□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
- 预填充数据库以提供真实场景的测试数据

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 API 参考

← 返回文档索引

SMS-C REST API 的参考，由实时路由表（`config/config.exs`）和控制器实现生成。下面的每个端点都是实际路由的服务。

该应用程序本身提供机器可读的 OpenAPI 规范和交互式 Swagger UI。这些始终与已部署的构建保持同步：

```
GET /api/schema      # OpenAPI JSON 规范
GET /api/docs        # Swagger UI
```

目录

- 概述
- 认证
- 响应约定
- 状态
- 消息
- 消息等待指示
- 位置
- 前端
- 事件
- 活动
- 其他端点系列
- HTTP 状态码
- 错误格式
- 最佳实践

概述

基础 URL

```
https://<host>:8443/api
```

默认端口：8443（可通过 `config :api_ex, api: %{port: ...}` 配置）协议：当 `enable_tls` 设置为 true 时使用 HTTPS（默认）。所有路由都挂载在 `/api` 前缀下。

内容类型

请求和响应体为 JSON：

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

没有基于 URL 的 API 版本控制。路由表是合同；请查阅 `/api/schema` 以获取确切的、已部署的形状。

认证

该 API 没有应用级认证。请求管道仅协商内容类型并附加 OpenAPI 规范；在路由前没有 API 密钥、承载令牌或会话检查。

因此，访问控制预计在应用程序外部处理：

- **TLS**：当 `enable_tls` 为 true 时，监听器使用配置的 `tls_cert_path` / `tls_key_path` 终止 HTTPS。请在生产中使用。
- 网络控制：通过防火墙或上游反向代理限制 API 端口仅对受信任的来源开放。多个端点记录调用者 IP（来自 `X-Forwarded-For` 或套接字）以进行审计，但不对此进行限制。

请勿将 API 端口直接暴露给不受信任的网络。

响应约定

控制器直接返回资源，而不是包装在 `data` 信封中。

单个资源：

```
{ "id": 12345, "status": "queued" }
```

集合作为裸 JSON 数组返回：

```
[ { "id": 12345 }, { "id": 12346 } ]
```

控制器级错误使用扁平的 `error` 字符串：

```
{ "error": "payload_hex is required. For plaintext, use POST
/api/messages_plaintext" }
```

框架级错误（路由失败、无法解析的主体）返回：

```
{ "status": "error", "response": "Not Found" }
```

状态

用于监控和负载均衡的健康检查。

请求：

```
GET /api/status
```

响应 (200 OK)：

```
{
  "status": "ok",
  "application": "OmniMessage",
  "timestamp": "2025-11-01T23:00:00Z"
}
```

示例：

```
curl https://host:8443/api/status
```

消息

消息队列是核心资源。消息内容以 **`payload_hex`**（TP-User-Data 字节）加上 **`dcs`**（数据编码方案）字节和可选的 **`udh_hex`**（用户数据头）传递。明文调用者应使用 提交明文，该方法为您执行编码、DCS 选择和多部分拆分。

消息对象

返回的消息字段（请参见 `ApiEx.Schemas.MessageSchema`）：

字段	类型	备注
`id`	integer	消息 ID
`source_msisdn`	string	源号码
`destination_msisdn`	string	目标号码
`payload_hex`	string	TP-User-Data 字节的十六进制（规范内容）
`dcs`	string	数据编码方案的十六进制字节（`00`=GSM7, `08`=UCS2, `04`=8bit, `C8`=MWI）
`udh_hex`	string / null	用户数据头字节的十六进
`message_text`	string / null	仅用于显示的派生 UTF-8 文本（不用于投递）
`source_smsc`	string	源 SMSC 标识符
`dest_smsc`	string / null	目标 SMSC 标识符
`status`	string	其中之一 `queued`, `backoff`, `sent`, `delivered`, `expired`, `dropped`, `auto_replied`, `balance_rejected`, `deleted`
`send_time`	datetime	
`deliver_time`	datetime / null	
`expires`	datetime	
`delivery_attempts`	integer	

字段	类型	备注
`deadletter`	boolean	
`message_parts`	integer / null	
`message_part_number`	integer / null	
`smsc_node_name`	string	拥有该消息的节点

列出消息

请求：

```
GET /api/messages
```

可选头：

- `smsc: <frontend_name>` - 轮询模式：返回目标为 SMSC 的消息。
- `include-unrouted: true|1` - 在轮询模式下，也返回没有位置注册的消息（默认：false）。
- `x-decode: true|1` - 包括每条消息负载的解码视图。

查询参数（列表模式，仅在没有 `smsc` 头时使用）：

- `status` - 按单个状态原子过滤（请参见上面的状态列表）。
- `limit` - 最大记录数（默认 100，最大 1000）。
- `offset` - 跳过的记录数（默认 0）。

该端点在列表模式下 始终分页：没有 `limit` 时最多返回 100 条记录，绝不会返回整个存储。

响应 (200 OK) - 消息对象的 JSON 数组。

示例：

```
# 轮询前端的消息
curl -H "smsc: uk_gateway" https://host:8443/api/messages

# 包括未路由消息的轮询
curl -H "smsc: uk_gateway" -H "include-unrouted: true"
https://host:8443/api/messages

# 列出已投递的消息
curl "https://host:8443/api/messages?status=delivered&limit=50"
```

获取消息

请求：

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

可选的 `x-decode: true` 头添加解码负载视图。

响应 (200 OK): 单个消息对象。**404** 时返回 `{"error": "Message not found"}` 如果 ID 不存在。

```
curl https://host:8443/api/messages/12345
```

提交消息

提交一个预编码的消息。**`payload_hex`** 是必需的；没有它的请求将被拒绝，返回 422。

请求：

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

主体：

```
{
  "source_msisdn": "15555551234",
  "destination_msisdn": "15555555678",
  "payload_hex": "c8329bfd06dddf723619",
  "dcs": "00",
  "source_smsc": "api_client"
}
```

字段：

- `source_msisdn`, `destination_msisdn` - 号码。
- `payload_hex` - 必需。TP-User-Data 字节的十六进制。
- `dcs` - 数据编码方案的十六进制字节（下游应用默认）。
- `udh_hex` - 可选的用户数据头十六进制。
- `source_smsc` - 标识提交系统。
- `source_type` - 可选的源分类。

响应 (201 Created): 创建的消息对象。拒绝返回 422 和 `error` 字符串（例如，缺少 `payload_hex`）。

```
curl -X POST https://host:8443/api/messages \
  -H "Content-Type: application/json" \
  -d '{
    "source_msisdn": "15555551234",
    "destination_msisdn": "15555555678",
    "payload_hex": "c8329bfd06dddf723619",
    "dcs": "00",
    "source_smsc": "api_client"
  }'
```

提交明文

提交一个 UTF-8 字符串，让 SMS-C 处理编码（GSM7 与 UCS2）、DCS 选择和多部分拆分。

请求：

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

主体：

```
{
  "source_msisdn": "15555551234",
  "destination_msisdn": "15555555678",
  "message": "Hello World",
  "source_smsc": "api_client"
}
```

消息文本可以作为 `message`、`text` 或 `body` 提供。`source_msisdn` 和 `destination_msisdn` 是必需的。

响应 (201 Created)：

- 单部分：创建的消息对象。
- 多部分：`{ "parts": <n>, "messages": [ ... ] }`。

```
curl -X POST https://host:8443/api/messages_plaintext \
  -H "Content-Type: application/json" \
  -d '{
    "source_msisdn": "15555551234",
    "destination_msisdn": "15555555678",
    "message": "Hello World",
    "source_smsc": "api_client"
  }'
```

提交原始 PDU

提交一个原始 SMS PDU（十六进制），以最大限度地兼容遗留和 SIP 前端。

请求：

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

主体：

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

字段：

- `message_body` - 必需。十六进制编码的 SMS PDU（或 TPDU，见下文）。
- `source_msisdn` - 可选。
- `source_smsc` / `smsc_node_name` - 在仅 TPDU 模式下必需。
- `rp_originator_address`, `rp_destination_address` - 提交裸 TPDU 时提供这些（RP 层被合成）。

调用者 IP 从 `X-Forwarded-For`（或套接字）中获取，以解析发起的前端。

响应：

- 201 Created, 返回 `{ "status": "success", "response": { "id": <id> } }`（生成时包含 `rp_ack` 十六进制字段）。
- 200 OK, 返回 `{ "status": "success", "response": { "status": "delivery_report_received" } }` 当 PDU 是投递报告而不是新消息时。
- 422, 返回缺失/无效的十六进制或解码失败的 `error` 字符串。

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

该服务解码 PDU（3GPP TS 23.040），提取号码/文本/DCS，检测投递报告，并应用路由。投递报告被记录但不作为新消息存储。

更新消息 (PATCH)

