_msisdn": "+447700900000",
  "message_body": "Hello World",
  "source_smsc": "api_client"
}
```

可选字段:

- `dest_smsc` - 覆盖路由决策
- `send_time` - 安排未来交付（ISO 8601）
- `message_parts` - 多部分消息的总部分
- `message_part_number` - 部分编号（1 索引）
- `tp_data_coding_scheme` - SMS DCS（默认: "00"）
- `source_imsi` - 源订阅者 IMSI
- `dest_imsi` - 目标订阅者 IMSI

响应 (201 Created):

```
{
  "data": {
    "id": 12345,
    "source_msisdn": "+15551234567",
    "destination_msisdn": "+447700900000",
    "message_body": "Hello World",
    "source_smsc": "api_client",
    "dest_smsc": "uk_gateway",
    "status": "pending",
    "send_time": "2025-10-30T12:00:00Z",
    "inserted_at": "2025-10-30T12:00:00Z"
  }
}
```

示例：

```
curl -X POST https://api.example.com:8443/api/messages \
  -H "Content-Type: application/json" \
  -d '{
    "source_msisdn": "+15551234567",
    "destination_msisdn": "+447700900000",
    "message_body": "Hello World",
    "source_smsc": "api_client"
  }'
```

性能：~70 消息/秒，平均响应时间 14ms

使用场景：

- 需要立即获取消息 ID
- 处理消息/秒
- 需要立即确认

提交消息（异步）

以高吞吐量提交消息（批处理）。

请求：

```
POST /api/messages/create_async
Content-Type: application/json
```

主体：与同步端点相同

响应 (202 Accepted)：

```
{
  "data": {
    "status": "accepted",
    "message": "□息已排队处理"
  }
}
```

示例：

```
curl -X POST
https://api.example.com:8443/api/messages/create_async \
  -H "Content-Type: application/json" \
  -d '{
    "source_msisdn": "+15551234567",
    "destination_msisdn": "+447700900000",
    "message_body": "批量通知□息",
    "source_smsc": "bulk_api"
  }'
```

性能：~4,650 消息/秒，平均响应时间 0.22ms

延迟：消息在 100ms 内出现在数据库中（可配置）

使用场景：

- 高容量批量消息（> 100 msg/sec）
- 不需要在 API 响应中获取消息 ID
- 吞吐量比即时确认更重要

更新消息

部分更新消息字段。

请求：

```
PATCH /api/messages/:id
Content-Type: application/json
```

主体：

```
{
  "dest_smsc": "alternate_gateway",
  "deliver_after": "2025-10-30T14:00:00Z"
}
```

可更新字段：

- `dest_smsc` - 更改目标
- `deliver_after` - 延迟交付
- `message_body` - 更新消息文本
- `status` - 更改状态

响应 (200 OK)：

```
{
  "data": {
    "id": 12345,
    "dest_smsc": "alternate_gateway",
    "deliver_after": "2025-10-30T14:00:00Z",
    ...
  }
}
```

示例：

```
curl -X PATCH https://api.example.com:8443/api/messages/12345 \
  -H "Content-Type: application/json" \
  -d '{
    "dest_smsc": "backup_gateway"
  }'
```

标记消息为已送达

将消息标记为成功送达。

请求：

```
POST /api/messages/:id/mark_delivered
Content-Type: application/json
```

主体：

```
{
  "dest_smsc": "uk_gateway"
}
```

响应 (200 OK):

```
{
  "data": {
    "id": 12345,
    "status": "delivered",
    "deliver_time": "2025-10-30T12:05:30Z",
    "dest_smsc": "uk_gateway",
    ...
  }
}
```

示例:

```
curl -X POST
https://api.example.com:8443/api/messages/12345/mark_delivered \
  -H "Content-Type: application/json" \
  -d '{
    "dest_smsc": "uk_gateway"
  }'
```

用例：在成功送达后由前端系统调用

增加交付尝试

增加重试计数器并应用指数退避。

请求:

```
PUT /api/messages/:id
```

响应 (200 OK):

```
{
  "data": {
    "id": 12345,
    "delivery_attempts": 2,
    "deliver_after": "2025-10-30T12:08:00Z",
    ...
  }
}
```

退避计算：

```
deliver_after = now + 2^(delivery_attempts) minutes
```

示例：

```
curl -X PUT https://api.example.com:8443/api/messages/12345
```

用例：在交付失败后由前端调用以安排重试

删除消息

从队列中删除消息。

请求：

```
DELETE /api/messages/:id
```

响应 (204 No Content)

示例：

```
curl -X DELETE https://api.example.com:8443/api/messages/12345
```

警告：删除消息会永久移除它们。请谨慎使用。

原始 SMS PDU API

以原始 PDU（协议数据单元）提交 SMS 消息，以最大限度地兼容旧系统。

提交原始 SMS（同步）

请求：

```
POST /api/messages_raw
Content-Type: application/json
```

主体：

```
{
  "pdu": "0001000B916407007009F0000004D4F29C0E",
  "source_smsc": "legacy_system"
}
```

PDU 格式：十六进制编码的 SMS TPDU（传输协议数据单元）

响应 (201 Created):

```
{
  "data": {
    "id": 12346,
    "source_msisdn": "+447700900000",
    "destination_msisdn": "+447700900000",
    "message_body": "Test",
    "source_smsc": "legacy_system",
    "raw_pdu": "0001000B916407007009F0000004D4F29C0E",
    ...
  }
}
```

示例：

```
curl -X POST https://api.example.com:8443/api/messages_raw \
  -H "Content-Type: application/json" \
  -d '{
    "pdu": "0001000B916407007009F0000004D4F29C0E",
    "source_smsc": "legacy_system"
  }'
```

提交原始 **SMS**（异步）

请求:

```
POST /api/messages_raw/async
Content-Type: application/json
```

主体: 与同步相同

响应 (202 Accepted):

```
{
  "data": {
    "status": "accepted",
    "message": "PDU 已排队处理"
  }
}
```

示例:

```
curl -X POST https://api.example.com:8443/api/messages_raw/async \
  -H "Content-Type: application/json" \
  -d '{
    "pdu": "0001000B916407007009F0000004D4F29C0E",
    "source_smsc": "legacy_gateway"
  }'
```

PDU 处理

系统自动：

1. 使用 SMS 标准解码 PDU（3GPP TS 23.040）
2. 提取电话号码、消息文本、DCS
3. 检测交付报告（CP-ACK、RP-ACK 等）
4. 如有需要，执行 IMSI 到 MSISDN 的查找
5. 应用路由规则
6. 存储原始 PDU 以供参考

交付报告检测：

- CP-ACK、CP-ERROR - 连接协议确认
- RP-ACK、RP-ERROR、RP-SMMA - 中继协议响应
- 交付报告被记录但不会作为消息存储

位置管理 API

管理用于移动终止消息交付的订阅者位置信息。

列出位置

请求：

```
GET /api/locations
```

响应 (200 OK):

```
{
  "data": [
    {
      "id": 1,
      "msisdn": "+15551234567",
      "imsi": "001001000000001",
      "location": "msc1.region1.example.com",
      "ran_location": "cell_tower_12345",
      "imei": "123456789012345",
      "ims_capable": true,
      "csfb": false,
      "registered": true,
      "expires": "2025-10-30T13:00:00Z",
      "user_agent": "Samsung Galaxy",
      "inserted_at": "2025-10-30T12:00:00Z",
      "updated_at": "2025-10-30T12:00:00Z"
    }
  ]
}
```


□□□

```
curl https://api.example.com:8443/api/locations
```

□□□□

□□□

```
GET /api/locations/:id
```

□□ (200 OK):

```
{
  "data": {
    "id": 1,
    "msisdn": "+15551234567",
    "imsi": "001001000000001",
    ...
  }
}
```

示例:

```
curl https://api.example.com:8443/api/locations/1
```

创建/更新位置

根据 IMSI（唯一标识符）创建新位置或更新现有位置。

请求:

```
POST /api/locations
Content-Type: application/json
```

主体:

```
{
  "msisdn": "+15551234567",
  "imsi": "001001000000001",
  "location": "msc1.region1.example.com",
  "ran_location": "cell_tower_12345",
  "imei": "123456789012345",
  "ims_capable": true,
  "csfb": false,
  "registered": true,
  "expires": "2025-10-30T13:00:00Z",
  "user_agent": "Samsung Galaxy"
}
```

必填字段：

- `imsi` - 唯一的订阅者标识符
- `msisdn` - 电话号码

可选字段：

- `location` - MSC/VLR 地址
- `ran_location` - 基站/扇区 ID
- `imei` - 设备标识符
- `ims_capable` - IMS VoLTE 能力
- `csfb` - 电路交换回退标志
- `registered` - 当前注册状态
- `expires` - 注册过期时间
- `user_agent` - 设备型号/信息

响应 (201 Created 或 200 OK)：

```
{
  "data": {
    "id": 1,
    "msisdn": "+15551234567",
    ...
  }
}
```

示例：

```
curl -X POST https://api.example.com:8443/api/locations \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "+15551234567",
    "imsi": "001001000000001",
    "location": "msc1.region1.example.com",
    "ims_capable": true,
    "registered": true
  }'
```

用例：当订阅者注册时由移动管理系统（HSS、MME 等）调用

更新位置

请求：

```
PATCH /api/locations/:id
Content-Type: application/json
```

主体：部分更新任何位置字段

响应 (200 OK)：

```
{
  "data": {
    "id": 1,
    ...
  }
}
```

示例：

```
curl -X PATCH https://api.example.com:8443/api/locations/1 \
  -H "Content-Type: application/json" \
  -d '{
    "location": "msc2.region2.example.com",
    "ran_location": "cell_tower_67890"
  }'
```

删除位置

请求：

```
DELETE /api/locations/:id
```

响应 (204 No Content)

示例:

```
curl -X DELETE https://api.example.com:8443/api/locations/1
```

用例：当订阅者注销或超时时调用

前端注册 API

跟踪和管理前端 SMSC 连接。

列出所有前端

请求:

```
GET /api/frontends
```

响应 (200 OK):

```
{
  "data": [
    {
      "id": 1,
      "frontend_name": "uk_gateway_1",
      "frontend_type": "smpp",
      "ip_address": "10.0.1.50",
      "hostname": "gateway1.uk.example.com",
      "uptime_seconds": 86400,
      "configuration": {
        "max_throughput": 1000,
        "bind_type": "transceiver"
      },
      "status": "active",
      "expires_at": "2025-10-30T12:02:00Z",
      "last_seen_at": "2025-10-30T12:00:30Z",
      "inserted_at": "2025-10-29T12:00:00Z",
      "updated_at": "2025-10-30T12:00:30Z"
    }
  ]
}
```


示例:

```
curl https://api.example.com:8443/api/frontends
```

仅列出活动前端

请求:

```
GET /api/frontends/active
```

响应 (200 OK): 相同格式，仅活动前端

示例:

```
curl https://api.example.com:8443/api/frontends/active
```

用例：获取可用目的地列表以进行路由

获取前端统计信息

请求：

```
GET /api/frontends/stats
```

响应 (200 OK)：

```
{
  "data": {
    "active_count": 5,
    "expired_count": 2,
    "unique_frontends": 7,
    "total_registrations": 1523
  }
}
```

示例：

```
curl https://api.example.com:8443/api/frontends/stats
```

获取前端历史

请求：

```
GET /api/frontends/history/:name
```

响应 (200 OK)：

```
{
  "data": [
    {
      "id": 1,
      "frontend_name": "uk_gateway_1",
      "status": "active",
      "inserted_at": "2025-10-30T12:00:00Z",
      ...
    },
    {
      "id": 2,
      "frontend_name": "uk_gateway_1",
      "status": "expired",
      "inserted_at": "2025-10-29T12:00:00Z",
      ...
    }
  ]
}
```

示例：

```
curl
https://api.example.com:8443/api/frontends/history/uk_gateway_1
```

注册前端

注册或更新前端连接。

请求：

```
POST /api/frontends/register
Content-Type: application/json
```

主体：

```
{
  "frontend_name": "uk_gateway_1",
  "frontend_type": "smpp",
  "ip_address": "10.0.1.50",
  "hostname": "gateway1.uk.example.com",
  "uptime_seconds": 86400,
  "configuration": {
    "max_throughput": 1000,
    "bind_type": "transceiver",
    "system_id": "gateway1"
  }
}
```

必填字段：

- `frontend_name` - 前端的唯一标识符
- `frontend_type` - 类型：`smpp`、`sip`、`http` 等。

可选字段：

- `ip_address` - 前端 IP
- `hostname` - 前端主机名
- `uptime_seconds` - 自启动以来的运行时间
- `configuration` - 自定义配置对象

响应 (201 Created)：

```
{
  "data": {
    "id": 1,
    "frontend_name": "uk_gateway_1",
    "status": "active",
    "expires_at": "2025-10-30T12:01:30Z",
    ...
  }
}
```

示例：

```
curl -X POST https://api.example.com:8443/api/frontends/register \
  -H "Content-Type: application/json" \
  -d '{
    "frontend_name": "uk_gateway_1",
    "frontend_type": "smpp",
    "ip_address": "10.0.1.50",
    "hostname": "gateway1.uk.example.com"
  }'
```

注册超时：90 秒（前端必须每 60-90 秒重新注册）

用例：前端系统定期调用以保持活动状态

事件日志 API

跟踪消息生命周期事件。

获取消息事件

请求：

```
GET /api/events/:message_id
```

响应 (200 OK):

```
{
  "data": [
    {
      "event_epoch": 1698672000,
      "name": "message_inserted",
      "description": "□息插入队列",
      "event_source": "node1@server.example.com"
    },
    {
      "event_epoch": 1698672001,
      "name": "message_routed",
      "description": "通过 route_id=42 路由到 uk_gateway",
      "event_source": "node1@server.example.com"
    },
    {
      "event_epoch": 1698672005,
      "name": "message_delivered",
      "description": "成功送达",
      "event_source": "node2@server.example.com"
    }
  ]
}
```

示例：

```
curl https://api.example.com:8443/api/events/12345
```

事件类型：

- `message_inserted` - 消息创建
- `message_routed` - 做出路由决策
- `message_delivered` - 成功送达
- `message_failed` - 交付失败
- `message_dropped` - 被路由丢弃
- `auto_reply_sent` - 自动回复触发
- `number_translated` - 应用号码转���
- `routing_failed` - 找不到路由
- `charging_failed` - 收费错误

记录事件

请求:

```
POST /api/events
Content-Type: application/json
```

主体:

```
{
  "message_id": 12345,
  "name": "custom_event",
  "description": "自定义事件描述",
  "event_source": "external_system"
}
```

响应 (201 Created):

```
{
  "data": {
    "message_id": 12345,
    "name": "custom_event",
    "description": "自定义事件描述",
    "event_source": "external_system",
    "event_epoch": 1698672010
  }
}
```

示例:

```
curl -X POST https://api.example.com:8443/api/events \
  -H "Content-Type: application/json" \
  -d '{
    "message_id": 12345,
    "name": "external_delivery_confirmed",
    "description": "由下游系统确认"
  }'
```

事件保留：7 天（可配置）

MMS 消息 API

管理多媒体消息服务（MMS）消息。

列出 MMS 消息

请求：

```
GET /api/mms_messages
```

响应 (200 OK)：类似于 SMS 消息，带有额外的 MMS 字段

创建 MMS 消息

请求：

```
POST /api/mms_messages
Content-Type: application/json
```

主体：

```
{
  "source_msisdn": "+15551234567",
  "destination_msisdn": "+447700900000",
  "subject": "照片",
  "content_type": "image/jpeg",
  "content_location": "https://cdn.example.com/media/12345.jpg",
  "message_size": 524288
}
```

响应 (201 Created)：完整的 MMS 消息对象

SS7 事件 API

跟踪 SS7 信令事件。

列出 SS7 事件

请求:

```
GET /api/ss7_events
```

响应 (200 OK):

```
{
  "data": [
    {
      "id": 1,
      "event_type": "MAP_UPDATE_LOCATION",
      "imsi": "001001000000001",
      "msisdn": "+15551234567",
      "timestamp": "2025-10-30T12:00:00Z",
      ...
    }
  ]
}
```

创建 SS7 事件

请求:

```
POST /api/ss7_events
Content-Type: application/json
```

主体:

```
{
  "event_type": "MAP_UPDATE_LOCATION",
  "imsi": "001001000000001",
  "msisdn": "+15551234567"
}
```

响应 (201 Created): 完整事件对象

错误代码

HTTP 状态代码

代码	含义	描述
200	OK	请求成功
201	Created	资源成功创建
202	Accepted	请求已接受处理
204	No Content	删除成功
400	Bad Request	请求格式无效
401	Unauthorized	需要认证
403	Forbidden	权限不足
404	Not Found	资源不存在
422	Unprocessable Entity	验证错误
429	Too Many Requests	超过速率限制
500	Internal Server Error	服务器错误
503	Service Unavailable	暂时不可用

错误响应格式

```
{
  "errors": {
    "detail": "验证失败: destination_msisdn 是必需的"
  }
}
```

常见错误消息

错误	原因	解决方案
"destination_msisdn is required"	缺少必填字段	在请求中包含 destination_msisdn
"Invalid phone number format"	格式错误的号码	使用 E.164 格式: +15551234567
"Message too long"	超过大小限制	拆分为多个部分
"No route found"	路由失败	检查路由配置
"Charging failed"	OCS 错误	验证收费系统连接性
"Message not found"	无效的消息 ID	验证 ID 是否存在
"Frontend not registered"	未知 SMSC	首先注册前端

速率限制

默认限制

端点	限制	窗口
POST /api/messages	100 req/sec	每个 IP
POST /api/messages/create_async	1000 req/sec	每个 IP
POST /api/messages_raw	100 req/sec	每个 IP
GET /api/*	1000 req/sec	每个 IP

速率限制头

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 95
X-RateLimit-Reset: 1698672060
```

超过速率限制

响应 (429 Too Many Requests):

```
{
  "errors": {
    "detail": "超出速率限制。请在 5 秒后重试。"
  }
}
```

□□□□

□□□□

1. □□□□□□□□□□□□ > 100 msg/sec □□ `/create_async`
2. □□ **source_smsc**□□□□□□□□□
3. □□□□□□□□ E.164 □□□+□□□□□
4. □□□□□□□ 5xx □□□□□□□□□
5. □□□□□□□□□□□□□□□□□

□□□□

1. □□□□□□□ 60 □□□□□
2. □□□□□□□□ `smsc` □□□□□□□□
3. □□□□ **include-unrouted**□□□□□□□□□□□□□□□□

5. 记录 **API** 活动：监控可疑模式

错误处理

1. 重试 **5xx** 错误：服务器错误通常是暂时的
2. 不要重试 **4xx**：客户端错误需要代码修复
3. 指数退避：在重试之间等待更长时间
4. 断路器：在重复失败后停止
5. 对模式进行警报：监控错误率

示例集成（**Python**）

```python
import requests
import time

class SMSCClient:
    def __init__(self, base_url, api_key=None):
        self.base_url = base_url
        self.session = requests.Session()
        if api_key:
            self.session.headers.update({"X-API-Key": api_key})

    def submit_message(self, from_num, to_num, text,
async_mode=False):
        endpoint = "/messages/create_async" if async_mode else
"/messages"
        url = f"{self.base_url}{endpoint}"

        payload = {
            "source_msisdn": from_num,
            "destination_msisdn": to_num,
            "message_body": text,
            "source_smsc": "python_client"
        }

        try:
            response = self.session.post(url, json=payload,
timeout=30)
            response.raise_for_status()
            return response.json()["data"]
        except requests.exceptions.RequestException as e:
            print(f"API 错误: {e}")
            return None

    def get_pending_messages(self, smsc_name,
include_unrouted=False):
        url = f"{self.base_url}/messages"
        headers = {"smsc": smsc_name}

        # 如果请求了包括未路由消息（向后兼容模式）
        if include_unrouted:
            headers["include-unrouted"] = "true"
```

```
        try:
            response = self.session.get(url, headers=headers,
timeout=30)
            response.raise_for_status()
            return response.json()["data"]
        except requests.exceptions.RequestException as e:
            print(f"API 错误: {e}")
            return []

    def mark_delivered(self, message_id, smsc_name):
        url = f"
{self.base_url}/messages/{message_id}/mark_delivered"
        payload = {"dest_smsc": smsc_name}

        try:
            response = self.session.post(url, json=payload,
timeout=30)
            response.raise_for_status()
            return True
        except requests.exceptions.RequestException as e:
            print(f"API 错误: {e}")
            return False

# 使用
client = SMSCClient("https://api.example.com:8443/api",
api_key="your_key")

# 提交单条消息
result = client.submit_message("+15551234567", "+447700900000",
"Hello")
print(f"消息 ID: {result['id']}")

# 提交批量消息（异步）
for i in range(1000):
    client.submit_message("+15551234567", f"+44770090{i:04d}",
f"Bulk {i}", async_mode=True)

# 前端轮询循环
while True:
    # 获取具有明确路由或位置注册的消息
    messages = client.get_pending_messages("my_gateway")

    # 或使用 include_unrouted=True 以获取向后兼容行为
    # messages = client.get_pending_messages("my_gateway",
```

```
include_unrouted=True)

    for msg in messages:
        # □□□□□□□□□□□
        success = deliver_via_smpp(msg)

        if success:
            client.mark_delivered(msg["id"], "my_gateway")
        else:
            # □□□□
            requests.put(f"
{client.base_url}/messages/{msg['id']}")

    time.sleep(5)  # □ 5 □□□□□
```

API □□□□

□□ 1□□□□

- □□□□
- □□□□ CRUD
- □□ PDU □□
- □□□□
- □□□□
- □□□□

□□□□

- □□□□□□□□□□□□□□□□□□□
- □□□□
- □□□□ API
- □□□□□ Webhook
- GraphQL API □□
- OAuth2 □□

□□□□ API □□□□□□□□□□ □□□□□□ ◆◆□□□□□

CDR（通话详细记录）架构参考

← 返回文档索引 | 主 **README**

完整参考用于长期消息存储、计费和分析的 CDR 数据库表。

目录

概述

`cdrs` 表存储系统处理的所有 SMS 消息的通话详细记录。CDR 在以下情况下被写入：

- 消息成功送达
- 消息在未送达的情况下过期
- 消息永久失败
- 消息被拒绝

CDR 提供与操作 Mnesia 数据库分开的长期存储，使得：

- 计费和开票
- 分析和报告
- 合规和审计
- 超过 Mnesia 保留期的消息历史

表架构

MySQL / MariaDB

```
CREATE TABLE cdrs (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,

  -- 消息识别
  message_id BIGINT NOT NULL,

  -- 电话号码
  calling_number VARCHAR(255) NOT NULL,
  called_number VARCHAR(255) NOT NULL,

  -- SMSC 路由
  source_smsc VARCHAR(255),
  dest_smsc VARCHAR(255),

  -- 节点信息（用于集群部署）
  origin_node VARCHAR(255),
  destination_node VARCHAR(255),

  -- 时间戳
  submission_time DATETIME NOT NULL,
  delivery_time DATETIME,
  expiry_time DATETIME,

  -- 状态和元数据
  status VARCHAR(50) NOT NULL,
  delivery_attempts INT DEFAULT 0,
  message_parts INT,
  deadletter BOOLEAN DEFAULT FALSE,

  -- 可选消息体（隐私控制）
  message_body TEXT,

  -- 审计时间戳
  inserted_at DATETIME NOT NULL,
  updated_at DATETIME NOT NULL,

  -- 索引
  INDEX idx_cdrs_message_id (message_id),
```

```
  INDEX idx_cdrs_calling_number (calling_number),
  INDEX idx_cdrs_called_number (called_number),
  INDEX idx_cdrs_status (status),
  INDEX idx_cdrs_submission_time (submission_time),
  INDEX idx_cdrs_dest_smsc (dest_smsc)
);
```

PostgreSQL

```
CREATE TABLE cdrs (
  id BIGSERIAL PRIMARY KEY,

  -- 消息识别
  message_id BIGINT NOT NULL,

  -- 电话号码
  calling_number VARCHAR(255) NOT NULL,
  called_number VARCHAR(255) NOT NULL,

  -- SMSC 路由
  source_smsc VARCHAR(255),
  dest_smsc VARCHAR(255),

  -- 节点信息（用于集群部署）
  origin_node VARCHAR(255),
  destination_node VARCHAR(255),

  -- 时间戳
  submission_time TIMESTAMP NOT NULL,
  delivery_time TIMESTAMP,
  expiry_time TIMESTAMP,

  -- 状态和元数据
  status VARCHAR(50) NOT NULL,
  delivery_attempts INTEGER DEFAULT 0,
  message_parts INTEGER,
  deadletter BOOLEAN DEFAULT FALSE,

  -- 可选消息体（隐私控制）
  message_body TEXT,

  -- 审计时间戳
  inserted_at TIMESTAMP NOT NULL,
  updated_at TIMESTAMP NOT NULL
);

-- 索引
CREATE INDEX idx_cdrs_message_id ON cdrs(message_id);
CREATE INDEX idx_cdrs_calling_number ON cdrs(calling_number);
CREATE INDEX idx_cdrs_called_number ON cdrs(called_number);
```

```
CREATE INDEX idx_cdrs_status ON cdrs(status);
CREATE INDEX idx_cdrs_submission_time ON cdrs(submission_time);
CREATE INDEX idx_cdrs_dest_smsc ON cdrs(dest_smsc);
```

字段描述

主键

字段	类型	可空	描述
`id`	BIGINT	NO	CDR 记录的自增主键

消息识别

字段	类型	可空	描述
`message_id`	BIGINT	NO	来自 SMS-C 消息队列的唯一消息标识符。引用 Mnesia 中的原始消息 ID。

电话号码

字段	类型	可空	描述
`calling_number`	VARCHAR(255)	NO	消息发送者的源 MSISDN（手机号码）。通常为 E.164 格式（例如，+15551234567）。
`called_number`	VARCHAR(255)	NO	消息接收者的目标 MSISDN（手机号码）。通常为 E.164 格式（例如，+15551234567）。

SMSC 路由

字段	类型	可空	描述
`source_smsc`	VARCHAR(255)	YES	提交消息的源 SMSC 的名称或标识符。如果通过 API 或其他非 SMSC 接口提交，则为 NULL。
`dest_smsc`	VARCHAR(255)	YES	交付（或尝试交付）消息的目标 SMSC 的名称或标识符。如果消息从未路由，则为 NULL。

节点信息

对于集群部署，跟踪处理消息的节点：

字段	类型	可空	描述
`origin_node`	VARCHAR(255)	YES	消息最初接收的 Erlang 节点名称（例如，“sms@node1.example.com”[对故障排除和负载分配分析非常有用。
`destination_node`	VARCHAR(255)	YES	消息交付的 Erlang 节点名称（如果与原始节点不同）对于单节点部署或消息从未交付的情况为 NULL。

时间戳

所有时间戳以 UTC 存储：

字段	类型	可空	描述
`submission_time`	DATETIME	NO	消息首次提交到 SMS-C 的时间。用作计费计算的开始时间。
`delivery_time`	DATETIME	YES	消息成功交付的时间。如果消息过期、失败或被拒绝，则为 NULL。
`expiry_time`	DATETIME	YES	消息过期（变得不可交付）的时间。如果消息已交付或仍在待处理状态，则为 NULL。

交付持续时间计算：

```
TIMESTAMPDIFF(SECOND, submission_time, delivery_time) AS
delivery_duration_seconds
```

状态和元数据

字段	类型	可空	描述
`status`	VARCHAR(50)	NO	最终消息状态。有效值：`delivered`、`expired`、`failed`、`rejected`
`delivery_attempts`	INT	NO	在最终状态之前进行的交付尝试次数。默认值：0。范围：通常为 0-255。
`message_parts`	INT	YES	连接消息的 SMS 段数。单部分消息为 1，多个部分为 2+。如果未知则为 NULL。
`deadletter`	BOOLEAN	NO	消息是否被移至死信队列。TRUE 表示消息无法交付并已耗尽所有重试。默认值：FALSE

状态值：

状态	描述	可计费	交付时间
`delivered`	成功交付给接收者	是	已设置
`expired`	超过有效期未交付	取决于计��政策	NULL
`failed`	永久交付失败（无效号码等）	取决于计费政策	NULL
`rejected`	被路由规则或验证拒绝	否	NULL

消息体

字段	类型	可空	描述
`message_body`	TEXT	YES	实际的 SMS 消息内容。如果启用了 `delete_message_body_after_delivery`，则可以为 NULL。最大长度因数据库而异（通常 TEXT 类型为 65,535 个字符）。

隐私模式：

- 完全保留：消息体存储在 CDR 中以满足合规/归档要求
- 隐私模式：当 `delete_message_body_after_delivery: true` 时，消息体设置为 NULL
- 合规模式：消息体以加密或哈希形式存储（需要自定义实现）

审计时间戳

字段	类型	可空	描述
`inserted_at`	DATETIME	NO	CDR 记录首次插入数据库的时间。通常与 delivery_time/expiry_time 相同或稍后。
`updated_at`	DATETIME	NO	CDR 记录最后一次更新的时间。如果从未更新，则与 inserted_at 相同。

SQL 示例

□本查询

查找特定电话号码的所有 **CDR**:

```
SELECT * FROM cdrs
WHERE calling_number = '+15551234567'
   OR called_number = '+15551234567'
ORDER BY submission_time DESC
LIMIT 100;
```

按状态计数消息:

```
SELECT status, COUNT(*) as count
FROM cdrs
GROUP BY status;
```

成功交付消息的平均交付时间:

```
SELECT AVG(TIMESTAMPDIFF(SECOND, submission_time, delivery_time))
AS avg_delivery_seconds
FROM cdrs
WHERE status = 'delivered'
  AND delivery_time IS NOT NULL;
```

计费查询

按目标 **SMSC** 的每日消息量:

```sql
SELECT
  DATE(submission_time) AS date,
  dest_smsc,
  COUNT(*) AS message_count,
  SUM(CASE WHEN status = 'delivered' THEN 1 ELSE 0 END) AS
delivered_count,
  SUM(message_parts) AS total_segments
FROM cdrs
WHERE submission_time >= DATE_SUB(NOW(), INTERVAL 30 DAY)
GROUP BY DATE(submission_time), dest_smsc
ORDER BY date DESC, message_count DESC;
```

按呼叫号码前缀的客户可计费消息：

```sql
SELECT
  DATE(submission_time) AS date,
  COUNT(*) AS message_count,
  SUM(message_parts) AS total_segments,
  SUM(message_parts) * 0.01 AS total_cost
FROM cdrs
WHERE calling_number LIKE '+1555%'
  AND status = 'delivered'
  AND submission_time >= '2025-10-01'
  AND submission_time < '2025-11-01'
GROUP BY DATE(submission_time);
```

路由性能分析：

```sql
SELECT
  dest_smsc,
  COUNT(*) AS total_messages,
  SUM(CASE WHEN status = 'delivered' THEN 1 ELSE 0 END) AS
delivered,
  ROUND(100.0 * SUM(CASE WHEN status = 'delivered' THEN 1 ELSE 0
END) / COUNT(*), 2) AS delivery_rate_pct,
  AVG(delivery_attempts) AS avg_attempts,
  AVG(TIMESTAMPDIFF(SECOND, submission_time, delivery_time)) AS
avg_delivery_seconds
FROM cdrs
WHERE submission_time >= DATE_SUB(NOW(), INTERVAL 7 DAY)
  AND dest_smsc IS NOT NULL
GROUP BY dest_smsc
ORDER BY delivery_rate_pct DESC;
```

分析查询

按小时的消息（流量模式）：

```sql
SELECT
  HOUR(submission_time) AS hour,
  COUNT(*) AS message_count
FROM cdrs
WHERE submission_time >= DATE_SUB(NOW(), INTERVAL 7 DAY)
GROUP BY HOUR(submission_time)
ORDER BY hour;
```

多部分消息分析：

```
SELECT
  message_parts,
  COUNT(*) AS message_count,
  AVG(TIMESTAMPDIFF(SECOND, submission_time, delivery_time)) AS
avg_delivery_seconds
FROM cdrs
WHERE message_parts IS NOT NULL
  AND status = 'delivered'
GROUP BY message_parts
ORDER BY message_parts;
```

失败消息分析：

```
SELECT
  called_number,
  COUNT(*) AS failure_count,
  AVG(delivery_attempts) AS avg_attempts,
  MAX(submission_time) AS last_failure
FROM cdrs
WHERE status IN ('failed', 'expired')
  AND submission_time >= DATE_SUB(NOW(), INTERVAL 7 DAY)
GROUP BY called_number
HAVING failure_count >= 5
ORDER BY failure_count DESC;
```

合规和审计查询

查找两个方之间在时间范围内的所有消息：

```sql
SELECT
  submission_time,
  calling_number,
  called_number,
  status,
  message_body,
  delivery_time
FROM cdrs
WHERE (
    (calling_number = '+15551234567' AND called_number =
'+15559876543')
    OR
    (calling_number = '+15559876543' AND called_number =
'+15551234567')
  )
  AND submission_time >= '2025-10-01'
  AND submission_time < '2025-11-01'
ORDER BY submission_time;
```

保留政策执行（删除旧 CDR）：

```sql
-- 查找超过保留期的记录（示例：2 年）
SELECT COUNT(*) FROM cdrs
WHERE submission_time < DATE_SUB(NOW(), INTERVAL 2 YEAR);

-- 删除旧记录（谨慎使用！）
DELETE FROM cdrs
WHERE submission_time < DATE_SUB(NOW(), INTERVAL 2 YEAR)
LIMIT 10000;  -- 批量删除以避免锁定
```

集群分析

消息在节点之间的分布：

```sql
SELECT
  origin_node,
  COUNT(*) AS message_count,
  SUM(CASE WHEN status = 'delivered' THEN 1 ELSE 0 END) AS
delivered_count
FROM cdrs
WHERE submission_time >= DATE_SUB(NOW(), INTERVAL 1 DAY)
GROUP BY origin_node;
```

□□

□□□□□□□□□□□□□□□□

□□□□	□	□□
`PRIMARY`	`id`	□□□□□□□□□
`idx_cdrs_message_id`	`message_id`	□□□□□ ID □□□ CDR
`idx_cdrs_calling_number`	`calling_number`	□□□□□□□□□
`idx_cdrs_called_number`	`called_number`	□□□□□□□□□
`idx_cdrs_status`	`status`	□□□□□□□
`idx_cdrs_submission_time`	`submission_time`	□□□□□□□□□□□
`idx_cdrs_dest_smsc`	`dest_smsc`	□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

```
CREATE INDEX idx_cdrs_billing ON cdrs(calling_number,
submission_time, status);
```

路由分析的复合索引:

```
CREATE INDEX idx_cdrs_route_perf ON cdrs(dest_smsc,
submission_time, status);
```

合规搜索的复合索引:

```
CREATE INDEX idx_cdrs_party_time ON cdrs(calling_number,
called_number, submission_time);
```

消息体搜索的全文索引（**MySQL**）:

```
ALTER TABLE cdrs ADD FULLTEXT INDEX idx_cdrs_message_body_ft
(message_body);

-- 用法:
SELECT * FROM cdrs
WHERE MATCH(message_body) AGAINST('keyword' IN NATURAL LANGUAGE
MODE);
```

按数据库的数据类型

支持的数据库之间的字段类型映射:

字段	**MySQL/MariaDB**	**PostgreSQL**	备注
`id`	BIGINT AUTO_INCREMENT	BIGSERIAL	64 位整数，自增
`message_id`	BIGINT	BIGINT	64 位整数
字符串字段	VARCHAR(255)	VARCHAR(255)	可变长度字符串，最大 255 个字符
`message_body`	TEXT	TEXT	大文本，MySQL 最大 65,535 字节，PostgreSQL 无限制
时间戳	DATETIME	TIMESTAMP	推荐使用 UTC 时间戳
整数	INT	INTEGER	32 位有符号整数
布尔值	BOOLEAN (TINYINT(1))	BOOLEAN	MySQL 存储为 0/1

隐私考虑

CDR 表可能包含敏感个人信息（电话号码、消息内容）。考虑以下隐私措施：

1. 消息体隐私

`config/runtime.exs` 中的配置选项：

```
config :sms_c,
  # 成功交付后删除□息体
  delete_message_body_after_delivery: true,

  # 在 UI 中隐藏□息体
  hide_message_body_in_ui: true,

  # 在导出中隐藏□息体
  hide_message_body_in_export: true
```

2. 电话号码掩码

对于不需要完整号码的分析：

```
-- 掩码电话号码的最后 4 位
SELECT
  CONCAT(SUBSTRING(calling_number, 1, LENGTH(calling_number) - 4),
'XXXX') AS masked_calling,
  CONCAT(SUBSTRING(called_number, 1, LENGTH(called_number) - 4),
'XXXX') AS masked_called,
  COUNT(*) AS message_count
FROM cdrs
GROUP BY masked_calling, masked_called;
```

3. 数据库加密

为数据库服务器启用静态加密��

MySQL：

```
-- 启用表加密
ALTER TABLE cdrs ENCRYPTION='Y';
```

PostgreSQL：使用 PostgreSQL 透明数据加密（TDE）或文件系统级别加密。

4. 访问控制

限制对 CDR 表的访问：

```
-- 创建只读计费用户
CREATE USER 'billing_ro'@'%' IDENTIFIED BY 'secure_password';
GRANT SELECT ON sms_c.cdrs TO 'billing_ro'@'%';

-- 创建有限的分析用户（无消息体访问）
CREATE USER 'analytics'@'%' IDENTIFIED BY 'secure_password';
GRANT SELECT (id, message_id, calling_number, called_number,
source_smsc,
              dest_smsc, submission_time, delivery_time, status,
              delivery_attempts, message_parts)
ON sms_c.cdrs TO 'analytics'@'%';
```

保留和归档

保留政策

根据监管和业务要求定义保留期：

行业	典型保留	监管依据
电信（美国）	18-24 个月	FCC，州法律
电信（欧盟）	6 个月 - 2 年	GDPR，电子隐私
金融	5-7 年	SOX，SEC
医疗保健	6 年	HIPAA

归档策略

1. 按日期分区（**MySQL 8.0+**，**PostgreSQL 11+**）

```sql
-- MySQL 按月分区
ALTER TABLE cdrs PARTITION BY RANGE (TO_DAYS(submission_time)) (
  PARTITION p202510 VALUES LESS THAN (TO_DAYS('2025-11-01')),
  PARTITION p202511 VALUES LESS THAN (TO_DAYS('2025-12-01')),
  PARTITION p202512 VALUES LESS THAN (TO_DAYS('2026-01-01')),
  PARTITION p_future VALUES LESS THAN MAXVALUE
);

-- 删除旧分区（快速归档）
ALTER TABLE cdrs DROP PARTITION p202510;
```

2. 归档到冷存储

```sql
-- 将旧 CDR 导出到归档表
CREATE TABLE cdrs_archive LIKE cdrs;

INSERT INTO cdrs_archive
SELECT * FROM cdrs
WHERE submission_time < DATE_SUB(NOW(), INTERVAL 2 YEAR);

-- 验证并从主表中删除
DELETE FROM cdrs
WHERE submission_time < DATE_SUB(NOW(), INTERVAL 2 YEAR);
```

3. 自动清理脚本

```
#!/bin/bash
# cleanup_old_cdrs.sh - 通过 cron 运行

MYSQL_USER="cleanup_user"
MYSQL_PASS="secure_password"
MYSQL_DB="sms_c"
RETENTION_DAYS=730  # 2 年

# 归档旧记录
mysql -u"$MYSQL_USER" -p"$MYSQL_PASS" "$MYSQL_DB" <<EOF
INSERT INTO cdrs_archive
SELECT * FROM cdrs
WHERE submission_time < DATE_SUB(NOW(), INTERVAL $RETENTION_DAYS
DAY)
LIMIT 100000;

DELETE FROM cdrs
WHERE submission_time < DATE_SUB(NOW(), INTERVAL $RETENTION_DAYS
DAY)
LIMIT 100000;
EOF
```

Cron 条目:

```
# 每天凌晨 2 点运行
0 2 * * * /usr/local/bin/cleanup_old_cdrs.sh >>
/var/log/sms_c/cleanup.log 2>&1
```

计费集成

费率卡架构

为计费创建单独的费率表:

```sql
CREATE TABLE billing_rates (
  id INT AUTO_INCREMENT PRIMARY KEY,
  destination_prefix VARCHAR(20) NOT NULL,
  description VARCHAR(255),
  rate_per_message DECIMAL(10, 6) NOT NULL,
  rate_per_segment DECIMAL(10, 6) NOT NULL,
  currency VARCHAR(3) DEFAULT 'USD',
  effective_date DATE NOT NULL,
  expiry_date DATE,
  INDEX idx_prefix (destination_prefix),
  INDEX idx_dates (effective_date, expiry_date)
);

-- 示例费率
INSERT INTO billing_rates (destination_prefix, description,
rate_per_message, rate_per_segment, effective_date) VALUES
('+1', '美国/加拿大', 0.0050, 0.0050, '2025-01-01'),
('+44', '英国', 0.0080, 0.0080, '2025-01-01'),
('+61', '澳大利亚', 0.0100, 0.0100, '2025-01-01'),
('+', '国际默认', 0.0150, 0.0150, '2025-01-01');
```