对存储的消息进行部分更新。

请求：

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

发送任何可写的消息字段（例如 `dest_smsc`、`deliver_after`、`status`）。元数据更改记录在事件日志中。

响应 (200 OK): 更新的消息对象。404 如果未找到，422 在拒绝更新时。

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

增加投递尝试

记录一次失败的投递：增加尝试计数，将消息移动到 `backoff`，并使用指数退避调度下次重试。

存在两种等效形式：

```
PUT  /api/messages/:id                    # 消息资源，PUT = 增加
POST /api/messages_increment_attempt         (id 在主体中)
PUT  /api/messages_increment_attempt/:id     (id 在 URL 中)
```

`PUT /api/messages/:id` 与 `PATCH /api/messages/:id` 是不同的PATCH 进行部分更新，PUT 增加投递尝试。传递 `keep_route=true` 以在重试时保留当前的 `dest_smsc`。

响应 (200 OK): 更新的消息，例如

```
{ "id": 12345, "status": "backoff", "delivery_attempts": 2 }
```

```
curl -X PUT https://host:8443/api/messages/12345
```

标记为已投递

将消息标记为已投递（本地存储或联合）。

```
POST /api/messages_mark_delivered           (id 在主体中)
PUT  /api/messages_mark_delivered/:id        (id 在 URL 中)
```

可选的 `smsc` 头选择联合 SMSC，当消息未在本地保存时。

响应 (200 OK): 已投递的消息对象。404 如果未找到。

```
curl -X POST https://host:8443/api/messages_mark_delivered \
  -H "Content-Type: application/json" \
  -d '{ "id": 12345 }'
```

重新路由消息

重新评估存储消息的路由规则并更新其 `dest_smsc`。

请求：

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

主体：`{ "id": 12345 }`

响应 (200 OK):

```
{ "message_id": 12345, "new_dest_smsc": "uk_gateway", "status":
"rerouted" }
```

如果未找到消息则返回 404，如果没有匹配的路由则返回 422。

清除字段

将特定消息字段重��为 null��

请求：

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

主体：`{ "id": 12345, "field": "dest_smsc" }`

`field` 必须是 `delivery_time`、`dest_smsc` 或 `deliver_after` 之一；任何其他值返回 400。更改记录在事件日志中。

响应 (200 OK)：更新的消息对象。

删除消息

请求：

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

响应 (204 No Content)。404 如果未找到。删除是永久性的。

```
curl -X DELETE https://host:8443/api/messages/12345
```

消息等待指示

提交 MWI（消息等待指示）通知。控制器根据 3GPP TS 23.038 构建 MWI PDU，并存储一个准备投递的消息，具有预计算的 `payload_hex` 和 `dcs`。

请求：

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

主体：

```
{
  "source_msisdn": "12345",
  "destination_msisdn": "15555555678",
  "indication_type": "voicemail",
  "indication_active": true,
  "message_body": "You have 1 new voicemail",
  "source_smsc": "IMS_SMSC"
}
```

字段：

- `indication_type` - `voicemail`（默认）、`fax`、`email` 或 `other`。
- `indication_active` - `true` 设置指示器，`false` 清除它（默认 `true`）。
- `message_body` - 可选的附带文本（GSM7 编码）。
- `source_msisdn`、`destination_msisdn`、`source_smsc`、`dest_smsc`。

响应 (201 Created): 创建的消息对象。422 在失败时。

位置

用于移动终止投递的用户位置记录。标准 REST 资源；`:id` 路径段是 IMSI。

```
GET    /api/locations            # 列表（分页：?limit=&offset=，x-total-
count 头）
GET    /api/locations/:id        # 按 IMSI 获取
POST   /api/locations            # 创建或更新（按 IMSI 键入）
PATCH  /api/locations/:id        # 部分更新
PUT    /api/locations/:id        # 更新
DELETE /api/locations/:id        # 删除
```

创建/更新主体：

```
{
  "msisdn": "15555551234",
  "imsi": "001001000000001",
  "location": "msc1.region1.example.com",
  "ran_location": "cell_tower_12345",
  "imei": "123456789012345",
  "ims_capable": true,
  "csfb": false,
  "registered": true,
  "expires": "2025-11-01T23:30:00Z",
  "user_agent": "Samsung Galaxy"
}
```

`imsi` 和 `msisdn` 是必需的；其余字段是可选的。响应为位置对象（200/201）；验证失败返回 422。由移动管理系统在用户注册或注销时调用。

前端

跟踪前端 SMSC 连接。资源仅公开 索引、显示和创建（没有 `active`、`stats`、`history` 或 `register` 子路由）。

```
GET  /api/frontends           # 列出所有注册
GET  /api/frontends/:id       # 获取一个注册
POST /api/frontends           # 注册或刷新前端
```

注册主体（`POST /api/frontends`）：

```
{
  "frontend_name": "smpp-gateway-1",
  "frontend_type": "SMPP",
  "ip_address": "10.0.1.5",
  "hostname": "smpp-gw-01",
  "uptime_seconds": 3600,
  "configuration": "{\"port\": 2775, \"system_id\":
\"smpp_user\"}"
}
```

`frontend_name` 是必需的。如果省略 `ip_address`，则从请求源 IP (`X-Forwarded-For` 或套接字) 填充。如果 `frontend_name` 已存在活动注册，则更新；否则创建新的注册。注册在大约 90 秒后过期，因□前端必须定期重新注册。

响应 (201 Created): 注册对象。

事件

每条消息的生命周期事件。事件保存在 ETS 中，具有滚动 TTL。

获取消息事件

请求：

```
GET /api/events/:id
```

`:id` 是数字消息 ID。

响应 (200 OK):

```
{
  "message_id": 12345,
  "events": [
    { "event_epoch": 1698672000, "name": "message_inserted",
"description": "..." },
    { "event_epoch": 1698672005, "name": "message_delivered",
"description": "..." }
  ]
}
```

如果 ID 不是整数则返回 400。

记录事件

请求：

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

主体：

```
{
  "message_id": 12345,
  "name": "external_delivery_confirmed",
  "description": "Confirmed by downstream system"
}
```

`message_id`、`name` 和 `description` 都是必需的。事件源从调用者 IP 自动检测（与活动前端注册匹配），如果没有匹配则回退到配置的节点名称。

响应 (201 Created)：

```
{
  "status": "success",
  "message": "Event recorded successfully",
  "event": {
    "message_id": 12345,
    "name": "external_delivery_confirmed",
    "description": "Confirmed by downstream system",
    "event_source": "smpp-gateway-1",
    "event_epoch": 1698672010
  }
}
```

活动

批量消息：上传一个命名的接收者列表，然后对其运行限速活动。有关概念（受众过滤、滴水速率、生命周期、统计信息），请参阅 批量消息 / 活动 指南。

接收者列表

```
GET    /api/campaign_lists            # 所有列表
GET    /api/campaign_lists/:id        # 一个列表，包括 "recipients"
POST   /api/campaign_lists            # 创建一个列表
PATCH  /api/campaign_lists/:id        # 附加接收者
DELETE /api/campaign_lists/:id        # 删除列表及其接收者
```

每个接收者最多可以携带四个模板变量（`var1`..`var4`）。将接收者作为 JSON 数组、原始 CSV 字符串或两者提供。JSON 接收者可以是普通 MSISDN 字符串或带有 `var1`..`var4` 的对象：

```
{
  "name": "VIP customers",
  "description": "High value subscribers",
  "recipients": [
    { "msisdn": "12025550101", "var1": "Alice", "var2": "Gold" },
    "12025550102"
  ]
}
```

对于 CSV，命名 `msisdn` / `destination_msisdn` / `number` / `phone` 列的标题行选择 MSISDN 列（否则使用第一列）；其余列按列顺序变为 `var1`..`var4`：

```
{ "name": "Imported", "csv":
"msisdn,first,plan\n12025550101,Alice,Gold\n" }
```

`PATCH` 附加接收者（不会替换它们）。

响应 (201 Created): `{ "id": 1, "name": "VIP customers", "recipient_count": 2 }`

活动

```
GET    /api/campaigns          # 所有活动（分页，最新优先）
GET    /api/campaigns/:id      # 一个活动，包括实时统计
POST   /api/campaigns          # 创建一个活动（开始为草稿）
DELETE /api/campaigns/:id      # 停止活动但保留其记录
```

创建主体：

```
{
  "name": "Maintenance notice",
  "message": "Planned maintenance tonight 02:00-03:00.",
  "source_msisdn": "12345",
  "source_smsc": "IMS_SMSC",
  "list_id": 1,
  "audience": "active_only",
  "drip_tps": 50
}
```

字段：

- `name`、`message`、`source_msisdn`、`source_smsc` - 必需。`message` 支持每个接收者的模板变量（`{{ var1 }}`..`{{ var4 }}`、`{{ msisdn }}`）。
- `list_id` - 接收者列表。省略以针对所有当前活动（注册）用户。
- `audience` - `active_only`（默认）或 `all`。省略 `list_id` 时忽略。
- `drip_tps` - 每秒提交到队列的消息数（默认来自配置）。
- `validity_hours` - 消息有效性（小时）；设置每条消息的 `expires`。
- `deliver_after` - 可选的 ISO 8601 时间，持有消息直到（计划发送）。

`DELETE` 停止投递但保留记录，因□历史和统计信息仍然可见；它返回已取消的活动（200），并在超过保留 TTL 后自动清除。

响应（`GET /api/campaigns/:id`）：

```
{
  "id": 7,
  "name": "Maintenance notice",
  "list_id": 1,
  "audience": "active_only",
  "status": "running",
  "drip_tps": 50,
  "total_targets": 1000,
  "stats": {
    "total": 1000, "pending": 600, "queued": 300,
    "delivered": 90, "failed": 10, "skipped_inactive": 0,
    "dispatched": 400, "progress_percent": 40, "delivery_percent":
22
  }
}
```

控制活动

请求：

```
POST /api/campaigns/control
Content-Type: application/json
```

主体：`{ "campaign_id": 7, "action": "start" }`

`action` 是 `start`、`pause`、`resume`、`cancel` 之一。返回更新的活动（200），如果未找到则返回 `404`，如果当前状态下操作无效则返回 `422`。

每个接收者结果

```
GET /api/campaign_targets/:id                # :id 是活动 ID
GET /api/campaign_targets/:id?state=failed # 按目标状态过滤
```

支持 `limit`/`offset` 分页；总计数（在任何 `state` 过滤后）在 `x-total-count` 响应头中。每个条目携带接收者的 `state`（`pending` / `queued` / `delivered` / `failed` / `skipped_inactive`）。

其他端点系列

以下路由系列是同一 API 的一部分，但在其专用指南中进行了深入文档。这里列出路径以供完整性。

路由

GET /api/routes # 列出路由
GET /api/routes/:id # 获取路由
POST /api/routes # 创建路由
PATCH /api/routes/:id # 更新
PUT /api/routes/:id # 更新
DELETE /api/routes/:id # 删除
GET /api/routes/export # 导出所有路由
POST /api/routes/import # 批量导入
POST /api/routes/simulate # 干运行路由决策

请参阅 SMS 路由指南。

号码翻译

GET /api/translations # 列出
GET /api/translations/:id # 获取
POST /api/translations # 创建
PATCH /api/translations/:id # 更新
DELETE /api/translations/:id # 删除
GET /api/translations/export # 导出
POST /api/translations/import # 批量导入
POST /api/translations/simulate # 干运行翻译

请参阅 号码翻译指南。

操作

```
GET  /api/operations                        # 操作索引
POST /api/operations/clear_messages         # 清除消息存储
POST /api/operations/clear_routes           # 清除路由
POST /api/operations/clear_translations     # 清除翻译
POST /api/operations/clear_frontends        # 清除前端注册
```

这些是破坏性维护操作；相应地保护 API 端口。

联合

```
GET  /api/federation/status           # 联合状态
GET  /api/federation/identity         # 此节点的联合身份
POST /api/federation/health           # 同行健康报告
POST /api/federation/registry         # 注册交换
POST /api/federation/notify           # 入站消息通知
POST /api/federation/delivery_status  # 投递状态回调
GET  /api/federation_messages         # 联合消息视图
```

请参阅 联合指南。

Diameter、日志和 DNS

```
GET  /api/diameter   # Diameter Sh 对等体/dip 状态
GET  /api/logs       # 最近的结构化日志
POST /api/naptr      # NAPTR 解析测试
```

Diameter Sh 用户查找在 HSS 用户查找指南 中描述。

HTTP 状态码

代码	含义	典型用途
200	OK	成功读取或更新
201	Created	资源已创建
202	Accepted	已接受进行异步处理
204	No Content	成功删除
400	Bad Request	输入格式错误（例如，非整数 ID，无效字段）
404	Not Found	未知资源
422	Unprocessable Entity	验证失败
500	Internal Server Error	意外的服务器错误

错误格式

控制器级错误返回扁平的 `error` 字符串：

```
{ "error": "Message not found" }
```

框架级错误（未匹配的路由、无法解析的主体）返回标准化的信封：

```
{ "status": "error", "response": "Not Found" }
```

最佳实践

消息提交

1. 预编码与明文: 当您已经编码了 TPDU 时，发送 payload_hex 到/api/messages；否则使用 /api/messages_plaintext 并让 SMS-C 编码和拆分。
2. 始终设置 source_smsc 以便事件和路由可以归属消息。
3. 一致使用 E.164 风格的号码。
4. 处理 422，通过检查 error 字符串而不是盲目重试。

前端集成

1. 在 ~90 秒过期窗口内重新注册 POST /api/frontends。
2. 使用 smsc 头轮询 获取您的消息；仅在您故意希望未路由流量时使用 include-unrouted。
3. 成功后通过 /api/messages_mark_delivered 标记为已投递。
4. 在失败时通过 PUT /api/messages/:id（或 /api/messages_increment_attempt）增加以调度退避重试。
5. 记录事件 以获取下游确认。

操作

1. 在网络层终止 TLS 并限制 API 端口；没有内置认证。
2. 重用 HTTP 连接并设置合理的客户端超时。
3. 监控 Prometheus 指标（请参阅 METRICS.md）以获取队列和投递健康状况。

ANSSI R226 拦截合规文档

文档目的：本文档提供了根据法国刑法第 R226-3 和 R226-7 条款，获得 ANSSI R226 授权所需的技术规范，适用于 OmniMessage 短信服务中心（SMSc）。

分类：法规合规文档

目标机构：法国国家信息系统安全局（ANSSI）

法规： R226 - 通信隐私保护与合法拦截

---

1. 详细技术规范

1.1 商业技术数据表

产品名称： OmniMessage SMSc（短信服务中心）

产品类型：电信消息中心

主要功能：短信消息路由、存储和交付

网络协议： REST API（HTTPS）、短信协议（SMPP、IMS、SS7/MAP 通过外部前端）

部署模型：本地服务器应用

技术栈： Elixir/Erlang、Phoenix 框架、Mnesia、MySQL/PostgreSQL

核心能力

消息处理：

- 集中的短信消息队列，支持 REST API
- 协议无关设计，支持 SMPP、IMS、SS7/MAP 前端
- 基于前缀的动态路由引擎
- 指数退���的重试逻辑
- 消息过期和死信队列处理
- 通话详细记录（CDR）生成和归档
- 性能：约 1,750 条消息/秒插入速率，150 万条消息/天容量

消息存储：

- 活动消息队列：Mnesia 内存数据库，支持磁盘持久化
  - 主要存储：RAM，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：消息在系统重启后依然存在
  - 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、失败或被拒绝时写入 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）
- 双层架构优势：
  - 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
- 集群支持高可用性（Mnesia 在节点间复制）

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
- 基于前缀的匹配（主叫/被叫号码）
- 源 SMSC 和类型过滤
- 基于优先级和权重的负载均衡
- 号码翻译和标准化
- ENUM（E.164 号码映射）DNS 查询支持
- 自动回复和消息丢弃能力
- 每条路由的计费控制（通过 Diameter Ro 的 OCS 集成）

 完整架构和功能详见 **README.md**

1.2 拦截能力

OmniMessage 实现了 ETSI TS 103 221 拦截接口（X1、X2、X3）。OmniMessage 作为网络元素和拦截点。执法监控设施通过 X1 配置目标。OmniMessage 通过 X2 以 3GPP TS 33.128 xIRI 交付拦截的短信。有关接口角色和完整参数参考，请参见 合法拦截（X1/X2/X3）。

1.2.1 消息获取

短信消息捕获：

OmniMessage SMSc 处理所有订阅者与外部网络之间的短信
完全访问消息元数据和内容，包括：
源 MSISDN（手机号码）
目标 MSISDN（手机号码）
源 IMSI（国际移动用户身份）
目标 IMSI
消息正文（文本内容）
原始 PDU（协议数据单元）数据
TP-DCS（数据编码方案）信息
消息编码（GSM7、UCS-2、8 位、Latin-1）
多部分消息指示符和重组数据
用户数据头（UDH）信息

消息元数据获取：

完整的通话详细记录（CDR）存储在数据库中，包括：
消息 ID（唯一标识符）
主叫号码（源 MSISDN）
被叫号码（目标 MSISDN）
提交时间戳（消息进入系统的时间）
交付时间戳（消息交付的时间）
过期时间戳（如果无法交付，消息过期的时间）状态（已交付、已过期、失败、被拒绝）
交付尝试次数
消息部分（用于连接/多部分 SMS）
源 SMSC 标识符