计费查询

将 CDR 与费率连接以进行开票:

```
SELECT
  DATE(c.submission_time) AS date,
  c.dest_smsc AS route,
  LEFT(c.called_number,
    CASE
      WHEN c.called_number LIKE '+1%' THEN 2
      WHEN c.called_number LIKE '+%' THEN
LENGTH(SUBSTRING_INDEX(c.called_number, '', 4))
      ELSE 0
    END
  ) AS destination_prefix,
  COUNT(*) AS message_count,
  SUM(c.message_parts) AS segment_count,
  COALESCE(r.rate_per_segment, 0.015) AS rate,
  SUM(c.message_parts) * COALESCE(r.rate_per_segment, 0.015) AS
total_cost
FROM cdrs c
LEFT JOIN billing_rates r ON c.called_number LIKE
CONCAT(r.destination_prefix, '%')
  AND c.submission_time >= r.effective_date
  AND (r.expiry_date IS NULL OR c.submission_time < r.expiry_date)
WHERE c.status = 'delivered'
  AND c.submission_time >= '2025-10-01'
  AND c.submission_time < '2025-11-01'
GROUP BY date, route, destination_prefix
ORDER BY date DESC, total_cost DESC;
```

□□□□□□□

CSV □□□

```
mysql -u billing_ro -p -D sms_c -e "
SELECT
  id,
  message_id,
  calling_number,
  called_number,
  dest_smsc,
  submission_time,
  delivery_time,
  status,
  message_parts
FROM cdrs
WHERE submission_time >= '2025-10-01'
  AND submission_time < '2025-11-01'
  AND status = 'delivered'
" --batch --silent | sed 's/\t/,/g' > billing_export_202510.csv
```

另见

- 配置指南 - 配置 CDR 导出设置
- 操作指南 - CDR 维护程序
- API 参考 - 通过 REST API 查询 CDR

SMS-C 配置参考

← 返回文档索引 | 主 **README**

完整的 SMS-C 配置选项参考，包括常见部署场景的示例。

目录

配置文件

SMS-C 使用三个主要配置文件：

config/config.exs

在编译时加载的静态配置。包含：

- 应用程序范围的默认值
- 日志记录配置
- 开发/测试设置

- 性能调优参数

config/runtime.exs

在启动时加载的运行时配置。包含：

- 数据库连接设置
- 集群配置
- 外部服务集成（OCS，ENUM）
- 初始路由和翻译规则
- 环境特定设置

config/prod.exs（可选）

生产特定的覆盖。

最佳实践：在 `runtime.exs` 中使用环境变量来处理敏感值，如密码和 API 密钥。

SQL CDR 存储配置

SMS-C 使用 **Mnesia** 处理操作数据（消息队列、路由规则、号码翻译），并支持外部 **SQL** 数据库进行长期 CDR（通话详细记录）存储、计费和分析。

支持的 SQL 数据库

系统支持以下 SQL 数据库进行 CDR 导出：

数据库	版本	适配器	默认端口	最佳用途
MySQL	8.0+	`Ecto.Adapters.MyXQL`	3306	通用，经过验证的可靠性
MariaDB	10.5+	`Ecto.Adapters.MyXQL`	3306	MySQL 兼容，开源
PostgreSQL	13+	`Ecto.Adapters.Postgres`	5432	高级功能，JSON 支持

注意：Mnesia 自动用于操作数据（消息队列、路由、翻译），无需配置。SQL 数据库仅用于 CDR 导出和长期存储。

MySQL / MariaDB 配置

```
# config/runtime.exs
config :sms_c, SmsC.Repo,
  adapter: Ecto.Adapters.MyXQL,
  username: System.get_env("DB_USERNAME") || "sms_user",
  password: System.get_env("DB_PASSWORD") || "secure_password",
  hostname: System.get_env("DB_HOSTNAME") || "localhost",
  port: String.to_integer(System.get_env("DB_PORT") || "3306"),
  database: System.get_env("DB_NAME") || "sms_c_prod",
  pool_size: String.to_integer(System.get_env("DB_POOL_SIZE") ||
"20")
```

PostgreSQL 配置

```
# config/runtime.exs
config :sms_c, SmsC.Repo,
  adapter: Ecto.Adapters.Postgres,
  username: System.get_env("DB_USERNAME") || "sms_user",
  password: System.get_env("DB_PASSWORD") || "secure_password",
  hostname: System.get_env("DB_HOSTNAME") || "localhost",
  port: String.to_integer(System.get_env("DB_PORT") || "5432"),
  database: System.get_env("DB_NAME") || "sms_c_prod",
  pool_size: String.to_integer(System.get_env("DB_POOL_SIZE") ||
"20")
```

选择 SQL 数据库

MySQL/MariaDB - 推荐用于大多数部署:

- 对 CDR 写入的优秀性能
- 在电信环境中的可靠性经过验证
- 对计费系统的广泛工具支持
- 简单的复制设置

PostgreSQL - 如果您需要:

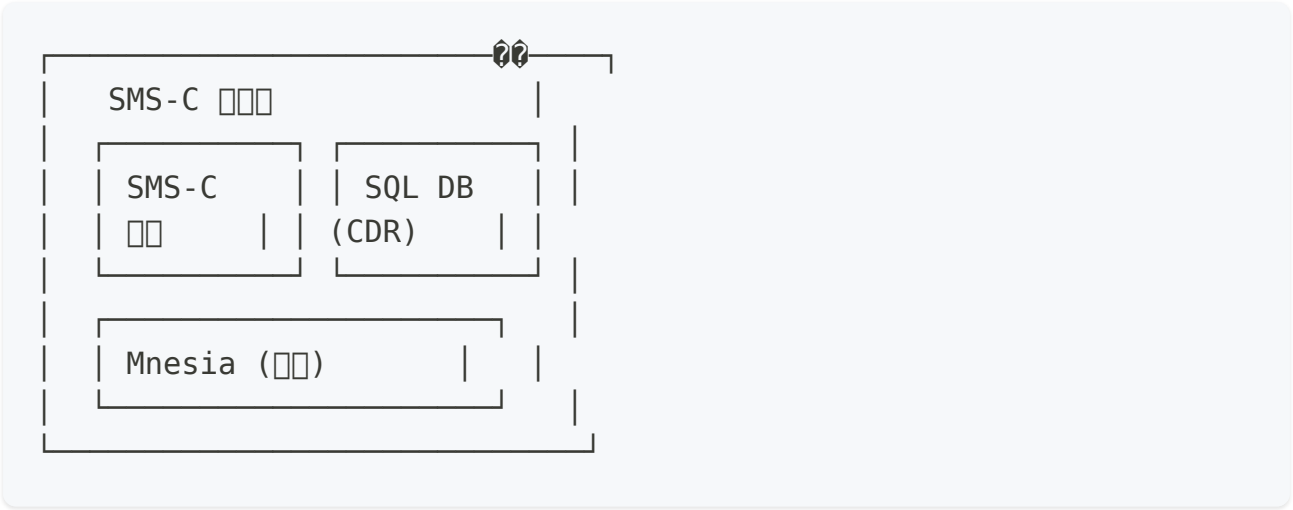
- 用于分析的高级 JSON/JSONB 功能

- □ CDR □□□□□□□□
- □□□ PostgreSQL □□□□
- □□□□□□□□ PostGIS

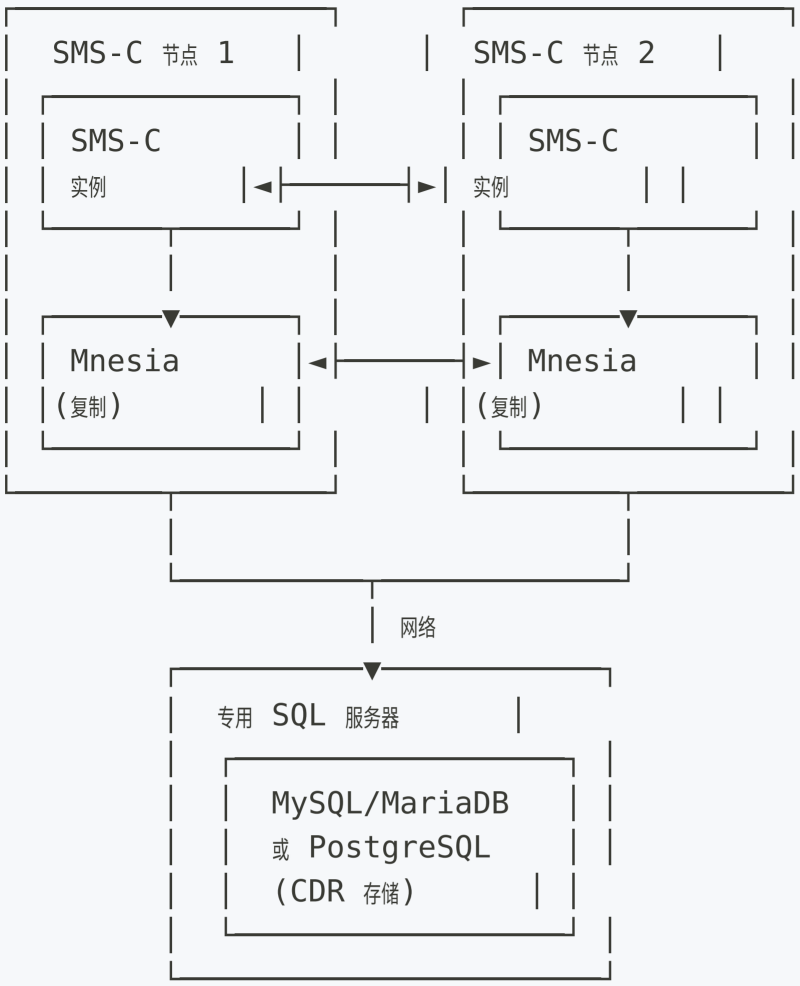
□□□□

□□□SQL CDR □□□□□□□□□□ SMS-C □□ □□□□□□□ □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□/□□□□



□□□□□□□□□ - □□□□



单独 **SQL** 服务器的好处：

- 性能隔离：CDR 写入不会影响消息处理
- 可扩展性：独立扩展数据库和消息处理
- 可靠性：数据库维护不会影响 SMS-C 的正常运行
- 数据管理：多个 SMS-C 实例的集中 CDR 存储
- 备份灵活性：独立的备份计划和保留策略

连接池大小指南

工作负载	池大小	描述
开发	5-10	最小并发
低流量（< 100 msg/sec）	10-15	小型部署
中等流量（100-1000 msg/sec）	20-30	典型生产
高流量（> 1000 msg/sec）	40-100	高吞吐量场景

计算：`pool_size = (预期并发 DB 操作) * 1.5`

数据库连接示例

使用环境变量（推荐用于生产）：

```
# 设置环境变量
export DB_USERNAME=sms_prod_user
export DB_PASSWORD=strong_password_here
export DB_HOSTNAME=db-primary.internal.example.com
export DB_PORT=3306
export DB_NAME=sms_c_production
export DB_POOL_SIZE=30
```

直接配置（仅限开发）：

```
config :sms_c, SmsC.Repo,
  username: "dev_user",
  password: "dev_password",
  hostname: "localhost",
  database: "sms_c_dev",
  pool_size: 5
```

连接池监控

通过 Prometheus 指标监控池使用情况:

- `ecto_pools_queue_time` - 等待连接的时间
- `ecto_pools_query_time` - 查询执行时间
- `ecto_pools_connected_count` - 活动连接数

如果等待时间持续超过 100ms - 表示需要更大的池。

API 配置

REST API 提供消息提交和管理功能。

本 API 配置

```
# config/runtime.exs
config :api_ex,
  port: String.to_integer(System.get_env("API_PORT") || "8443"),
  listen_ip: System.get_env("API_LISTEN_IP") || "0.0.0.0",
  enable_tls: System.get_env("API_ENABLE_TLS") != "false"
```

TLS/SSL 配置

生产设置带 **TLS**（推荐）:

```
config :api_ex,
  port: 8443,
  listen_ip: "0.0.0.0",
  enable_tls: true,
  tls_cert_path: "/etc/sms_c/certs/server.crt",
  tls_key_path: "/etc/sms_c/certs/server.key"
```

开发设置不带 **TLS**:

```
config :api_ex,
  port: 8080,
  listen_ip: "127.0.0.1",
  enable_tls: false
```

API 证书设置

生成自签名证书用于测试：

```
# 创建证书目录
mkdir -p priv/cert

# 生成私钥
openssl genrsa -out priv/cert/server.key 2048

# 生成证书签名请求
openssl req -new -key priv/cert/server.key -out priv/cert/server.csr \
  -subj "/C=US/ST=State/L=City/O=Organization/CN=sms-api.example.com"

# 生成自签名证书（有效期 365 天）
openssl x509 -req -days 365 -in priv/cert/server.csr \
  -signkey priv/cert/server.key -out priv/cert/server.crt

# 设置权限
chmod 600 priv/cert/server.key
chmod 644 priv/cert/server.crt
```

对于生产环境，使用受信任的 CA（Let's Encrypt、商业 CA 等）提供的证书。

API 访问控制

IP 白名单（应用防火墙）：

```
# 使用 iptables（Linux）
iptables -A INPUT -p tcp --dport 8443 -s 10.0.0.0/8 -j ACCEPT
iptables -A INPUT -p tcp --dport 8443 -j DROP

# 使用 firewalld（Red Hat/CentOS）
firewall-cmd --permanent --add-rich-rule='rule family="ipv4"
source address="10.0.0.0/8" port protocol="tcp" port="8443"
accept'
firewall-cmd --reload
```

API 密钥认证（应用级别）：

通过自定义插件在路由器中配置 - 有关实现细节，请参阅操作指南。

Web UI 配置

Web 界面提供路由管理、消息浏览和监控。

□本 Web UI 配置

```
# config/runtime.exs
config :control_panel,
  port: String.to_integer(System.get_env("WEB_PORT") || "80"),
  hostname: System.get_env("WEB_HOSTNAME") || "localhost",
  enable_tls: System.get_env("WEB_ENABLE_TLS") == "true"
```

生产 Web UI 设置

```
config :control_panel,
  port: 443,
  hostname: "sms-admin.example.com",
  enable_tls: true,
  tls_cert_path: "/etc/sms_c/certs/web.crt",
  tls_key_path: "/etc/sms_c/certs/web.key"
```

反向代理设置（推荐）

使用 Nginx 或 Apache 作为反向代理以增强安全性和功能：

Nginx 配置示例：

```
upstream sms_web {
  server 127.0.0.1:4000;
  keepalive 32;
}

server {
  listen 80;
  server_name sms-admin.example.com;
  return 301 https://$server_name$request_uri;
}

server {
  listen 443 ssl http2;
  server_name sms-admin.example.com;

  ssl_certificate /etc/letsencrypt/live/sms-
admin.example.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/sms-
admin.example.com/privkey.pem;
  ssl_protocols TLSv1.2 TLSv1.3;
  ssl_ciphers HIGH:!aNULL:!MD5;

  # □□□□□□□□□□
  auth_basic "SMS-C Admin";
  auth_basic_user_file /etc/nginx/.htpasswd;

  location / {
    proxy_pass http://sms_web;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
  }

  # LiveView □ WebSocket □□
  location /live {
    proxy_pass http://sms_web;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
```

```
    proxy_read_timeout 86400;
  }
}
```

集群配置

SMS-C 支持多节点集群以实现高可用性和负载分配。

单节点设置

```
# config/runtime.exs
config :sms_c,
  cluster_nodes: [],  # 空表 = 单节点模式
  smsc_node_name: "node1"
```

多节点静态集群

```
# 节点 1: config/runtime.exs
config :sms_c,
  cluster_nodes: [
    :"sms@node1.internal.example.com",
    :"sms@node2.internal.example.com",
    :"sms@node3.internal.example.com"
  ],
  smsc_node_name: "node1"

# 节点 2: config/runtime.exs
config :sms_c,
  cluster_nodes: [
    :"sms@node1.internal.example.com",
    :"sms@node2.internal.example.com",
    :"sms@node3.internal.example.com"
  ],
  smsc_node_name: "node2"
```

于 DNS 的自动发现

```
config :sms_c,
  dns_cluster_query: "sms-cluster.internal.example.com",
  smsc_node_name: System.get_env("NODE_NAME") || "node1"
```

DNS 自动发现的设置:

```
# 为集群节点配置 SRV 或 A 记录
# SRV 记录（优先）：
_sms._tcp.sms-cluster.internal.example.com. IN SRV 0 0 0
node1.internal.example.com.
_sms._tcp.sms-cluster.internal.example.com. IN SRV 0 0 0
node2.internal.example.com.
_sms._tcp.sms-cluster.internal.example.com. IN SRV 0 0 0
node3.internal.example.com.

# A 记录（替代）：
sms-cluster.internal.example.com. IN A 10.0.1.10
sms-cluster.internal.example.com. IN A 10.0.1.11
sms-cluster.internal.example.com. IN A 10.0.1.12
```

Erlang 分布式配置

使用正确名称启���节点:

```
# 节点 1
export NODE_NAME=sms@node1.internal.example.com
export ERLANG_COOKIE=shared_secret_cookie_here
elixir --name $NODE_NAME --cookie $ERLANG_COOKIE -S mix phx.server

# 节点 2
export NODE_NAME=sms@node2.internal.example.com
export ERLANG_COOKIE=shared_secret_cookie_here
elixir --name $NODE_NAME --cookie $ERLANG_COOKIE -S mix phx.server
```

重要：集群中的所有节点必须使用相同的 Erlang cookie 以确保安全。

集群网络要求

在集群节点之间打开这些端口：

端口范围	协议	目的
4369	TCP	Erlang 端口映射守护进程（EPMD）
9100-9200	TCP	Erlang 分布

防火墙配置示例：

```
# 允许来自内部网络的集群流量
iptables -A INPUT -p tcp -s 10.0.0.0/8 --dport 4369 -j ACCEPT
iptables -A INPUT -p tcp -s 10.0.0.0/8 --dport 9100:9200 -j ACCEPT
```

消息队列配置

控制消息保留和过期行为。

消息过期

```
# config/runtime.exs
config :sms_c,
  dead_letter_time_minutes: 1440  # 24 小时
```

常见值：

- **60** - 1 小时（测试/开发）
- **1440** - 24 小时（典型生产）
- **4320** - 3 天（延长保留）
- **10080** - 7 天（最大保留）

超过□值的消息将变得不可交付，并标记□□清理。

投递重试配置

重试行为使用指数回退：

```
重试延迟 = 2^(尝试次数) 分钟
```

尝试	延迟
1	2 分钟
2	4 分钟
3	8 分钟
4	16 分钟
5	32 分钟
6	64 分钟
7	128 分钟
8	256 分钟

在死信之前的最大尝试次数：受 `dead_letter_time_minutes` 限制。

清理配置

```
# config/config.exs
config :sms_c,
  cleanup_interval_minutes: 10,
  fingerprint_ttl_minutes: 5,
  event_ttl_days: 7
```

清理间隔：

- **cleanup_interval_minutes**□□□□□□□□□□□□□□□□□□□10□
- **fingerprint_ttl_minutes**□□□□□□□□□□□□5□
- **event_ttl_days**□□□□□□□□□□□□7□

□□□□

□ OCS □□□□□□□□□□□□□

□□□□

```
# config/runtime.exs
config :sms_c,
  default_charging_enabled: true,
  ocs_url: "http://ocs.internal.example.com:2080/jsonrpc",
  ocs_tenant: "sms.example.com",
  ocs_destination: "default",
  ocs_source: "sms_platform",
  ocs_subject: "sms_user",
  ocs_account: "default_account"
```

□□□□

```
# config/runtime.exs
config :sms_c,
  default_charging_enabled: false
```

□□□□□□□□□□□□□□□□□□□

每租户计费配置

```
config :sms_c,
  ocs_url: System.get_env("OCS_URL") ||
"http://localhost:2080/jsonrpc",
  ocs_tenant: System.get_env("OCS_TENANT") ||
"tenant1.example.com",
  ocs_account: System.get_env("OCS_ACCOUNT") || "default"
```

按租户的环境变量：

```
# 租户 1
export OCS_TENANT=tenant1.example.com
export OCS_ACCOUNT=tenant1_account

# 租户 2
export OCS_TENANT=tenant2.example.com
export OCS_ACCOUNT=tenant2_account
```

计费失败行为

配置计费失败时发生的事情：

```
config :sms_c,
  charging_failure_action: :allow  # 或 :deny
```

- **:allow** - 即使计费失败也处理消息（记录错误）
- **:deny** - 如果计费失败则拒绝消息

OCS 连接示例

测试 **OCS** 连接性：

```
# 测试 OCS API
curl -X POST http://ocs.internal.example.com:2080/jsonrpc \
  -H "Content-Type: application/json" \
  -d '{
    "method": "SessionSv1.AuthorizeEvent",
    "params": [{
      "Tenant": "sms.example.com",
      "Account": "test_account",
      "Destination": "1234567890",
      "Usage": 100
    }],
    "id": 1
  }'
```

预期响应：

```
{
  "id": 1,
  "result": {
    "Attributes": {},
    "MaxUsage": 100,
    ...
  }
}
```

ENUM 配置

基于 DNS 的 E.164 号码查找以实现智能路由。

禁用 ENUM（默认）

```
# config/runtime.exs
config :sms_c,
  enum_enabled: false
```

启用带默认 DNS 的 ENUM

```
config :sms_c,
  enum_enabled: true,
  enum_domains: ["e164.arpa", "e164.org"],
  enum_dns_servers: [],  # 使用系统默认 DNS
  enum_timeout: 5000  # 5 秒
```

启用带自定义 DNS 服务器的 ENUM

```
config :sms_c,
  enum_enabled: true,
  enum_domains: ["e164.internal.example.com", "e164.arpa"],
  enum_dns_servers: [
    {"10.0.1.53", 53},  # 内部 DNS 服务器
    {"8.8.8.8", 53},    # Google 公共 DNS（后备）
    {"1.1.1.1", 53}     # Cloudflare DNS（后备）
  ],
  enum_timeout: 3000  # 3 秒（更快的故障转移）
```

ENUM 域优先级

域按顺序查询，直到成功查找：

```
config :sms_c,
  enum_domains: [
    "e164.internal.example.com",  # 首先尝试内部
    "e164.carrier.net",             # 然后是运营商
    "e164.arpa"                     # 然后是公共注册
  ]
```

ENUM 性能调优

对于低延迟网络：

```
enum_timeout: 2000  # 2 秒
```

对于高延迟/卫星链接：

```
enum_timeout: 10000  # 10 秒
```

ENUM DNS 设置示例

配置私有 **ENUM** 区域（BIND9 格式）：

```
; e164.internal.example.com 的区域文件
$ORIGIN e164.internal.example.com.
$TTL 300

; 号码: +1-555-0100 变为 0.0.1.0.5.5.5.1.e164.internal.example.com
0.0.1.0.5.5.5.1.e164.internal.example.com. IN NAPTR 100 10 "u"
"E2U+sip" "!^.*$!sip:15550100@voip-gateway.example.com!" .
0.0.1.0.5.5.5.1.e164.internal.example.com. IN NAPTR 100 20 "u"
"E2U+pstn" "!^.*$!pstn:gateway-a.example.com!" .

; 号码: +1-555-0200
0.0.2.0.5.5.5.1.e164.internal.example.com. IN NAPTR 100 10 "u"
"E2U+sip" "!^.*$!sip:15550200@voip-gateway.example.com!" .
```

测试 **ENUM** 解析：

```
# 查询 ENUM 域
dig @10.0.1.53 NAPTR 0.0.1.0.5.5.5.1.e164.internal.example.com

# 预期输出包括 NAPTR 记录:
# 0.0.1.0.5.5.5.1.e164.internal.example.com. 300 IN NAPTR 100 10
"u" "E2U+sip" "!^.*$!sip:15550100@voip-gateway.example.com!" .
```

号码翻译配置

基于正则表达式的号码规范化，在路由之前应用。

禁用号码翻译

```
# config/runtime.exs
config :sms_c,
  translation_rules: []
```

本号码翻译示例

为本地号码添加国家代码：

```
config :sms_c,
  translation_rules: [
    %{
      calling_prefix: nil,
      called_prefix: "",
      source_smsc: nil,
      calling_match: "^(\d{10})$",            # 匹配 10 位数字
      calling_replace: "+1\1",                 # 前面加上 +1
      called_match: "^(\d{10})$",
      called_replace: "+1\1",
      priority: 100,
      description: "为 10 位北美号码添加 +1",
      enabled: true
    }
  ]
```

规范化国际格式：

```
%{
  calling_prefix: nil,
  called_prefix: nil,
  source_smsc: nil,
  calling_match: "^00(\d+)$",                    # 匹配 00 前缀
  calling_replace: "+\1",                        # 替换为 +
  called_match: "^00(\d+)$",
  called_replace: "+\1",
  priority: 10,
  description: "将 00 国际前缀转换为 +",
  enabled: true
}
```

移除格式字符:

```
%{
  calling_prefix: nil,
  called_prefix: nil,
  source_smsc: nil,
  calling_match: "^\+?1?[\s\-\.\(\)]*(\d{3})[\s\-\.\)\(]*(\d{3})
[\s\-\.\(\)]*(\d{4})$",
  calling_replace: "+1\1\2\3",
  called_match: "^\+?1?[\s\-\.\(\)]*(\d{3})[\s\-\.\)\(]*(\d{3})
[\s\-\.\(\)]*(\d{4})$",
  called_replace: "+1\1\2\3",
  priority: 50,
  description: "规范化美国电话号码格式",
  enabled: true
}
```

运营商特定翻译

路由代码剥离:

```
%{
  calling_prefix: nil,
  called_prefix: "101",                        # 仅适用于 101 前缀
  source_smsc: "carrier_a",                    # 仅来自此运营商
  calling_match: nil,                          # 不更改呼叫
  calling_replace: nil,
  called_match: "^101(\d+)$",                 # 剥离 101 路由代码
  called_replace: "\1",
  priority: 5,
  description: "从被叫号码中剥离运营商路由代码",
  enabled: true
}
```

多规则翻译

规则按优先级顺序进行评估（数字越小，优先级越高）：

```
config :sms_c,
  translation_rules: [
    # □□□ 1□□□□□□□□□
    %{
      calling_prefix: "1555",
      called_prefix: nil,
      source_smsc: nil,
      calling_match: "^(1555\d{7})$",
      calling_replace: "+\1",
      called_match: nil,
      called_replace: nil,
      priority: 1,
      description: "□□□□□□□",
      enabled: true
    },

    # □□□ 50□□□□□
    %{
      calling_prefix: nil,
      called_prefix: nil,
      source_smsc: nil,
      calling_match: "^(\d{10})$",
      calling_replace: "+1\1",
      called_match: "^(\d{10})$",
      called_replace: "+1\1",
      priority: 50,
      description: "□□ 10 □□□□□□",
      enabled: true
    }
  ]
```

□□□□

□□□□□□□□□

从配置加载路由

```
# config/runtime.exs
config :sms_c,
  sms_routes: [
    # 地理路由示例
    %{
      calling_prefix: nil,
      called_prefix: "+1",
      source_smsc: nil,
      dest_smsc: "north_america_gateway",
      source_type: nil,
      enum_domain: nil,
      auto_reply: false,
      auto_reply_message: nil,
      drop: false,
      charged: :default,
      weight: 100,
      priority: 50,
      description: "北美路由",
      enabled: true
    },

    # 负载均衡路由示例
    %{
      calling_prefix: nil,
      called_prefix: "+44",
      source_smsc: nil,
      dest_smsc: "uk_gateway_1",
      source_type: nil,
      enum_domain: nil,
      auto_reply: false,
      auto_reply_message: nil,
      drop: false,
      charged: :default,
      weight: 70,
      priority: 50,
      description: "英国主网关（70%）",
      enabled: true
    },
    %{
      calling_prefix: nil,
      called_prefix: "+44",
```

```
      source_smsc: nil,
      dest_smsc: "uk_gateway_2",
      source_type: nil,
      enum_domain: nil,
      auto_reply: false,
      auto_reply_message: nil,
      drop: false,
      charged: :default,
      weight: 30,
      priority: 50,
      description: "英国备份网关（30%）",
      enabled: true
    }
  ]
```

跳过初始路由加载

```
# 不从配置加载路由（仅通过 Web UI 管理）
config :sms_c,
  sms_routes: []
```

仅在路由表为空（首次启动）时加载配置中定义的路由。

性能调优配置

有关详细优化策略，请参阅 性能调优指南。

批量插入工作程序

```
# config/config.exs
config :sms_c,
  batch_insert_batch_size: 100,            # 每批□息数量
  batch_insert_flush_interval_ms: 100      # 最大等待时间（毫秒）
```

性能配置文件：

配置文件	批量大小	间隔	吞吐量	延迟
高流量	200	200ms	~5,000 msg/sec	最多 200ms
平衡	100	100ms	~4,500 msg/sec	最多 100ms
低延迟	50	20ms	~3,000 msg/sec	最多 20ms
实时	10	10ms	~1,500 msg/sec	最多 10ms

日志配置

日志级别

```
# config/config.exs
config :logger, :console,
  level: :info,  # :debug, :info, :warning, :error
  format: "$time $metadata[$level] $message\n",
  metadata: [:request_id, :message_id, :route_id]
```

生产推荐: `:info` 或 `:warning` 开发推荐: `:debug`

日志输出目标

仅控制台（开发）：

```
config :logger,
  backends: [:console]
```

文件日志（生产）：

```
config :logger,
  backends: [:console, {LoggerFileBackend, :file_log}]

config :logger, :file_log,
  path: "/var/log/sms_c/application.log",
  level: :info,
  format: "$time $metadata[$level] $message\n",
  metadata: [:request_id, :message_id]
```

日志轮换

使用 **logrotate**（Linux）：

```
# /etc/logrotate.d/sms_c
/var/log/sms_c/*.log {
    daily
    rotate 30
    compress
    delaycompress
    notifempty
    create 0644 sms_user sms_group
    sharedscripts
    postrotate
        # 信号应用程序重新打开日志文件
        systemctl reload sms_c
    endscript
}
```

常见配置场景

高流量聚合器

优化以实现最大吞吐量（5,000+ 消息/秒）：

```
# 数据库
config :sms_c, SmsC.Repo,
  pool_size: 50

# 批量工作程序
config :sms_c,
  batch_insert_batch_size: 200,
  batch_insert_flush_interval_ms: 200

# □息保留
config :sms_c,
  dead_letter_time_minutes: 1440  # 24 小时

# 计费（为性能禁用）
config :sms_c,
  default_charging_enabled: false

# 清理（延长间隔）
config :sms_c,
  cleanup_interval_minutes: 30
```

企业实时消息

优化以实现低延迟（< 20ms）：

```
# 数据库
config :sms_c, SmsC.Repo,
  pool_size: 20

# 批量工作程序（低延迟）
config :sms_c,
  batch_insert_batch_size: 20,
  batch_insert_flush_interval_ms: 10

# □息保留
config :sms_c,
  dead_letter_time_minutes: 4320  # 3 天

# 计费（启用）
config :sms_c,
  default_charging_enabled: true,
  ocs_url: "http://ocs.local:2080/jsonrpc"
```

开发/测试

优化以进行调试和可见性：

```
# 数据库
config :sms_c, SmsC.Repo,
  pool_size: 5

# 批量工作程序（立即）
config :sms_c,
  batch_insert_batch_size: 1,
  batch_insert_flush_interval_ms: 10

# 日志（详细）
config :logger, :console,
  level: :debug

# □息保留（短）
config :sms_c,
  dead_letter_time_minutes: 60  # 1 小时

# 计费（禁用）
config :sms_c,
  default_charging_enabled: false
```

多租户服务提供商

每个租户的单独配置：

```
# 租户 1 环境
export DB_NAME=sms_c_tenant1
export OCS_TENANT=tenant1.example.com
export OCS_ACCOUNT=tenant1_account
export NODE_NAME=sms_tenant1@node1.example.com

# 租户 2 环境
export DB_NAME=sms_c_tenant2
export OCS_TENANT=tenant2.example.com
export OCS_ACCOUNT=tenant2_account
export NODE_NAME=sms_tenant2@node1.example.com
```

地理冗余

跨区域集群：

```
# 美国东部集群
config :sms_c,
  cluster_nodes: [
    :"sms@us-east-1a.example.com",
    :"sms@us-east-1b.example.com",
    :"sms@us-west-1a.example.com"  # 跨区域用于灾难恢复
  ],
  smsc_node_name: "us-east-1a"
```

配置验证

在部署之前测试配置：

```
# 检查配置语法
mix compile

# 验证数据库连接
mix ecto.create
mix ecto.migrate

# 测试 OCS 连接性（如果启用）
curl -X POST http://localhost:2080/jsonrpc -H "Content-Type:
application/json" \
  -d '{"method":"SessionSv1.Ping","params":[],"id":1}'

# 以交互模式启动应用程序
iex -S mix phx.server
```

环境变量参考

配置中使用的常见环境变量：

变量	目的	示例
`DB_USERNAME`	数据库用户名	`sms_prod_user`
`DB_PASSWORD`	数据库密码	`strong_password`
`DB_HOSTNAME`	数据库主机	`db.internal.example.com`
`DB_PORT`	数据库端口	`3306`
`DB_NAME`	数据库名称	`sms_c_production`
`DB_POOL_SIZE`	连接池大小	`30`
`API_PORT`	API 监听端口	`8443`
`API_LISTEN_IP`	API 监听 IP	`0.0.0.0`
`WEB_PORT`	Web UI 端口	`443`
`NODE_NAME`	Erlang 节点名称	`sms@node1.example.com`
`ERLANG_COOKIE`	集群密钥	`shared_cookie_value`
`OCS_URL`	OCS API URL	`http://ocs.local:2080/jsonrpc`
`OCS_TENANT`	OCS 租户	`sms.example.com`

配置最佳实践

1. 使用环境变量处理敏感值（密码、API 密钥）
2. 在生产之前测试配置更改于暂存环境
3. 在部署说明中记录自定义设置
4. 对配置文件进行版本控制（不包括机密）
5. 在更改后进行监控以检测性能回归

6. 保留工作配置的备份
7. 在重启之前进行验证以避免启动失败
8. 在各个环境中使用一致的命名
9. 根据硬件设置资源限制
10. 定期审查以删除未使用的功能

故障排除配置问题

症状	可能原因	解决方案
应用程序无法启动	配置中的语法错误	检查日志，验证语法
数据库连接失败	错误的凭据/主机	验证 DB_* 环境变量
API 无法访问	错误的端口/IP 绑定	检查 API_PORT 和 listen_ip
集群节点无法连接	Cookie 不匹配，防火墙	验证 ERLANG_COOKIE，检查端口 4369、9100-9200
计费失败	OCS 无法访问	测试 ocs_url 的连接性
ENUM 查找失败	DNS 服务器无法访问	测试 DNS 连接性，检查超时
性能不佳	错误的批量设置	查看性能调优指南
消息未路由	路由未加载	检查 sms_routes 配置或 Web UI

如需更多帮助，请参阅 故障排除指南。

消息存储配置（Mnesia）

消息保留

消息存储在 Mnesia 中以便快速访问，并具有可配置的自动清理。

```
config :sms_c,
  # 在 Mnesia 中保留□息的时间（小时）
  message_retention_hours: 24,

  # 检查旧□息的频率（分钟）
  retention_check_interval_minutes: 60
```

建议：

- 生产：24-72 小时（平衡操作需求与内存）
- 开发：4-8 小时（更快的清理以进行测试）
- 高流量：12-24 小时（节省内存）

内存影响：

- 平均消息：~1KB
- 10,000 消息：~10MB
- 100,000 消息：~100MB

CDR（通话详细记录）导出

当消息被投递或过期时，CDR 可以自动写入您的 Ecto 数据库以进行长期存储和计费分析。

```
config :sms_c,
  # 启用/禁用 CDR 写入
  cdr_enabled: true
```

CDR 记录包括：

- 消息 ID，呼叫/被叫号码

- 源/目的 SMSC
- 源/目的节点（对于集群）
- 提交、投递、过期时间戳
- 状态，投递尝试
- 可选消息正文（见隐私控制）

何时禁用：

- ��需要 CDR 的测试环境
- 临时故障排除以减少数据库负载

隐私控制

配置消息正文的可见性和保留，以符合隐私合规性。

```
config :sms_c,
  # 成功投递后从 Mnesia 中删除□息正文
  delete_message_body_after_delivery: false,

  # 在 Web UI 中隐藏□息正文
  hide_message_body_in_ui: false,

  # 在 CSV 导出中隐藏□息正文
  hide_message_body_in_export: false
```

用例：

配置	用例
`delete_message_body_after_delivery: true`	节省 Mnesia 空间，符合隐私合规
`hide_message_body_in_ui: true`	防止操作员查看消息内容
`hide_message_body_in_export: true`	数据导出合规，清理报告

示例配置：

□□□□□□□□

```
config :sms_c,
  delete_message_body_after_delivery: true,
  hide_message_body_in_ui: true,
  hide_message_body_in_export: true,
  cdr_enabled: true  # 保留没有正文的 CDR
```

□□□□□□□□□

```
config :sms_c,
  delete_message_body_after_delivery: false,
  hide_message_body_in_ui: false,
  hide_message_body_in_export: false,
  cdr_enabled: true
```

启动日志

在应用程序启动时，配置状态会被记录：

```
[info] □息存储: Mnesia（保留: 24小时）
[info] CDR ���出: 已启用
[info] 投递后删除正文: 已禁用
[info] OCS 计费: 已启用（url: http://..., tenant: ...）
```

这提供了对活动功能的即时可见性。

SMS-C Prometheus 指标文档

← 返回文档索引 | 主 README

概述

本文档描述了 SMS-C 系统暴露的所有 Prometheus 指标。这些指标旨在供运维人员监控系统健康、性能，并排查问题。

访问指标

Prometheus 指标端点可在以下地址访问：

```
http://localhost:9568/metrics
```

端点以 Prometheus 文本格式暴露指标，可以被 Prometheus 服务器抓取。指标在系统处理消息时实时更新。

指标命名约定

所有指标遵循以下模式：`sms_c.<category>.<metric_name>.<type>`

类别：

- `license` - 许可证状态指标
- `message` - 消息处理指标
- `routing` - 路由决策指标
- `enum` - ENUM/NAPTR 查找指标
- `delivery` - 消息投递指标
- `queue` - 队列管理指标
- `charging` - 计费/收费指标
- `mnesia` - 数据库指标

- `frontend` - 前端连接指标
- `location` - 位置/注册指标
- `phoenix.endpoint` - HTTP API 请求指标
- `vm` - Erlang VM 系统指标

许可证指标

sms_c_license_status

类型: Gauge

描述: OmniMessage SMS-C 系统的当前许可证状态。

值:

- `1` - 有效许可证
- `0` - 无效/过期许可证

标签: 无

产品名称: `omnimessage`

用例: 监控许可证有效性，以确保系统在有效许可证下运行。当无效时，消息仍然会被接收，但会被路由到目的地 "NOLICENCE" 而不是正常路由。

许可证无效时的行为:

- 入站消息被 接受 并存储
- 消息目的地 (`dest_smsc`) 自动设置为 **"NOLICENCE"**
- 正常路由被 绕过
- UI 和监控保持 可访问
- 数据库和所有服务保持 正常运行

告警:

```
- alert: SMS_C_License_Invalid
  expr: sms_c_license_status == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: "SMS-C 许可证无效或过期"
    description: "许可证状态无效 - □息被路由到 NOLICENCE"
```

示例 **Prometheus** 查询:

```
# 检查许可证是否有效
sms_c_license_status == 1

# 对无效许可证发出警报
sms_c_license_status == 0

# 计算路由到 NOLICENCE 的□息数量（表明许可证问题）
sms_c_routing_route_matched_count{dest_smsc="NOLICENCE"}
```

消息处理指标

sms_c_message_received_count

类型: Counter

描述: SMS-C 从所有来源接收到的消息总数。

标签:

- `source_smsc`: 发送消息的源 SMSC 名称
- `source_type`: 源连接类型 (ims, circuit_switched, smpp)
- `message_type`: 消息类型 (sms, mms)

用例: 按来源和类型监控入站消息量。用于检测流量模式、识别繁忙时段，并发现消息流中的异常。

告警：对突然下降（潜在的源连接问题）或激增（潜在的攻击/垃圾邮件）设置告警。

---

sms_c_message_validated_count

类型：Counter

描述：执行的消息验证总数。

标签：

- `valid`：验证是否通过 (true 或 false)

用例：跟踪验证成功/失败率。高失败率可能表明消息格式错误或集成问题。

告警：当验证失败率超过阈值时发出警报（例如，> 5% 失败）。

---

sms_c_message_processing_stop_duration

类型：Histogram

描述：从接收消息到完成处理所需的时间（包括验证、路由和排队）。

单位：毫秒

桶：10, 50, 100, 250, 500, 1000, 2500, 5000 ms

标签：

- `success`：处理是否成功 (true 或 false)

用例：监控端到端消息处理性能。识别处理管道中的减速。

告警：当 p95 或 p99 延迟超过 SLA 阈值时发出警报。

路由指标

sms_c_routing_route_matched_count

类型: Counter

描述：特定路由被匹配并选择用于消息路由的总次数。

标签：

- `route_id`: 匹配路由的唯一标识符
- `dest_smsc`: 路由选择的目标 SMSC
- `priority`: 匹配路由的优先级值

用例：理解哪些路由被使用得最频繁。识别未充分利用或过载的路由。对容量规划和路由优化很有用。

告警：如果高优先级路由很少被匹配（可能表明路由配置错误）则发出警报。

---

sms_c_routing_failed_count

类型: Counter

描述：找不到合适路由的路由失败总数。

标签：

- `reason`: 失败原因 (no_route_found, validation_failed, 等)