- 目标 SMSC 标识符
- 源节点（Erlang 集群节点名称）
- 目标节点（用于分布式部署）
- 死信标志（重试耗尽指示符）

 完整 **CDR** 架构详见 **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** 文档见 **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：内存中，带有 disc_copies 备份
  - 主要存储在 RAM 中以获得最大速度
  - 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
- 操作：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/交付过程中绝不查询 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），根据法规要求可配置
- 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��路由操作从不接触 SQL（没有数据库瓶颈）
2. 可扩展性： Mnesia 在没有 SQL 开销的情况下处理 1,750+ 条消息/秒
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. 消息提交 → 存储在 Mnesia（RAM + 磁盘备份）
2. 消息路由 → Mnesia 查询（超快）
3. 消息交付/过期 → CDR 写入 SQL（异步）
4. 24 小时后 → 消息从 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）
- 全文搜索能力（如果在 CDR 数据库上启用）
- 索引 CDR 字段以加快合法拦截查询

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）

 完整监控指南见 **OPERATIONS_GUIDE.md**
 指标文档见 **METRICS.md**

历史分析：

- CDR 数据库可按以下条件查询：
  - 时间范围
  - 主叫/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

用户跟踪：

- 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、时机模式）

网络分析：

- 路由性能指标
- 网关可用性和健康
- 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
- 号码翻译和重写规则
- ENUM DNS 查询用于路由智能
- 基于前缀的路由决策

 号码翻译指南见 **number_translation_guide.md**
 路由指南见 **sms_routing_guide.md**

1.3 对策能力

1.3.1 隐私保护机制

通信机密性：

HTTPS/TLS 用于 REST API 通信
基于证书的身份验证
数据库连接加密（支持 TLS）
可配置的消息正文交付后删除

访问控制：

Web UI 访问控制
API 身份验证机制
数据库访问控制
前端注册身份验证

审计日志：

完整的系统事件日志
消息提交/交付日志
配置更改跟踪
管理操作日志
结构化日志记录，支持可配置级别

1.3.2 数据保护特性

消息隐私：

可配置的消息正文交付后删除
消息正文可选不显示在 UI 中
消息正文可选不包含在导出中
CDR 消息正文字段可以设置为 NULL 以保护隐私

数据库安全：

MySQL 表加密支持（ENCRYPTION='Y'）

- PostgreSQL 透明数据加密支持
- 数据库访问角色分离
- 分析用只读用户帐户
- 限制对消息内容的访问

系统加固：

- 最小化暴露的网络端口
- TLS 证书管理
- 安全配置存储
- 基于环境的配置分离
- 使用 Erlang 分布式协议的集群安全

1.4 存储架构：Mnesia + SQL 双层设计

概述

OmniMessage SMSc 采用独特的双层存储架构，专门设计用于高性能操作消息处理与长期合规和归档存储分开。

层 1：**Mnesia** 内存消息队列

什么是 **Mnesia**？

- 内置于 Erlang/OTP 运行时的分布式数据库
- 混合存储：主要在内存中，自动磁盘备份
- ACID 兼容事务
- 跨多个节点的集群复制

存储模式：**disc_copies**

- 内存主存：所有活动消息存储在 RAM 中
  - 闪电般快速的读/写操作（亚毫秒）
  - 在正常消息路由操作期间没有磁盘 I/O
  - 实现 1,750+ 条消息/秒的吞吐量
- 磁盘备份（自动）： Mnesia 将 RAM 同步到磁盘
  - 写入在后台异步进行
  - 每次事务提交时更新磁盘副本

- 崩溃恢复：系统重启时所有消息完好无损
- 位置：应用数据中的 `Mnesia.*/` 目录

Mnesia 中的消息生命周期：

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. 清理工作者（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/交付过程中绝不查询 **SQL** 数据库
- SQL 在操作消息处理□□程中完全被绕过
- 这消除了传统 SMS-C 瓶颈（数据库 I/O）

层 **2**：用于 **CDR** 导出/归档的 **SQL** 数据库

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**：仅在消息达到最终状态时创建 CDR 记录：

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

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

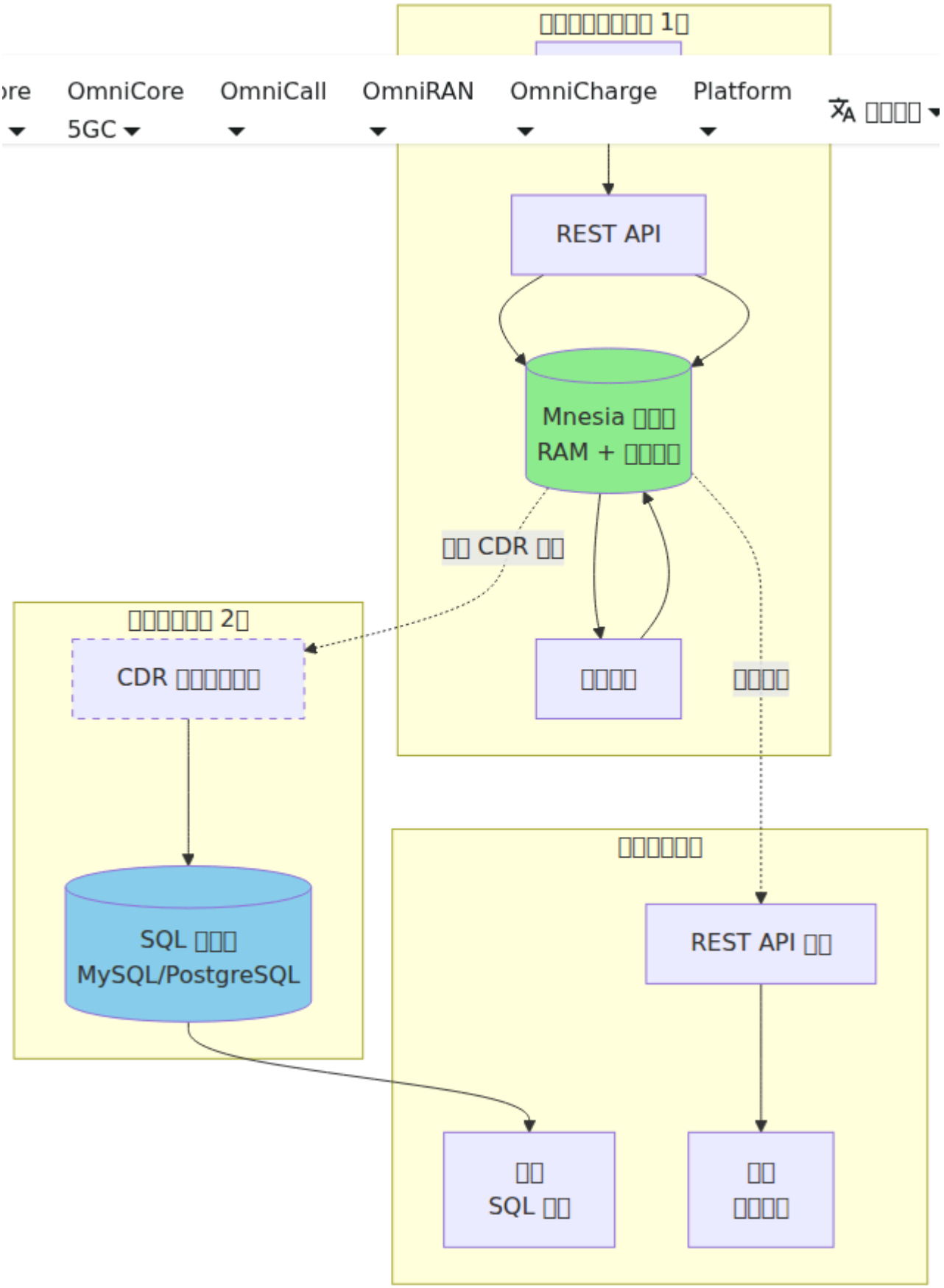
- □□□□□ □□ 100ms□□□□□
- □□□□□ □□□□□ CDR □□□□□□□□□□

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

SQL □□□□□□□

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

□□□


OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
REST API
Mnesia
RAM +
CDR
CDR
SQL
MySQL/PostgreSQL
REST API
SQL

-

虚线：异步操作（后台）
绿色：高性能层（内存）
蓝色：归档层（持久 SQL）

合法拦截的影响

最近消息（< 24 小时）：

通过 Mnesia 访问（REST API 查询）
超快检索
完整的消息内容可用
可进行实时监控

历史消息（> 24 小时）：

通过 SQL 数据库访问（CDR 表）
标准 SQL 查询性能
始终可用完整的消息元数据
消息正文可用（除非启用隐私模式）

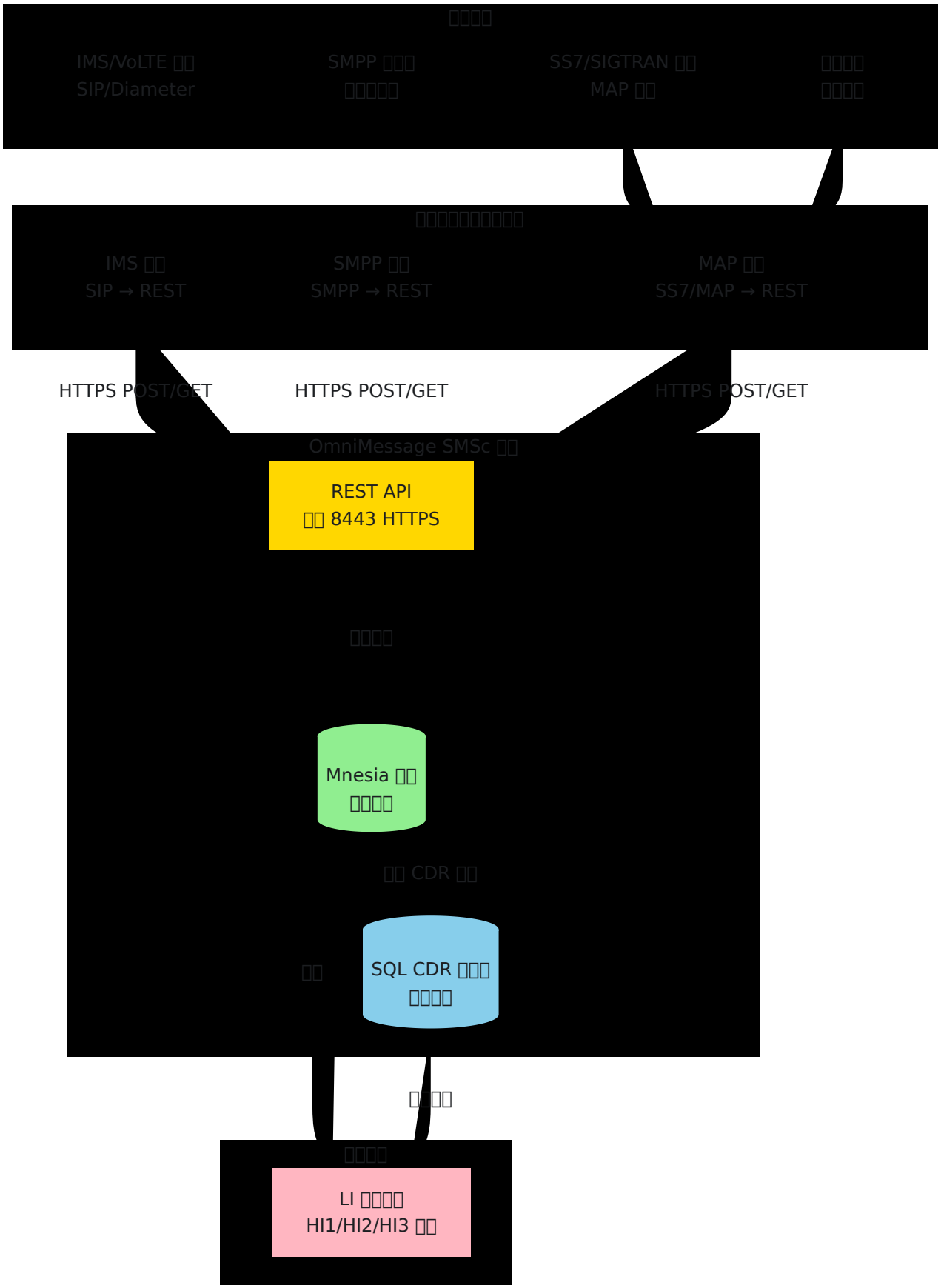
合规性好处：

1. 无数据丢失： disc_copies 模式确保消息在崩溃时存活
2. 永久审计跟踪： CDR 在 SQL 数据库中保留多年
3. 性能：合法拦截查询不会影响消息路由
4. 灵活性：最近消息（Mnesia）+ 历史消息（SQL）均可访问

1.5 多协议前端集成架构

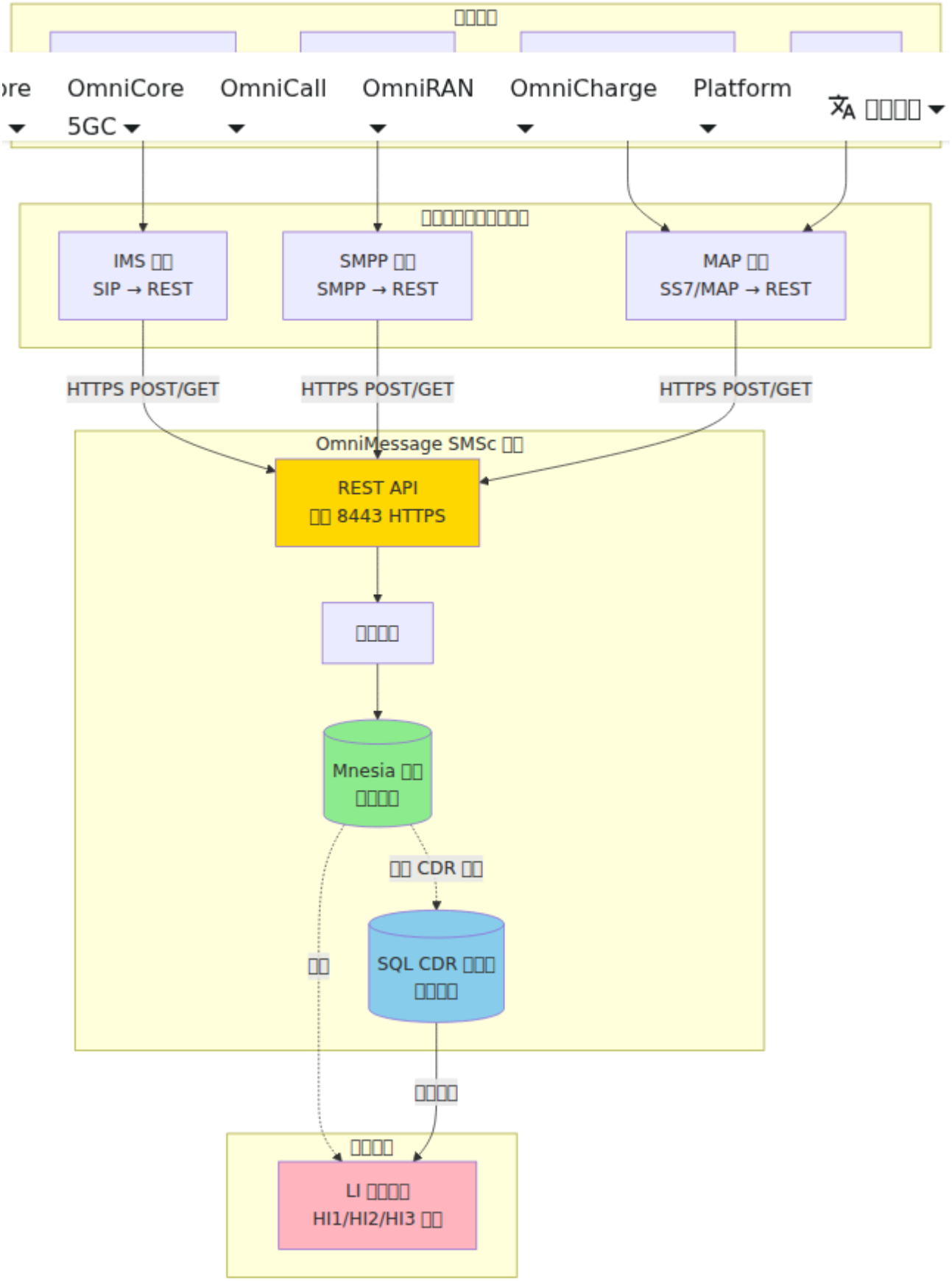
OmniMessage SMSc 采用协议无关的核心设计，通过统一的 REST API 与外部协议特定网关（前端）接口。该架构允许合法拦截捕获消息，无论使用何种电信协议发送或接收消息。

架构概述


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
LI
HI1/HI2/HI3

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。


OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
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
HI1/HI2/HI3