用例：跟踪路由失败以识别配置缺口或意外流量模式。

告警：对任何路由失败发出警报，因为它们表明消息无法送达。

---

sms_c_routing_action_count

类型: Counter

描述：采取的特殊路由操作总数。

标签：

- `action`：操作类型 (drop, auto_reply, forward)
- `route_id`：触发操作的路由

用例：监控丢弃规则（反垃圾邮件）、自动回复使用情况和转发模式。

告警：对丢弃操作的意外激增发出警报（可能表明垃圾邮件攻击）。

---

sms_c_routing_stop_duration

类型：Histogram

描述：评估所有路由并选择最佳匹配所需的时间。

单位：毫秒

桶：1, 5, 10, 25, 50, 100, 250, 500 ms

标签：

- `dest_smsc`：选择的目标 SMSC

用例：监控路由引擎性能。缓慢的路由表明路由过多或匹配逻辑复杂。

告警：当路由所需时间持续超过预期时发出警报（例如，p95 > 50ms）。

ENUM/NAPTR 查找指标

sms_c_enum_cache_hit_count

类型：Counter

描述：从缓存中服务的 ENUM 查找总数（不需要 DNS 查询）。

标签：

- `domain`: □□□ ENUM □

□□: □□□□□□□□□□□□□□□□□ DNS □□□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

---

sms_c_enum_cache_miss_count

□□: Counter

□□: □□ DNS □□□ ENUM □□□□□□□□□□□□□

□□:

- `domain`: □□□ ENUM □

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□: `cache_hit_rate = hits / (hits + misses)`

---

sms_c_enum_cache_size_size

□□: Gauge

□□: ENUM □□□□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

---

sms_c_enum_lookup_stop_duration

□□: Histogram

□□: □□ ENUM □□□□□□□□□□□□□□□□□ DNS □□□□

□□: □□

桶: 10, 50, 100, 250, 500, 1000, 2500, 5000 ms

标签:

- `domain`: 查询的 ENUM 域
- `success`: 查找是否成功 (true 或 false)
- `cache_hit`: 结果是否来自缓存 (true 或 false)

用例: 监控 ENUM 查找性能。识别缓慢的 DNS 服务器或网络问题。

告警: 当 p95 查找时间超过超时阈值时发出警报。

---

sms_c_enum_naptr_records_record_count

类型: Histogram

描述: 成功 ENUM 查找返回的 NAPTR 记录数量。

桶: 0, 1, 2, 3, 5, 10

标签:

- `domain`: 查询的 ENUM 域

用例: 理解 ENUM 记录分布。大多数查找应返回 1-3 条记录。

告警: 如果经常返回 0 条记录发出警报（DNS 配置问题）。

投递指标

sms_c_delivery_queued_count

类型: Counter

描述: 排队等待投递到目标 SMSC 的消息总数。

标签:

- `dest_smsc`: 目标 SMSC 名称

用例：监控每个目标的消息流。对容量规划很有用。

告警：与投递成功/失败计数进行比较以检测积累。

---

sms_c_delivery_attempted_count

类型: Counter

描述：进行的投递尝试总数（包括重试）。

标签：

- `dest_smsc`: 目标 SMSC 名称

用例：跟踪投递尝试量。相对于排队计数的高尝试计数表明重试行为。

---

sms_c_delivery_succeeded_count

类型: Counter

描述：成功投递到目标 SMSC 的消息总数。

标签：

- `dest_smsc`: 目标 SMSC 名称

用例：跟踪每个目标的成功投递。主要成功指标。

告警：如果成功率低于 SLA 阈值发出警报。

计算: `success_rate = succeeded / queued`

---

sms_c_delivery_failed_count

类型: Counter

描述：在所有重试尝试后未能投递的消息总数。

标签：

- `dest_smsc`：目标 SMSC 名称
- `reason`：失败原因

用例：跟踪投递失败以识别问题目标或失败模式。

告警：对高失败率或特定失败原因发出警报。

---

sms_c_delivery_dead_letter_count

类型：Counter

描述：移动到死信队列（无法投递）的消息总数。

标签：

- `reason`：死信原因（例如，`max_retries_exceeded`，`expired`）

用例：监控需要手动干预的无法投递的消息。

告警：对任何死信事件发出警报，因为它们表示完全投递失败。

---

sms_c_delivery_succeeded_duration

类型：Histogram

描述：从消息排队到成功投递的端到端时间。

单位：毫秒

桶：100, 500, 1000, 5000, 10000, 30000, 60000 ms

标签：

- `dest_smsc`：目标 SMSC 名称

用例：监控投递延迟。识别缓慢的目标或网络问题。

告警：当 p95 投递时间超过 SLA 阈值时发出警报。

---

sms_c_delivery_succeeded_attempt_count

类型：Histogram

描述：成功投递前所需的投递尝试次数。

桶：1, 2, 3, 5, 10

标签：

- `dest_smsc`：目标 SMSC 名称

用例：理解重试行为。大多数投递应在第一次尝试时成功。

告警：如果平均尝试次数超过 2 发出警报（表明目标可靠性问题）。

---

sms_c_delivery_failed_attempt_count

类型：Histogram

描述：最终失败前进行的投递尝试次数。

桶：1, 2, 3, 5, 10

标签：

- `dest_smsc`：目标 SMSC 名称

用例：理解在放弃之前发生多少次重试。

队列指标

sms_c_queue_size_size

类型: Gauge

描述：当前队列中消息的总数（所有状态合并）。

标签：

- `queue_type`：队列类型 (message_queue, dead_letter)

用例：监控队列深度以检测积压或处理问题。

告警：当队列大小超过容量阈值时发出警报。

---

sms_c_queue_size_pending

类型: Gauge

描述：当前待投递的消息数量（尚未尝试）。

标签：

- `queue_type`：队列类型

用例：监控待处理消息计数。高待处理计数表明处理延迟。

告警：当待处理计数在较长时间内超过阈值时发出警报。

---

sms_c_queue_size_failed

类型: Gauge

描述：当前处于失败状态的消息数量（等待重试）。

标签：

- `queue_type`: □□□□

□□: □□□□□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□

---

sms_c_queue_size_delivered

□□: Gauge

□□: □□□□□□□/□□□□□□□□□□□□□□□□

□□:

- `queue_type`: □□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□

---

sms_c_queue_oldest_message_age_seconds

□□: Gauge

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□:

- `queue_type`: □□□□

□□: □��□□□□□□□□□□□□ SLA □□□□□□□□

□□: □□□□□□□□□□□□ SLA □□□□□□□□□□□□□> 300 □□□

计费指标

sms_c_charging_requested_count

类型: Counter

描述: 向 OCS 或计费系统发出的计费请求总数。

标签:

- `account`: 被计费的账户标识符

用例: 跟踪每个账户的计费量。对账单对账很有用。

---

sms_c_charging_succeeded_count

类型: Counter

描述: 成功的计费操作总数。

标签:

- `account`: 被计费的账户标识符

用例: 监控每个账户的计费成功率。

计算: `success_rate = succeeded / requested`

---

sms_c_charging_failed_count

类型: Counter

描述: 失败的计费操作总数。

标签:

- `account`: 账户标识符

- `reason`: □□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□

---

sms_c_charging_succeeded_duration

□□: Histogram

□□: □□□□□□□□□□□□□□□□

□□: □□

□: 10, 50, 100, 250, 500, 1000, 2500, 5000 ms

□□:

- `account`: □□□□□

□□: □□□□□□□□□□□□□□□□□□□□

告警：对意外表增长率发出警报。

---

sms_c_frontend_status_count

类型：Gauge

描述：每个连接状态下的前端数量。

标签：

- `frontend_name`：前端标识符
- `status`：连接状态 (connected, disconnected)

用例：监控前端连接性。检测连接失败。

告警：当预期的前端断开连接时发出警报。

---

sms_c_location_registered_count

类型：Counter

描述：系统接收到的位置信息/用户注册的总数。

标签：

- `location`：用户注册的前端/SMSC 名称
- `ims_capable`：用户是否支持 IMS (true/false)

用例：监控用户注册活动。跟踪 IMS 与非 IMS 用户。检测注册风暴或失败。

告警：设置告警以监控：

- 注册率下降（可能表明网络问题）
- 注册的异常激增
- 非 IMS 注册的高比例（旧设备涌入）

示例查询：

```
# 每分钟注册率
rate(sms_c_location_registered_count[1m])

# IMS 与非 IMS 注册比例
sum(rate(sms_c_location_registered_count{ims_capable="true"}[5m]))
/
sum(rate(sms_c_location_registered_count[5m]))
```

HTTP API 请求指标

phoenix_endpoint_stop_duration

类型: Distribution (Histogram)

描述: HTTP 请求处理持续时间（以毫秒为单位），从请求开始到响应完成。

标签:

- `route`: API 端点路由（例如，`/api/messages`, `/api/frontends`）

桶: 10ms, 50ms, 100ms, 250ms, 500ms, 1s, 2.5s, 5s

用例: 监控 API 性能。识别缓慢的端点。跟踪响应时间 SLA。

告警: 设置告警以监控:

- 关键端点的 P95 延迟 > 500ms
- 任何端点的 P99 延迟 > 1s
- 延迟趋势上升

示例查询:

```
# 按端点的 P95 响应时间
histogram_quantile(0.95,
  rate(phoenix_endpoint_stop_duration_bucket[5m]))

# 响应时间超过 1 秒的请求
sum(rate(phoenix_endpoint_stop_duration_bucket{le="1000"}[5m]))
```

---

phoenix_endpoint_stop_count

类型: Counter

描述: 按路由和 HTTP 状态码分类的完成 HTTP 请求总数。

标签:

- `route`: API 端点路由
- `status`: HTTP 状态码 (200, 201, 400, 404, 500, 等)

用例: 监控 API 请求量和成功率。按端点跟踪错误率。

告警: 设置告警以监控:

- 任何端点的错误率 > 5%
- 关键端点的 5xx 错误
- 请求量的突然下降

示例查询:

```
# 每个端点的请求率
sum by (route) (rate(phoenix_endpoint_stop_count[5m]))

# 按端点的错误率
sum by (route) (rate(phoenix_endpoint_stop_count{status=~"5.."}
[5m])) /
sum by (route) (rate(phoenix_endpoint_stop_count[5m]))

# 成功率
sum(rate(phoenix_endpoint_stop_count{status=~"2.."}[5m])) /
sum(rate(phoenix_endpoint_stop_count[5m]))
```

---

phoenix_router_dispatch_exception_count

类型: Counter

描述: 在 HTTP 请求处理期间引发的异常/错误总数。

标签:

- `route`: 发生异常的 API 端点路由
- `kind`: 异常类型 (error, exit, throw)

用例: 跟踪应用程序错误。识别问题端点。监控系统稳定性。

告警: 对关键端点的任何非零值设置告警。

示例查询:

```
# 按端点的异常率
rate(phoenix_router_dispatch_exception_count[5m])

# 最近一小时的总异常
increase(phoenix_router_dispatch_exception_count[1h])
```

Erlang VM □□

vm_memory_total

□□: Gauge

□□: Erlang VM □□□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□: □□□□□□ > 80% □□□□□□□□□□□□□□□

---

vm_memory_processes

□□: Gauge

□□: Erlang □□□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□□

---

vm_total_run_queue_lengths_total

□□: Gauge

□□: □□ CPU □□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□ CPU □□□

□□: □□□ > 10 * CPU □□□□□□□□□□

---

vm_system_counts_process_count

□□: Gauge

描述：当前在 VM 中运行的进程数量。

用例：监控进程创建模式。检测进程泄漏。

告警：当接近进程限制（默认 262,144）时发出警报。

指标收集和轮询

系统每 10 秒自动收集以下指标：

- 队列大小和年龄
- Mnesia 表大小
- ENUM 缓存统计

所有其他指标都是事件驱动的，并在相应操作发生时发出。

常见监控模式

按目标的投递成功率

跟踪每个目标 SMSC 的消息投递成功率：

公式：`(sms_c_delivery_succeeded_count) / (sms_c_delivery_queued_count)`

解释：应该 > 95% 以确保目标健康。较低的比率表明投递问题。

端到端消息延迟

监控从消息接收到投递的总时间：

指标：

- `sms_c_message_processing_stop_duration`（处理）
- `sms_c_delivery_succeeded_duration`（投递）

解释：总和表示用户面临的延迟。

ENUM 缓存有效性

测量 ENUM 缓存的性能：

公式：`(sms_c_enum_cache_hit_count) / (sms_c_enum_cache_hit_count + sms_c_enum_cache_miss_count)`

解释：应该 > 80% 在预热后。较低的比率可能表明短 TTL 或高流量波动。

路由利用率

识别哪些路由处理最多流量：

指标：`sms_c_routing_route_matched_count` 按 `route_id` 分组

解释：用于识别热路由以进行优化和容量规划。

队列积压趋势

监控消息队列是增长（积压）还是缩小（赶上）：

指标：

- `sms_c_queue_size_pending` (当前待处理)
- `sms_c_queue_oldest_message_age_seconds` (年龄趋势)

解释：增长的待处理计数 + 增加的年龄 = 积压形成。

重试率

了解投递重试的频率：

指标：`sms_c_delivery_succeeded_attempt_count` histogram percentiles

□□: □□ p95 > 1□□□□□□□□□□□□□□□□□□□□□

推荐告警

告警	条件	严重性	描述
高路由失败率	`routing_failed_count` 增加	Critical	消息无法路由
队列积压	`queue_size_pending` > 阈值	Warning	消息积累
队列中的旧消息	`queue_oldest_message_age_seconds` > 300	Critical	SLA 违规
投递失败激增	`delivery_failed_count` 激增	High	目标问题
死信事件	`delivery_dead_letter_count` > 0	High	无法投递的消息
ENUM 查找超时	`enum_lookup_stop_duration` p95 > 5000ms	Warning	DNS 问题
低缓存命中率	ENUM 缓存命中率 < 0.7	Warning	缓存无效
前端断开连接	`frontend_status_count{status="disconnected"}` > 0	High	连接丢失
计费失败	`charging_failed_count` > 阈值	High	计费问题
消息处理缓慢	`message_processing_stop_duration` p95 > 1000ms	Warning	性能下降

仪表板推荐

运维仪表板

目的：实时系统健康监控

面板：

1. 消息吞吐量（每分钟接收/处理/投递）
2. 队列大小（待处理、失败、已投递）
3. 按目标的投递成功率
4. p95 处理和投递延迟
5. 活跃前端状态
6. 当前告警

性能仪表板

目的：系统性能分析

面板：

1. 消息处理持续时间直方图
2. 路由持续时间直方图
3. ENUM 查找���续时间直方图
4. 计费持续时间直方图
5. 投递尝试分布
6. 缓存命中率

业务仪表板

目的：流量和使用分析

面板：

1. 按源 SMSC 的消息
2. 按目标 SMSC 的消息
3. 路由利用率热图
4. 自动回复和丢弃操作计数
5. ENUM 使用统计
6. 按账户的计费量

指标保留

推荐的 Prometheus 保留设置:

- 原始指标: 15 天
- **5** 分钟聚合: 90 天
- **1** 小时聚合: 2 年

这提供了详细的近期历史，同时保持长期趋势以进行容量规划。

使用指标进行故障排除

场景：消息未能投递

调查步骤:

1. 检查 `sms_c_message_received_count` - 消息是否被接收?
2. 检查 `sms_c_routing_failed_count` - 它们是否被路由?
3. 检查 `sms_c_delivery_queued_count` - 它们是否被排队?
4. 检查 `sms_c_delivery_failed_count` - 投递尝试是否失败?
5. 检查 `dest_smsc` 标签以识别问题目标

□□□□□□□□□

□□□□□

1. □□ `sms_c_message_processing_stop_duration` □□□ - □□□□□□□
2. □□ `sms_c_routing_stop_duration` - □□□□□□□□
3. □□ `sms_c_enum_lookup_stop_duration` - ENUM □□□□□□□
4. □□ `sms_c_charging_succeeded_duration` - □□□□□□□□
5. □□□□□□□□□□□□

□□□□□□□□□

□□□□□

1. □□ `sms_c_queue_size_pending` □□ - □□□□□□□
2. □□ `sms_c_delivery_attempted_count` - □□□□□□□□□
3. □□ `sms_c_delivery_failed_count` - □□□□□□□
4. □□ `sms_c_delivery_succeeded_duration` - □□□□□□□□□□□
5. □□ `dest_smsc` □□□□□□□□□□

Prometheus □□□

□□□□

□□□□□□□□**5** □□□□□:

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

□□□□□□□□□**1** □□□□□:

```
rate(sms_c_message_received_count[1h]) * 60
```

今天的总消息：

```
increase(sms_c_message_received_count[24h])
```

按源类型的消息：

```
sum by (source_type) (rate(sms_c_message_received_count[5m]))
```

按源 **SMSC** 的消息：

```
sum by (source_smsc) (rate(sms_c_message_received_count[5m]))
```

投递性能

投递成功率（百分比）：

```
(rate(sms_c_delivery_succeeded_count[5m]) /
rate(sms_c_delivery_queued_count[5m])) * 100
```

投递失败率（百分比）：

```
(rate(sms_c_delivery_failed_count[5m]) /
rate(sms_c_delivery_queued_count[5m])) * 100
```

平均投递尝试（**p95**）：

```
histogram_quantile(0.95,
sms_c_delivery_succeeded_attempt_count_bucket)
```

按目标的投递成功：

```
sum by (dest_smsc) (rate(sms_c_delivery_succeeded_count[5m]))
```

投递失败原因：

```
sum by (reason) (rate(sms_c_delivery_failed_count[5m]))
```

投递时间（**p95**）：

```
histogram_quantile(0.95, sms_c_delivery_succeeded_duration_bucket)
```

投递时间（**p99**）：

```
histogram_quantile(0.99, sms_c_delivery_succeeded_duration_bucket)
```

队列指标

当前待处理消息：

```
sms_c_queue_size_pending
```

等待重试的失败消息：

```
sms_c_queue_size_failed
```

最旧消息的年龄（分钟）：

```
sms_c_queue_oldest_message_age_seconds / 60
```

队列增长率（消息/小时）：

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

进入队列的消息：

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

离开队列的消息：

```
rate(sms_c_delivery_succeeded_count[5m]) +
rate(sms_c_delivery_failed_count[5m])
```

队列积压（进入 - 离开）：

```
rate(sms_c_delivery_queued_count[5m]) -
(rate(sms_c_delivery_succeeded_count[5m]) +
rate(sms_c_delivery_failed_count[5m]))
```

路由性能

路由成功率：

```
(1 - (rate(sms_c_routing_failed_count[5m]) /
(rate(sms_c_routing_route_matched_count[5m]) +
rate(sms_c_routing_failed_count[5m])))) * 100
```

使用最多的路由：

```
topk(10, sum by (route_id, dest_smsc)
(rate(sms_c_routing_route_matched_count[1h])))
```

路由延迟（**p50, p95, p99**）：

```
histogram_quantile(0.50, sms_c_routing_stop_duration_bucket)
histogram_quantile(0.95, sms_c_routing_stop_duration_bucket)
histogram_quantile(0.99, sms_c_routing_stop_duration_bucket)
```

每分钟的路由失败：

```
rate(sms_c_routing_failed_count[5m]) * 60
```

每小时的丢弃操作：

```
increase(sms_c_routing_action_count{action="drop"}[1h])
```

每小时的自动回复操作：

```
increase(sms_c_routing_action_count{action="auto_reply"}[1h])
```

ENUM 性能

ENUM 缓存命中率：

```
rate(sms_c_enum_cache_hit_count[5m]) /
(rate(sms_c_enum_cache_hit_count[5m]) +
rate(sms_c_enum_cache_miss_count[5m]))
```

ENUM 缓存命中百分比：

```
(rate(sms_c_enum_cache_hit_count[5m]) /
(rate(sms_c_enum_cache_hit_count[5m]) +
rate(sms_c_enum_cache_miss_count[5m]))) * 100
```

ENUM 查找延迟（**p95**）：

```
histogram_quantile(0.95, sms_c_enum_lookup_stop_duration_bucket)
```

每秒的 **ENUM** 查找（缓存与未缓存）：

```
# 缓存（快速）
rate(sms_c_enum_cache_hit_count[5m])

# 未缓存（需要 DNS 查询）
rate(sms_c_enum_cache_miss_count[5m])
```

返回的平均 **NAPTR** 记录：

```
rate(sms_c_enum_naptr_records_record_count_sum[5m]) /
rate(sms_c_enum_naptr_records_record_count_count[5m])
```

ENUM 缓存大小：

```
sms_c_enum_cache_size_size
```

处理性能

消息处理延迟（**p95**）：

```
histogram_quantile(0.95,
sms_c_message_processing_stop_duration_bucket)
```

消息处理延迟（**p99**）：

```
histogram_quantile(0.99,
sms_c_message_processing_stop_duration_bucket)
```

处理失败：

```
rate(sms_c_message_processing_stop_duration_count{success="false"}
[5m])
```

验证失败率：

```
rate(sms_c_message_validated_count{valid="false"}[5m]) /
rate(sms_c_message_validated_count[5m])
```

计费指标

计费成功率：

```
rate(sms_c_charging_succeeded_count[5m]) /
rate(sms_c_charging_requested_count[5m])
```

每分钟的计费失败：

```
rate(sms_c_charging_failed_count[5m]) * 60
```

计费延迟（**p95**）：

```
histogram_quantile(0.95, sms_c_charging_succeeded_duration_bucket)
```

按账户的计费量：

```
sum by (account) (rate(sms_c_charging_requested_count[1h]))
```

前端健康

活跃前端：

```
sum(sms_c_frontend_status_count{status="connected"})
```

断开连接的前端：

```
sum(sms_c_frontend_status_count{status="disconnected"})
```

按名称的前端：

```
sum by (frontend_name)
(sms_c_frontend_status_count{status="connected"})
```

系统健康

Mnesia 表大小：

```
sms_c_mnesia_table_size_record_count
```

路由计数：

```
sms_c_mnesia_table_size_record_count{table="sms_route"}
```

翻译规则计数：

```
sms_c_mnesia_table_size_record_count{table="translation_rule"}
```

Grafana 仪表板示例

仪表板 1：实时运维

目的：监控当前系统活动和健康。

面板：

1. 消息吞吐量（图形）
    - 查询：`rate(sms_c_message_received_count[5m])`
    - 查询：`rate(sms_c_delivery_succeeded_count[5m])`
    - 单位：messages/second
    - 图例：{{source_type}}

2. 投递成功率（仪表）
    - 查询: `(rate(sms_c_delivery_succeeded_count[5m]) / rate(sms_c_delivery_queued_count[5m])) * 100`
    - 单位: percent (0-100)
    - 阈值:
        - 红色: < 90
        - 黄色: 90-95
        - 绿色: > 95
3. 队列深度（图形）
    - 查询: `sms_c_queue_size_pending`
    - 查询: `sms_c_queue_size_failed`
    - 单位: messages
    - 图例: {{queue_type}}
4. 最旧消息年龄（状态）
    - 查询: `sms_c_queue_oldest_message_age_seconds / 60`
    - 单位: minutes
    - 阈值:
        - 绿色: < 5
        - 黄色: 5-10
        - 红色: > 10
5. 活跃前端（状态）
    - 查询: `sum(sms_c_frontend_status_count{status="connected"})`
    - 单位: count
    - 颜色: 蓝色
6. 路由失败（图形）
    - 查询: `rate(sms_c_routing_failed_count[5m]) * 60`
    - 单位: failures/minute
    - 警报阈值: > 0

仪表板 2：性能分析

目的：分析系统性能并识别瓶颈。

面板：

1. 端到端延迟（图形）
    - 查询：`histogram_quantile(0.50, sms_c_message_processing_stop_duration_bucket)` (p50)
    - 查询：`histogram_quantile(0.95, sms_c_message_processing_stop_duration_bucket)` (p95)
    - 查询：`histogram_quantile(0.99, sms_c_message_processing_stop_duration_bucket)` (p99)
    - 单位：milliseconds
    - 图例：Percentile
2. 组件延迟（条形仪表）
    - 路由：`histogram_quantile(0.95, sms_c_routing_stop_duration_bucket)`
    - ENUM：`histogram_quantile(0.95, sms_c_enum_lookup_stop_duration_bucket)`
    - 计费：`histogram_quantile(0.95, sms_c_charging_succeeded_duration_bucket)`
    - 投递：`histogram_quantile(0.95, sms_c_delivery_succeeded_duration_bucket)`
    - 单位：milliseconds
    - 水平条形
3. 投递尝试分布（热图）
    - 查询：`sms_c_delivery_succeeded_attempt_count_bucket`
    - 显示通常需要多少次尝试
    - 颜色比例：蓝色（1 次尝试）到红色（多次尝试）
4. **ENUM** 缓存性能（图形）

- 命中率：`rate(sms_c_enum_cache_hit_count[5m]) / (rate(sms_c_enum_cache_hit_count[5m]) + rate(sms_c_enum_cache_miss_count[5m]))`
- 缓存大小：`sms_c_enum_cache_size_size`
- 双 Y 轴（速率与大小）

5. 处理成功率（仪表）

   - 查询：`(rate(sms_c_message_processing_stop_duration_count{success="true"}[5m]) / rate(sms_c_message_processing_stop_duration_count[5m])) * 100`
   - 单位：percent
   - 阈值：
     - 红色：< 95
     - 黄色：95-99
     - 绿色：> 99

仪表板 3：流量分析

目的：分析消息流量模式和路由分布。

面板：

1. 按源类型的消息（饼图）

   - 查询：`sum by (source_type) (increase(sms_c_message_received_count[1h]))`
   - 显示分布：IMS vs CS vs SMPP

2. 按源 **SMSC** 的消息（条形图）

   - 查询：`sum by (source_smsc) (rate(sms_c_message_received_count[1h]))`
   - 前 10 个源
   - 水平条形

3. 路由利用率（表格）

- 列：
    - 路由 ID
    - 目标 SMSC
    - 消息（1h）：`sum by (route_id, dest_smsc) (increase(sms_c_routing_route_matched_count[1h]))`
    - 优先级
    - 成功率
- 按消息计数排序

4. 按目标的投递（图形）

    - 查询：`sum by (dest_smsc) (rate(sms_c_delivery_succeeded_count[5m]))`
    - 单位：messages/second
    - 堆叠面积图
    - 图例：{{dest_smsc}}

5. 丢弃/自动回复操作（状态）

    - 丢弃：`increase(sms_c_routing_action_count{action="drop"}[1h])`
    - 自动回复：`increase(sms_c_routing_action_count{action="auto_reply"}[1h])`
    - 并排状态

6. 每小时流量模式（图形）

    - 查询：`rate(sms_c_message_received_count[1h]) * 3600`
    - 时间范围：最近 7 天
    - 显示每日模式

仪表板 4：容量与资源

目的：监控资源使用情况和容量限制。

面板：

1. 队列容量（图形）

    - 当前：`sms_c_queue_size_size`
    - 容量线：基于系统限制的固定值
    - 显示利用率趋势

2. 数据库表增长（图形）

    - 消息：`sms_c_mnesia_table_size_record_count{table="sms_route"}`
    - 翻译：`sms_c_mnesia_table_size_record_count{table="translation_rule"}`
    - 最近 30 天的趋势

3. 消息积压趋势（图形）

    - 查询：`rate(sms_c_delivery_queued_count[5m]) - (rate(sms_c_delivery_succeeded_count[5m]) + rate(sms_c_delivery_failed_count[5m]))`
    - 正值 = 积压增长
    - 负值 = 赶上

4. 峰值流量（状态）

    - 查询：`max_over_time(rate(sms_c_message_received_count[5m])[24h:])`
    - 显示最近 24 小时内的最高 5m 速率
    - 单位：messages/second

5. 容量利用率（仪表）

    - 查询：`(rate(sms_c_message_received_count[5m]) / MAX_CAPACITY) * 100`
    - 将 MAX_CAPACITY 替换为您的系统限制
    - 单位：percent
    - 阈值：
        - 绿色：< 70
        - 黄色：70-85
        - 红色：> 85

仪表板 5: SLA 合规性

目的：跟踪 SLA 指标和合规性。

面板：

1. **SLA** 合规性（仪表）
   - 投递成功：`(rate(sms_c_delivery_succeeded_count[1h]) / rate(sms_c_delivery_queued_count[1h])) * 100`
   - 目标线在 99%
   - 阈值：
     - 红色：< 95
     - 黄色：95-99
     - 绿色：>= 99
2. 在 **SLA** 内投递的消息（状态）
   - 查询：`count(sms_c_delivery_succeeded_duration_bucket{le="5000"}) / count(sms_c_delivery_succeeded_duration_bucket)`
   - 显示在 5 秒内投递的百分比
   - 单位：percent
3. **SLA** 违规（计数器）
   - 超过 5 分钟的消息：`increase(sms_c_queue_oldest_message_age_seconds{} > 300)[24h:]`
   - 应为 0
4. 正常运行时间（状态）
   - 查询：`up{job="sms-c"}`
   - 二进制：1 = 正常，0 = 异常
   - 显示当前状态
5. 每日成功率趋势（图形）
   - 查询：`avg_over_time((rate(sms_c_delivery_succeeded_count[1h]) / rate(sms_c_delivery_queued_count[1h]))[24h:1h])`

- 时间范围：最近 30 天
- SLA 线在 99%

告警规则示例

关键告警

路由失败：

```
alert: RoutingFailuresDetected
expr: increase(sms_c_routing_failed_count[5m]) > 0
for: 2m
labels:
  severity: critical
annotations:
  summary: "{{ $value }} 路由失败在过去 5 分钟内"
  description: "□息无法路由。检查路由配置。"
```

队列积压：

```
alert: MessageQueueBacklog
expr: sms_c_queue_size_pending > 10000
for: 5m
labels:
  severity: critical
annotations:
  summary: "□息队列有 {{ $value }} 条待处理□息"
  description: "队列正在积压。检查投递性能。"
```

队列中的旧消息：

```yaml
alert: OldMessagesInQueue
expr: sms_c_queue_oldest_message_age_seconds > 300
for: 2m
labels:
  severity: critical
annotations:
  summary: "最旧□息已存在 {{ $value }} 秒"
  description: "□息未被投递。检查前端。"
```

所有前端断开连接：

```yaml
alert: NoActiveFrontends
expr: sum(sms_c_frontend_status_count{status="connected"}) == 0
for: 1m
labels:
  severity: critical
annotations:
  summary: "没有前端连接"
  description: "没有可用的投递路径。检查前端连接。"
```

死信队列增长：

```yaml
alert: DeadLetterMessagesIncreasing
expr: rate(sms_c_delivery_dead_letter_count[10m]) > 0
for: 5m
labels:
  severity: critical
annotations:
  summary: "{{ $value }} 条□息移动到死信队列"
  description: "□息变得无法投递。调查失败原因。"
```

警告告警

低投递成功率：

```
alert: LowDeliverySuccessRate
expr: (rate(sms_c_delivery_succeeded_count[10m]) /
rate(sms_c_delivery_queued_count[10m])) < 0.95
for: 10m
labels:
  severity: warning
annotations:
  summary: "投递成功率为 {{ $value | humanizePercentage }}"
  description: "成功率低于 95%。调查投递失败。"
```

高重试率：

```
alert: HighDeliveryRetryRate
expr: histogram_quantile(0.95,
sms_c_delivery_succeeded_attempt_count_bucket) > 2
for: 15m
labels:
  severity: warning
annotations:
  summary: "95th 百分位投递尝试次数: {{ $value }}"
  description: "□息需要多次尝试。检查目标可靠性。"
```

消息处理缓慢：

```
alert: SlowMessageProcessing
expr: histogram_quantile(0.95,
sms_c_message_processing_stop_duration_bucket) > 1000
for: 10m
labels:
  severity: warning
annotations:
  summary: "95th 百分位处理时间: {{ $value }}ms"
  description: "□息处理缓慢。检查系统资源。"
```

ENUM 查找失败：

```
alert: HighEnumFailureRate
expr: rate(sms_c_enum_lookup_stop_duration_count{success="false"}
[10m]) > 0.1
for: 10m
labels:
  severity: warning
annotations:
  summary: "ENUM查找失败率：{{ $value }}"
  description: "DNS查找失败。检查DNS服务器。"
```

低**ENUM**缓存命中率：

```
alert: LowEnumCacheHitRate
expr: rate(sms_c_enum_cache_hit_count[10m]) /
(rate(sms_c_enum_cache_hit_count[10m]) +
rate(sms_c_enum_cache_miss_count[10m])) < 0.70
for: 30m
labels:
  severity: warning
annotations:
  summary: "ENUM缓存命中率：{{ $value | humanizePercentage }}"
  description: "缓存效率低。可能表明唯一号码流量。"
```

计费失败：

```
alert: ChargingFailuresDetected
expr: rate(sms_c_charging_failed_count[10m]) > 0.05
for: 10m
labels:
  severity: warning
annotations:
  summary: "计费失败率：{{ $value }}"
  description: "计费系统错误。检查OCS连接。"
```

附加说明

- 所有持续时间指标在内部使用纳秒精度，但在报告时转换为毫秒

- 计数器指标是累积的，应在 Prometheus 查询中使用 `rate()` 或 `increase()` 函数
- Gauge 指标表示在收集时的瞬时值
- 直方图指标提供百分位数计算（p50, p95, p99），并可用于创建热图
- 所有指标均包括 Prometheus 添加的默认标签（实例、作业等）
- 创建仪表板时，使用适当的时间范围：实时使用 5m，趋势使用 1h，容量规划使用 24h+
- 在 Prometheus 中设置记录规则以提高仪表板性能，适用于频繁使用的复杂查询
- 在 Grafana 中使用变量模板以实现动态仪表板（选择 dest_smsc、source_smsc 等）

SMS-C号码转换指南

← 返回文档索引 | 主README

概述

SMS-C号码转换系统在路由之前提供灵活的基于正则表达式的电话号码转换。转换规则可以规范化号码、添加国际前缀、为特定网关格式化号码，并将多个转换串联在一起。规则存储在Mnesia中以确保持久性，并可以在运行时修改而不会中断服务。

主要特点

- □于前缀的匹配：在应用转换之前通过前缀匹配号码
- □于正则表达式的转换：强大的模式匹配和替换，带有捕获组
- 源**SMSC**过滤：根据消息来源应用不同的转换
- □于优先级的评估：通过可配置的优先级（1-255）控制规则顺序
- 规则链式处理：通过循环预防继续处理多个规则
- 独立的呼叫/被叫转换：对发起和目的号码进行独立转换
- 配置文件加载：在首次启动时从`runtime.exs`加载初始规则
- 运行时配置：在不重启的情况下添加、修改或禁用规则
- **Web UI**：���整的CRUD接口用于规则管理
- 模拟工具：逐步评估测试转换逻辑
- 备份/恢复：导出和导入转换配置
- 预路由集成：在路由之前应用转换以确保一致的号码格式

架构

数据模型

每个转换规则包含以下字段：

字段	类型	描述	必需
`rule_id`	整数	自动递增的唯一标识符	是（自动）
`calling_prefix`	字符串/nil	呼叫号码的前缀匹配（nil = 通配符）	否
`called_prefix`	字符串/nil	被叫号码的前缀匹配（nil = 通配符）	否
`source_smsc`	字符串/nil	源SMSC名称（nil = 通配符）	否
`calling_match`	字符串/nil	匹配呼叫号码的正则表达式模式	否
`calling_replace`	字符串/nil	呼叫号码的替换模式	否
`called_match`	字符串/nil	匹配被叫号码的正则表达式模式	否
`called_replace`	字符串/nil	被叫号码的替换模式	否
`priority`	整数	规则优先级（1-255，数字越小优先级越高）	是
`description`	字符串	人类可读的描述	否
`enabled`	布尔值	启用/禁用规则	是
`continue`	布尔值	匹配后继续评估规则（默认：false）	否

注意：规则按优���级顺序评估（数字越小越先）。只有启用的规则才会被评估。

转换算法

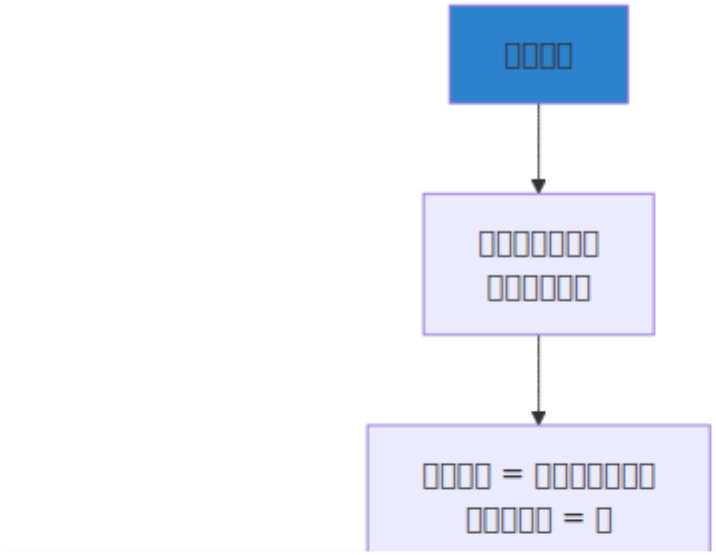
在转换号码时，系统：

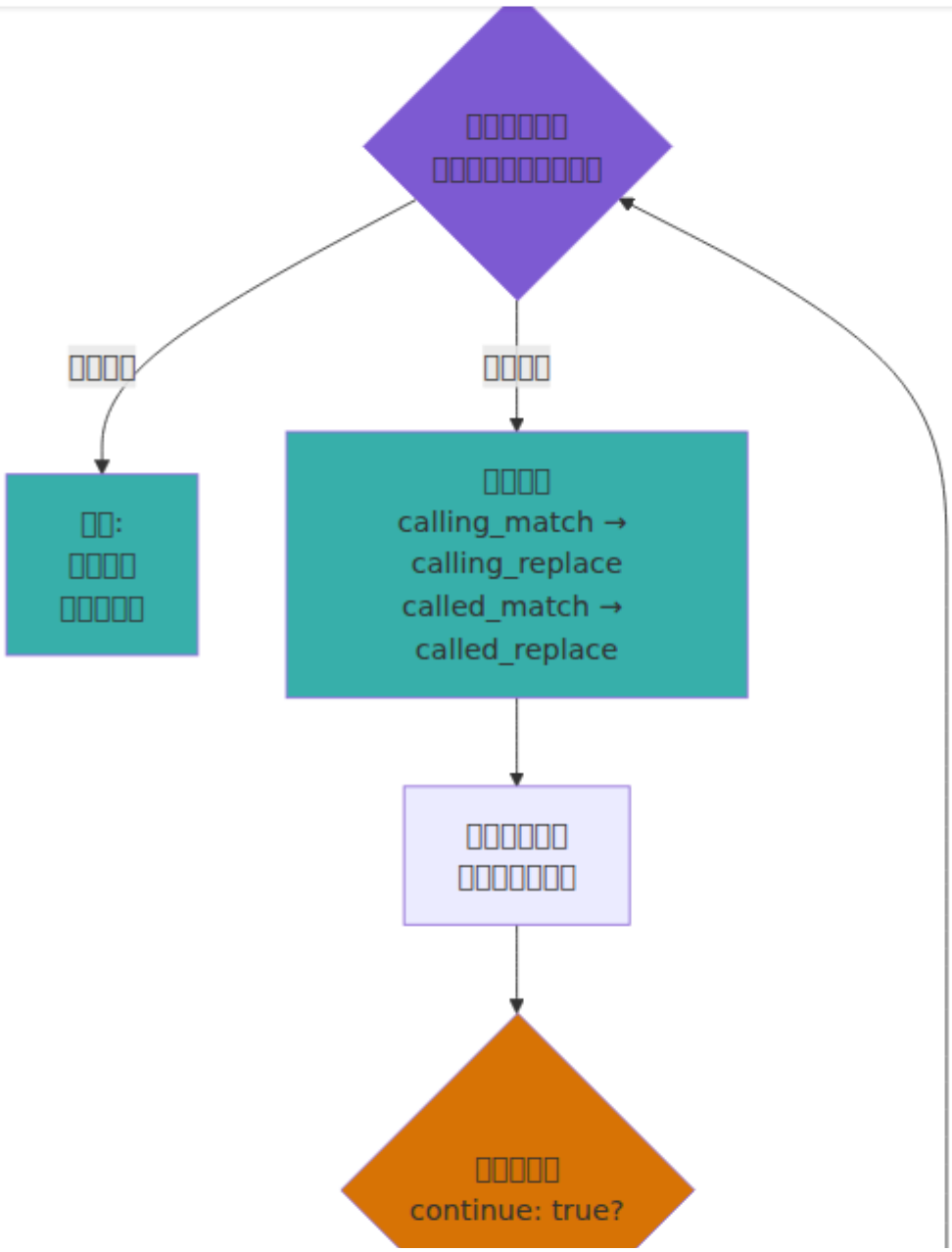
1. 检索启用的规则，按优先级排序（最低优先级优先）
2. 依次评估规则与消息参数：
    - 匹配`calling_prefix`（如果指定）
    - 匹配`called_prefix`（如果指定）
    - 匹配`source_smsc`（如果指定）

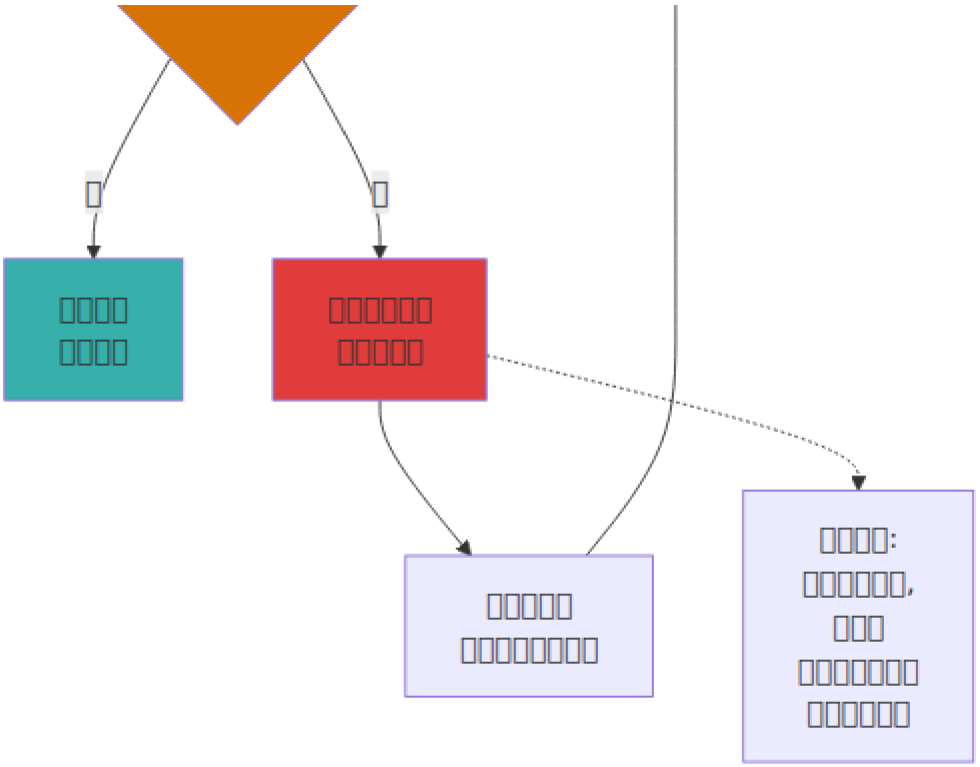
3. 应用第一个匹配的规则：
    - 使用`calling_match`和`calling_replace`转换呼叫号码
    - 使用`called_match`和`called_replace`转换被叫号码
4. 检查继续标志：
    - 如果`continue: false` → 停止处理，返回结果
    - 如果`continue: true` → 从可用规则中移除匹配的规则，使用转换后的号码继续进行步骤2
5. 返回最终号码和所有应用规则的列表

带循环预防的规则链式处理

`continue`标志启用强大的规则链式处理，同时防止无限循环：


□□□□
□□□□□□□□
□□□□□□□
□□□□ = □□□□□□□□
□□□□□ = □


□□□□□□
□□□□□□□□□□□□
□□□□
□□:
□□□□
□□□□□
□□□□
calling_match →
calling_replace
called_match →
called_replace
□□□□□□
□□□□□□□□
□□□□□□
continue: true?