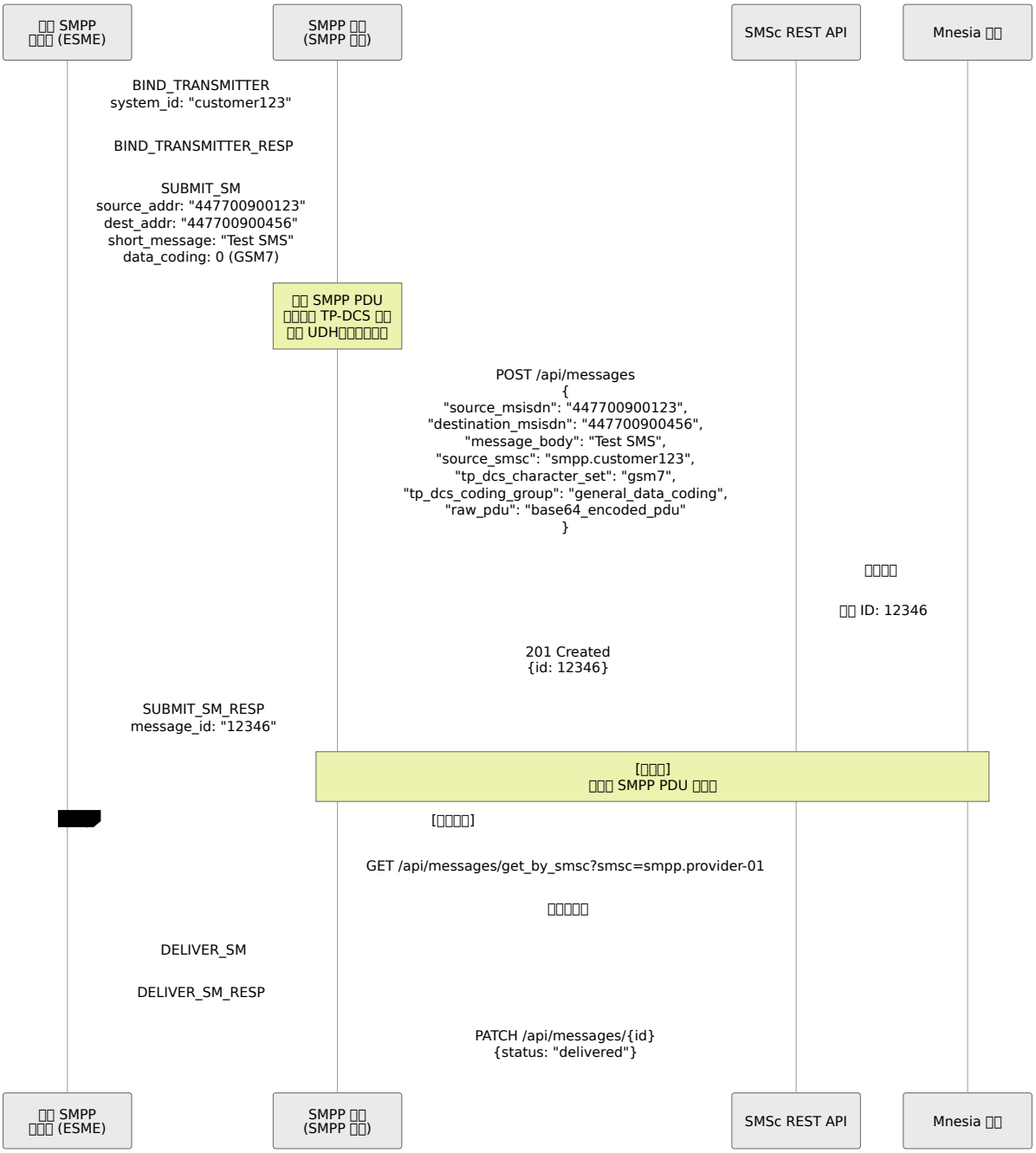
IMS

- /IMSI IMS

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

2. SMPP □□□□

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

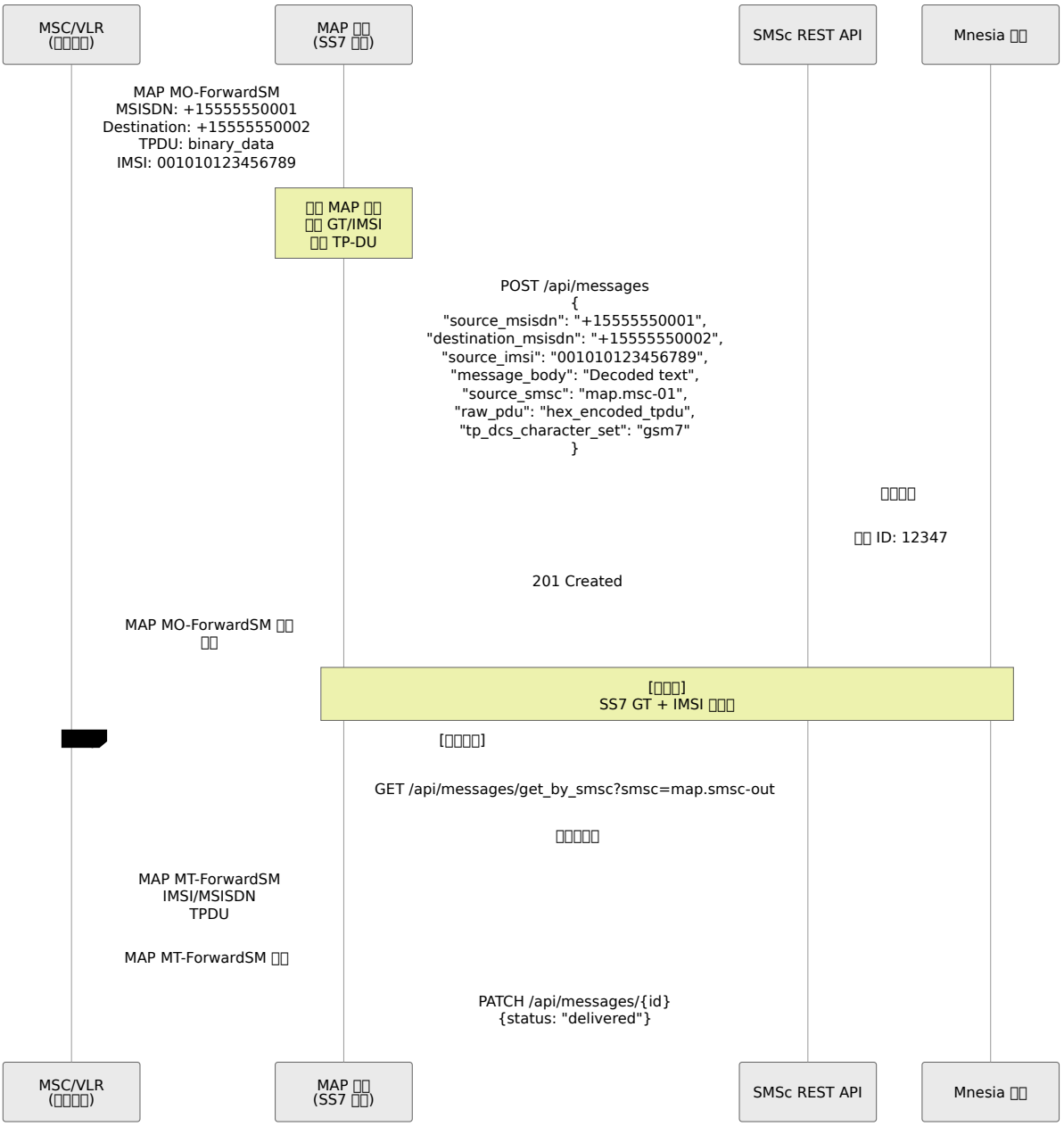


SMPP □□□□□□□

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

3. SS7/MAP □□□□

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



SS7/MAP □□□□□□□

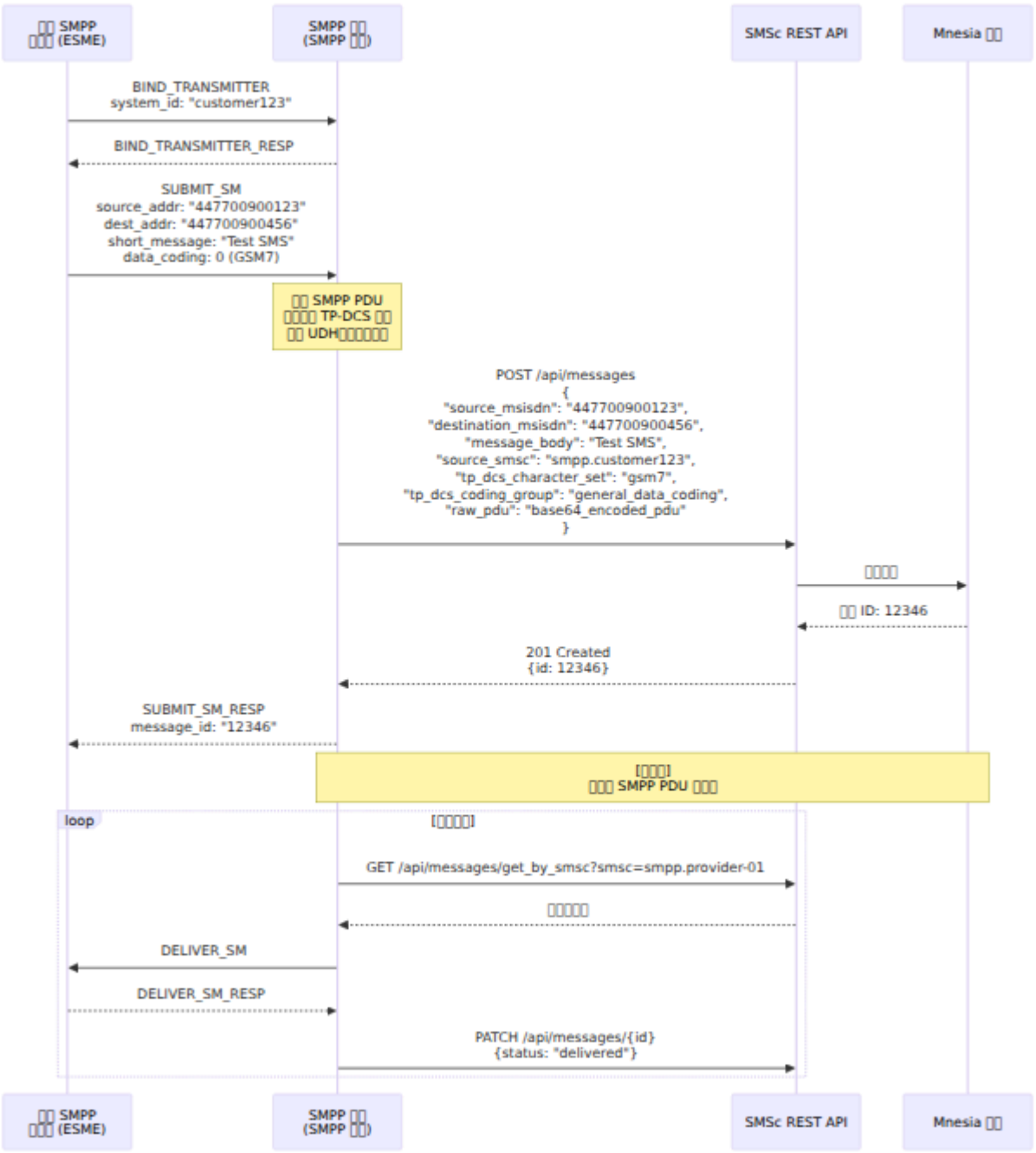
- MAP 消息中的 IMSI
- 全球标题（GT）地址
- MSC/VLR 地址（网络元素识别）
- SCCP 呼叫/被叫方地址
- MAP 操作码
- TP-用户数据二进制格式

所有协议的统一拦截

合法拦截的关键好处：无论使用哪种协议（IMS/SIP、SMPP 或 SS7/MAP），所有消息都在 SMSc 核心中汇聚，具有标准化的数据结构，从而实现：

1. 协议无关的监控：单一拦截点捕获所有息类型
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”）
- 使按协议类型进行过滤和分析成为可能