通配符

- `nil`或空值作为通配符，匹配任何值
- 没有匹配条件的规则是一个捕获所有规则
- 没有转换模式（nil匹配/替换）的规则将号码原样传递

示例：规则链式处理场景

Parse error on line 20: ...] style R1 fill:#38B2AC style R ----------^
Expecting 'SOLID_OPEN_ARROW', 'DOTTED_OPEN_ARROW', 'SOLID_ARROW', 'BIDIRECTIONAL_SOLID_ARROW', 'DOTTED_ARROW', 'BIDIRECTIONAL_DOTTED_ARROW', 'SOLID_CROSS', 'DOTTED_CROSS', 'SOLID_POINT', 'DOTTED_POINT', got 'TXT'

□试

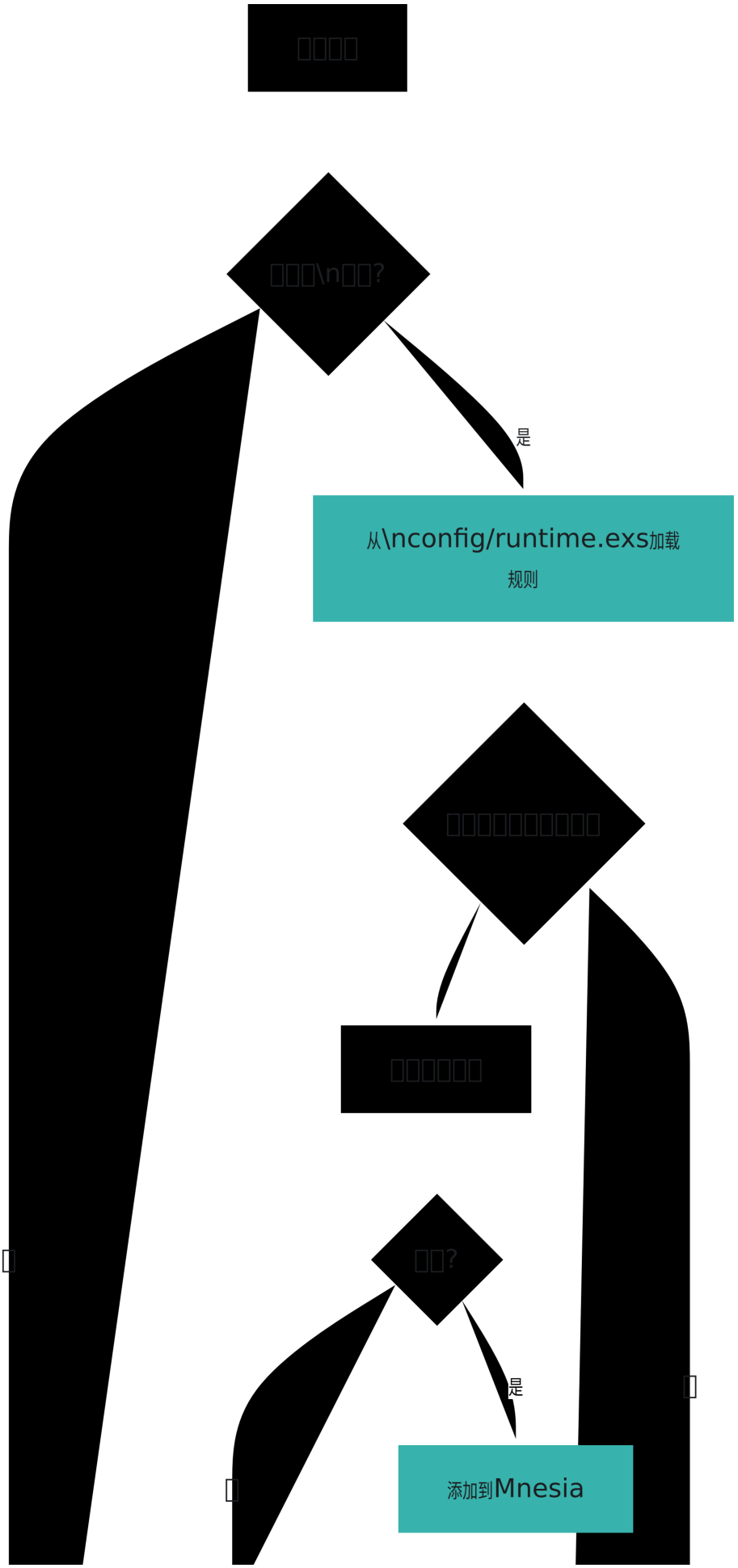
配置

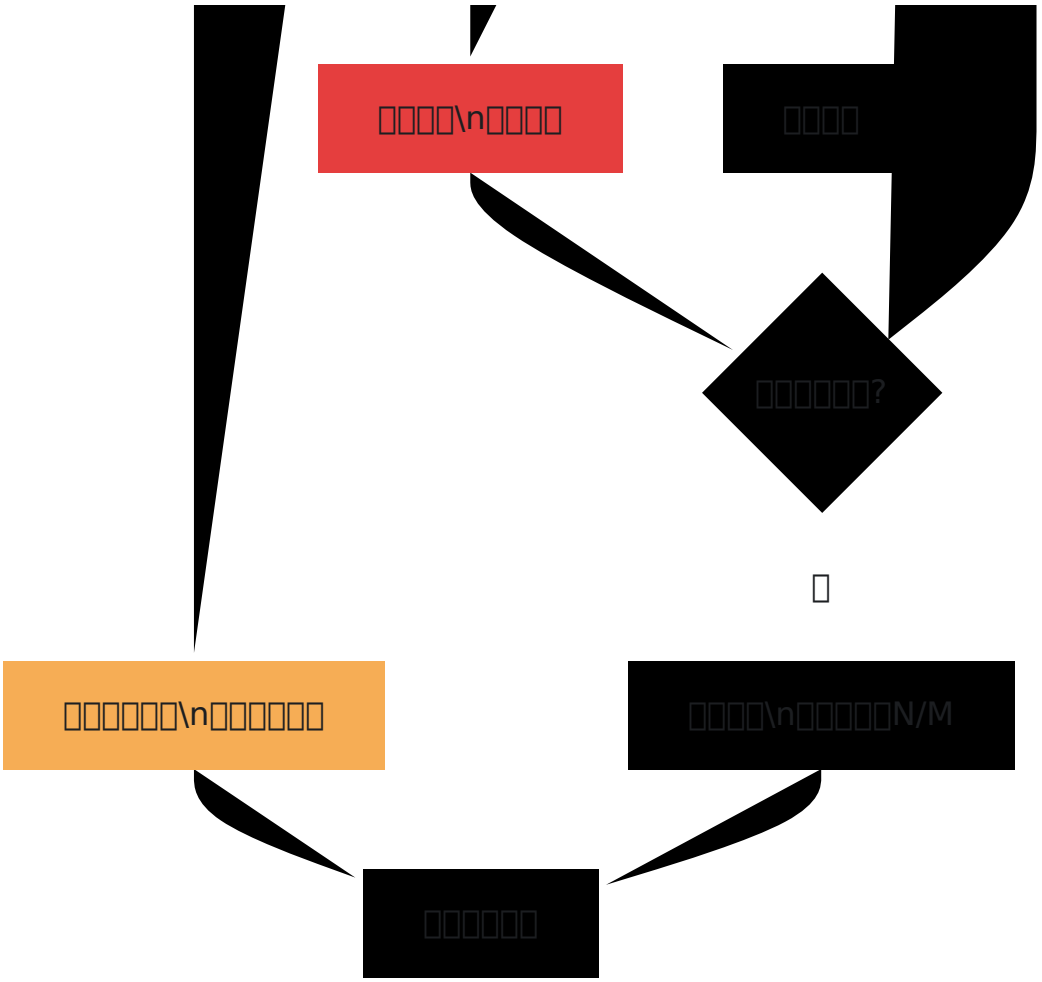
从配置文件加载规则

转换规则可以在`config/runtime.exs`中定义，并将在首次启动时自动加载。

重要：仅在转换表为空时加载配置中的规则（首次启动）。这保留了在运行时通过Web UI添加的规则，并防止在重启时出现重复。

配置加载流程


是
从\nconfig/runtime.exs加载
规则
是
添加到Mnesia


□□□□\n□□□□
□□□□
□□□□□□□?
□
□□□□□□\n□□□□□□
□□□□\n□□□□□N/M
□□□□□□

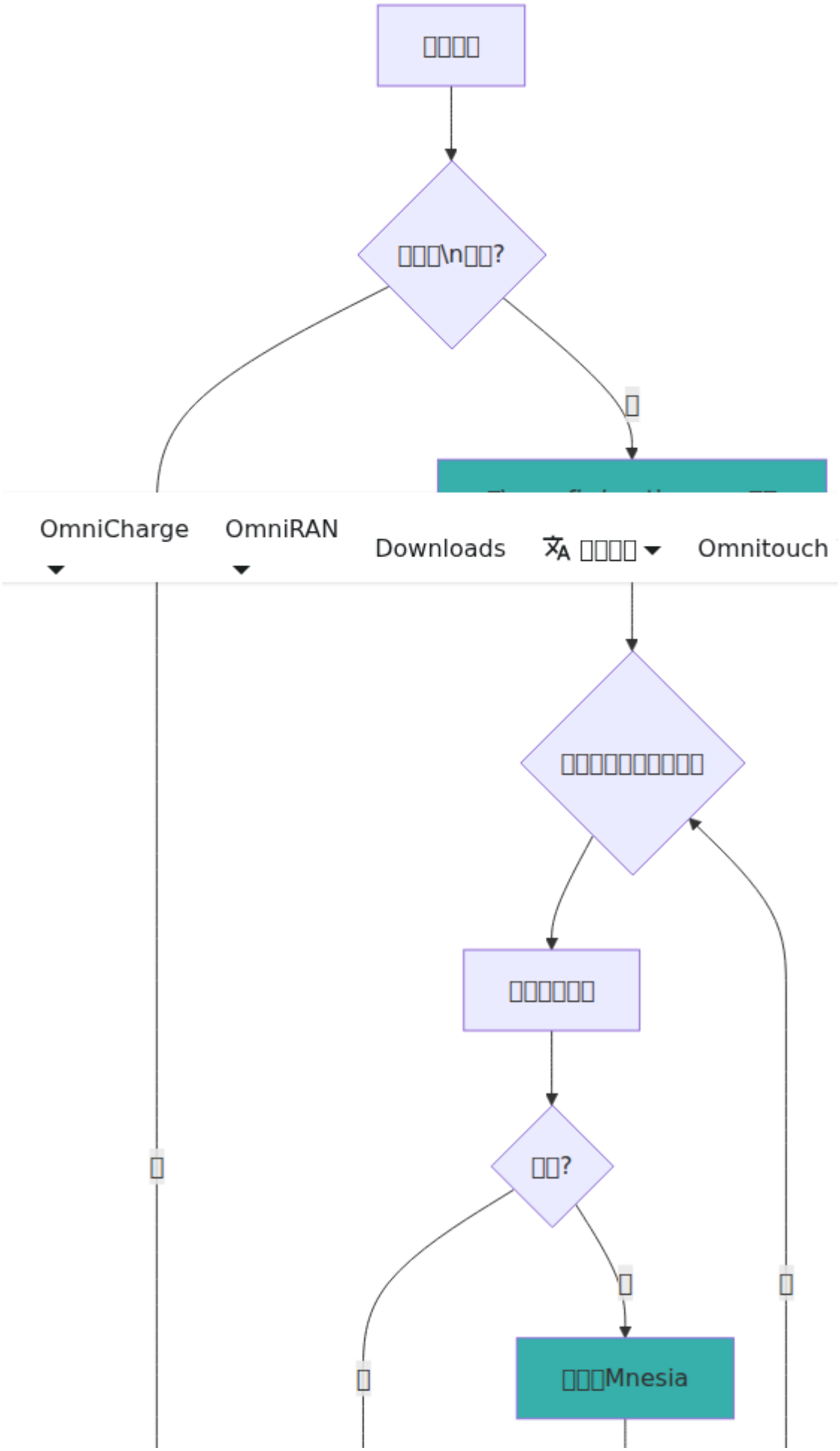
□□□□

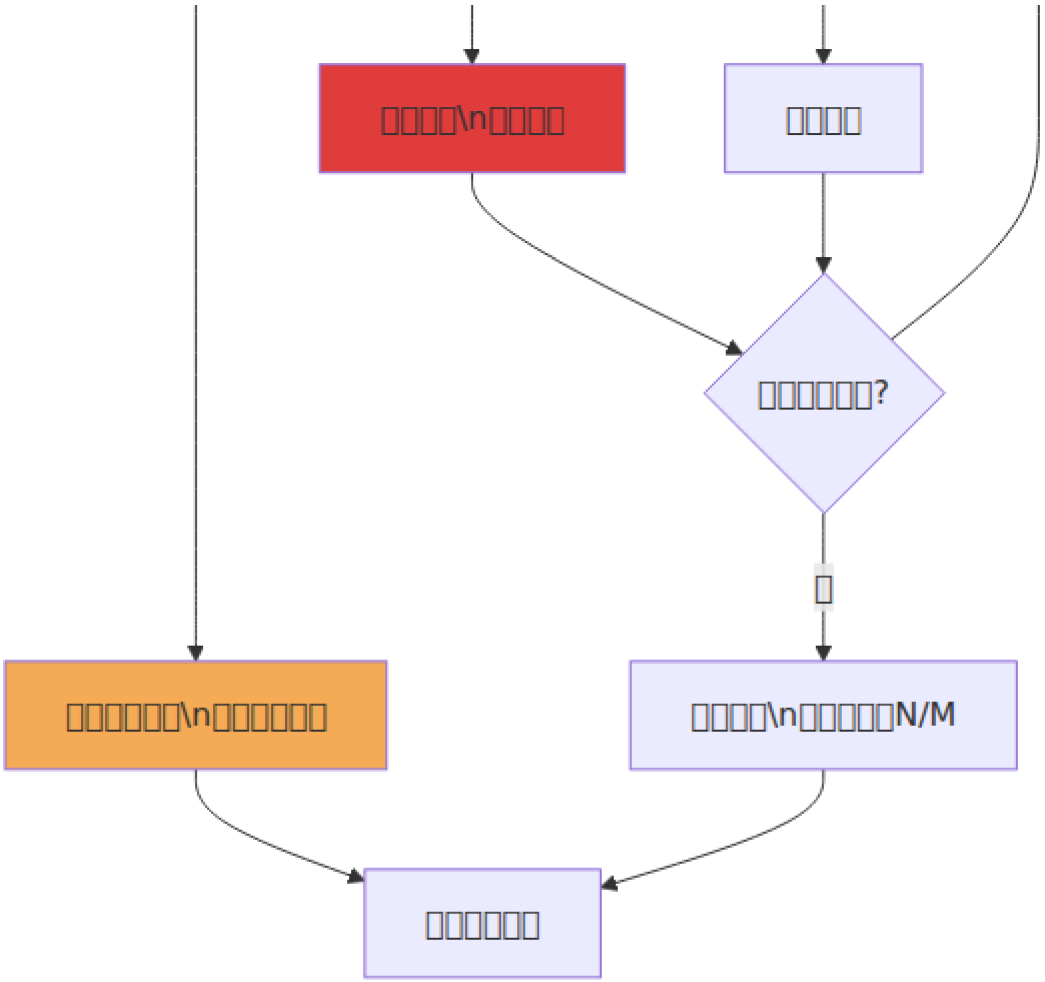
```
# config/runtime.exs
config :sms_c, :translation_rules, [
  # 为10位美国号码添加+1
  %{
    calling_prefix: nil,
    called_prefix: nil,
    source_smsc: "us_domestic_smsc",
    calling_match: "^(\d{10})$",
    calling_replace: "+1\1",
    called_match: "^(\d{10})$",
    called_replace: "+1\1",
    priority: 10,
    description: "为来自国内SMSC的10位美国号码添加+1",
    enabled: true,
    continue: false
  },

  # 去掉国际格式的前导零
  %{
    calling_prefix: "00",
    called_prefix: nil,
    source_smsc: nil,
    calling_match: "^00(.+)$",
    calling_replace: "+\1",
    called_match: nil,
    called_replace: nil,
    priority: 5,
    description: "将00国际前缀转换为+",
    enabled: true,
    continue: true  # 继续应用更多格式
  },

  # 为特定网关格式化英国号码
  %{
    calling_prefix: "+44",
    called_prefix: "+44",
    source_smsc: nil,
    calling_match: "^\+44(.*)$",
    calling_replace: "0044\1",
    called_match: "^\+44(.*)$",
    called_replace: "0044\1",
    priority: 20,
    description: "为传统网关格式化英国号码",
```

```
    enabled: true,
    continue: false
  }
]
```

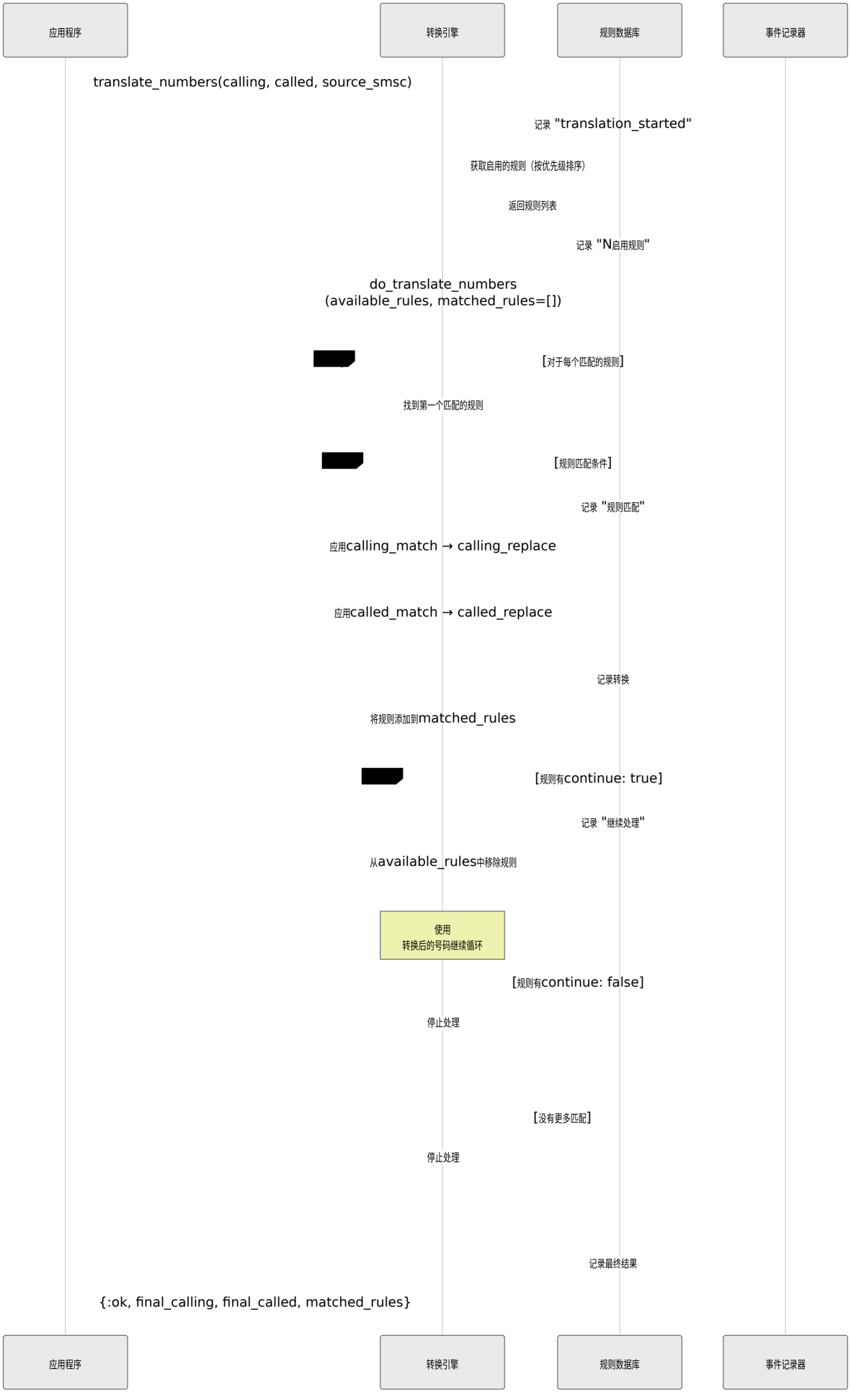
□□

□□□□□


□□□□
□□□\n□□?
□
OmniCharge
OmniRAN
Downloads
□□□□
Omnitouch
□□□□□□□□□□□□
□□□□□□□
□□?
□
□
□
□
□
□□□Mnesia



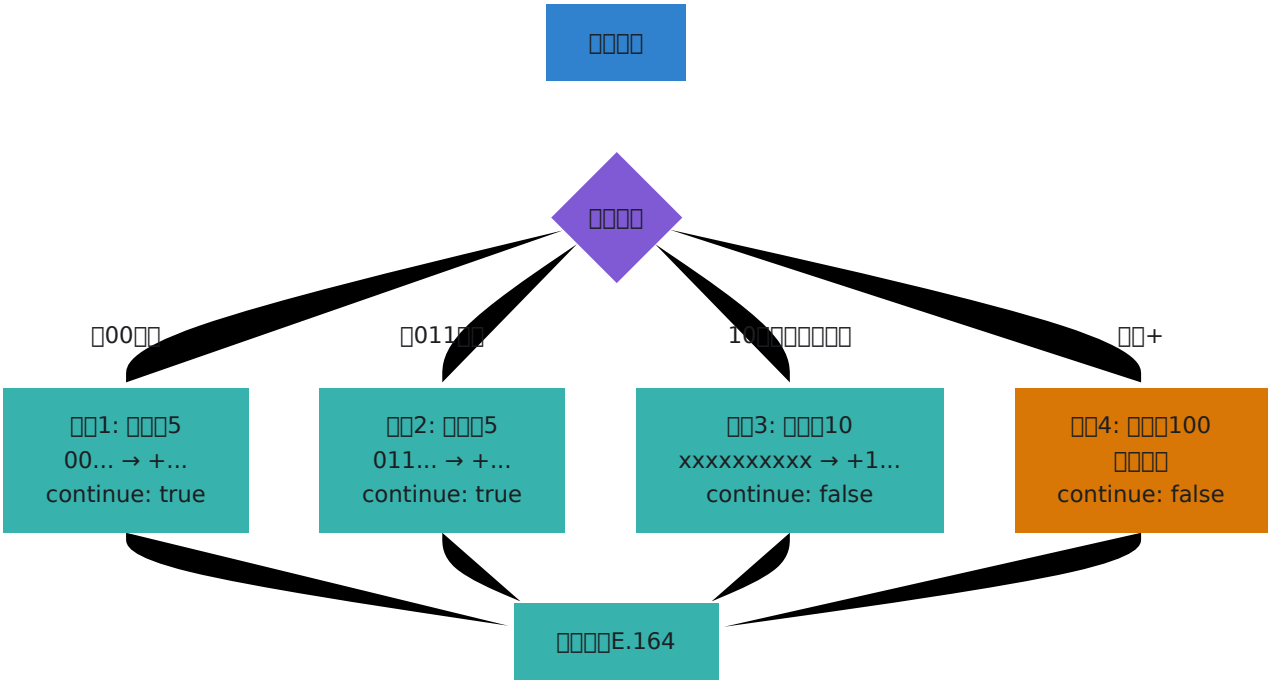
消息转换流程


应用程序
转换引擎
规则数据库
事件记录器
translate_numbers(calling, called, source_smsc)
记录 "translation_started"
获取启用的规则（按优先级排序）
返回规则列表
记录 "N启用规则"
do_translate_numbers
(available_rules, matched_rules=[])
[对于每个匹配的规则]
找到第一个匹配的规则
[规则匹配条件]
记录 "规则匹配"
应用calling_match → calling_replace
应用called_match → called_replace
记录转换
将规则添加到matched_rules
[规则有continue: true]
记录 "继续处理"
从available_rules中移除规则
使用
转换后的号码继续循环
[规则有continue: false]
停止处理
[没有更多匹配]
停止处理
记录最终结果
{:ok, final_calling, final_called, matched_rules}
应用程序
转换引擎
规则数据库
事件记录器

□□□□

□□□□□□□□

□□□□□□□□□□□□□E.164□



□□□□□□□□

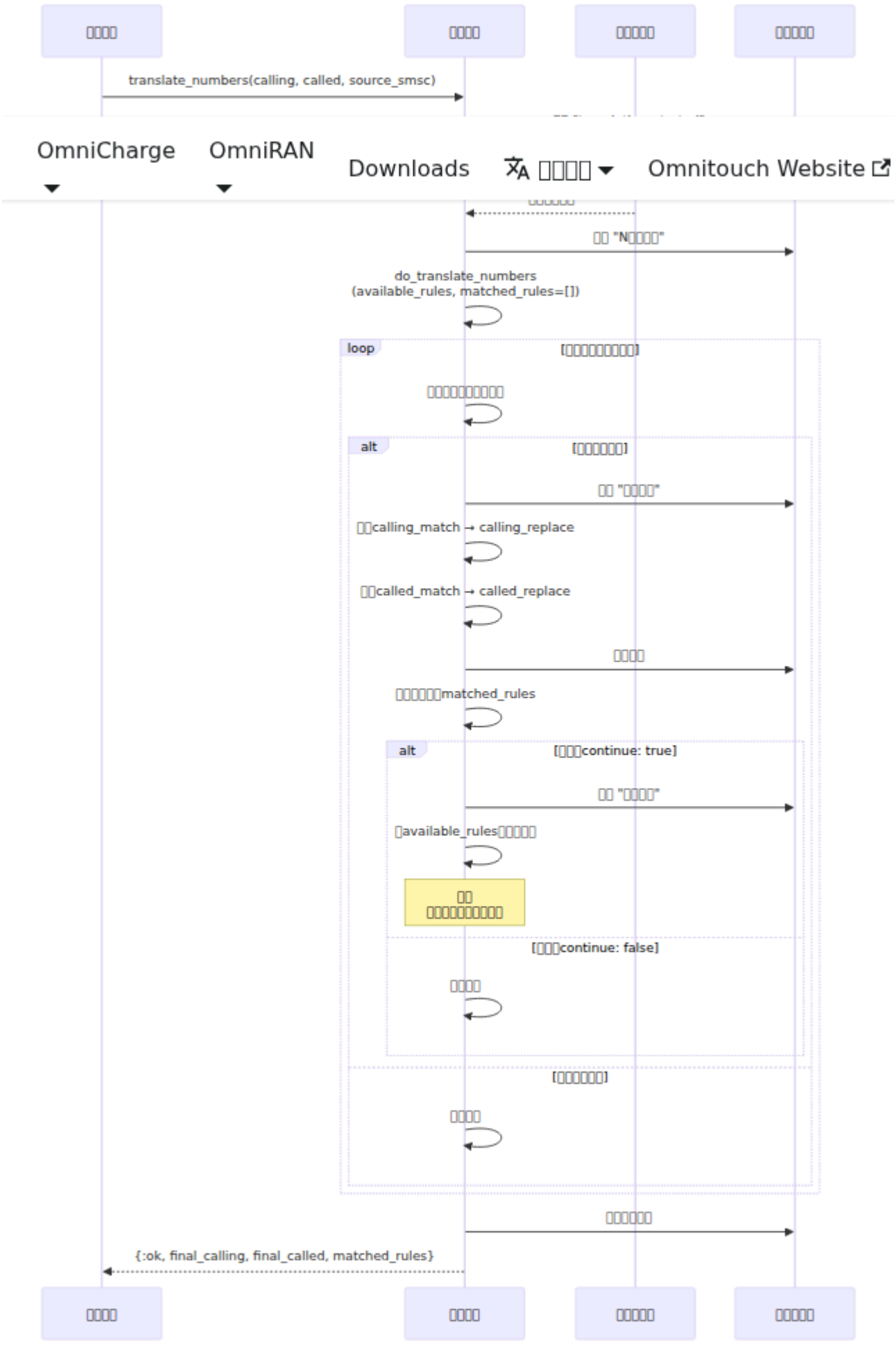
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Parse error on line 2: ...chart TD I[□□: "5551234567"] --> S1[----------
^ Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND',
'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND',
'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'STR'

□□

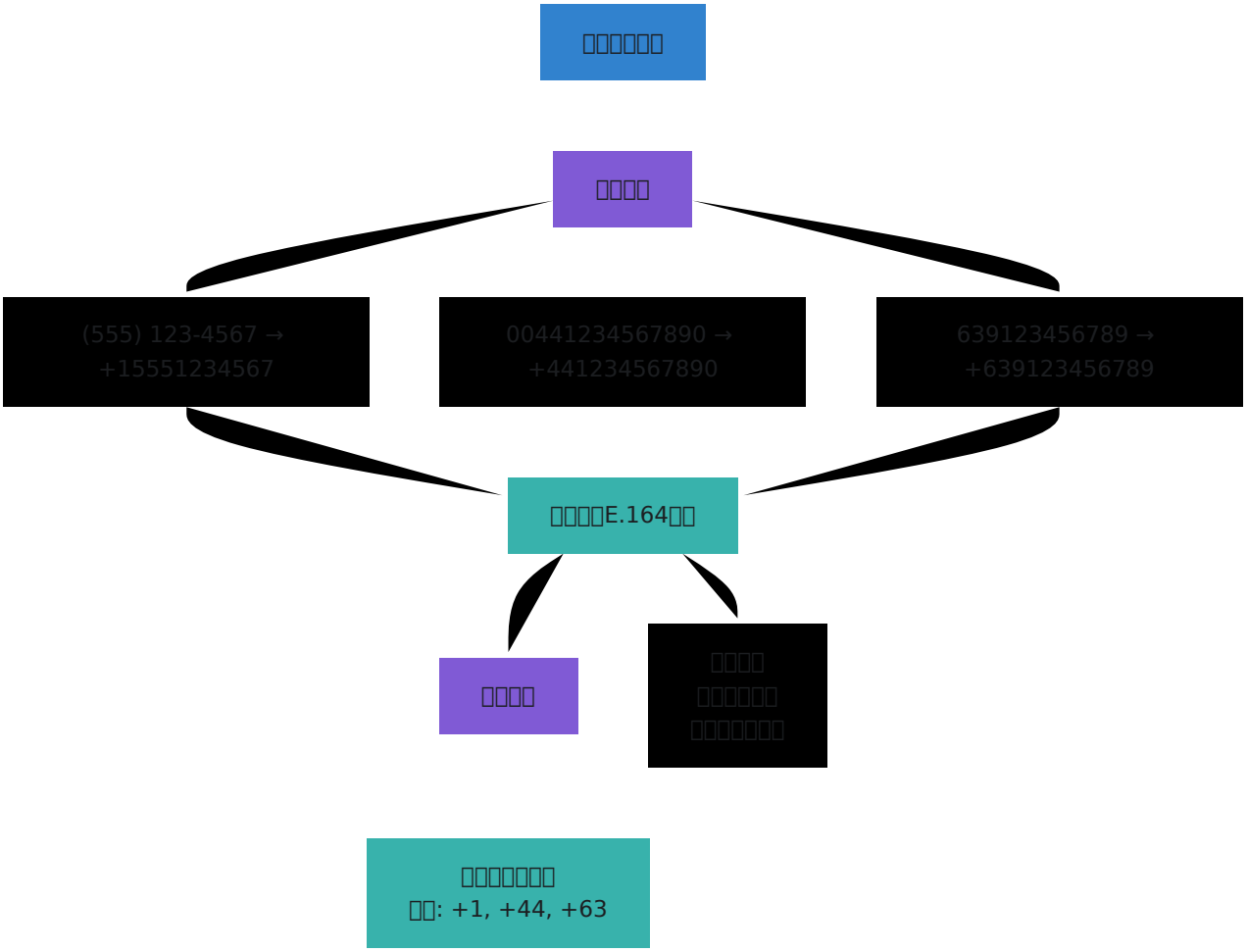
SMSC□□□□

□□□□□□□□□□□□□□□□


translate_numbers(calling, called, source_smsc)
OmniCharge
OmniRAN
Downloads
Omnitouch Website
do_translate_numbers
(available_rules, matched_rules=[])
loop
alt
alt
{:ok, final_calling, final_called, matched_rules}

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□



□□□□□□□□□

□□□□□□□□□□□□□□□□□

Parse error on line 18: ... style Input fill:#3182CE style R ----------^ Expecting 'SOLID_OPEN_ARROW', 'DOTTED_OPEN_ARROW', 'SOLID_ARROW', 'BIDIRECTIONAL_SOLID_ARROW', 'DOTTED_ARROW', 'BIDIRECTIONAL_DOTTED_ARROW', 'SOLID_CROSS', 'DOTTED_CROSS', 'SOLID_POINT', 'DOTTED_POINT', got 'TXT'

□□

Web界面

转换规则管理UI

通过导航菜单访问规则管理界面 `/number_translation`:

功能:

- 在可排序的表格中查看所有规则，按优先级排序
- 通过表单验证添加新规则
- 编辑现有规则
- 启用/��用规则而不删除
- 确认后删除规则
- 对于`continue: true`的规则提供可视指示
- 以JSON格式导入/导出规则

添加规则:

1. 填写匹配条件（可选）:
   - 呼叫前缀（例如，“+1”，“44”）
   - 被叫前缀（例如，“+639”，“1555”）
   - 源SMSC（留空表示任何）
2. 定义转换（可选）:
   - 呼叫号码正则匹配和替换
   - 被叫号码正则匹配和替换
3. 设置优先级（1-255，数字越小优先级越高）
4. 设置状态:
   - 启用：规则处于活动状态
   - 继续处理：在□规则匹配后继续评估更多规则
5. 添加描述
6. 点击“添加规则”或“更新规则”

继续处理切换:

- 停止（默认）：在□规则匹配后停止处理
- 继续：应用□规则并继续评估剩余规则

- 启用继续的规则在表格中显示蓝色“↓ 继续”徽章

编辑规则：

1. 点击规则旁边的“编辑”
2. 根据需要修改字段
3. 点击“更新规则”

规则表指示：

- 启用/禁用徽章显示规则状态
- ↓ 继续徽章显示将继续处理的规则
- 优先级徽章显示评估顺序
- 正则模式以等宽字体显示以提高清晰度

转换模拟器

通过导航菜单访问模拟器 `/translation_simulator`：

功能：

- 使用实际号码测试转换逻辑
- 逐步转换显示应用的每个规则
- 查看每个转换的前后值
- 查看匹配的规则及其原因
- 加载示例场景以快速测试
- 查看测试历史（最近10次测试）

使用模拟器：

1. 输入测试参数：
    - 呼叫号码（来自）
    - 被叫号码（到）
    - 源SMSC（可选）
2. 点击“测试转换”
3. 查看全面的结果：
    - 转换结果：所有转换后的最终号码

    - 应用的规则：所有匹配的规则的计数和列表
    - 逐步转换：每个规则的详细视图：
        - 步骤编号和规则信息
        - 规则描述
        - 呼叫和被叫号码的前后值
        - 对于继续处理的规则的“↓ 继续”指示
        - 以绿色突出显示的转换
        - 标记为“直接通过”的未更改值

4. 使用示例按钮加载预配置的示例
5. 查看测试历史以比较不同场景

示例输出：

转换结果

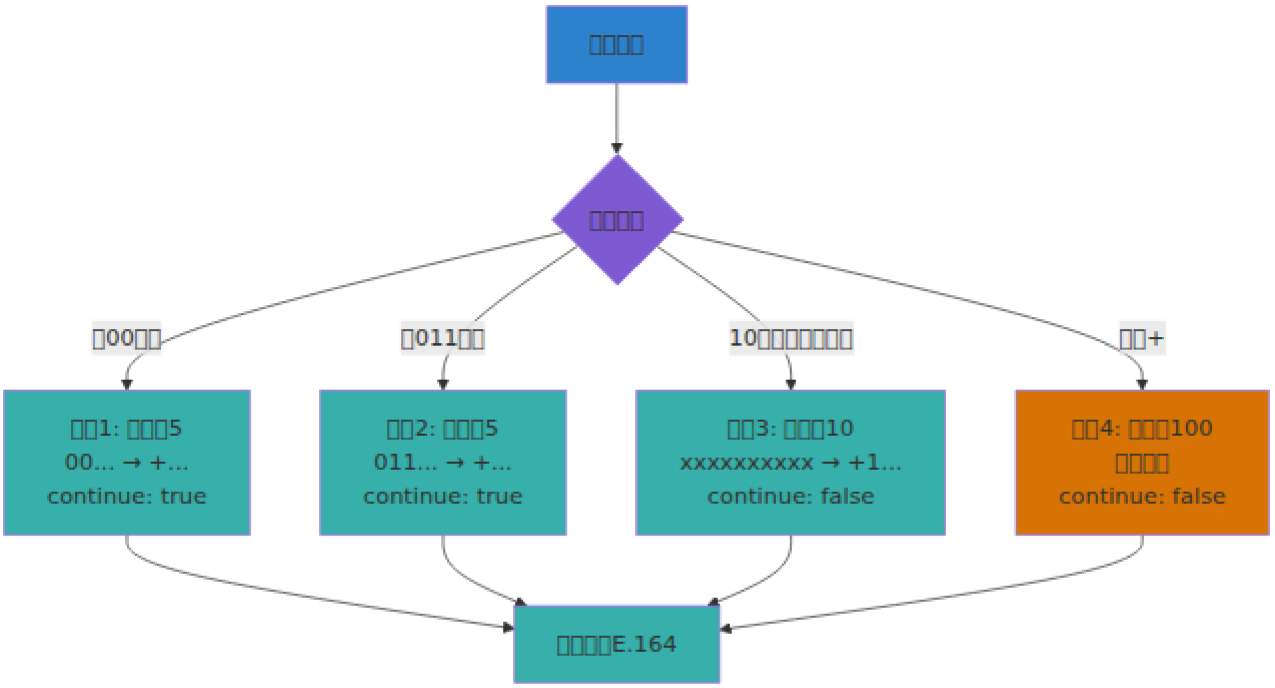
```
═══════════════════════════════════════════════════════

呼叫号码:  5551234567 → +1-555-123-4567
被叫号码:   9078720155 → +1-907-872-0155
✓ 由3条规则翻译
```

逐步转换

```
═══════════════════════════════════════════════════════

┌─ 步骤1 ──────────────────────────────────────────┐
│ 规则 #1 (优先级10)            ↓ 继续 │
│ 为10位数字添加国家代码                  │
│                                                  │
│ 被叫: 9078720155 → +19078720155        │
└──────────────────────────────────────────────────┘

┌─ 步骤2 ──────────────────────────────────────────┐
│ 规则 #2 (优先级20)            ↓ 继续 │
│ 使用破折号格式化区号                    │
│                                                  │
│ 被叫: +19078720155 → +1-907-8720155    │
└──────────────────────────────────────────────────┘

┌─ 步骤3 ──────────────────────────────────────────┐
│ 规则 #3 (优先级30)                      │
│ 网关的最终格式化                        │
│                                                  │
│ 被叫: +1-907-8720155 → +1-907-872-0155│
└──────────────────────────────────────────────────┘
```