- 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

两层存储架构提供多个合法拦截的访问点，针对实时监控（Mnesia）和历史分析（SQL）进行了优化。

1. REST API 访问最近消息（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
- 所有查询都访问 Mnesia（内存中）以获得即时响应

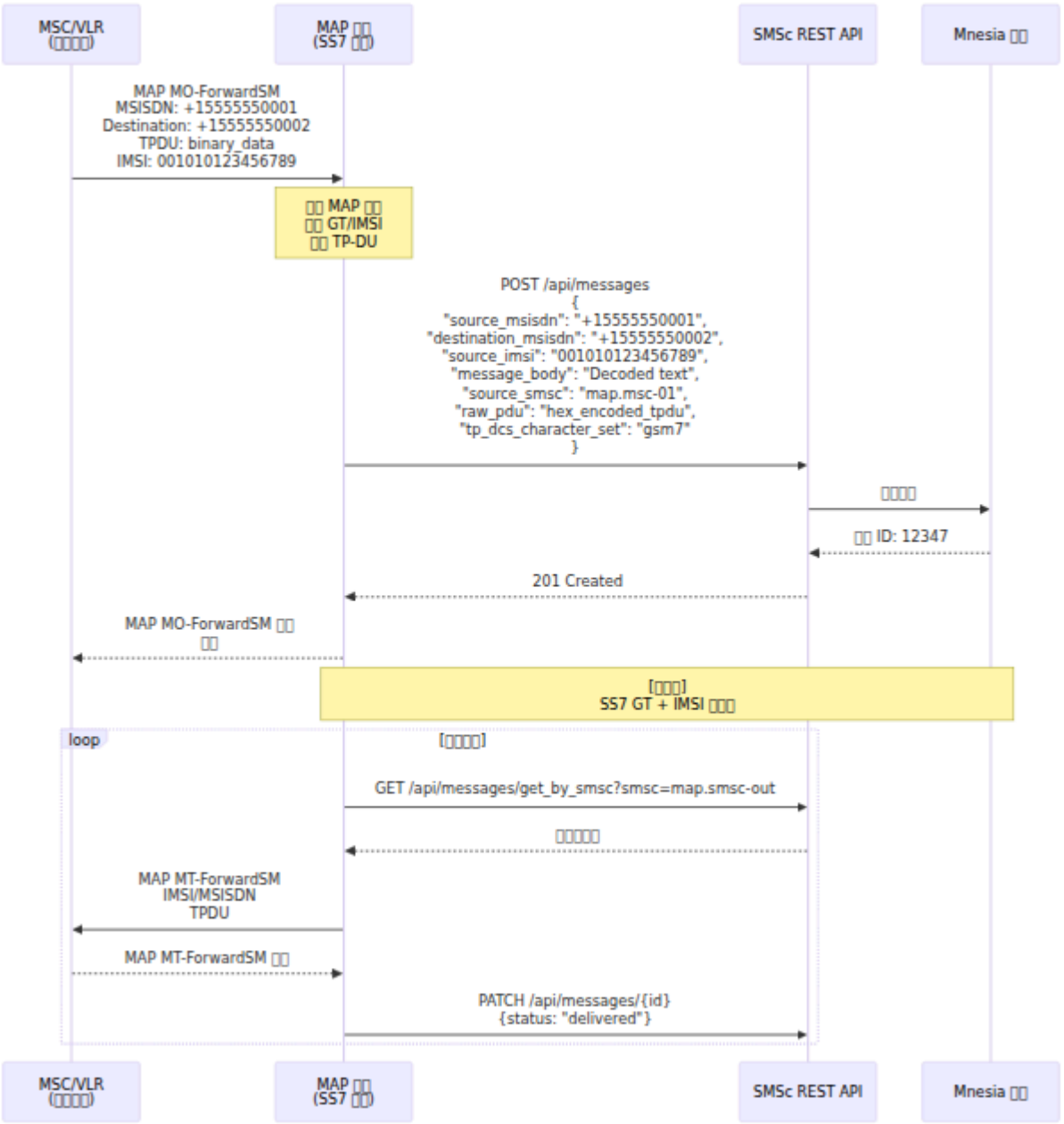
注意：这些端点查询活动的 Mnesia 消息队列，提供对当前正在处理或最近交付的消息的访问（在保留期内）。

查询参数：

- 按源/目的 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）：



直接 SQL 访问：

仅限只读数据库凭据供授权系统使用
SQL 查询访问 cdrs 表（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）
为高效搜索而建立索引的字段：
calling_number（索引） - 源电话号码 called_number（索引） - 目的电话号码
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/数年的数据）。

3. 实时消息馈送（**PubSub**）：

- Phoenix PubSub 集成实时事件
- 消息提交通知
- 消息交付通知
- 消息状态更改事件
- 按标准可配置事件过滤
- WebSocket 支持实时监控