API参考

核心操作概述



转换参数

translate_numbers接受以下参数:

- `calling_number` (可选): 发��电话号码
- `called_number` (可选): 目的电话号码
- `source_smsc` (可选): 源SMSC标识符
- `message_id` (可选): 用于事件记录

返回:

- `{:ok, translated_calling, translated_called, [rules_applied]}` - 始终成功
- 如果没有规则匹配，则返回原始号码
- 返回所有应用的规则列表（按顺序）

```
# 示例用法
{:ok, new_calling, new_called, rules} =
  NumberTranslation.translate_numbers(
    calling_number: "5551234567",
    called_number: "9078720155",
    source_smsc: "domestic_gateway",
    message_id: "msg_123"
  )

# 检查是否发生了任何转换
if rules != [] do
  Logger.info("应用了 #{length(rules)} 条转换规则")
  Enum.each(rules, fn rule ->
    Logger.info("  - 规则 ##{rule.rule_id}: #{rule.description}")
  end)
end
```

规则管理操作

```
# 添加新规则
{:ok, rule} = NumberTranslation.add_rule(%{
  calling_prefix: nil,
  called_prefix: nil,
  source_smsc: "gateway1",
  calling_match: "^(\d{10})$",
  calling_replace: "+1\1",
  called_match: "^(\d{10})$",
  called_replace: "+1\1",
  priority: 10,
  description: "为10位数字添加+1",
  enabled: true,
  continue: false
})

# 更新规则
{:ok, updated_rule} = NumberTranslation.update_rule(rule_id, %{
  enabled: false,
  description: "用于测试而禁用"
})

# 删除规则
:ok = NumberTranslation.delete_rule(rule_id)

# 获取特定规则
rule = NumberTranslation.get_rule(rule_id)

# 列出所有规则
all_rules = NumberTranslation.list_rules()

# 仅列出启用的规则（按优先级排序）
enabled_rules = NumberTranslation.list_enabled_rules()
```

导入/导出操作

```
# 导出所有规则
backup = NumberTranslation.export_rules()
# 返回: %{
#   version: "1.0",
#   exported_at: ~U[2024-01-15 10:30:00Z],
#   count: 5,
#   rules: [...]
# }

# 保存为JSON文件
json = Jason.encode!(backup, pretty: true)
File.write!("translation_rules_backup.json", json)

# 导入规则（与现有规则合并）
{:ok, %{imported: 3, failed: 0}} =
  NumberTranslation.import_rules(backup, mode: :merge)

# 导入规则（替换所有现有规则）
{:ok, %{imported: 5, failed: 0}} =
  NumberTranslation.import_rules(backup, mode: :replace)
```

最佳实践

规则设计

1. 保持优先级有序:
    - **1-10**: 关键规范化规则（添加国家代码，修复格式）
    - **11-50**: 特定网关格式化
    - **51-100**: 可选转换
    - **101+**: 捕获所有或调试规则
2. 战略性使用**continue**:
    - 对于准备进一步处理的规范化规则启用`continue: true`
    - 对于最终格式化规则禁用`continue: false`

    - 避免长链（最多3-4条规则）以保持性能

3. 记录规则：

    - 始终添加清晰的描述
    - 在描述中包含示例（例如，“5551234567 → +15551234567”）
    - 记录目的和预期输入/输出

4. 测试正则模式：

    - 在部署之前使用模拟器测试模式
    - 使用捕获组（\1，\2）进行灵活的转换
    - 转义特殊正则字符（点、括号等）

性能

1. 最小化规则数量：

    - 尽可能合并相似规则
    - 使用前缀匹配以减少正则评估
    - 删除或禁用未使用的规则

2. 优化正则模式：

    - 首先使用前缀匹配（比正则更快）
    - 保持正则模式简单
    - 避免重回溯的模式

3. 限制规则链式处理：

    - 长链（5条以上规则）可能影响性能
    - 考虑将多个步骤合并为一条规则
    - 使用Telemetry指标监控转换延迟

操作

1. 在部署之前测试：

    - 使用模拟器进行真实示例测试

    - 测试边缘情况（空号码，特殊字符）
    - 验证continue标志行为

2. 定期备份：

    - 在进行重大更改之前导出规则
    - 对导出进行版本控制
    - 在非生产环境中测试导入

3. 监控转换：

    - 启用message_id记录以进行调试
    - 检查事件日志以了解转换决策
    - 监控应用的规则

4. 逐步推出：

    - 首先添加新规则时禁用
    - 使用模拟器测试
    - 启用并监控
    - 根据需要进行调整

正则提示

1. 常见模式：

    - 10位美国号码：`^(\d{10})$`
    - 国际格式：`^\+(\d+)$`
    - 去掉前导零：`^0+(.+)$`
    - 添加破折号：`^(\d{3})(\d{3})(\d{4})$` → `\1-\2-\3`

2. 捕获组：

    - 使用括号进行捕获：`^(\d{3})(\d{7})$`
    - 在替换中引用：`+1\1\2`
    - 多个捕获：`^\+(\d{1,3})(\d+)$` → `00\1\2`

3. 转义特殊字符：

  - 字面点：`\`
  - 字面加号：`\+`
  - 字面括号：`\(`或`\)`

故障排除

规则未匹配

症状：预期规则未匹配，号码未转换

可能原因：

- 前缀不匹配（检查是否精确匹配前缀）
- 源SMSC不匹配
- 正则模式未匹配输入格式
- 规则已禁用
- 优先级更高的规则优先匹配（带有continue: false）

解决方案：

1. 使用模拟器查看评估的规则
2. 检查规则状态（启用/禁用）
3. 验证前缀匹配（区分大小写）
4. 单独测试正则模式
5. 检查优先级顺序

错误的转换应用

症状：号码已转换但结果不正确

可能原因：

- 正则模式匹配但替换模式错误
- 多条规则以意外顺序应用
- 捕获组引用不正确（\1, \2等）

解决方案：

1. 使用模拟器查看逐步转换
2. 验证正则模式捕获正确的组
3. 检查替换模式语法
4. 在在线正则测试器中测试正则
5. 审查规则优先级和continue标志

无限循环/性能下降

症状：转换耗时很长或似乎挂起

注意：由于循环预防，这不应该发生，但如果发生：

可能原因：

- 循环预防逻辑中的错误
- 极其复杂的正则评估
- 非常长的规则链

解决方案：

1. 检查应用程序日志以查找错误
2. 审查具有continue: true的规则
3. 简化正则模式
4. 减少链式规则的数量
5. 如果循环预防失败，请报告错误

意外的规则链式处理

症状：应用的规则比预期的多

可能原因：

- 规则在不应该的情况下具有continue: true
- 优先级排序允许多个匹配
- 转换后的号码匹配额外规则

□□□□□

1. □□□□□□□□□□□□□□□□
2. □□□□□□□□continue□□
3. □□□□□□□□□□□□
4. □□□□□□□□□continue: false

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

□□□□□

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

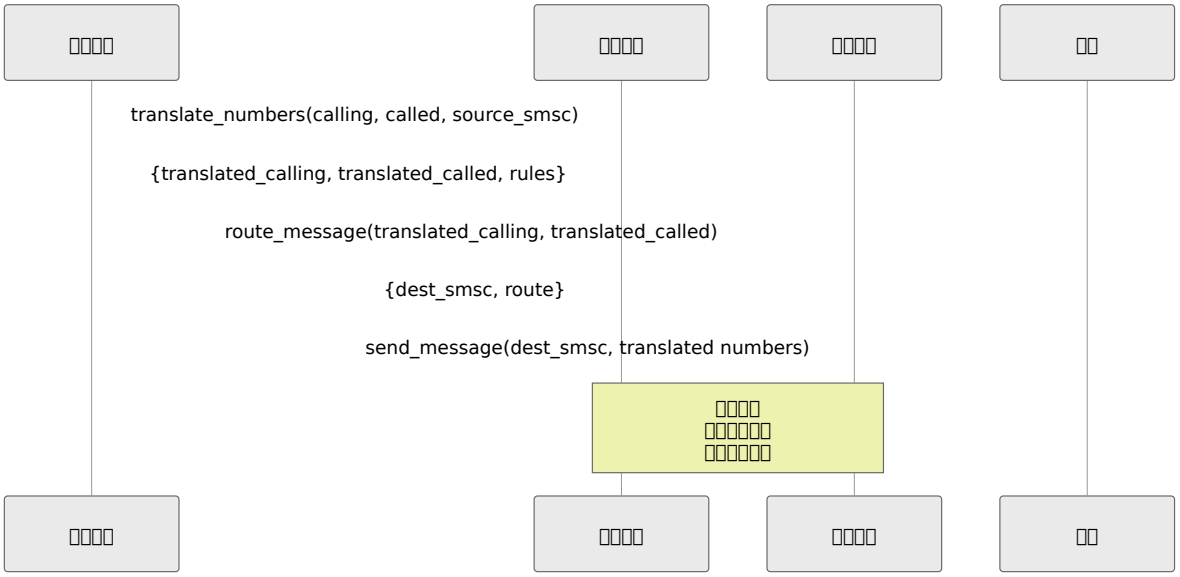
□□□□□

1. □□□□□□□□□□□□□□□□□□□□□□□□□□□
2. □□□□□□□□□
3. □□□□□□□□□□□□□□□□
4. □□□□□□□□□□□□□translate_numbers

□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□



□□□□

□□EventLogger□□□□□□□□

- `translation_started`: □□□□
- `translation_candidates`: □□□□□□□
- `translation_matched`: □□□□□□□□
- `translation_calling`: □□□□□□□
- `translation_called`: □□□□□□□
- `translation_continue`: □□□□□continue=true□□□□□
- `translation_none`: □□□□□□

□□□`message_id`□□□`translate_numbers/1`□□□□□

Telemetry□□

□□Telemetry□□□□□□□

```
:telemetry.attach(
  "number-translation-handler",
  [:sms_c, :number_translation, :translate, :stop],
  fn _event_name, measurements, metadata, _config ->
    # measurements: %{duration: 微秒}
    # metadata: %{rules_applied: 计数, ...}
  end,
  nil
)
```

需要监控的关键指标：

- 转换持续时间（p50，p95，p99）
- 每条消息应用的规则
- 匹配的规则与未匹配的规则
- continue标志的使用情况

集群

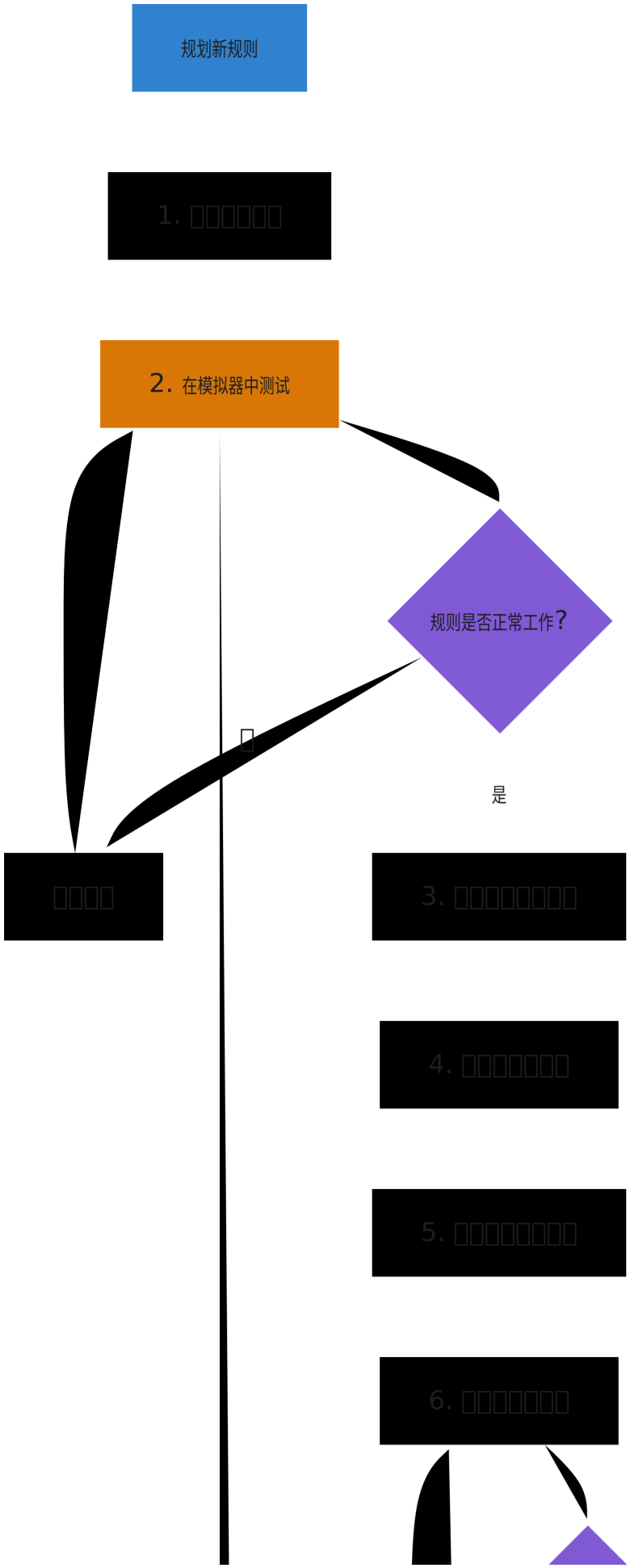
Mnesia表自动分布在集群节点之间。转换规则为高可用性而复制。

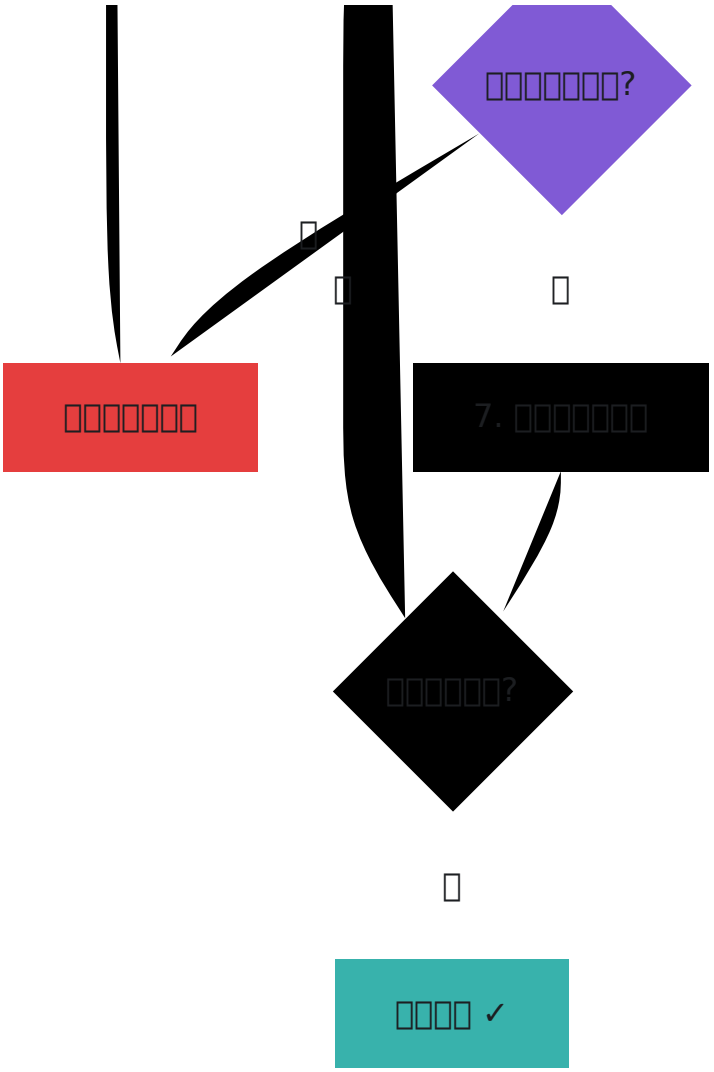
Parse error on line 25: ... style New fill:#3182CE style P ----------^ Expecting 'SOLID_OPEN_ARROW', 'DOTTED_OPEN_ARROW', 'SOLID_ARROW', 'BIDIRECTIONAL_SOLID_ARROW', 'DOTTED_ARROW', 'BIDIRECTIONAL_DOTTED_ARROW', 'SOLID_CROSS', 'DOTTED_CROSS', 'SOLID_POINT', 'DOTTED_POINT', got 'TXT'

□试

迁移策略

在部署新转换规则时：


规划新规则
1. □□□□□□□
2. 在模拟器中测试
规则是否正常工作?
□
是
□□□□
3. □□□□□□□□□
4. □□□□□□□□
5. □□□□□□□□□
6. □□□□□□□□



□□

□□1□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□E.164 (+1XXXXXXXXXX)

```
# 规则1: 10位数字（最高优先级）
%{
  calling_match: "^(\d{10})$",
  calling_replace: "+1\1",
  called_match: "^(\d{10})$",
  called_replace: "+1\1",
  priority: 5,
  description: "为裸10位数字添加+1",
  enabled: true,
  continue: false
}

# 规则2: 1 + 10位数字（中等优先级）
%{
  calling_match: "^1(\d{10})$",
  calling_replace: "+1\1",
  called_match: "^1(\d{10})$",
  called_replace: "+1\1",
  priority: 10,
  description: "将1XXXXXXXXXX转换为+1XXXXXXXXXX",
  enabled: true,
  continue: false
}

# 测试用例:
# "5551234567" → "+15551234567"（规则1）
# "15551234567" → "+15551234567"（规则2）
# "+15551234567" → "+15551234567"（未匹配，直接通过）
```

示例2：国际前缀转换与链式处理

需求：将00前缀转换为+，然后格式化为网关

```
# □□1□□00□□□+□□□□□□□□□□
%{
  calling_match: "^00(.+)$",
  calling_replace: "+\1",
  called_match: "^00(.+)$",
  called_replace: "+\1",
  priority: 5,
  description: "□00□□□□□□□+",
  enabled: true,
  continue: true  # □□□□□
}

# □□2□□□□□□□□□□□□□□□□□□
%{
  calling_match: "^\+(\d+)$",
  calling_replace: "00\1",
  called_match: "^\+(\d+)$",
  called_replace: "00\1",
  priority: 10,
  description: "□+□□□□□□□□□□00",
  enabled: true,
  continue: false  # □□
}

# □□□□□
# □□1□"00441234567890" → "+441234567890"□□□1□□□□
# □□2□"+441234567890" → "00441234567890"□□□2□□□□
# □□□"00441234567890"
# □□□□□□□[□□1□□□2]
```

□□3□SMSC□□□□

□□□□□□□SMSC□□□□□□□□

```
# 规则1: 受信任的SMSC - 直接通过（优先级5）
%{
  source_smsc: "trusted_gateway",
  calling_match: nil,  # 无转换
  calling_replace: nil,
  called_match: nil,
  called_replace: nil,
  priority: 5,
  description: "通过来自受信任网关的号码",
  enabled: true,
  continue: false
}

# 规则2: 不受信任的SMSC - 规范化（优先级10）
%{
  source_smsc: "untrusted_gateway",
  calling_match: "^(.*)$",
  calling_replace: "+VALIDATE\1",
  called_match: "^(.*)$",
  called_replace: "+VALIDATE\1",
  priority: 10,
  description: "为不受信任的来源添加验证前缀",
  enabled: true,
  continue: false
}

# 规则3: 其他SMSC的捕获所有（优先级100）
%{
  source_smsc: nil,  # 通配符
  calling_match: "^(\d{10})$",
  calling_replace: "+1\1",
  called_match: "^(\d{10})$",
  called_replace: "+1\1",
  priority: 100,
  description: "默认: 为10位数字添加+1",
  enabled: true,
  continue: false
}
```

示例4：多步骤格式化链

需求：规范化 → 添加国家代码 → 使用破折号格式化

```
# 规则1: 去掉前导零（继续）
%{
  calling_match: "^0+(.+)$",
  calling_replace: "\1",
  called_match: "^0+(.+)$",
  called_replace: "\1",
  priority: 5,
  description: "去掉前导零",
  enabled: true,
  continue: true
}

# 规则2: 如果缺失则添加国家代码（继续）
%{
  calling_match: "^(\d{10})$",
  calling_replace: "+1\1",
  called_match: "^(\d{10})$",
  called_replace: "+1\1",
  priority: 10,
  description: "为10位数字添加+1",
  enabled: true,
  continue: true
}

# 规则3: 使用破折号格式化（停止）
%{
  calling_match: "^\+1(\d{3})(\d{3})(\d{4})$",
  calling_replace: "+1-\1-\2-\3",
  called_match: "^\+1(\d{3})(\d{3})(\d{4})$",
  called_replace: "+1-\1-\2-\3",
  priority: 15,
  description: "格式化为+1-XXX-XXX-XXXX",
  enabled: true,
  continue: false
}

# 测试用例:
# 输入: "005551234567"
# 步骤1: "005551234567" → "5551234567"（规则1，继续）
# 步骤2: "5551234567" → "+15551234567"（规则2，继续）
# 步骤3: "+15551234567" → "+1-555-123-4567"（规则3，停止）
```

```
# 结果："+1-555-123-4567"
# 应用的规则：[规则1，规则2，规则3]
```

支持

如有问题或疑问：

- 检查测试套件 `test/sms_c/messaging/number_translation_test.exs` 中的示例
- 使用模拟器调试转换逻辑
- 查看事件日志以获取转换决策
- 检查Mnesia表内容：`:mnesia.table_info(:translation_rule, :size)`
- 监控Telemetry指标以查找性能问题

SMS-C 操作指南

← 返回文档索引 | 主 README

SMS-C 操作团队的日常操作程序、监控和维护任务。

目录

日常操作

早晨健康检查

在每一天开始时执行以下检查：

1. 检查系统状态

```
# API 健康检查
curl https://api.example.com:8443/api/status

# 预期响应：
# {"status":"ok","application":"OmniMessage","timestamp":"2025-10-30T08:00:00Z"}
```

2. □□ Prometheus □□

□□ Prometheus □□□□□□□□

- □□□□□□□□□ 24 □□□
- □□□□□□□□ < 1%□
- □□□□□□□ < 1000 □□□□
- □□□□□□□□ > 95%□
- □□□□□□□□□

- 收费失败
- 数据库连接问题
- 集群节点问题

消息量监控

检查每小时消息计数：

使用 Prometheus 查询：

```
# 每小时接收的息
increase(sms_c_message_received_count[1h])

# 每小时投递的息
increase(sms_c_delivery_succeeded_count[1h])

# 计算投递率
rate(sms_c_delivery_succeeded_count[1h]) /
rate(sms_c_message_received_count[1h])
```

预期模式：

- 工作时间：较高的消息量
- 夜间/周末：较低的消息量
- 投递率：应 > 95%

警报条件：

- 消息突然下降（> 50% 减少）
- 消息突然激增（> 200% 增加）
- 投递率降至 90% 以下

监控

关键指标监控

消息处理指标

接收消息计数 (`sms_c_message_received_count`):

- 什么: 进入系统的总消息
- 警报: 突然下降或激增
- 查询: `rate(sms_c_message_received_count[5m])`

消息处理时长 (`sms_c_message_processing_stop_duration`):

- 什么: 端到端处理时间
- 警报: p95 > 1000ms
- 查询: `histogram_quantile(0.95, sms_c_message_processing_stop_duration)`

路由指标

路由失败 (`sms_c_routing_failed_count`):

- 什么: 无法路由的消息
- 警报: 任何失败（> 0）
- 查询: `increase(sms_c_routing_failed_count[5m])`

路由匹配 (`sms_c_routing_route_matched_count`):

- 什么: 正在使用的路由
- 警报: 高优先级路由未匹配
- 查询: `sms_c_routing_route_matched_count`

投递指标

投递成功率:

- 什么: 成功投递的百分比
- 警报: 率 < 95%
- 查询: `rate(sms_c_delivery_succeeded_count[5m]) / rate(sms_c_delivery_queued_count[5m])`

投递尝试 (`sms_c_delivery_succeeded_attempt_count`):

- 什么: 投递所需的重试次数

- □□□p95 > 2□□□□□□□□□
- □□□ `histogram_quantile(0.95, sms_c_delivery_succeeded_attempt_count)`

□□□□□

□□□□ (`sms_c_queue_size_size`)□

- □□□□□□□□□□□□
- □□□□□ > 10,000
- □□□ `sms_c_queue_size_size`

□□□□□□ (`sms_c_queue_oldest_message_age_seconds`)□

- □□□□□□□□□□□□□□□
- □□□□□ > 300 □
- □□□ `sms_c_queue_oldest_message_age_seconds`

□□□□□

□□□□□□□□□

1. □□□□□□□□□□
    - □□□□□□□5 □□□□□
    - □□□□□□□5 □□□□□
    - □□□□□□□□ 24 □□
2. □□□□□□□□□□□
    - □□□□□□□□
    - □□□□□□□
    - □□□□□□□
3. □□□□□□□□□
    - □□□□□□□□□
    - □□□□□□□□□

    - 时间范围：过去 24 小时

4. 路由状态（表格）
    - 路由 ID
    - 匹配计数（过去一小时）
    - 目标 SMSC
    - 优先级
5. 前端状态（表格）
    - 前端名称
    - 状态（活动/过期）
    - 最后见到
    - 消息计数（过去一小时）
6. 系统健康（单个统计）
    - API 响应时间（p95）
    - 数据库查询时间（p95）
    - ENUM 查找时间（p95）

警报配置

关键警报（需要立即响应）：

```yaml
# 未找到路由 - 消息无法投递
- alert: RoutingFailures
  expr: increase(sms_c_routing_failed_count[5m]) > 0
  severity: critical
  description: "{{ $value }} 息在过去 5 分钟内路由失败"

# 队列积压 - 处理滞后
- alert: QueueBacklog
  expr: sms_c_queue_size_pending > 10000
  severity: critical
  description: "队列有 {{ $value }} 待处理息"

# 消息老化 - 投递卡住
- alert: OldMessagesInQueue
  expr: sms_c_queue_oldest_message_age_seconds > 300
  severity: critical
  description: "最旧息已 {{ $value }} 秒"

# 前端断开连接 - 无投递路径
- alert: FrontendDisconnected
  expr: sms_c_frontend_status_count{status="disconnected"} > 0
  severity: critical
  description: "{{ $value }} 个前端断开连接"
```

警告警报（需要调查）：

```
# 投递成功率下降
- alert: LowDeliveryRate
  expr: rate(sms_c_delivery_succeeded_count[10m]) / rate(sms_c_delivery_queued_count[10m]) < 0.90
  severity: warning
  description: "投递成功率为 {{ $value }}"

# 重试次数过多
- alert: HighRetryRate
  expr: histogram_quantile(0.95, sms_c_delivery_succeeded_attempt_count) > 2
  severity: warning
  description: "95th 百分位投递尝试次数: {{ $value }}"

# ENUM 查找缓慢或失败
- alert: SlowEnumLookups
  expr: histogram_quantile(0.95, sms_c_enum_lookup_stop_duration) > 5000
  severity: warning
  description: "ENUM 查找耗时 > 5 秒"

# ENUM 缓存命中率低
- alert: LowEnumCacheHitRate
  expr: rate(sms_c_enum_cache_hit_count[10m]) / (rate(sms_c_enum_cache_hit_count[10m]) + rate(sms_c_enum_cache_miss_count[10m])) < 0.70
  severity: warning
  description: "ENUM 缓存命中率: {{ $value }}"
```

消息跟踪

查找特定消息

按消息 **ID**:

1. **Web UI**: 导航到 `/message_queue`
2. 在搜索框中输入消息 ID
3. 查看完整详细信息和事件历史

通过 **API**:

```
curl https://api.example.com:8443/api/messages/12345
```

按电话号码:

1. **Web UI**: 导航到 `/message_queue`
2. 在搜索框中输入电话号码
3. 查看该号码的所有消息

跟踪消息生命周期

查看事件历史:

1. **Web UI**: 点击队列中的消息，查看“事件”部分
2. **API**: `GET /api/events/12345`

常见事件序列:

```
1. message_inserted - 息已创建
   ↓
2. number_translated - 号码已标准化（如果已配置）
   ↓
3. message_routed - 已做出路由决策
   ↓
4. charging_attempted - 收费检查（如果启用）
   ↓
5. message_delivered - 成功投递
```

投递失败序列:

```
1. message_inserted
   ↓
2. message_routed
   ↓
3. delivery_attempt_1 - 第一次尝试失败
   ↓
4. delivery_attempt_2 - 第二次尝试失败（延迟 2 分钟）
   ↓
5. delivery_attempt_3 - 第三次尝试失败（延迟 4 分钟）
   ↓
6. message_dead_letter - 超过重试限制
```

检查投递状态

待处理消息：

- 状态：“待处理”
- deliver_after：未来时间戳
- delivery_attempts：0 或较低数字

已投递消息：

- 状态：“已投递”
- deliver_time：投递时间戳
- dest_smsc：投递的前端

失败消息：

- 状态：“待处理”且 delivery_attempts 较高
- deadletter：true（如果过期）
- 检查事件日志以获取失败原因

于位置的消息路由

SMS-C 支持基于位置的消息检索，允许前端自动接收目的地为注册在其位置的订阅者的消息。

工作原理：

当前端使用 `get_messages_for_smsc(smsc_name)` 查询待处理消息时，系统以两种方式返回消息：

1. 显式路由 - `dest_smsc` 明确匹配前端名称的消息
2. 基于位置的路由 - 消息满足以下条件：
   - `dest_smsc` 为 `null`（未明确路由）
   - `destination_msisdn` 有活动位置记录
   - 位置的 `location` 字段与前端名称匹配
   - 位置未过期

示例场景：

MSISDN 为 `+447700900123` 的订阅者在前端 `uk_gateway` 注册：

```
# 订阅者注册（创建位置记录）
POST /api/locations
{
  "msisdn": "+447700900123",
  "imsi": "234150123456789",
  "location": "uk_gateway",
  "expires": "2025-11-01T12:00:00Z"
}
```

当一条消息到达该订阅者而没有明确路由时：

```
# 消息提交时没有 dest_smsc
POST /api/messages
{
  "source_msisdn": "+15551234567",
  "destination_msisdn": "+447700900123",
  "message_body": "Hello",
  "source_smsc": "api"
  # 注意: dest_smsc 为 null
}
```

当 `uk_gateway` 前端轮询时，将自动接收消息：

```
# 前端轮询□息
GET /api/messages/queue?smsc=uk_gateway

# 返回□息，即使 dest_smsc 为 null
# 因为目标订阅者在 uk_gateway 注册
```

位置要求：

为了使基于位置的路由工作：

- `locations` 表必须有 `destination_msisdn` 的条目
- `location` 字段必须与查询的 SMSC 名称匹配
- `expires` 时间戳必须在未来

监控□于位置的路由：

检查位置记录：

```
# 通过 API
GET /api/locations/{msisdn}

# 检查位置是否过期
# expires 字段应 > 当前时间
```

常见问题：

- 消息未投递：检查位置是否已过期
- 错误的前端：验证 `location` 字段是否与预期的前端名称匹配
- 未找到位置：订阅者可能需要重新注册

手动干预

重试失败消息：

```
# 重置 delivery_attempts 和 deliver_after
curl -X PATCH https://api.example.com:8443/api/messages/12345 \
  -H "Content-Type: application/json" \
  -d '{
    "delivery_attempts": 0,
    "deliver_after": "2025-10-30T12:00:00Z"
  }'
```

更改目标：

```
# 路由到不同的 SMSC
curl -X PATCH https://api.example.com:8443/api/messages/12345 \
  -H "Content-Type: application/json" \
  -d '{
    "dest_smsc": "backup_gateway"
  }'
```

删除卡住的消息：

```
curl -X DELETE https://api.example.com:8443/api/messages/12345
```

路由管理

查看当前路由

Web UI：导航到 `/sms_routing`

通过 **API**：

```
# 列出所有路由
curl https://api.example.com:8443/api/routes
```

检查路由使用情况：

Prometheus 查询：

```
# 每条路由路由的□息（过去一小时）
increase(sms_c_routing_route_matched_count[1h])
```

添加新路由

Web UI:

1. 导航到 `/sms_routing`
2. 点击“添加新路由”
3. 填写字段：
    - 呼叫前缀：源号码前缀（可选）
    - 被叫前缀：目的号码前缀（地理路由必需）
    - 源 **SMSC**：源系统过滤器（可选）
    - 目标 **SMSC**：目的网关（除非自动回复/丢弃，否则必需）
    - 优先级：路由优先级（1-255，数字越小优先级越高）
    - 权重：负载均衡权重（1-100）
    - 描述：人类可读的描述
    - 启用：勾选以立即激活
4. 点击“保存路由”

示例：地理路由：

- 被叫前缀：`+44`
- 目标 SMSC：`uk_gateway`
- 优先级：50
- 权重：100
- 描述：“英国路由”

示例：负载均衡路由：

创建两个具有相同标准但不同权重的路由：

路由 1：

- 被叫前缀：`+44`
- 目标 SMSC：`uk_primary`

- 优先级：50
- 权重：70
- 描述："英国主要（70%）"

路由 2：

- 被叫前缀：`+44`
- 目标 SMSC：`uk_backup`
- 优先级：50
- 权重：30
- 描述："英国备份（30%）"

测试路由

路由模拟器：

1. 导航到 `/simulator`
2. 输入测试参数：
    - 呼叫号码：`+15551234567`
    - 被叫号码：`+447700900000`
    - 源 SMSC：（可选）
    - 源类型：（可选）
3. 点击"模拟路由"
4. 查看结果：
    - 选择的路由：选择了哪个路由
    - 所有匹配：哪些路由匹配了标准
    - 评估：每条路由匹配或不匹配的原因

生产前测试：

- 在模拟器中测试所有新路由
- 验证选择了正确的路由
- 检查优先级排序
- 验证权重分配

修改现有路由

Web UI:

1. 导航到 `/sms_routing`
2. 在列表中找到路由
3. 点击“编辑”
4. 修改字段
5. 点击“保存路由”

常见修改:

- 禁用路由: 取消勾选“启用”(临时移除)
- 调整权重: 更改负载均衡分配
- 更改优先级: 重新排序路由评估
- 更新目标: 切换到不同的 SMSC

删除路由

Web UI:

1. 导航到 `/sms_routing`
2. 在列表中找到路由
3. 点击“删除”
4. 确认删除

警告: 删除路由是永久性的。考虑改为禁用。

导出/导入路由

导出路由（备份）:

1. 导航到 `/sms_routing`
2. 点击“导出路由”
3. 保存 JSON 文件

导入路由:

1. □□□ `/sms_routing`
2. □□“□□□□”
3. □□ JSON □□
4. □□□□□□□
   - □□□□□□□□□□□
   - □□□□□□□□□□□

□□□

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

□□□□

□□□□□□

Web UI□□□□ `/frontend_status`

□□□

- □□□□□□□□□“□□”
- □□□□□□□□□□□< 90 □□
- □□□□□□□□□

□□ **API**□

```
# □□□□□□□
curl https://api.example.com:8443/api/frontends/active

# □□□□□□□
curl https://api.example.com:8443/api/frontends/stats
```

调查断开连接

前端过期：

1. 检查前端日志以获取错误
2. 验证与 SMS-C 的网络连接
3. 确认前端正在运行
4. 检查前端注册逻辑（应每 60 秒重新注册）

注册未显示：

1. 验证前端是否调用 POST `/api/frontends/register`
2. 检查 API 日志以获取注册错误
3. 验证 JSON 有效负载格式
4. 使用 curl 手动测试注册

示例手动注册：

```
curl -X POST https://api.example.com:8443/api/frontends/register \
  -H "Content-Type: application/json" \
  -d '{
    "frontend_name": "test_gateway",
    "frontend_type": "smpp",
    "ip_address": "10.0.1.50",
    "hostname": "gateway.example.com"
  }'
```

查看前端历史

Web UI:

1. 导航到 `/frontend_status`
2. 在列表中找到前端
3. 点击“历史”
4. 查看过去的注册

通过 **API**:

```
curl https://api.example.com:8443/api/frontends/history/uk_gateway
```

用例：

- 调查连接可靠性
- 跟踪前端正常运行模式
- 识别配置更改

号码翻译管理

号码翻译规则通过 `config/runtime.exs` 管理。更改需要重启应用程序。

查看活动翻译规则

检查配置文件：

```
cat config/runtime.exs | grep -A 20 "translation_rules:"
```

常见翻译任务

为本地号码添加国家代码：

编辑 `config/runtime.exs`：

```
%{
  calling_prefix: nil,
  called_prefix: nil,
  source_smsc: nil,
  calling_match: "^(\d{10})$",
  calling_replace: "+1\1",
  called_match: "^(\d{10})$",
  called_replace: "+1\1",
  priority: 100,
  description: "为 10 位美国号码添加 +1",
  enabled: true
}
```

标准化国际格式：

```
%{
  calling_prefix: nil,
  called_prefix: nil,
  source_smsc: nil,
  calling_match: "^00(\d+)$",
  calling_replace: "+\1",
  called_match: "^00(\d+)$",
  called_replace: "+\1",
  priority: 10,
  description: "将 00 前缀转换为 +",
  enabled: true
}
```

运营商特定代码剥离：

```
%{
  calling_prefix: nil,
  called_prefix: "101",
  source_smsc: "carrier_a",
  calling_match: nil,
  calling_replace: nil,
  called_match: "^101(\d+)$",
  called_replace: "\1",
  priority: 5,
  description: "□□□□ A □□□□□□□□□",
  enabled: true
}
```

□□□□□□

□□□□□□□□

1. □□□□□□□□□□□□□□□
2. □□□□□□□□□□/□□□□□□□
3. □□□□□□□□□□ `number_translated` □□
4. □□□□□□□□□□□□

□□□□□□

□□□□□□□ `enabled: false`□

```
%{
  ...
  enabled: false
}
```

□□□□□□□□

系统维护

数据库维护

检查数据库大小：

使用数据库管理工具监控 CDR 存储大小：

- **MySQL/MariaDB**：查询 `information_schema.tables` 获取数据库大小
- **PostgreSQL**：使用 `pg_database_size()` 函数或在 psql 中使用 `\l+` 命令

清理旧的 **CDR** 记录：

根据保留政策定期归档和清除 CDR 记录：

- 根据业务需求配置自动归档（通常在操作数据库中保留 30-90 天）
- 将旧记录归档到数据仓库或冷存储
- 从操作数据库中批量删除归档记录，以避免锁争用

优化表：

定期优化数据库表以保持性能：

- **MySQL/MariaDB**：在低流量期间运行 `OPTIMIZE TABLE` 命令
- **PostgreSQL**：定期运行 `VACUUM ANALYZE`（或启用自动真空）

每周运行 在低流量期间以保持最佳性能。

Mnesia 数据库维护

检查 **Mnesia** 表大小：

```
# 在 IEx 控制台中
:mnesia.table_info(:sms_route, :size)
:mnesia.table_info(:translation_rule, :size)
```

备份 **Mnesia** 表：

```
# 导出路由（Web UI）
# 导航到 /sms_routing
# 点击“导出路由”

# 或通过 Mnesia 备份
:mnesia.backup("/var/backups/sms_c/mnesia_backup.bup")
```

恢复 **Mnesia**:

```
# 通过 Web UI 导入
# 或恢复备份:
:mnesia.restore("/var/backups/sms_c/mnesia_backup.bup", [])
```

日志轮换

为应用程序日志配置 logrotate:

```
# /etc/logrotate.d/sms_c
/var/log/sms_c/*.log {
    daily
    rotate 30
    compress
    delaycompress
    notifempty
    create 0644 sms_user sms_group
    sharedscripts
    postrotate
        systemctl reload sms_c || true
    endscript
}
```

重启应用程序

优雅重启（集群中零停机时间）：

```
# 一次重启一个节点
systemctl restart sms_c

# 等待节点加入集群
# 对每个节点重复
```

紧急重启（所有节点）：

```
systemctl restart sms_c
```

重启后:

- 验证所有前端重新连接
- 检查 Prometheus 以确保指标连续
- 监控日志以获取错误
- 验证消息处理恢复

备份和恢复

备份内容

1. 配置文件:
    - `config/runtime.exs`
    - `config/config.exs`
    - `config/prod.exs`（如果存在）
2. 路由表（Mnesia）：
    - 通过 Web UI 导出
    - 或 Mnesia 备份命令
3. **SQL CDR** 数据库:
    - 每日全量备份
    - 事务日志备份（持续）

4. **TLS** 证书:
    - `priv/cert/*.crt`
    - `priv/cert/*.key`

备份程序

每日配置备份:

```
#!/bin/bash
# /opt/sms_c/scripts/backup_config.sh

BACKUP_DIR="/var/backups/sms_c/$(date +%Y%m%d)"
mkdir -p $BACKUP_DIR

# 备份配置
cp -r /opt/sms_c/config $BACKUP_DIR/

# 备份证书
cp -r /opt/sms_c/priv/cert $BACKUP_DIR/

# 设置权限
chmod 600 $BACKUP_DIR/cert/*

echo "配置备份完成: $BACKUP_DIR"
```

数据库备份:

```
#!/bin/bash
# /opt/sms_c/scripts/backup_database.sh

BACKUP_DIR="/var/backups/sms_c/database"
DATE=$(date +%Y%m%d_%H%M%S)

mkdir -p $BACKUP_DIR

# □□ SQL CDR □□□
# MySQL/MariaDB□□□ mysqldump □ --single-transaction □□□□□□
# PostgreSQL□□□ pg_dump -F c □□□□□□

# □□□□□□□□□□□□□□□□□□□□□
# - □□□□□□□□□□mysqldump□pg_dump□
# - □□□□□□□□□□□□□□□
# - □□□□□□□□□□
# - □□□□□□□□□□30 □□

# □□□□□
find $BACKUP_DIR -name "sms_c_*.gz" -mtime +30 -delete

echo "□□□□□□□□sms_c_${DATE}"
```

□□□□□□□

```
#!/bin/bash
# /opt/sms_c/scripts/backup_routes.sh

BACKUP_DIR="/var/backups/sms_c/routes"
DATE=$(date +%Y%m%d)

mkdir -p $BACKUP_DIR

# □□ API □□
curl https://api.example.com:8443/api/routes/export \
  > $BACKUP_DIR/routes_${DATE}.json

echo "□□□□□□□□routes_${DATE}.json"
```

□□□□□crontab□□

```
# 每天凌晨 2 点
0 2 * * * /opt/sms_c/scripts/backup_config.sh
0 2 * * * /opt/sms_c/scripts/backup_database.sh
0 2 * * * /opt/sms_c/scripts/backup_routes.sh
```

恢复程序

恢复配置：

```
# 停止应用程序
systemctl stop sms_c

# 恢复配置文件
cp -r /var/backups/sms_c/20251030/config/* /opt/sms_c/config/

# 恢复证书
cp -r /var/backups/sms_c/20251030/cert/* /opt/sms_c/priv/cert/

# 启动应用程序
systemctl start sms_c
```

恢复 **SQL CDR** 数据库：

使用适当的恢复工具恢复数据库：

- **MySQL/MariaDB**：解压并通过 mysql 客户端进行管道
- **PostgreSQL**：使用 pg_restore 进行自定义格式转储

重要：在恢复数据库之前停止 SMS-C 应用程序，以防止数据冲突。

恢复路由表：

1. 导航到 Web UI `/sms_routing`
2. 点击“导入路由”
3. 选择备份 JSON 文件
4. 选择“替换”模式
5. 确认导入

容量规划

监控增长趋势

消息量趋势:

Prometheus 查询（30 天平均）：

```
avg_over_time(sms_c_message_received_count[30d])
```

数据库增长率:

```
-- 每月数据增长
SELECT
  DATE_FORMAT(inserted_at, '%Y-%m') AS month,
  COUNT(*) AS message_count,
  ROUND(SUM(LENGTH(message_body)) / 1024 / 1024, 2) AS data_mb
FROM message_queues
GROUP BY month
ORDER BY month DESC
LIMIT 12;
```

容量指标

CPU 使用率:

- 正常: < 50% 平均
- 高: > 70% 持续
- 关键: > 90%

内存使用率:

- 正常: < 70% 可用
- 高: > 80%
- 关键: > 90%

磁盘使用率:

- 正常: < 60% 已满
- 高: > 75%
- 关键: > 85%

队列深度:

- 正常: < 1000 待处理
- 高: > 5000 待处理
- 关键: > 10,000 待处理

扩展建议

何时进行垂直扩展（升级资源）:

- CPU 持续 > 70%
- 内存持续 > 80%
- 单节点瓶颈

何时进行水平扩展（添加节点）:

- 所有节点 CPU > 50%
- 消息量 > 5000 msg/sec
- 需要地理分布
- 需要高可用性

数据库扩展:

- 用于报告查询的只读副本
- 连接池优化
- 索引优化
- 按日期对大表进行分区

事件响应

严重性级别

关键（立即响应）：

- 无消息被投递
- 所有前端断开连接
- 数据库不可用
- API 完全宕机

高（1 小时内响应）：

- 投递成功率 < 80%
- 多个前端断开连接
- 路由失败 > 10%
- 队列积压增长

中（4 小时内响应）：

- 单个前端断开连接
- 投递成功率 80-95%
- 消息处理缓慢
- ENUM 查找失败

低（24 小时内响应）：

- 轻微性能下降
- 单条路由问题
- 非关键警告警报

事件检查清单

1. 评估严重性:

- 检查 Prometheus 警报
- 审查仪表板指标

- 检查消息队列状态
- 验证前端连接

2. 收集信息：

- 最近的配置更改？
- 最近的部署？
- 外部依赖状态（OCS、DNS）？
- 日志中的错误消息？

3. 立即行动：

- 停止正在进行的更改
- 如果怀疑是原因，回滚最近的部署
- 如果需要，启用详细日志记录
- 通知利益相关者

4. 调查：

- 审查应用日志
- 检查系统资源使用情况
- 检查数据库性能
- 测试外部依赖

5. 解决方案：

- 应用修复
- 在模拟器中测试
- 部署到生产
- 监控改进情况

6. 事件后：

- 记录根本原因
- 更新监控/警报
- 实施预防措施
- 更新操作手册

常见事件

高队列积压：

1. 检查投递成功率
2. 验证前端是否连接并轮询
3. 检查数据库性能
4. 审查 Prometheus 以查找瓶颈
5. 考虑增加批量大小/间隔

路由失败：

1. 审查路由配置
2. 在路由模拟器中测试
3. 检查缺失的路由
4. 验证是否存在捕获所有路由
5. 检查事件日志以获取失败原因

前端断开连接：

1. 检查前端系统状态
2. 验证网络连接
3. 审查前端日志
4. 测试手动 API 注册
5. 检查防火墙规则

消息处理缓慢：

1. 检查数据库查询性能
2. 审查批处理工作者配置
3. 验证资源是否充足（CPU/内存）
4. 检查 ENUM 查找延迟
5. 审查收费系统性能

有关详细故障排除程序，请参见 故障排除指南。

性能调优指南

← 返回文档索引 | 主 **README**

本指南解释了如何优化 SMS-C 在不同工作负载场景下的性能。

性能概述

SMS-C 使用 Mnesia 进行内存消息存储，并自动将 SQL 数据库归档以保留 CDR，从而提供 **1,750** 消息/秒 的吞吐量。

关键性能指标

在 Intel i7-8650U @ 1.90GHz (8 核) 上测量:

操作	吞吐量	延迟（平均）	改进
消息插入（带路由）	1,750 消息/秒	0.58 毫秒	比 SQL 快 21 倍
消息插入（简单）	1,750 消息/秒	0.57 毫秒	比 SQL 快 21 倍
获取 SMSC 消息	800 消息/秒	1.25 毫秒	内存查询
每次插入内存	62 KB	-	减少 50%

容量: ~每天 1.5 亿条消息，单节点

目录

消息存储架构

SMS-C 使用双存储架构以实现最佳性能：

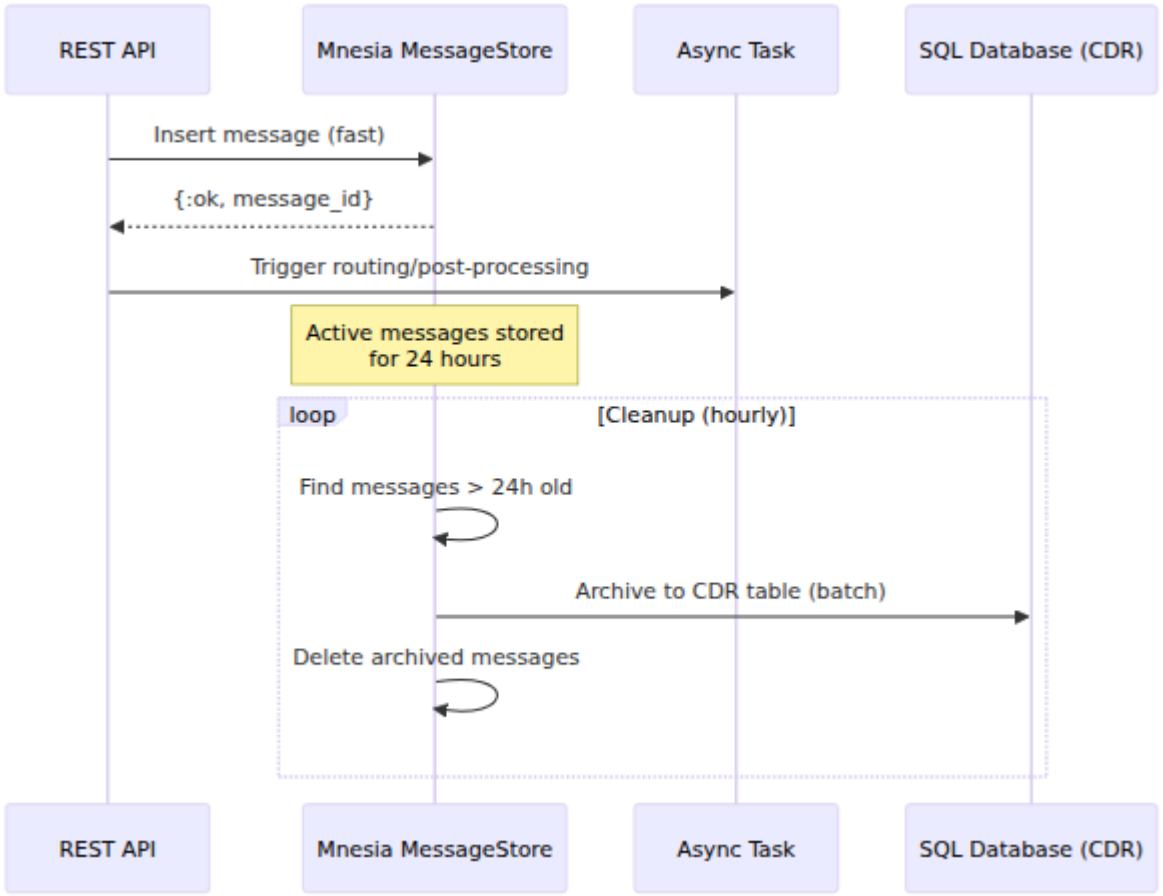
活动消息存储 **(Mnesia)**

- 目的：超快的消息插入、路由和交付
- 存储：内存中带磁盘持久性 (disc_copies)
- 性能：1,750 消息/秒插入吞吐量，0.58 毫秒延迟
- 保留：可配置（默认：24 小时）
- 集群：支持分布式 Mnesia 以实现水平扩展

CDR 归档 **(SQL** 数据库**)**

- 目的：长期消息历史和报告
- 存储：SQL 数据库 (MySQL/MariaDB 或 PostgreSQL) 以实现持久归档
- 性能：批量写入以最小化数据库负载
- 保留：永久（或根据数据保留政策）
- 查询：分析、报告、合规

数据流



Mnesia 优化

消息保留配置

```
# config/runtime.exs
config :sms_c,
  message_retention_hours: 24  # 默认: 24 小时
```

调优指南:

- 高容量 **(>1M 消息/天)**: 12-24 小时保留
    - 最小化 Mnesia 表大小
    - 更快的查询
    - 更频繁地归档到 MySQL

- 中等容量 **(100K-1M 消息/天)**: 24-48 小时保留
    - 对于大多数部署来说是良好的平衡
    - 适当的重试逻辑缓冲
- 低容量 **(<100K 消息/天)**: 48-168 小时保留
    - 在快速存储中保留更长的消息历史
    - 较少的归档频率

Mnesia 表索引

MessageStore 自动创建以下索引:

- `status` - 用于过滤待处理/已交付消息
- `dest_smsc` - 用于特定 SMSC 的查询
- `expires` - 用于过期处理
- `destination_msisdn` - 用于订阅者查询
- `source_msisdn` - 用于订阅者查询

Mnesia 磁盘持久性

消息以 `disc_copies` 存储，提供:

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

CDR 归档配置

`BatchInsertWorker` 处理使用批量写入将 CDR 归档到 MySQL:

```
# config/runtime.exs
config :sms_c,
  batch_insert_batch_size: 100,           # CDR □□□□
  batch_insert_flush_interval_ms: 100     # □□□□□□□
```

CDR □□□□

□□□□□

```
batch_insert_batch_size: 200
batch_insert_flush_interval_ms: 200
```

- □□���□□□□ MySQL □□
- CDR □□□□□□□□□□□□□□□□□

□□□□□□

```
batch_insert_batch_size: 100
batch_insert_flush_interval_ms: 100
```

- □□□□□□□□□□□□□□□□□
- CDR □ 100 □□□□□

□□ **CDR** □□

```
batch_insert_batch_size: 20
batch_insert_flush_interval_ms: 20
```

- □□□ CDR □□□□□□□□□
- □□ MySQL □□□□

查询优化

有效使用 Mnesia 索引

使用索引字段的查询是最快的：

```
# 快速查询（使用索引）
MessageStore.list(status: :pending)
MessageStore.list(dest_smsc: "gateway-1")
Messaging.get_messages_for_smsc("gateway-1")

# 较慢的查询（全表扫描）
MessageStore.list(limit: :infinity)  # 返回所有□息
```

MySQL 连接池

对于 CDR 查询和归档，配置 MySQL 连接池：

```
# config/runtime.exs
config :sms_c, SmsC.Repo,
  pool_size: 10  # 对于重度 CDR 报告增加
```

指南：

- 标准部署：`pool_size: 10`
- 重度 CDR 报告：`pool_size: 20-30`
- 仅归档：`pool_size: 5`

□准测试

运行□准测试

该项目包括基于 Benchee 的基准测试以进行性能测试：

```
# 原始 SMS API 基准测试（比较同步与异步）
mix run benchmarks/raw_sms_bench.exs

# 一般□息 API 基准测试
mix run benchmarks/message_api_bench.exs
```

解释结果

示例输出：

```
Name                                          ips        average
deviation         median         99th %
submit_message_raw_async (batch)        4.65 K        0.22 ms
±41.72%       0.184 ms        0.55 ms
submit_message_raw (sync)             0.0696 K       14.36 ms
±33.42%       12.57 ms       33.71 ms
```

关键指标：

- **ips**：每秒迭代次数（越高越好）
- **average**：平均执行时间（越低越好）
- **median**：中间值，对于偏态分布比平均值更具代表性
- **99th %**：99 百分位延迟（对 SLA 合规性很重要）

性能□线

在现代硬件（Intel i7-8650U，8 核）上的预期性能：

指标	**insert_message (Mnesia)**	之前 **(MySQL)**
吞吐量 (带路由)	1,750 消息/秒	83 消息/秒
吞吐量 (简单)	1,750 消息/秒	89 消息/秒
响应时间 (平均)	0.58 毫秒	16 毫秒
响应时间 (p99)	<5 毫秒	30 毫秒
每次操作内存	62 KB	121 KB
性能提升	快 **21** 倍	-

关键改进:

- □ □□□□□□□□□□□□
- □ □□□□□□□□□□□□□□□□
- □ Mnesia 内存□□□储与 MySQL 磁盘 I/O
- □ 50% 内存减少

监控

运行时统计

检查批处理工作者统计信息:

```
SmsC.Messaging.BatchInsertWorker.stats()
```

返回:

```
%{
  total_enqueued: 10000,
  total_flushed: 9900,
  total_batches: 99,
  current_queue_size: 100,
  flush_errors: 0,
  last_flush_at: ~U[2025-10-22 12:34:56Z],
  last_flush_count: 100,
  last_flush_duration_ms: 45
}
```

关键指标监控

1. 队列大小：`current_queue_size` - 大部分时间应低于 `batch_size`
2. 刷新持续时间：`last_flush_duration_ms` - 对于 batch_size=100 应 < 100 毫秒
3. 刷新错误：`flush_errors` - 应为 0 或非常低
4. 吞吐量：`total_flushed / uptime` - 应与预期负载匹配

警报

设置监控警报以便于：

- 队列大小持续达到最大（表示背压）
- 刷新持续时间增加（数据库性能下降）
- 刷新错误 > 0（数据库连接问题）
- 吞吐量低于预期（性能下降）

故障排除

症状：低吞吐量

可能原因：

1. 数据库连接池耗尽：增加 `pool_size`
2. 数据库缓慢：检查查询性能，添加索引

3. 网络延迟：优化到数据库的网络路径
4. 批量大小太小：增加 `batch_insert_batch_size`

症状：高延迟

可能原因：

1. 刷新间隔太高：减少 `batch_insert_flush_interval_ms`
2. 批量大小太高：减少 `batch_insert_batch_size`
3. 数据库写入缓慢：检查磁盘 I/O，优化表
4. 在需要同步时使用异步 API：切换到同步端点

症状：内存问题

可能原因：

1. 队列备份：消息积累速度快于刷新
2. 批量大小太大：减少 `batch_insert_batch_size`
3. 刷新失败：检查统计信息中的 `flush_errors`
4. 需要重启工作者：`Supervisor.terminate_child/2` 并重启

最佳实践

1. 从默认值开始（100/100ms），根据观察到的行为进行调整
2. 在生产环境中监控至少 1 周，然后再进行优化
3. 在预发布环境中测试配置更改，使用类似生产的负载
4. 使用□准测试验证配置更改
5. 记录您的调优决策以供将来参考
6. 在优化之前设置警报以捕捉回归
7. 考虑时区 - 峰值负载因地区而异

示例配置

配置：高容量聚合器

```
# config/prod.exs
config :sms_c,
  batch_insert_batch_size: 200,
  batch_insert_flush_interval_ms: 200

config :sms_c, SmsC.Repo,
  pool_size: 50
```

配置：企业实时消息传递

```
# config/prod.exs
config :sms_c,
  batch_insert_batch_size: 20,
  batch_insert_flush_interval_ms: 10

config :sms_c, SmsC.Repo,
  pool_size: 20
```

配置：开发/测试

```
# config/dev.exs
config :sms_c,
  batch_insert_batch_size: 10,
  batch_insert_flush_interval_ms: 50

config :sms_c, SmsC.Repo,
  pool_size: 5
```

进一步阅读

- Ecto 性能指南

- Benchee □□
- Phoenix □□□□

SMS-C 路由指南

← 返回文档索引 | 主 README

概述

SMS-C 路由系统提供基于多个标准（包括号码前缀、SMSC 标识符、连接类型等）灵活、高性能的 SMS 消息路由。路由存储在 Mnesia 中以确保持久性，并且可以在运行时进行修改而不会中断服务。

主要特点

- □于前缀的路由：基于呼叫/被叫号码前缀进行路由，采用最长匹配优先的逻辑
- □于 **SMSC** 的路由：基于源或目的地 SMSC 进行路由
- □于类型的路由：基于源连接类型（IMS、电路交换、SMPP）进行路由
- □于优先级的路由：通过可配置的优先级控制路由选择顺序
- □于权重的负载均衡：使用权重在多个路由之间分配流量
- 自动回复路由：自动向消息发起者发送回复
- 丢弃路由：丢弃符合特定标准的消息（垃圾邮件过滤等）
- 计费控制：为每条路由配置计费行为（是/否/默认）
- 配置文件加载：在首次启动时从 `runtime.exs` 加载初始路由
- ��行时配置：在不重启的情况下添加、修改或禁用路由
- **Web UI**：具有前端下拉菜单的完整 CRUD 接口，用于路由管理
- 模拟工具：在部署前测试路由逻辑
- 备份/恢复：导出和导入路由配置
- **ENUM** 支持：基于 DNS 的号码查找（未来实施）

架构

数据模型

每条路由包含以下字段：

字段	类型	描述	
`route_id`	integer	自增唯一标识符	是（[
`calling_prefix`	string/nil	呼叫号码的前缀匹配（nil = 通配符）	否
`called_prefix`	string/nil	被叫号码的前缀匹配（nil = 通配符）	否
`source_smsc`	string/nil	源 SMSC 名称（nil = 通配符）	否
`dest_smsc`	string/nil	目的地 SMSC 名称（除非 auto_reply 或 drop 为真，否则必需）	有条[
`source_type`	atom/nil	源类型：`:ims`、`:circuit_switched`、`:smpp` 或 nil	否
`enum_domain`	string/nil	用于查找的 DNS ENUM 域	否
`auto_reply`	boolean	如果为真，则向发起者发送回复	否（[
`auto_reply_message`	string/nil	自动回复的消息文本（如果 auto_reply 为真，则必需）	有条[
`drop`	boolean	如果为真，则丢弃消息（垃圾邮件过滤）	否（[
`charged`	atom	计费行为：`:yes`、`:no` 或 `:default`	否（[认：
`weight`	integer	负载均衡权重（1-100，默认 100）	是
`priority`	integer	路由优先级（1-255，数字越小优先级越高）	是
`description`	string	人类可读的描述	否
`enabled`	boolean	启用/禁用路由	是

注意：路由必须是以下三种类型之一：

1. 正常路由：`auto_reply=false`，`drop=false`，需要 `dest_smsc`
2. 自动回复：`auto_reply=true`，需要 `auto_reply_message`
3. 丢弃：`drop=true`，丢弃消息

路由算法

在路由消息时，系统遵循以下优先级顺序：

优先级 **1**：□于位置的路由（最高）

1. 检查订阅者注册：如果目的地 MSISDN 在位置表中注册
2. 直接路由到服务前端：跳过所有路由规则，直接发送到服务该订阅者的前端
3. 此操作发生在号码转换之后，以确保与位置注册的一致性

优先级 **2**：标准路由规则（如果未找到位置注册）

1. 过滤启用的路由，匹配所有指定标准
2. 按特异性排序（更特定的路由优先）：
   - 更长的被叫前缀 = 更高的特异性（×100 分）
   - 更长的呼叫前缀 = 中等特异性（×50 分）
   - 指定源 SMSC = +25 分
   - 指定 ENUM 结果域 = +15 分
   - 指定源类型 = +10 分
   - 指定 ENUM 域 = +5 分
3. 按优先级分组（数字越小 = 优先级越高）
4. 从最高优先级组中选择，使用加权随机选择
5. 执行路由操作：
   - 正常路由：返回目的地 SMSC 以进行消息投递
   - 自动回复路由：异步向发起者发送回复
   - 丢弃路由：丢弃消息并记录事件

通配符

- `nil` 或空值作为通配符，匹配任何值
- 没有指定标准的路由是一个捕获所有路由

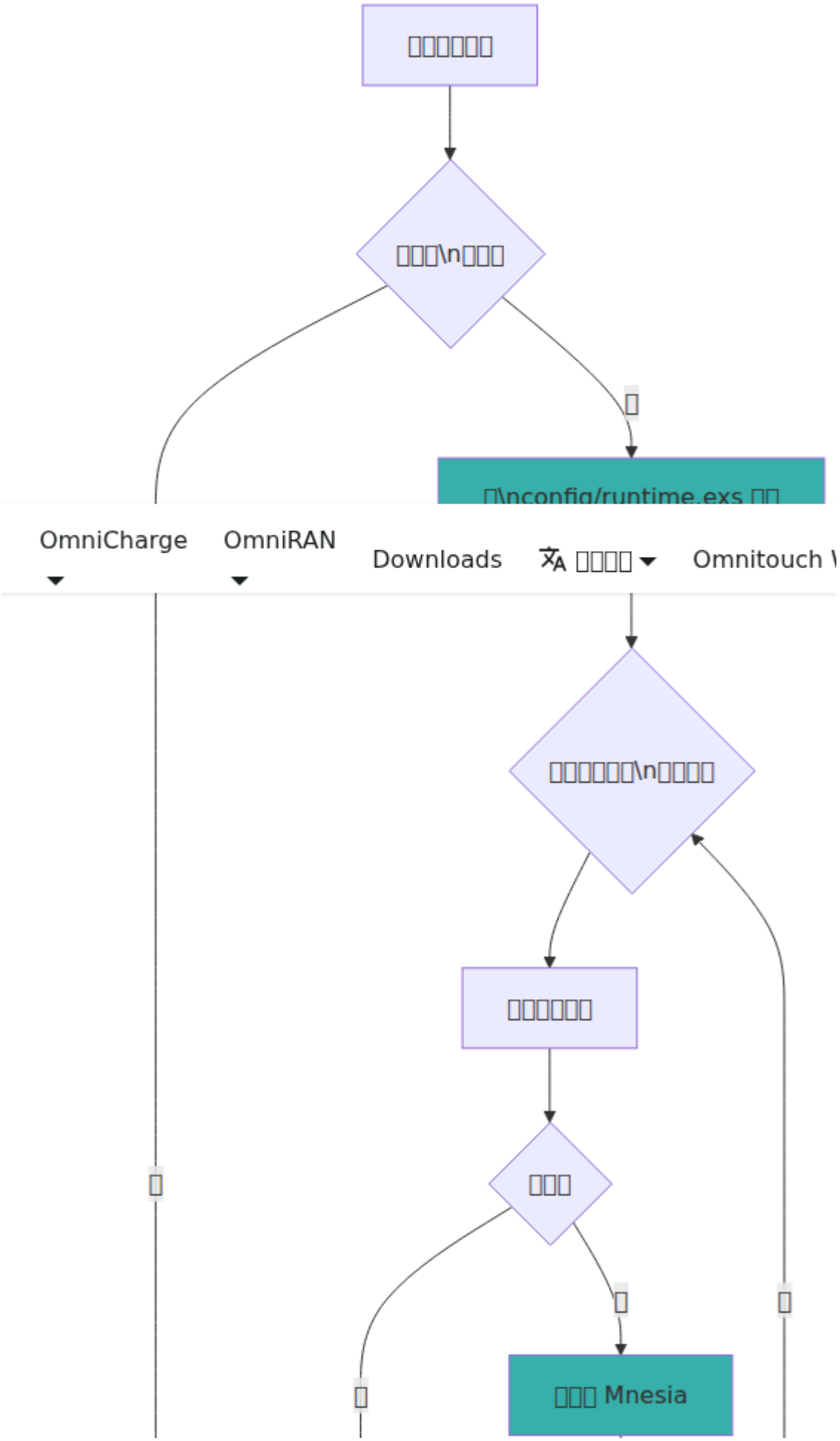
配置

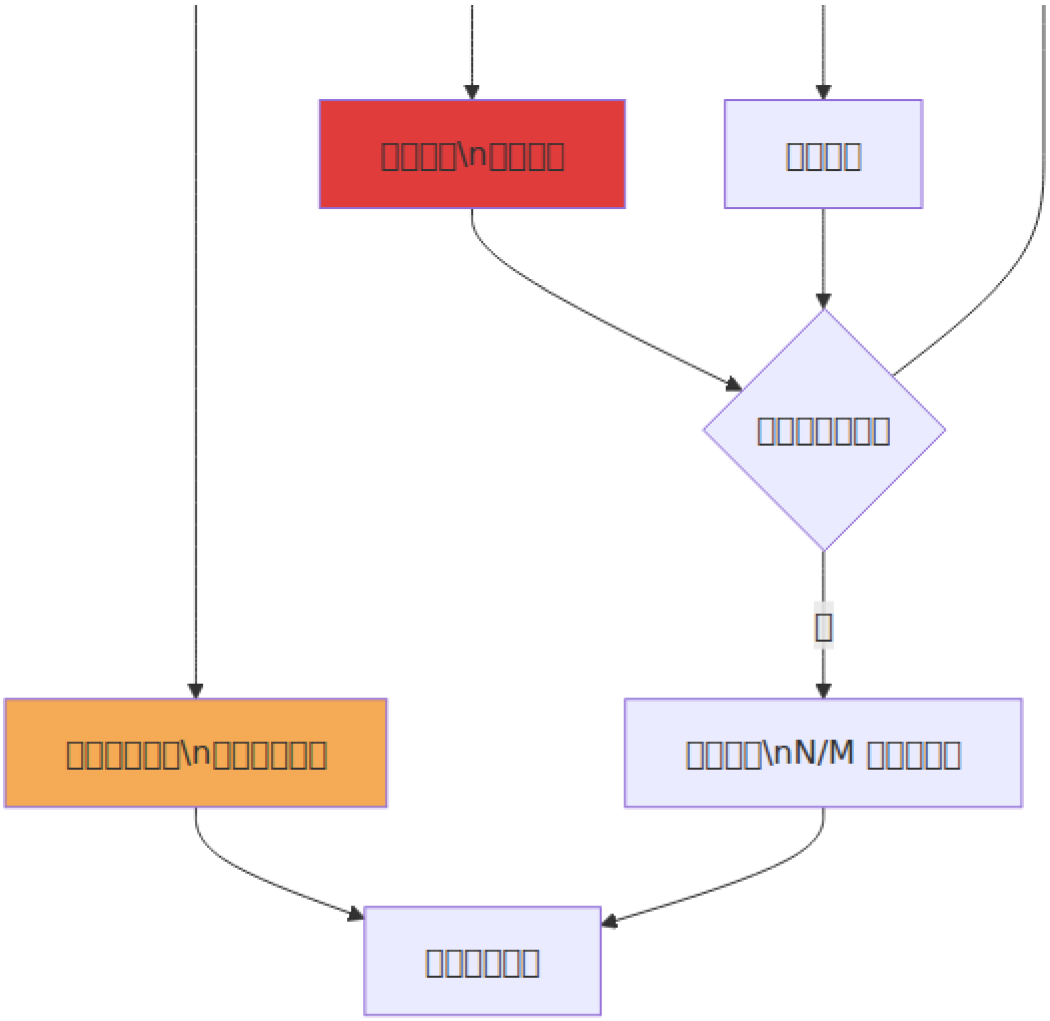
从配置文件加载路由

路由可以在 `config/runtime.exs` 中定义，并将在首次启动时自动加载。这对于定义系统首次启动时应存在的基线路由规则非常有用。

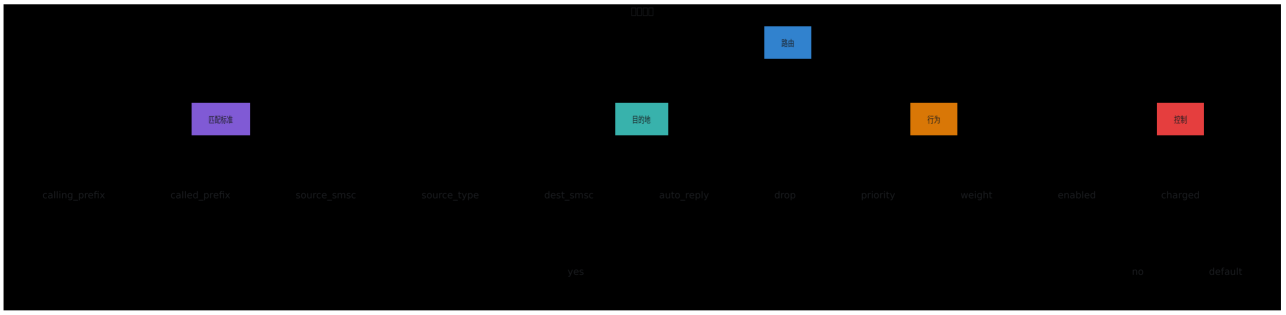
重要：仅在路由表为空（首次启动）时加载配置中的路由。这保留了在运行时通过 Web UI 添加的路由，并防止在重启时出现重复。

配置加载流程


□□□□□□
□□□\n□□□
□
□\nconfig/runtime.exs □□
OmniCharge
OmniRAN
Downloads
□□□□
Omnitouch
□□□□□□\n□□□□
□□□□□□
□□□
□
□
□
□
□□□ Mnesia



示例路由配置结构

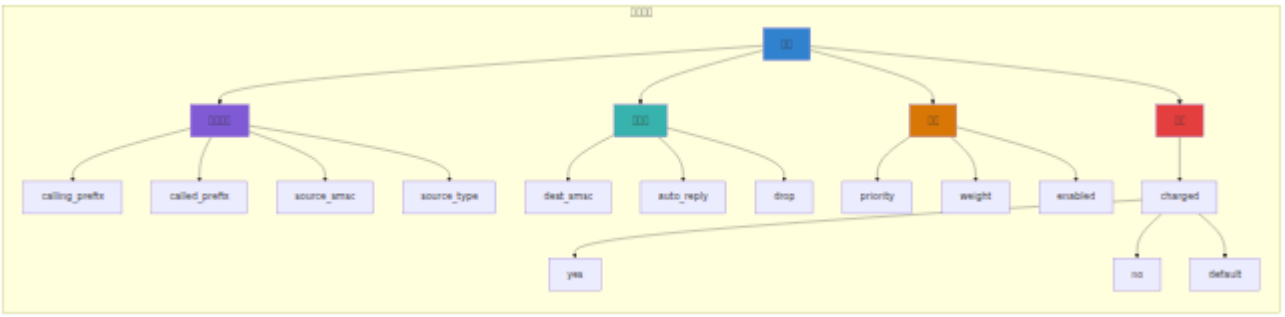


请参见 `config/runtime.exs` 和 `config/sms_routes.example.exs` 获取完整示例，包括：

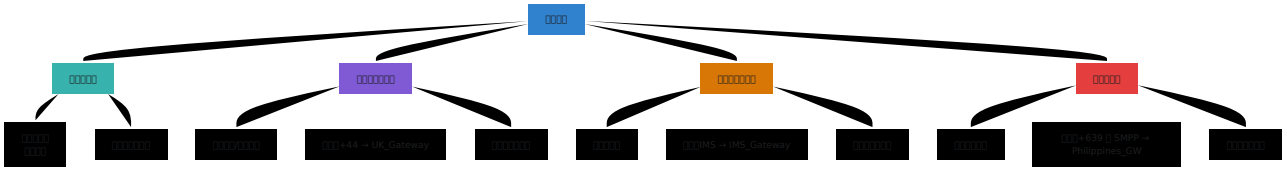
- 地理路由
- 自动回复路由
- 丢弃路由（垃圾邮件过滤）
- 负载均衡路
- 计费的优质号码路由

□□

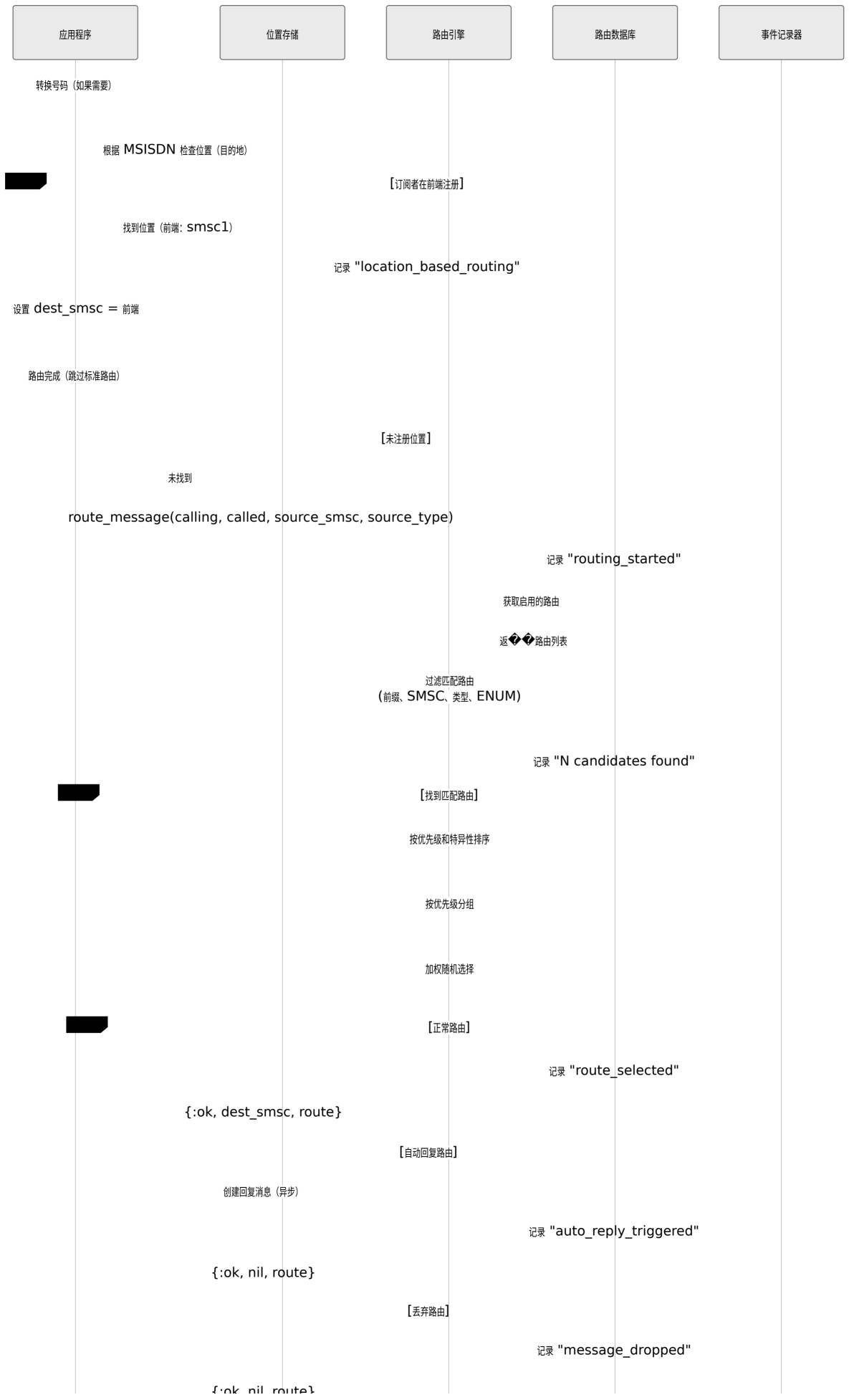
□□□□□

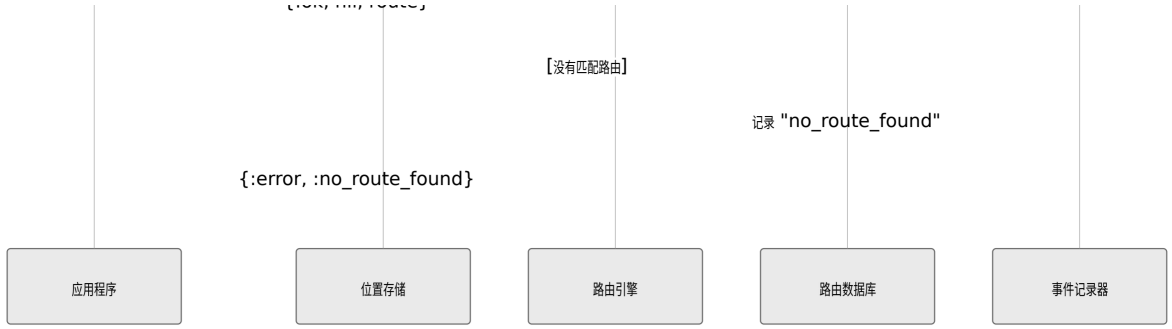

□□□□
□□
□□□□
□□□
□□
□□
calling_prefix
called_prefix
source_smsc
source_type
dest_smsc
auto_reply
drop
priority
weight
enabled
changed
yes
no
default

□□□□□□


□□□□
□□□□□
□□□□□□□
□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□□□

□□□□□□


应用程序
位置存储
路由引擎
路由数据库
事件记录器
转换号码（如果需要）
根据 MSISDN 检查位置（目的地）
[订阅者在前端注册]
找到位置（前端: smsc1）
记录 "location_based_routing"
设置 dest_smsc = 前端
路由完成（跳过标准路由）
[未注册位置]
未找到
route_message(calling, called, source_smsc, source_type)
记录 "routing_started"
获取启用的路由
返回路由列表
过滤匹配路由
(前缀、SMSC、类型、ENUM)
记录 "N candidates found"
[找到匹配路由]
按优先级和特异性排序
按优先级分组
加权随机选择
[正常路由]
记录 "route_selected"
{:ok, dest_smsc, route}
[自动回复路由]
创建回复消息（异步）
记录 "auto_reply_triggered"
{:ok, nil, route}
[丢弃路由]
记录 "message_dropped"
{:ok, nil, route}



常见用例

□于位置的路由（最高优先级）

将消息直接路由到服务注册订阅者的前端，绕过所有路由规则：

Parse error on line 4: ...-->|订阅者在前端注册
于 "ims-core-1"| REG[基于位 ----------^ Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND', 'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND', 'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'STR'

□试

工作原理：

1. 消息到达，带有目的地号码
2. 号码被转换（如果配置了）
3. 系统检查转换后的目的地 MSISDN 是否在位置表中
4. 如果注册，消息直接路由到服务该订阅者的前端
5. 标准路由规则完全跳过
6. 如果未注册，则应用正常路由规则

好处：

- 保证投递到正确的前端，针对注册的订阅者
- 最快的路由 - 不需要路由表评估
- 准确的路由 - 订阅者位置是事实来源
- 覆盖���有路由规则 - 确保订阅者可达性

用例：

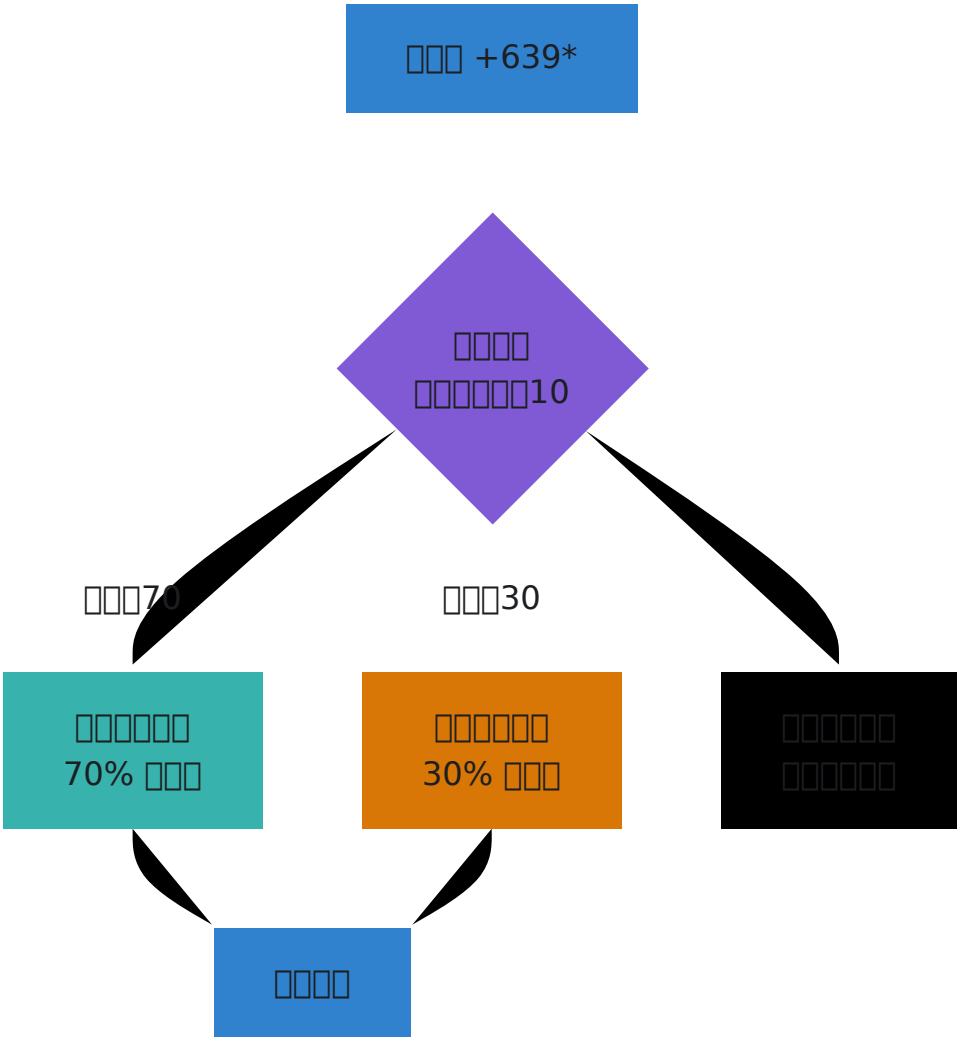
- □□□ IMS □□□□□ IMS/VoLTE □□□
- □□□□□ MSC □□□□□□
- □□□□□□□□□□□□ SIP □□□

□□□□

□□□□□□□□□□□□□□□ SMSC□

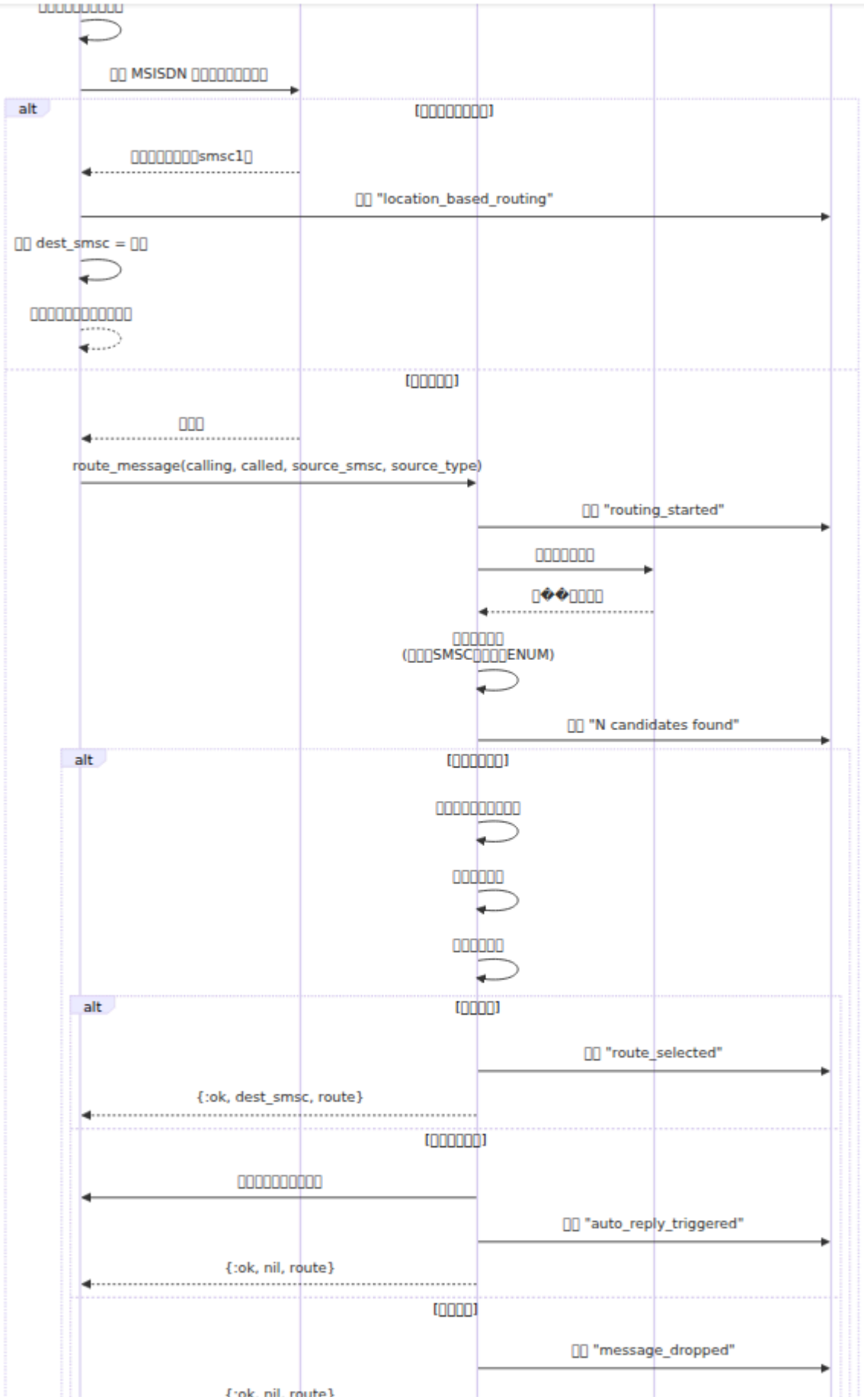
□□□□

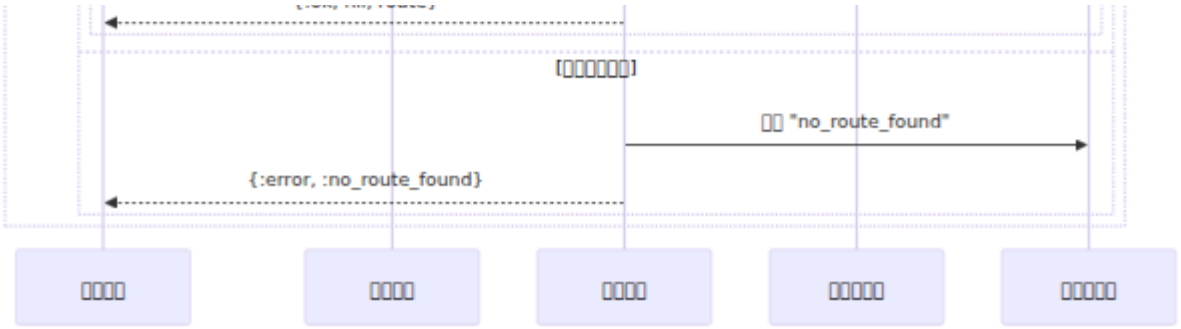
□□□□□□□ SMSC □□□□□□□



优质号码路由

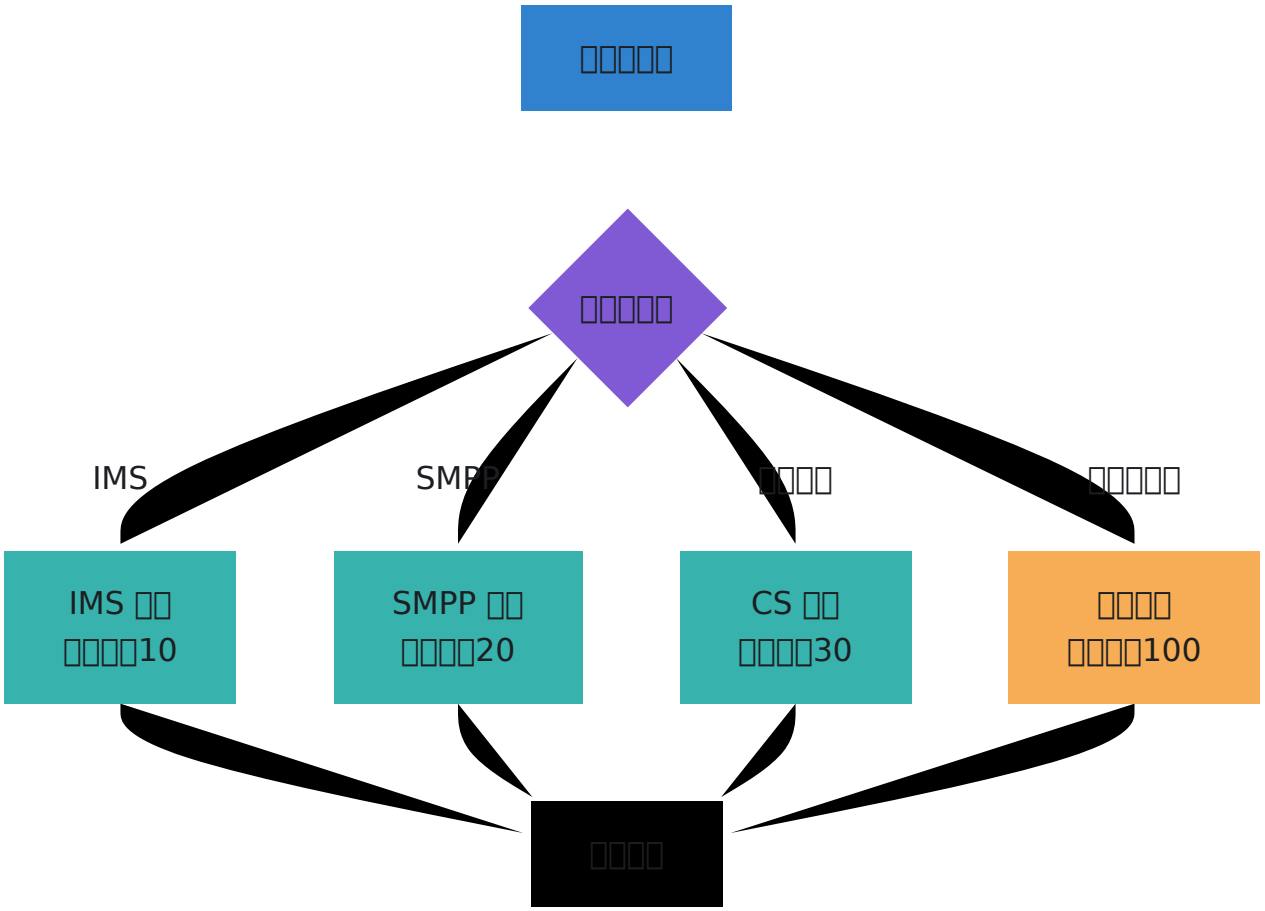
将优质号码路由到特殊处理，优先级更高：


□□□□□□□□□□
□□ MSISDN □□□□□□□□□□
alt
[□□□□□□□□□]
□□□□□□□□□smsc1□
□□ "location_based_routing"
□□ dest_smsc = □□
□□□□□□□□□□□□□□
[□□□□□□]
□□□□
route_message(calling, called, source_smsc, source_type)
□□ "routing_started"
□□□□□□□□
□□□□□□□
(□□□SMSC□□□□□ENUM)
□□ "N candidates found"
alt
[□□□□□□□]
□□□□□□□□□□□□
□□□□□□□
□□□□□□□
alt
[□□□□□]
□□ "route_selected"
{:ok, dest_smsc, route}
[□□□□□□□]
□□□□□□□□□□□□
□□ "auto_reply_triggered"
{:ok, nil, route}
[□□□□□]
□□ "message_dropped"
{:ok, nil, route}


[□□□□□□□]
□□ "no_route_found"
{:error, :no_route_found}
□□□□
□□□□
□□□□
□□□□□
□□□□□

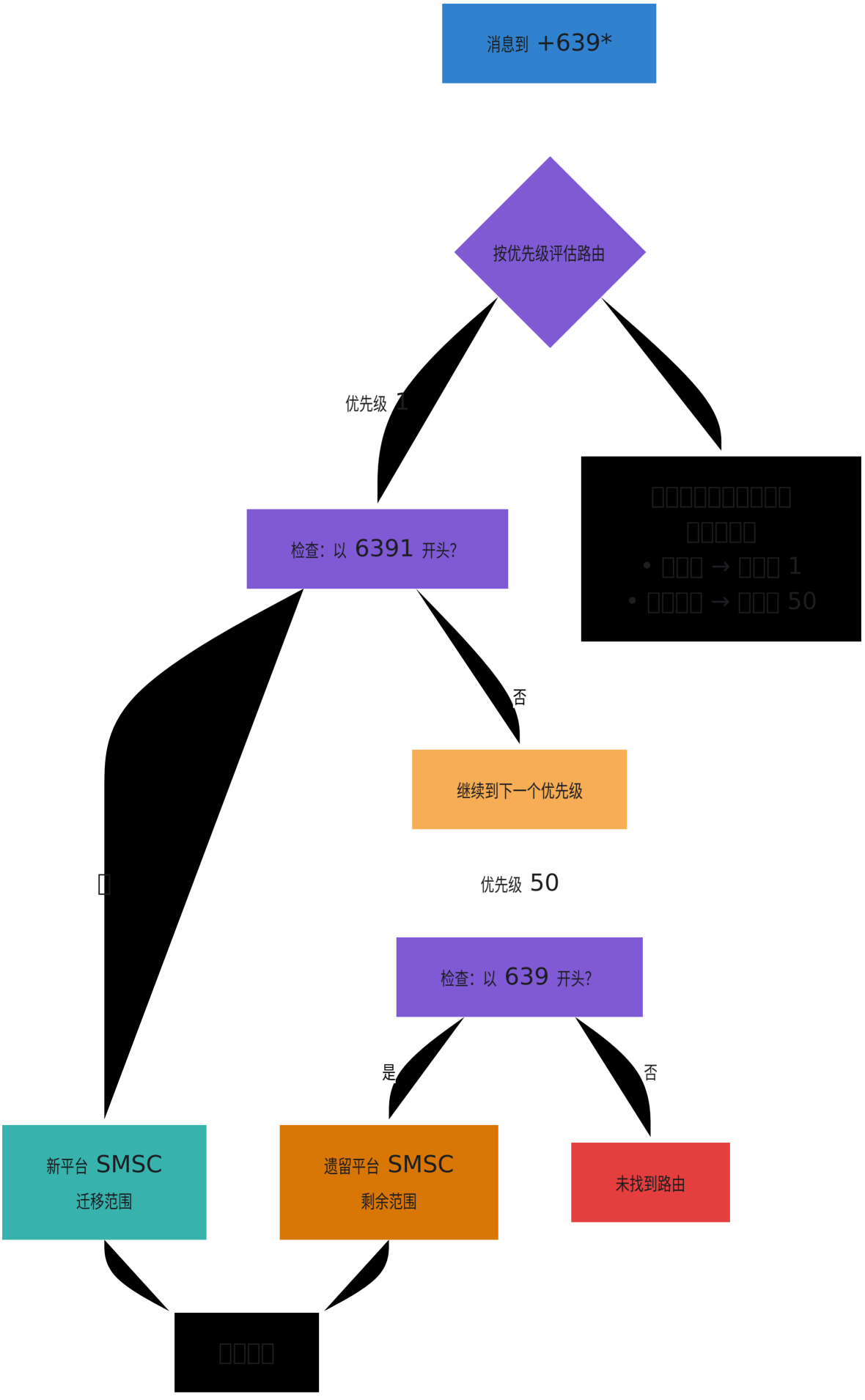
□□□□□□□

□□□□□□□□□□□□□


□□□□□
□□□□□
IMS
SMPP
□□□□
□□□□□
IMS □□
□□□□10
SMPP □□
□□□□20
CS □□
□□□□30
□□□□
□□□□100
□□□□

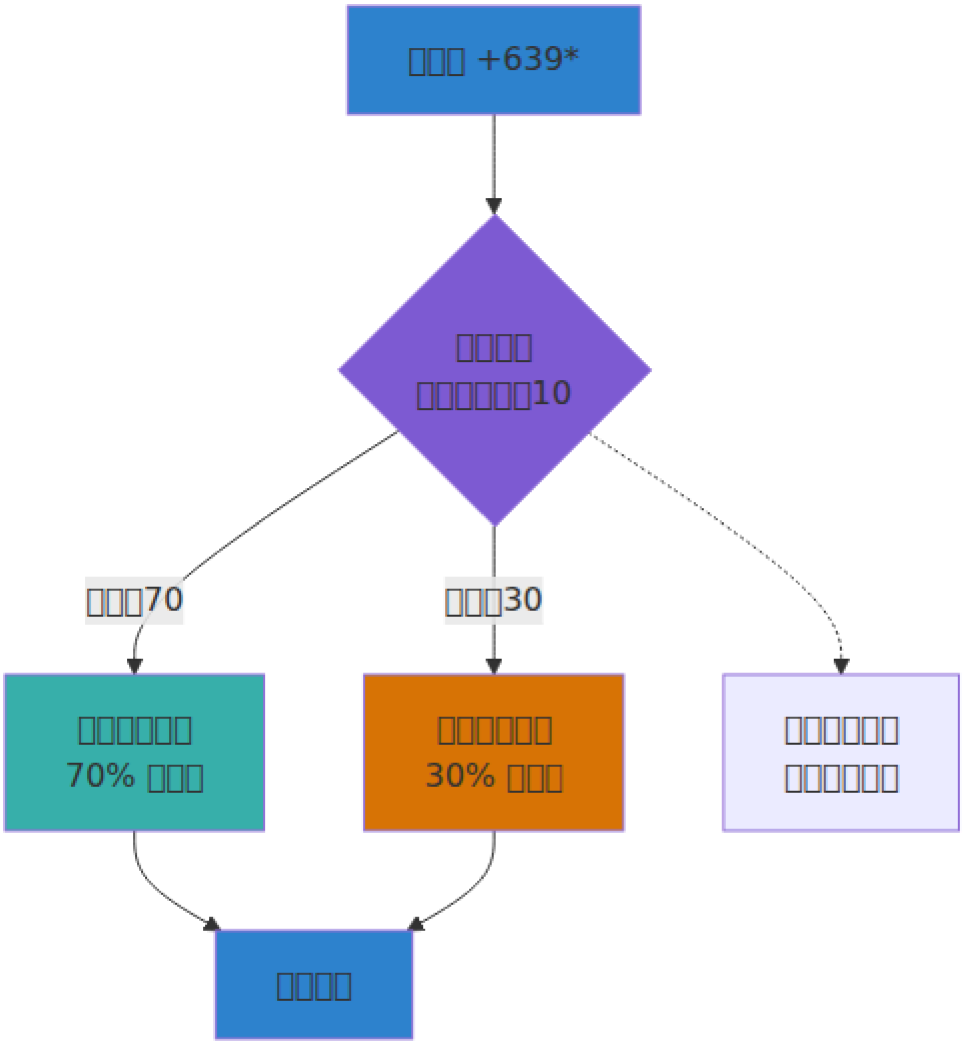
□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□


消息到 +639*
按优先级评估路由
优先级 1
检查：以 6391 开头?
□□□□□□□□□□
□□□□□
• □□□ → □□□ 1
• □□□□ → □□□ 50
否
继续到下一个优先级
优先级 50
检查：以 639 开头?
是
否
新平台 SMSC
迁移范围
遗留平台 SMSC
剩余范围
未找到路由
□□□□

复杂多标准路由

组合多个标准以实现精细控制：



Web 界面

路由管理 UI

访问路由管理界面位于 `/sms_routing`（在路由器中配置）：

功能：

- 在可排序的表格中查看所有路由
- 使用表单验证添加新路由
- 编辑现有路由

- □□/□□□□□□□□□□□□□□□□
- □□□□□□□□
- □□□□□□□5 □□□□□

□□□□□□

1. □□“□□□□□□”
2. □□□□□□□□□□□□□□□□ SMSC □□□□□□
3. □□□□□□□1-100□□□□ 100□□□□□□□□□1-255□□□□ 100□
4. □□“□□”□□□□□□
5. □□“□□□□”

□□□□□□

1. □□□□□□□□□“□□”
2. □□□□□□□□□□□
3. □□“□□□□”

□□□□□□

- □□“□□”□□□□□□□□□□□□
- □□“□□”□□□□□□

□□□□□

□□□□□□□□□ `/simulator`□□□□□□□□□□□

□□□

- □□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□
- □□/□□□□□□□□□□□□□
- □□□□□□□□□□□□□□
- □□□□□□□□□□□□ 10 □□□□□

□□□□□□□

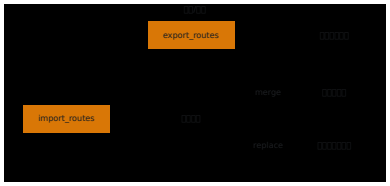
1. □□□□□□□□
    - □□□□□□□□□□
    - □□□□□□□□
    - □ SMSC□□□□
    - □□□□□□□/IMS/□□□□/SMPP□
2. □□“□□□□”
3. □□□□□□□□
    - □□□□□□□□□□□□□□□□□□□“□□□□□”□
    - □□□□□□□□□□□□□□□□□□□
        - ✓ □□□□ = ��□□□
        - ✗ □□ X = □□□□□
        - □□□□□□□/□□□□□□□
    - □□□□□□
        - □□□□ + “SELECTED” □□ = □□□□□□□
        - □□□□ + “MATCHED” □□ = □□□□□□□□□
        - □□□□ = □□□□□□
4. □□□□□□□□□□□□□□□
5. □□□□□□□□□□□□□□□

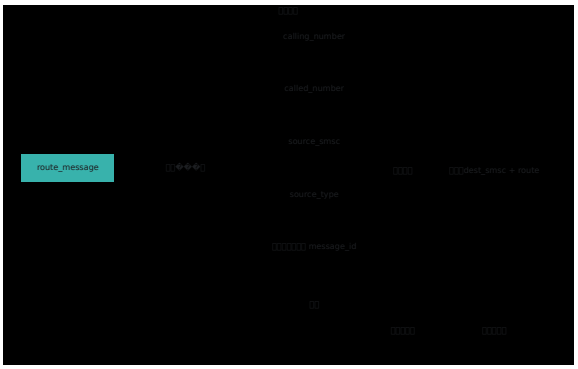
□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

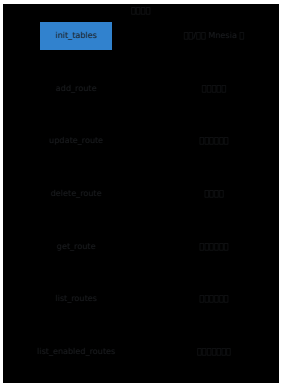
- □□□□□“□□□□ '1234'”□“□□ '44' □□”
- □□□□□“□□□□□□□□□□”□“□□ '639' □□”
- □ **SMSC**□“□□ 'smsc1'”□“□□ 'untrusted_smsc'□□□ 'none'”
- □□□□“□□□□□□□□□□”□“□□ 'smpp'□□□ 'IMS'”

API □□

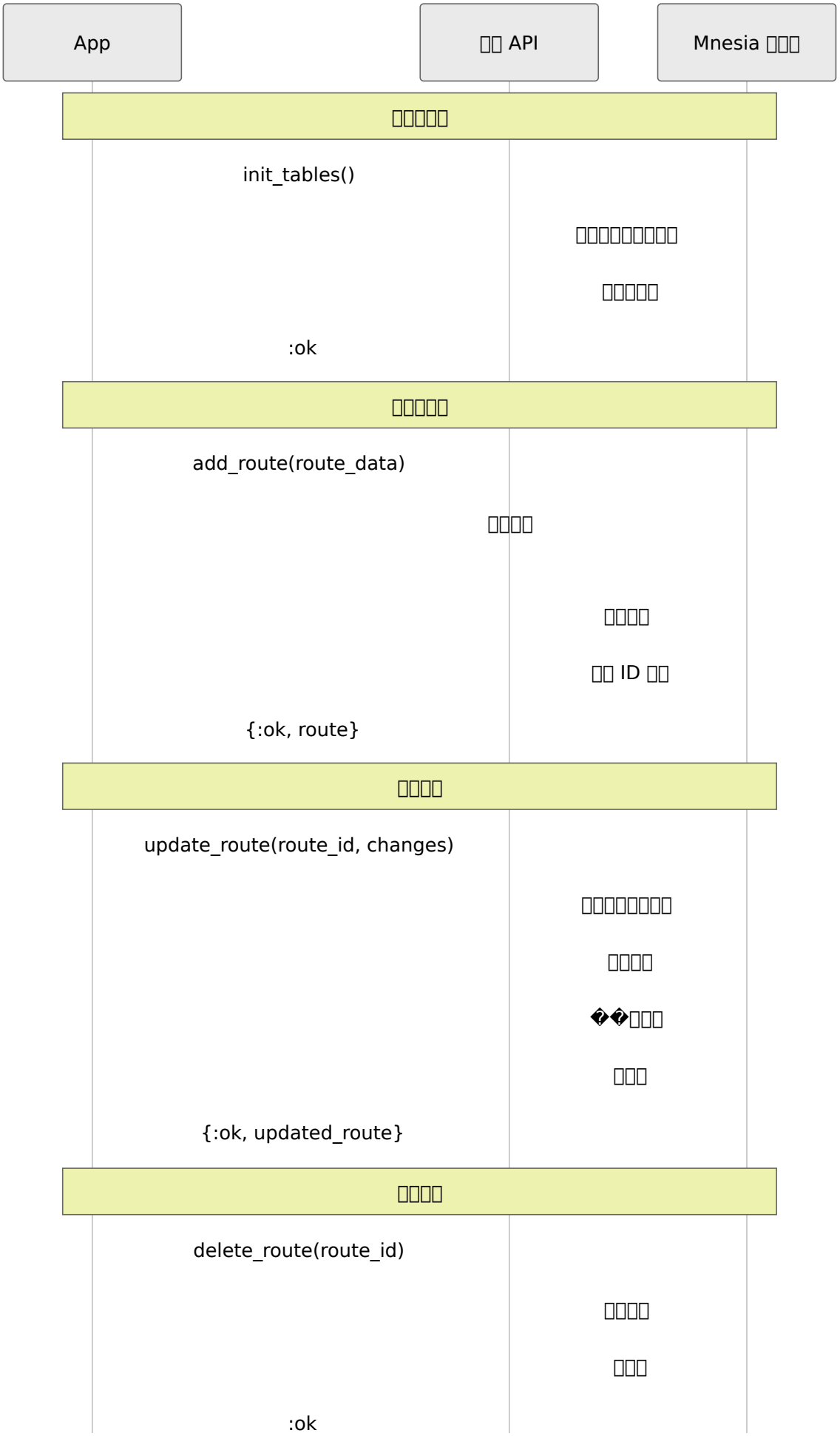
□□□□□□

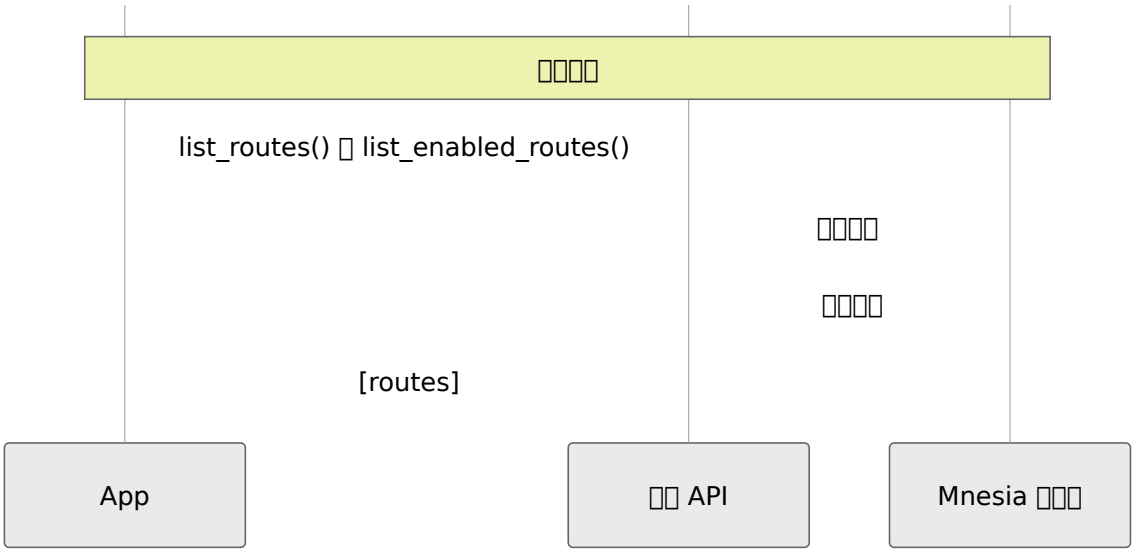

export_routes
merge
import_routes
replace


calling_number
called_number
source_smsc
route_message
source_type


init_tables
add_route
update_route
delete_route
get_route
list_routes
list_enabled_routes

□□□□□□


App
□□ API
Mnesia □□□
□□□□□□
init_tables()
□□□□□□□□□□□□
□□□□□□
:ok
□□□□□□
add_route(route_data)
□□□□□
□□□□□
□□ ID □□
{:ok, route}
□□□□□
update_route(route_id, changes)
□□□□□□□□□□
□□□□□
□□□□
□□□
{:ok, updated_route}
□□□□□
delete_route(route_id)
□□□□□
□□□
:ok



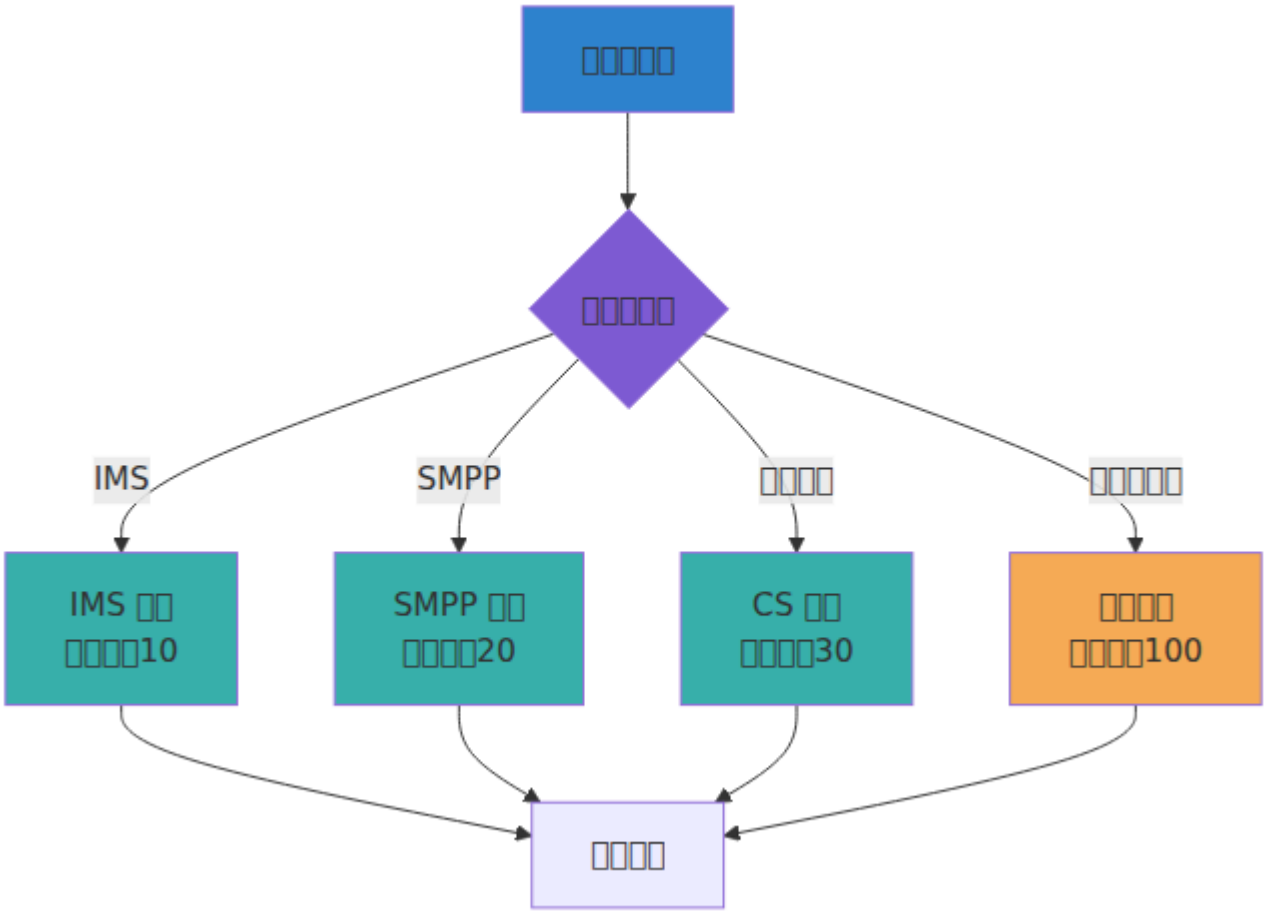
消息路由参数

route_message 接受以下参数:

- `calling_number`（可选）：发起电话号码
- `called_number`（可选）：目的电话号码
- `source_smsc`（可选）：源 SMSC 标识符
- `source_type`（可选）：连接类型（`:ims`、`:circuit_switched`、`:smpp`）
- `message_id`（可选）：用于事件日志记录

返回:

- `{:ok, dest_smsc, route}` - 找到并选择了路由
- `{:error, :no_route_found}` - 未找到匹配路由



导入/导出操作

最佳实践

路由设计

1. 明智地使用优先级：将低优先级（1-10）保留给关键路由
2. 保持简单：从广泛的路由开始，按需添加特定路由
3. 记录路由：始终为路由添加描述
4. 使用捕获所有：始终有一个低优先级的默认路由

□□

1. □□□□□□□□□□□□□□□□□□□□□□
2. □□□□□□□□□□□□□□□□□□□□□□□□
3. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

1. □□□□□□□□□□□□□□□□□□□□□□
2. □□□□□□□□□□□□□□□□□□□□□□
3. □□□□□□□□□□□□□□□□□□□□□□
4. □□□□□□□□□□□□□□□□□□□□□□□□

□□

1. □□□□□□□□□□□□□□□□□□□□□□
2. □□□□□□□□□□□□□□□□□□□□□□
3. □□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□

□□□□□□ `{:error, :no_route_found}`

□□□□□□

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

□□□□□□

1. □□□□□□□□□□□ `SmsRouting.list_enabled_routes()`

2. 使用模拟器测试实际消息参数的路由
3. 添加捕获所有路由以进行调试：`add_route(%{dest_smsc: "debug_smsc", priority: 255})`
4. 检查事件日志以获取路由评估详细信息

选择了错误的路由

症状：消息路由到意外的目的地

可能原因：

- 优先级配置错误
- 通配符路由具有更高的优先级
- 特异性计算偏向不同的路由
- 使用权重的多条路由具有相同标准

解决方案：

1. 使用模拟器查看所有匹配的路由
2. 检查优先级值（数字越小优先级越高）
3. 验证模拟器中的特异性分数
4. 审查负载均衡路由的权重分布

性能问题

症状：路由缓慢

可能原因：

- 数据库中的路由数量过多
- 复杂的路由模式
- Mnesia 表未正确索引

解决方案：

1. 合并相似路由
2. 删除不再需要的禁用路由

3. 确保创建 Mnesia 索引（在 `init_tables` 中自动）
4. 考虑缓存频繁使用的路由决策

高级主题

ENUM/NAPTR 集成

ENUM（E.164 号码映射）提供基于 DNS 的号码查找，使用 NAPTR 记录。SMS-C 包含完整的 ENUM 支持，具有缓存、可配置的 DNS 服务器和基于 ENUM 查找结果的路由匹配。

什么是 ENUM？

ENUM 将 E.164 电话号码映射到 DNS 名称，使用简单的转换：

- 电话号码：+1-212-555-1234
- **ENUM** 查询：4.3.2.1.5.5.5.2.1.2.1.e164.arpa
- **DNS** 记录类型：NAPTR（命名权威指针）
- 结果：SIP URI、路由信息或其他服务数据

配置

ENUM 功能在 `config/runtime.exs` 中配置：

启用 **ENUM** 查找：

设置 `enum_enabled: true` 以在路由之前启用 ENUM 查找。启用后，系统将对传入消息执行 DNS ENUM 查找，并在路由决策中使用结果。

ENUM 域：

按优先级顺序列出要查询的 ENUM 域。系统将尝试每个域，直到成功查找为止。

常见的 ENUM 域：

- `e164.arpa` - 官方 IETF ENUM 域
- `e164.org` - 替代 ENUM 注册
- 自定义私有 ENUM 域

DNS 服务器:

为 ENUM 查询配置特定的 DNS 服务器。格式: `{ip_address, port}`

留空或设置为 `[]` 以使用系统默认 DNS 服务器。

示例自定义 DNS 配置:

- Google 公共 DNS: `{"8.8.8.8", 53}`, `{"8.8.4.4", 53}`
- Cloudflare DNS: `{"1.1.1.1", 53}`, `{"1.0.0.1", 53}`
- 自定义 ENUM DNS: `{"10.0.0.53", 53}`

超时:

设置 DNS 查询超时（默认: 5000ms）。对于慢速网络增加，对于快速故障转移减少。

ENUM 查找的工作原理

Parse error on line 37: ... style Router fill:#3182CE style C ----------------------^ Expecting 'SOLID_OPEN_ARROW', 'DOTTED_OPEN_ARROW', 'SOLID_ARROW', 'BIDIRECTIONAL_SOLID_ARROW', 'DOTTED_ARROW', 'BIDIRECTIONAL_DOTTED_ARROW', 'SOLID_CROSS', 'DOTTED_CROSS', 'SOLID_POINT', 'DOTTED_POINT', got 'TXT'

□试

ENUM 缓存

系统缓存 ENUM 查找结果 15 分钟，以提高性能并减少 DNS 负载。

缓存好处:

- 减少 DNS 查询负载
- 提高路由延迟
- 保护免受 DNS 服务器故障（缓存结果保持可用）

缓存统计:

- 在 NAPTR 测试页面查看缓存大小和状态

- 通过 Prometheus 指标监控缓存命中/未命中率
- 如有需要手动清除缓存（配置更改、测试等）

缓存行为：

- 成功和失败的查找都被缓存
- 失败的查找被缓存，以避免对无效号码的重复查询
- 缓存在 15 分钟后自动过期
- 缓存在应用程序重启后仍然存在（存储在 ETS 中）

在路由中使用 ENUM

路由可以使用 `enum_result_domain` 字段与 ENUM 查找结果匹配：

示例场景：

ENUM 查找 +1-555-0100 返回 NAPTR 记录：

- 服务：E2U+sip
- 替换：sip:customer@voip-carrier.com
- 结果域：voip-carrier.com

路由配置：

创建一个路由，`enum_result_domain: "voip-carrier.com"`，以匹配 ENUM 查找返回该域的消息。

匹配逻辑：

- 如果路由具有 `enum_result_domain: nil` - 匹配所有消息（通配符）
- 如果路由具有 `enum_result_domain: "specific.com"` - 仅在 ENUM 返回该域时匹配
- 具有匹配 ENUM 域的路由获得更高的特异性分数

优先级计算：

具有 ENUM 结果域的路由获得 +15 的特异性分数，使其优先于通用路由。

测试 ENUM 查找

访问 NAPTR 测试页面位于 `/naptr_test`（通过导航菜单）。

□□□

- □□□□□ DNS □□□□□□□ ENUM □□
- □□□□□ NAPTR □□□□
- □□□ NAPTR □□□□□□□□□
- □□□□□□
- □□□□□□□□□

□□□□□□□

1. □□□□□□□□□□□ + □□□
2. □□ ENUM □□□□□e164.arpa□
3. □□“□□□□”
4. □□□□□
    - □□□ NAPTR □□
    - □□□□□□□
    - □□□□□E2U+sip□E2U+tel □□
    - □□□□□
    - □□□
    - □□□□□□□□□□□□□□□

□□□□□□□

- □□□□□ DNS □□□□□“□□□□”□
- □□□□
- □□□□□□□
- □□□□□□

□□□□

□□ NAPTR □□□□□

- □□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□u=□□□s=□□□
- □□□□□□□□□□E2U+sip□E2U+tel □□

- 正则表达式：替代表达式
- 替换：替代域或地址
- 结果域：提取的域，用于路由匹配

常见 ENUM 用例

1. VoIP 对等

使用 ENUM 识别托管在 SIP/VoIP 网络上的号码，并直接路由到 VoIP 网关：

- ENUM 返回 SIP URI：sip:number@voip-carrier.com
- 结果域：voip-carrier.com
- 路由与 `enum_result_domain: "voip-carrier.com"` 被选择
- 流量发送到直接 VoIP 对等网关

2. 运营商识别

识别服务某个号码的运营商并相应地路由：

- ENUM 返回运营商信息
- 结果域：carrier-a.com
- 路由到运营商 A 的互联
- 优化路由成本和质量

3. 号码可携带性

处理在运营商之间迁移的已携带号码：

- ENUM 查找返回当前运营商
- 自动路由到正确的目的地
- 无需手动更新路由表

4. 最低成本路由

将 ENUM 与多个路由结合：

- ENUM 识别目的地网络
- 针对同一域的多个路由，具有不同成本
- 使用优先级和权重来优先选择低成本路由

5. 紧急服务

将紧急号码（911、112 等）路由到适当的紧急服务：

- ENUM 查找识别本地紧急网关
- 高优先级路由确保立即路由
- 无需正常路由评估的延迟

ENUM 路由策略

推荐配置：

1. 高优先级 **ENUM** 路由（优先级 **1-10**）
    - 匹配特定 ENUM 结果域的路由
    - 用于直接对等、VoIP 路由
    - 最高特异性，首先选择
2. 中优先级前缀路由（优先级 **50-100**）
    - 标准基于前缀的路由
    - 在 ENUM 查找失败或返回无记录时使用
    - 可靠的后备
3. 低优先级捕获所有（优先级 **200+**）
    - 默认路由，用于其他所有情况
    - 确保没有消息未路由

示例路由层次结构：

- 优先级 1：`enum_result_domain: "sip.carrier.com"` → 直接 VoIP 网关
- 优先级 10：`enum_result_domain: "tel.carrier.com"` → 运营商的 PSTN 网关
- 优先级 50：`called_prefix: "+1"` → 北美默认网关
- 优先级 100：`called_prefix: "+"` → 国际默认网关
- 优先级 200：无标准 → 最终后备

性能考虑

DNS 查询延迟:

ENUM 查找为路由添加 DNS 查询时间:

- 缓存: < 1ms（快速）
- 未缓存: 10-100ms（取决于 DNS 服务器）

建议:

- 使用地理位置接近的 DNS 服务器
- 配置适当的超时（默认 5000ms）
- 监控缓存命中率（目标 > 80%）
- 考虑为已知号码预热缓存

可扩展性:

缓存系统处理高流量场景:

- 缓存在所有进程之间共享
- 读取并发的 ETS 表以提高性能
- 通过 TTL 自动清理缓存
- 可扩展到数百万个缓存条目

故障处理:

ENUM 失败优雅地回退到常规路由:

- DNS 超时 → 转到下一个路由
- 无 NAPTR 记录 → 使用基于前缀的路由
- NAPTR 格式无效 → 记录错误，继续路由
- DNS 服务器不可用 → 使用缓存结果或后备

监控 ENUM 操作

使用 Prometheus 指标监控 ENUM 性能:

- `sms_c_enum_lookup_stop_duration` - 查找延迟
- `sms_c_enum_cache_hit_count` - 缓存命中
- `sms_c_enum_cache_miss_count` - 缓存未命中

- `sms_c_enum_cache_size_size` - 当前缓存大小
- `sms_c_enum_naptr_records_record_count` - 每次查找的 NAPTR 记录

关键指标：

- 缓存命中率：应在预热后 > 70%
- 查找持续时间 **p95**：应 < 1000ms
- 失败的查找：监控 DNS 问题

请参见 `docs/METRICS.md` 获取完整的指标文档。

故障排除 **ENUM**

问题：未找到 **NAPTR** 记录

- 验证 ENUM 域配置
- 测试 DNS 服务器连接性
- 检查号码是否实际在 ENUM 注册中
- 尝试替代 ENUM 域（例如，e164.org）
- 使用 NAPTR 测试页面进行诊断

问题：**ENUM** 查找缓慢

- 检查 DNS 服务器延迟
- 验证网络连接
- 如有需要增加超时
- 考虑使用更接近的 DNS 服务器
- 检查缓存命中率

问题：**ENUM** 后选择了错误的路由

- 验证路由中的 `enum_result_domain` 字段
- 使用路由模拟器测试路由逻辑
- 检查结果域提取是否正确
- 查看测试页面中的 NAPTR 记录格式

问题：**ENUM** 查找被禁用

- 验证 `enum_enabled: true` 在 config/runtime.exs 中
- 检查 `enum_domains` 列表是否为空
- 配置更改后重启应用程序
- 检查应用程序日志以获取 ENUM 初始化信息

安全考虑

DNS 缓存中毒:

- 仅使用受信任的 DNS 服务器
- 如果可用，考虑使用 DNSSEC
- 验证 NAPTR 记录格式
- 监控意外结果域

资源耗尽:

- 缓存限制防止内存耗尽
- 超时防止在慢 DNS 上挂起
- 失败的查找被缓存以防止重试风暴

信息泄露:

- ENUM 查找向 DNS 服务器揭示路由意图
- 对于敏感路由，使用私有 DNS 服务器
- 考虑 VPN/加密 DNS 以保护隐私

事件日志记录

路由决策通过 EventLogger 记录:

- `sms_routing_started`: 路由评估开始
- `sms_routing_candidates`: 找到的启用路由数量
- `sms_routing_matches`: 匹配的路由数量
- `sms_routing_selected`: 选择的路由详细信息
- `sms_routing_failed`: 未找到路由

通过将 `message_id` 传递给 `route_message/1` 启用日志记录。

集群

Mnesia 表自动分布在集群节点之间。路由被复制以确保高可用性。

Parse error on line 25: ... style New fill:#3182CE style P ----------------------^ Expecting 'SOLID_OPEN_ARROW', 'DOTTED_OPEN_ARROW', 'SOLID_ARROW', 'BIDIRECTIONAL_SOLID_ARROW', 'DOTTED_ARROW', 'BIDIRECTIONAL_DOTTED_ARROW', 'SOLID_CROSS', 'DOTTED_CROSS', 'SOLID_POINT', 'DOTTED_POINT', got 'TXT'

试

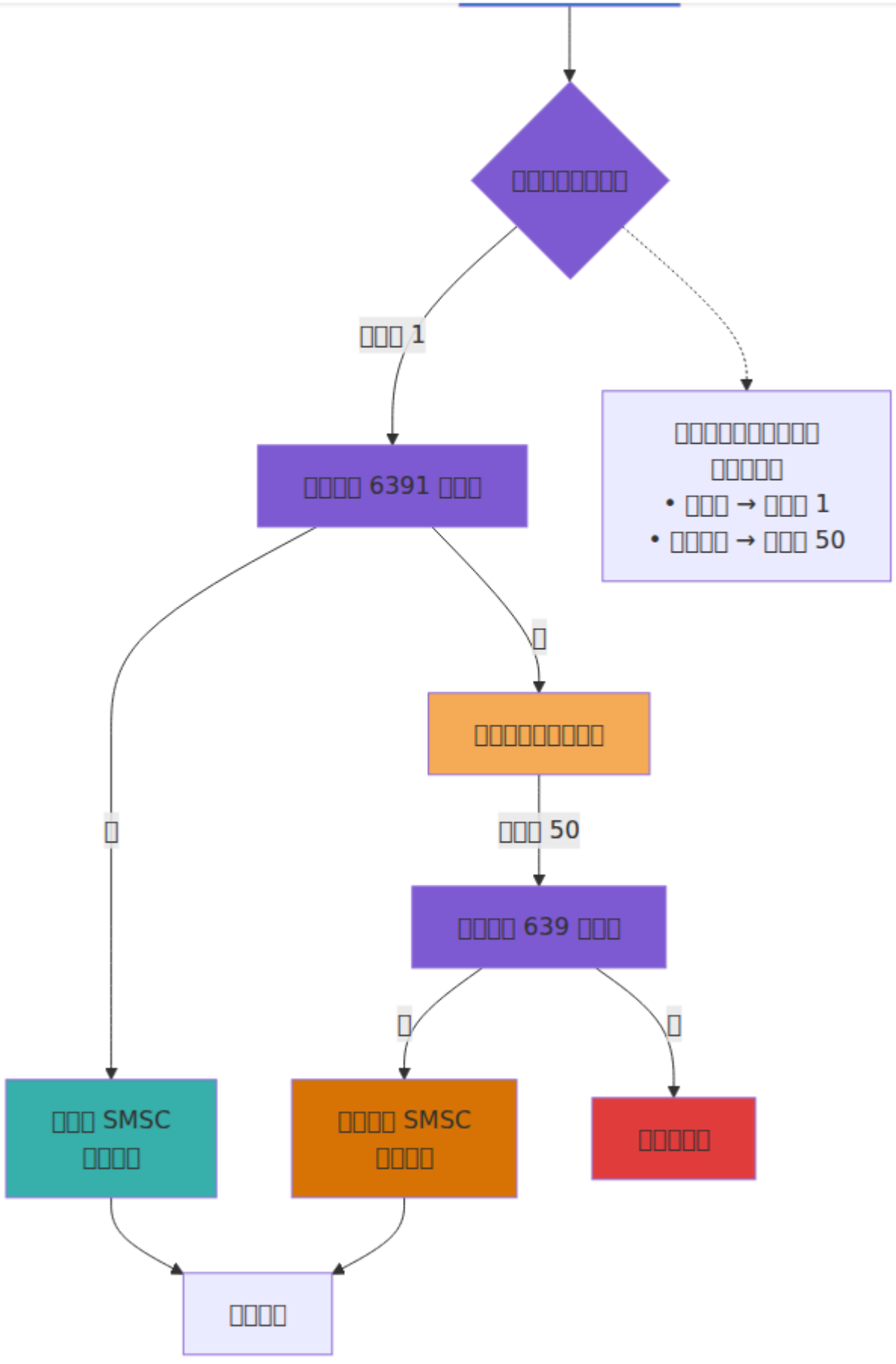
示例

请参见测试套件 `test/sms_c/messaging/sms_routing_test.exs`，获取全面的示例，包括：

- 前缀匹配
- 基于优先级的路由
- 基于权重的负载均衡
- 多标准路由
- 边缘情况

从旧路由迁移

如果从旧的基于配置的路由迁移，请遵循以下流程：


□□□□□□□□□□
□□□ 1
□□□□ 6391 □□□
□□□□□□□□□□□□
□□□□□□
• □□□ → □□□ 1
• □□□□ → □□□ 50
□
□□□□□□□□□□□
□
□□□ 50
□□□□ 639 □□□
□
□
□□□ SMSC
□□□□
□□□□ SMSC
□□□□
□□□□□
□□□□

迁移步骤详细信息

1. 初始化表
    - 创建 Mnesia 路由表
    - 准备系统以进行新路由
2. 分析旧路由
    - 正则模式 → 基于前缀的路由
    - 罐装响应 → 自动回复路由
    - 自定义逻辑 → 多标准路由
3. 彻底测试
    - 使用路由模拟器
    - 验证所有场景
    - 检查边缘情况
4. 更新代码
    - 替换旧的路由调用
    - 使用 `route_message/1` API
    - 更新错误处理
5. 部署与监控
    - 部署新路由系统
    - 监控问题
    - 初始时保留旧配置作为备份
6. 清理
    - 删除旧路由配置
    - 删除迁移代码
    - 更新文档

支持

如有问题或疑问：

- 检查测试套件以获取示例
- 使用模拟器调试路由逻辑
- 查看事件日志以获取路由决策
- 检查 Mnesia 表内容：`:mnesia.table_info(:sms_route, :size)`

SMS-C 故障排除指南

← 返回文档索引 | 主 **README**

全面指南，用于诊断和解决常见的 SMS-C 问题。

目录

诊断工具

快速健康检查

```
# 1. 检查 API 状态
curl https://api.example.com:8443/api/status

# 2. 检查 Prometheus 指标端点
curl https://api.example.com:9568/metrics | grep sms_c

# 3. 检查应用程序日志
tail -f /var/log/sms_c/application.log

# 4. 检查进程状态
systemctl status sms_c

# 5. 检查 SQL CDR 数据库连接性 (MySQL/MariaDB)
mysql -u sms_user -p -h db.example.com -e "SELECT 1"

# 对于 PostgreSQL:
# psql -U sms_user -h db.example.com -d sms_c_prod -c "SELECT 1"
```

日志分析

查看最近错误:

```
# 最近 100 条错误级别日志条目
tail -1000 /var/log/sms_c/application.log | grep "\[error\]"

# 搜索特定错误模式
grep "routing_failed" /var/log/sms_c/application.log

# 查找 SQL 数据库错误
grep -i "database\|sql\|ecto" /var/log/sms_c/application.log |
grep error
```

实时监控日志:

```
# 带过滤器跟踪日志
tail -f /var/log/sms_c/application.log | grep -E "
(error|warning|critical)"
```

指标查询

检查消息处理速率:

```
# 每秒□息数
rate(sms_c_message_received_count[5m])

# 投递成功率
rate(sms_c_delivery_succeeded_count[5m]) /
rate(sms_c_delivery_queued_count[5m])
```

检查队列状态:

```
# 当前队列深度
sms_c_queue_size_pending

# 最旧□息的年龄（秒）
sms_c_queue_oldest_message_age_seconds
```

检查系统性能:

```
# □息处理延迟 (p95)
histogram_quantile(0.95,
sms_c_message_processing_stop_duration_bucket)

# 路由延迟 (p95)
histogram_quantile(0.95, sms_c_routing_stop_duration_bucket)
```

□□□□□□

□□□□□□

□□□

- □□□□“□□□”□□
- □□□□□□□□□□
- □□□□□□

□□□□□

1. □□□□□□□□

```
curl https://api.example.com:8443/api/frontends/active
```

□□□□□□□□□□
□□□□□□□□□□□□□□□□□

2. □□□□□□□□

□□ Web UI: `/message_queue`

- □□□□□□□“□□□”
- □□ `dest_smsc` □
- □□ `deliver_after` □□□□

3. □□□□□□

□□ Web UI: `/simulator`

- □□□□□□□□□□□□□□
- □□□□□□□□□□□□□□

4. □□□□□□□□

□□□□□□□□□□

- □□□□□□□□ `/api/messages`□

- 前端是否正确发送 `smsc` 头?

解决方案:

没有前端连接:

```
# 检查前端系统状态
systemctl status frontend_service

# 验证前端是否可以访问 API
curl -k https://api.example.com:8443/api/status

# 手动注册前端
curl -X POST https://api.example.com:8443/api/frontends/register \
  -H "Content-Type: application/json" \
  -d '{
    "frontend_name": "test_gateway",
    "frontend_type": "smpp",
    "ip_address": "10.0.1.50"
  }'
```

消息路由到错误的 **SMSC**:

- 查看路由配置
- 检查路由优先级
- 在路由模拟器中测试
- 验证前端名称与消息中的 `dest_smsc` 匹配

消息计划在未来:

- 检查 `deliver_after` 时间戳
- 如有必要重置:

```
curl -X PATCH https://api.example.com:8443/api/messages/12345 \
  -H "Content-Type: application/json" \
  -d '{"deliver_after": "2025-10-30T12:00:00Z"}'
```

消息重试失败

症状:

- `delivery_attempts` 计数器增加
- 尝试计数高的消息（> 3）
- 指数退避延迟

诊断步骤:

1. 检查事件日志:

```
curl https://api.example.com:8443/api/events/12345
```

查找:

- 投递失败事件
- 错误描述
- 重试时间戳

2. 检查前端日志:

- 前端为什么无法投递?
- 网络错误?
- 协议错误?
- 下游系统不可用?

解决方案:

临时网络问题:

- 等待重试（自动）
- 监控成功投递

持续失败:

```
# □□□□□□□□
curl -X PATCH https://api.example.com:8443/api/messages/12345 \
  -H "Content-Type: application/json" \
  -d '{"dest_smsc": "backup_gateway"}'

# □□□□□□□□
curl -X PATCH https://api.example.com:8443/api/messages/12345 \
  -H "Content-Type: application/json" \
  -d '{"delivery_attempts": 0, "deliver_after": "2025-10-
30T12:00:00Z"}'
```

□□□□□□□□

- □□□□□□
- □□□□□□□□
- □□□□□□□□□□□

□���□□

□□□

- □□□□ `deadletter: true`
- □□□□□□□□
- □□□□“□□□”

□□□□□

1. □□□□□□□

□□ Web UI: `/message_queue`

- □□□□□□□
- □□□□□□□

2. □□□□□□□

- □□□□□□
- □□□□□□□□
- □□□□□□□□

解决方案：

延长过期时间：

```
# 将过期时间延长 24 小时
curl -X PATCH https://api.example.com:8443/api/messages/12345 \
  -H "Content-Type: application/json" \
  -d '{"expires": "2025-10-31T12:00:00Z", "deadletter": false}'
```

路由问题

找不到路由

症状：

- 错误：`no_route_found`
- `sms_c_routing_failed_count` 指标增加
- 事件日志显示“routing_failed”

诊断步骤：

1. 检查路由是否存在：

访问 Web UI：`/sms_routing`

- 验证路由是否配置
- 检查至少一个路由是否启用

2. 测试路由：

访问 Web UI：`/simulator`

- 输入消息参数（呼叫号码，被叫号码，源 SMSC）
- 查看评估结果
- 检查为什么路由没有匹配

3. 检查路由标准：

- 前缀匹配是否必要?
- 源 SMSC 过滤是否过于严格?
- 所有路由是否禁用?

解决方案:

没有配置路由:

添加通用路由:

```
呼叫前缀: (空)
被叫前缀: (空)
源 SMSC: (空)
目标 SMSC: default_gateway
优先级: 255
权重: 100
启用: ✓
描述: 通用默认路由
```

路由过于具体:

添加更广泛的路由:

```
被叫前缀: +
目标 SMSC: international_gateway
优先级: 200
权重: 100
启用: ✓
描述: 国际通用路由
```

所有路由都被禁用:

- 通过 Web UI 启用适当的路由
- 检查配置是否意外禁用路由

选择错误的路由

症状:

- 消息路由到意外的目的地
- 错误的网关接收流量
- 负载均衡未按预期分配

诊断步骤：

1. 使用路由模拟器：

访问 Web UI: `/simulator`

- 使用实际消息参数进行测试
- 查看“所有匹配”部分
- 检查优先级和特异性分数

2. 检查路由优先级：

- 数字越低 = 优先级越高
- 路由按优先级顺序评估
- 在同一优先级内，权重适用

3. 检查路由特异性：

特异性评分：

- 更长的被叫前缀：每个字符 +100 分
- 更长的呼叫前缀：每个字符 +50 分
- 指定源 SMSC：+25 分
- 指定源类型：+10 分
- 指定 ENUM 域：+15 分

解决方案：

调整优先级：

使特定路由优先级更高：

```
优质路由：
   被叫前缀： +1555
   优先级： 10（���优先级）

一般路由：
   被叫前缀： +1
   优先级： 50（低优先级）
```

调整权重：

更改负载均衡分配：

```
主路由 (70%)：
   权重： 70

备用路由 (30%)：
   权重： 30
```

添加更具体的路由：

为特定情况覆盖一般路由：

```
特定路由：
   被叫前缀： +15551234
   目标 SMSC： dedicated_gateway
   优先级： 1

一般路由：
   被叫前缀： +1
   目标 SMSC： general_gateway
   优先级： 50
```

自动回复不工作

症状：

- 配置了自动回复路由但未触发
- 没有回复消息被发送

- 事件日志缺少自动回复事件

诊断步骤：

1. 检查路由配置：

- `auto_reply: true`
- `auto_reply_message` 包含文本
- 路由已启用
- 路由匹配消息标准

2. 在模拟器中测试：

- 验证路由是否被选择
- 检查是否有“auto_reply”指示

3. 检查事件日志：

```
curl https://api.example.com:8443/api/events/12345 | grep
auto_reply
```

解决方案：

路由不匹配：

- 扩大标准（移除过滤器）
- 检查优先级（应高于正常路由）
- 验证启用状态

消息未设置：

编辑路由，添加消息：

```
自动回复： ✓
自动回复□息： "感谢您的□息。我们将尽快回复。"
```

优先级错误���

自动回复路由应具有高优先级（低数字）：

```
自动回复路由:
  优先级: 10

正常路由:
  优先级: 50
```

性能问题

高消息处理延迟

症状:

- `sms_c_message_processing_stop_duration` p95 > 1000ms
- API 响应缓慢
- 队列积压

诊断步骤:

1. 检查组件延迟:

```
# 路由延迟
histogram_quantile(0.95, sms_c_routing_stop_duration_bucket)

# ENUM 查找延迟
histogram_quantile(0.95, sms_c_enum_lookup_stop_duration_bucket)

# 收费延迟
histogram_quantile(0.95, sms_c_charging_succeeded_duration_bucket)

# 投递延迟
histogram_quantile(0.95, sms_c_delivery_succeeded_duration_bucket)
```

2. 检查系统资源:

```
# CPU 使用率
top -b -n 1 | grep sms_c

# 内存使用率
ps aux | grep beam.smp
```

解决方案:

路由缓慢（许多路由）:

- 减少启用的路由数量
- 合并相似路由
- 优化路由标准

ENUM 查找缓慢:

- 检查 DNS 服务器延迟
- 增加超时
- 使用更快/更近的 DNS 服务器
- 如果不需要，禁用 ENUM

收费缓慢:

- 检查 OCS 性能
- 增加 OCS 超时
- 如果不需要，禁用收费
- 使用异步收费

数据库缓慢:

- 增加连接池大小
- 添加索引
- 优化查询
- 升级数据库资源

配置更改:

```
# config/config.exs
# 增加批处理大小以提高吞吐量
config :sms_c,
  batch_insert_batch_size: 200,
  batch_insert_flush_interval_ms: 200

# 增加数据库池
config :sms_c, SmsC.Repo,
  pool_size: 50
```

低消息吞吐量

症状:

- 处理 < 100 msg/sec
- 使用异步 API 但仍然缓慢
- 高 API 响应时间

诊断步骤:

1. 检查批处理工作者:

```
# 在生产控制台 (iex)
SmsC.Messaging.BatchInsertWorker.stats()
```

查找:

- `current_queue_size` 接近最大值
- `flush_errors` > 0
- `last_flush_duration_ms` 非常高

2. 检查瓶颈:

```
# 数据库查询时间
ecto_pools_query_time

# 连接池排队时间
ecto_pools_queue_time
```

解决方案：

数据库瓶颈：

增加池大小：

```
config :sms_c, SmsC.Repo,
  pool_size: 50  # 从 20 增加
```

批处理配置：

调整以提高吞吐量：

```
config :sms_c,
  batch_insert_batch_size: 200,  # 更大的批次
  batch_insert_flush_interval_ms: 200  # 更长的间隔
```

使用异步端点：

```
# 高吞吐量：使用 /create_async
curl -X POST
https://api.example.com:8443/api/messages/create_async

# NOT: /api/messages (同步)
```

队列积压增长

症状：

- `sms_c_queue_size_pending` 增加
- 最旧消息的年龄增加

- 处理无法跟上到达速率

诊断步骤：

1. 检查到达速率与投递速率：

```
# 到达速率
rate(sms_c_message_received_count[5m])

# 投递速率
rate(sms_c_delivery_succeeded_count[5m])
```

2. 检查前端容量：

- 前端轮询是否足够频繁？
- 前端处理消息是否足够快？
- 有前端错误吗？

3. 检查投递成功率：

```
rate(sms_c_delivery_succeeded_count[5m]) /
rate(sms_c_delivery_attempted_count[5m])
```

解决方案：

前端未轮询：

- 检查前端连接性
- 验证轮询间隔（应为 5-10 秒）
- 重启前端服务

前端太慢：

- 添加更多前端实例
- 优化前端处理
- 增加前端并发性

高重试率：

- 调查投递失败
- 修复下游问题
- 路由到备用网关

临时峰值：

- 等待队列排空
- 监控直到正常
- 如果重复考虑容量升级

数据库问题

连接失败

症状：

- 错误：“无法连接到数据库”
- API 返回 500 错误
- 应用程序无法启动

诊断步骤：

1. 检查 **SQL CDR** 数据库状态：

```
# MySQL/MariaDB
systemctl status mysql

# PostgreSQL
systemctl status postgresql

# 测试连接性 (MySQL/MariaDB)
mysql -u sms_user -p -h db.example.com -e "SELECT 1"

# 测试连接性 (PostgreSQL)
psql -U sms_user -h db.example.com -d sms_c_prod -c "SELECT 1"
```

2. 检查网络：

```
# Ping 数据库主机
ping db.example.com

# 检查端口连接性 (MySQL/MariaDB: 3306, PostgreSQL: 5432)
telnet db.example.com 3306
# 或
telnet db.example.com 5432
```

3. 检查凭证:

```
# 验证环境变量
echo $DB_USERNAME
echo $DB_HOSTNAME
echo $DB_PORT

# 尝试使用相同凭证手动连接 (MySQL/MariaDB)
mysql -u $DB_USERNAME -p$DB_PASSWORD -h $DB_HOSTNAME

# 对于 PostgreSQL:
# psql -U $DB_USERNAME -h $DB_HOSTNAME -d sms_c_prod
```

解决方案:

数据库关闭:

```
# 启动数据库 (MySQL/MariaDB)
systemctl start mysql

# 启动数据库 (PostgreSQL)
systemctl start postgresql
```

凭证错误:

更新配置:

```
export DB_USERNAME=correct_user
export DB_PASSWORD=correct_password

# 重启应用程序
systemctl restart sms_c
```

网络问题：

- 检查防火墙规则
- 验证安全组（云）
- 检查 VPN/网络连接性

连接池耗尽：

增加池大小：

```
config :sms_c, SmsC.Repo,
  pool_size: 50  # 从当前值增加
```

查询缓慢

症状：

- 数据库查询时间高
- API 响应缓慢
- 连接池排队增加

诊断步骤：

1. 检查慢查询日志：

```
-- MySQL/MariaDB: 启用慢查询日志
SET GLOBAL slow_query_log = 'ON';
SET GLOBAL long_query_time = 1;  -- 记录 > 1 秒的查询

-- 查看慢查询 (MySQL/MariaDB)
SELECT * FROM mysql.slow_log ORDER BY query_time DESC LIMIT 10;

-- PostgreSQL: 在 postgresql.conf 中启用慢查询日志
-- log_min_duration_statement = 1000  # 毫秒
-- 然后检查 PostgreSQL 日志
```

2. 检查缺失的索引:

```
-- 检查表索引
SHOW INDEX FROM message_queues;

-- 预期索引:
-- - source_smsc
-- - dest_smsc
-- - send_time
-- - inserted_at
```

3. 检查表统计信息:

```
-- 表大小 (MySQL/MariaDB)
SELECT
  table_name,
  table_rows,
  ROUND(data_length / 1024 / 1024, 2) AS data_mb,
  ROUND(index_length / 1024 / 1024, 2) AS index_mb
FROM information_schema.tables
WHERE table_schema = 'sms_c_prod';

-- 表大小 (PostgreSQL)
-- SELECT schemaname, tablename,
--
pg_size_pretty(pg_total_relation_size(schemaname||'.'||tablename))
AS size
-- FROM pg_tables WHERE schemaname = 'public';
```

解决方案：

缺失索引：

```
CREATE INDEX idx_message_queues_source_smsc ON
message_queues(source_smsc);
CREATE INDEX idx_message_queues_dest_smsc ON
message_queues(dest_smsc);
CREATE INDEX idx_message_queues_send_time ON
message_queues(send_time);
CREATE INDEX idx_message_queues_status ON message_queues(status);
```

表碎片化：

```
-- MySQL/MariaDB
OPTIMIZE TABLE message_queues;
OPTIMIZE TABLE frontend_registrations;

-- PostgreSQL
-- VACUUM ANALYZE message_queues;
-- VACUUM ANALYZE frontend_registrations;
```

数据过多：

清理旧记录：

```
-- 删除超过 30 天的已投递消息
DELETE FROM message_queues
WHERE status = 'delivered'
AND deliver_time < DATE_SUB(NOW(), INTERVAL 30 DAY)
LIMIT 10000;
```

磁盘空间已满

症状：

- 错误：“磁盘已满”
- 无法写入数据库

- 应用程序崩溃

诊断步骤:

1. 检查磁盘使用情况:

```
df -h

# 检查 SQL 数据库目录 (MySQL/MariaDB)
du -sh /var/lib/mysql

# 检查 SQL 数据库目录 (PostgreSQL)
du -sh /var/lib/postgresql
```

2. 查找大文件:

```
# 查找最大文件 (MySQL/MariaDB)
find /var/lib/mysql -type f -exec du -h &#123;&#125; + | sort -rh
| head -20

# 查找最大文件 (PostgreSQL)
find /var/lib/postgresql -type f -exec du -h &#123;&#125; + | sort
-rh | head -20

# 检查日志文件
du -sh /var/log/sms_c/*
```

解决方案:

清理旧数据:

```
-- 删除旧消息
DELETE FROM message_queues
WHERE inserted_at < DATE_SUB(NOW(), INTERVAL 90 DAY)
LIMIT 100000;
```

轮换日志:

```
# 强制 logrotate
logrotate -f /etc/logrotate.d/sms_c

# 清除旧日志文件
find /var/log/sms_c -name "*.log.*" -mtime +30 -delete
```

扩展磁盘：

- 调整卷大小（云）
- 添加新磁盘并扩展卷
- 将数据移动到更大磁盘

前端连接问题

前端未显示为活动

症状：

- 前端状态显示“过期”
- 前端不在活动列表中
- 消息未投递到前端

诊断步骤：

1. 检查注册：

```
curl https://api.example.com:8443/api/frontends/active | grep
frontend_name
```

2. 检查前端日志：

- 前端是否调用 `/api/frontends/register`？
- 有任何 API 错误吗？
- 注册频率（应每 60 秒一次）

3. 检查 **API** 日志：

```
grep "frontend.*register" /var/log/sms_c/application.log | tail
-20
```

□□□□□

□□□□□□

□□□□□□□□

```
curl -X POST https://api.example.com:8443/api/frontends/register \
  -H "Content-Type: application/json" \
  -d '&#123;
    "frontend_name": "uk_gateway",
    "frontend_type": "smpp",
    "ip_address": "10.0.1.50"
  &#125;'
```

□□□□□□□□□□□□□□□/□□□□

□□□□□

□□□ 90 □□□□□□□□□ 60 □□□□□□□

```
# □□□□ 60 □□□□□
while True:
    register_with_smsc()
    time.sleep(60)
```

□□□□□

- □□□□□ API □□□□□□
- □□ DNS □□
- □□□□□□□□ curl

□□□□□□/□□

□□□

- 前端状态在活动/过期之间切换
- 注册历史中的高计数
- 连接不稳定

诊断步骤：

1. 检查前端健康：

- 前端进程是否稳定？
- 有崩溃或重启吗？
- 资源问题（CPU/内存）？

2. 检查网络稳定性：

```
# 检查丢包
ping -c 100 api.example.com

# 检查连接重置
netstat -s | grep -i reset
```

3. 检查注册时间：

- 太频繁？（每几秒）
- 太不频繁？（> 90 秒）

解决方案：

前端不稳定：

- 修复前端应用程序问题
- 增加前端资源
- 检查前端日志中的错误

网络问题：

- 检查是否存在间歇性连接
- 查看防火墙日志
- 检查负载均衡器健康检查

注册间隔错误：

正确间隔：

```
REGISTRATION_INTERVAL = 60  # 秒
```

计费/收费问题

收费失败

症状：

- `sms_c_charging_failed_count` 增加
- 事件日志显示“charging_failed”
- 消息标记为 `charge_failed: true`

诊断步骤：

1. 检查 **OCS** 连接��：

```
# 测试 OCS API
curl -X POST http://ocs.example.com:2080/jsonrpc \
  -H "Content-Type: application/json" \
  -d '&amp;#123;
    "method": "SessionSv1.Ping",
    "params": [],
    "id": 1
  &amp;#125;'
```

预期：`{"result":"Pong"}`

2. 检查 **OCS** 日志：

```
tail -f /var/log/ocs/ocs.log
```

3. 检查配置：

```
# 验证 OCS URL
grep ocs_url config/runtime.exs
```

解决方案：

OCS 不可用：

```
# 检查 OCS 状态
systemctl status ocs

# 如有需要，启动
systemctl start ocs
```

配置错误：

更新配置：

```
config :sms_c,
  ocs_url: "http://correct-host:2080/jsonrpc",
  ocs_tenant: "correct_tenant"
```

暂时禁用收费：

```
config :sms_c,
  default_charging_enabled: false
```

重启应用程序。

账户问题：

- 检查 OCS 中账户是否存在
- 验证账户是否有余额
- 检查计费计划是否配置

收费太慢

症状：

- `sms_c_charging_succeeded_duration` p95 > 500ms
- 启用收费时消息处理缓慢
- 禁用时快速

诊断步骤：

1. 检查收费延迟：

```
histogram_quantile(0.95, sms_c_charging_succeeded_duration_bucket)
```

2. 检查 **OCS** 性能：

```
# OCS 响应时间
curl -w "%&#123;time_total&#125;\n" -X POST
http://ocs.example.com:2080/jsonrpc \
  -H "Content-Type: application/json" \
  -d '&#123;"method":"SessionSv1.Ping","params":[],"id":1&#125;'
```

3. 检查网络延迟：

```
# Ping OCS 主机
ping -c 10 ocs.example.com
```

解决方案：

OCS 慢：

- 优化 OCS 配置
- 添加 OCS 资源
- 使用更快的计费引擎

网络延迟：

- 将 OCS 部署到更靠近 SMS-C 的位置

- 使用直接网络路径
- 尽量避免 VPN/隧道

超时太短：

增加超时：

```
config :sms_c,
  ocs_timeout: 5000  # 5 秒
```

ENUM 查找问题

ENUM 查找失败

症状：

- `sms_c_enum_lookup_stop_duration` 显示失败
- 事件日志显示 ENUM 错误
- 带有 `enum_result_domain` 的路由未匹配

诊断步骤：

1. 检查 **ENUM** 配置：

```
grep -A 10 "enum_" config/runtime.exs
```

2. 测试 **DNS** 连接性：

```
# 测试 DNS 服务器
dig @8.8.8.8 e164.arpa

# 测试 ENUM 查询
# 对于 +15551234567:
dig @8.8.8.8 NAPTR 7.6.5.4.3.2.1.5.5.5.1.e164.arpa
```

3. 检查 **DNS** 服务器：

```
# 自定义 DNS 是否可达?
ping 10.0.1.53

# 测试端口
nc -zv 10.0.1.53 53
```

解决方案:

DNS 服务器不可达:

使用备用 DNS:

```
config :sms_c,
  enum_dns_servers: [
    &#123;"8.8.8.8", 53&#125;,  # Google 公共 DNS
    &#123;"1.1.1.1", 53&#125;   # Cloudflare DNS
  ]
```

ENUM 域错误:

更新域:

```
config :sms_c,
  enum_domains: ["e164.arpa"]  # 使用标准域
```

超时太短:

增加超时:

```
config :sms_c,
  enum_timeout: 10000  # 10 秒
```

禁用 **ENUM**（如果不需要）:

```
config :sms_c,
  enum_enabled: false
```

ENUM 缓存问题

症状:

- 低缓存命中率 (< 70%)
- 缓存大小无限增长
- 内存使用高

诊断步骤:

1. 检查缓存统计信息:

```
# 缓存命中率
rate(sms_c_enum_cache_hit_count[5m]) /
(rate(sms_c_enum_cache_hit_count[5m]) +
rate(sms_c_enum_cache_miss_count[5m]))

# 缓存大小
sms_c_enum_cache_size_size
```

2. 检查流量模式:

- 数字是否重复?
- 缓存 TTL 是否合适?

解决方案:

低命中率（预期）:

- 流量到唯一数字（正常）
- 监控但如果 < 70% 不要报警

缓存增长:

通过 NAPTR 测试页面清除缓存或重启应用程序。

高内存使用:

- 大缓存时预期
- 监控整体系统内存

- 考虑调整 TTL

集群问题

节点无法加入集群

症状：

- 单个节点运行
- 集群查询仅返回本地结��
- Erlang 分布错误

诊断步骤：

1. 检查节点名称：

```
# 在 IEx 控制台中
Node.self()
# 预期: :sms@node1.example.com

Node.list()
# 预期：其他节点的列表
```

2. 检查 **Erlang Cookie**：

```
# 检查 cookie 文件
cat ~/.erlang.cookie

# 在所有节点上验证相同
```

3. 检查网络：

```
# 节点之间可以相互访问吗?
ping node2.example.com

# 检查端口
nc -zv node2.example.com 4369
nc -zv node2.example.com 9100-9200
```

解决方案:

Cookie 不匹配:

在所有节点上设置相同的 cookie:

```
export ERLANG_COOKIE=same_secret_value_here

# 或更新 ~/.erlang.cookie
echo "same_secret_value_here" > ~/.erlang.cookie
chmod 400 ~/.erlang.cookie
```

防火墙阻塞:

打开所需端口:

```
# EPMD
iptables -A INPUT -p tcp --dport 4369 -j ACCEPT

# Erlang 分布
iptables -A INPUT -p tcp --dport 9100:9200 -j ACCEPT
```

DNS 问题:

使用 IP 地址而不是主机名:

```
config :sms_c,
  cluster_nodes: [
    :"sms@10.0.1.10",
    :"sms@10.0.1.11"
  ]
```

□□□□□

□□□

- □□□□□□□□□□
- □□□□□□□□□□□
- Mnesia □□□

□□□□□

1. □□□□□□□□□

```
# □□□□□□□□ (IEx)
Node.list()
```

2. □□ **Mnesia**□

```
:mnesia.system_info(:running_db_nodes)
```

□□□□□

□□□□□□□

```
# □□□□□□□
systemctl stop sms_c

# □□□□□□□□□
systemctl start sms_c  # □ node1 □

# □□□□□□□□□□□□□□□□□□□□
systemctl start sms_c  # □ node2 □
systemctl start sms_c  # □ node3 □
```

Mnesia □□□□

- □□□□□□□□□□
- □□□□□□
- □□ Mnesia □□

- 启动节点
- 导入路由

API 问题

API 不响应

症状:

- 连接超时
- 连接被拒绝
- 无响应

诊断步骤:

1. 检查 **API** 进程:

```
# 应用程序是否在运行?
systemctl status sms_c

# 检查监听端口
netstat -tlnp | grep 8443
```

2. 检查防火墙:

```
# 检查 iptables
iptables -L -n | grep 8443

# 测试本地连接
curl -k https://localhost:8443/api/status
```

3. 检查 **TLS** 配置:

```
# 检查证书是否存在
ls -l priv/cert/server.crt priv/cert/server.key

# 检查证书有效性
openssl x509 -in priv/cert/server.crt -noout -dates
```

解决方案：

应用程序未运行：

```
systemctl start sms_c
```

防火墙阻塞：

```
# 允许 API 端口
iptables -A INPUT -p tcp --dport 8443 -j ACCEPT
```

证书问题：

生成新证书（请参见配置指南）。

端口错误：

检查配置：

```
grep "port:" config/runtime.exs
```

API 返回 500 错误

症状：

- 内部服务器错误
- 500 状态码
- 日志中有错误

诊断步骤：

1. 检查应用程序日志:

```
tail -100 /var/log/sms_c/application.log | grep "\[error\]"
```

2. 检查数据库:

```
mysql -u sms_user -p -e "SELECT 1"
```

3. 检查资源:

```
# 内存
free -h

# CPU
top -b -n 1

# 磁盘
df -h
```

解决方案:

数据库不可用:

- 启动数据库
- 修复连接配置

内存不足:

- 重启应用程序
- 增加系统内存
- 检查内存泄漏

应用程序错误:

- 检查日志中的具体错误
- 修复配置问题
- 重启应用程序

Web UI 问题

无法访问 Web UI

症状：

- 连接超时
- 404 未找到
- 页面无法加载

诊断步骤：

1. 检查应用程序状态：

```
systemctl status sms_c
```

2. 检查端口：

```
netstat -tlnp | grep 80
```

3. 检查 **URL**：

- 主机名正确吗？
- 端口正确吗？
- HTTP 与 HTTPS？

解决方案：

错误的端口：

检查配置：

```
grep "control_panel" config/runtime.exs
```

在正确的端口访问（默认：80 或 4000）。

应用程序未运行：

```
systemctl start sms_c
```

防火墙：

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

LiveView 未更新

症状：

- 页面加载但不更新
- 数据过时
- 浏览器控制台中的 WebSocket 错误

诊断步骤：

1. 检查浏览器控制台：

- 打开开发者工具 (F12)
- 查找 WebSocket 错误
- 检查网络选项卡中的失败请求

2. 检查代理配置：

如果使用反向代理，确保支持 WebSocket：

```
location /live &#123;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
&#125;
```

解决方案：

WebSocket 被阻止：

- 配置代理以支持 WebSocket
- 检查防火墙
- 检查浏览器扩展

刷新页面：

- 硬刷新 (Ctrl+F5)
- 清除浏览器缓存

系统资源问题

高 CPU 使用率

症状：

- CPU 持续 > 80%
- 系统缓慢
- 应用程序无响应

诊断步骤：

1. 检查进程：

```
top -b -n 1 | grep beam.smp
```

2. 检查指标：

```
# 息处理速率
rate(sms_c_message_received_count[5m])

# 路由操作
rate(sms_c_routing_route_matched_count[5m])
```

解决方案：

高流量：

- 横向扩展（添加节点）
- 纵向扩展（增加 CPU）

路由效率低：

- 减少路由数量
- 优化路由标准

ENUM 查找过多：

- 检查缓存命中率
- 考虑禁用如果不需要

高内存使用

症状：

- 内存使用 > 90%
- 应用程序崩溃
- 内存不足错误

诊断步骤：

1. 检查内存：

```
free -h

ps aux | grep beam.smp
```

2. 检查缓存大小：

```
sms_c_enum_cache_size_size
```

解决方案：

ENUM 缓存过大：

- 清除缓存

- 减少 TTL
- 如果不需要，禁用 ENUM

批处理队列增长：

```
# 检查工作者统计信息 (IEx)
SmsC.Messaging.BatchInsertWorker.stats()
```

如果队列很大，手动刷新或重启。

增加内存：

- 纵向扩展
- 添加交换空间（临时）

内存泄漏：

- 重启应用程序
- 报告问题以供调查

如需更多帮助，请咨询：

- 操作指南 - 日常程序
- 配置指南 - 配置选项
- 指标指南 - 监控设置
- 应用程序日志 - `/var/log/sms_c/application.log`