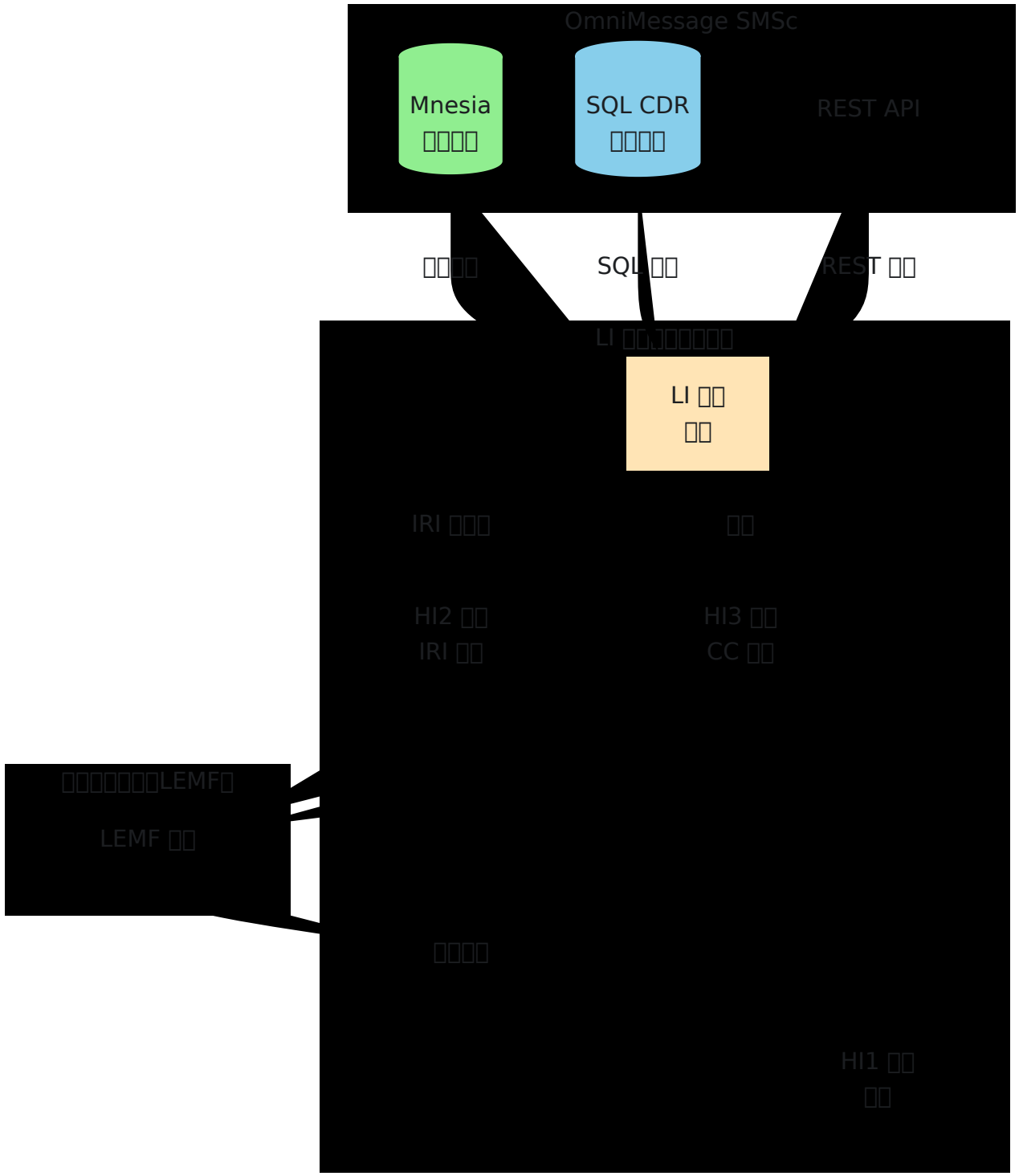
4. 批量导出接口：

- CDR 记录的 CSV 导出
- 用于程序访问的 JSON 导出
- 可配置的导出字段
- 基于时间范围的导出
- 隐私保护导出（可选消息体排除）

ETSI 合法拦截标准接口

OmniMessage SMSc 提供了实施 ETSI 兼容合法拦截接口的基础。SMSc 核心不原生实现 HI1/HI2/HI3 移交接口。它提供所有必要的数据访问点。您可以将这些���问点与外部合法拦截中介功能（LIMF）系统集成。

标准 **ETSI LI** 接口：



主な機能

HI1 送信 - 加入者管理

- 加入者 プロビジョニング、アクティベーション、ディアクティベーション
- 加入者 LEMF → LIMF関連付け
- 加入者
  - 追加/削除/変更通知（MSISDN、IMSI）

设置拦截持续时间和有效期
配置过滤标准（电话号码、时间窗口）
检索拦截状态
与 SMSc 的集成：
LIMF 维护目标列表（授权数据库）
LIMF 查询 SMSc CDR/API 以匹配消息
LIMF 根据 HI1 提供的标准进行过滤

HI2 接口 - IRI（拦截相关信息）交付：

目的：向执法机关交付消息元数据
方向： LIMF → LEMF（单向）
数据格式：符合 ETSI TS 102 232-x 的 XML/ASN.1
来自 SMSc CDR 的内容：
消息 ID
呼叫号码（源 MSISDN）
被叫号码（目的 MSISDN）
IMSI（源和目的，如果可用）
提交时间戳
交付时间戳
消息状态（已交付/失败/过期）
交付尝试
SMSC 网关信息（源/目的）
网络位置（如果可用）
与 SMSc 的集成：
LIMF 查询 CDR 数据库以获取目标电话号码 LIMF 将 CDR 记录转换为 ETSI IRI 格式 LIMF 通过 HI2 将 IRI 交付给 LEMF

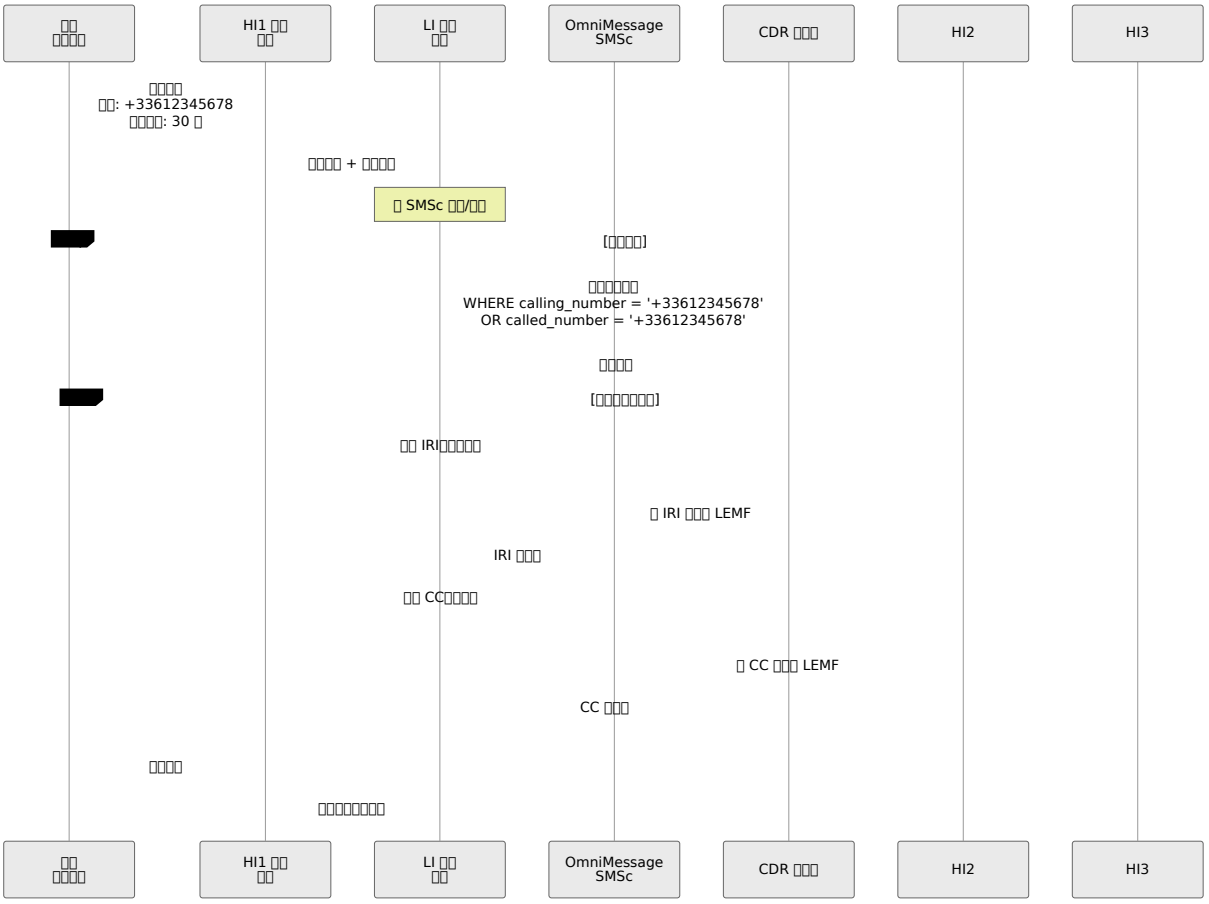
HI3 接口 - CC（通信内容）交付：

目的：向执法机关交付实际消息内容
方向： LIMF → LEMF（单向）
数据格式：符合 ETSI TS 102 232-x
来自 SMSc 的内容：

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

- □ **SMSc □□□□**
  - LIMF □ CDR `message_body` □□□□□□□□□
  - LIMF □□□□□□□□□ PDU ��□
  - LIMF □□□□□□□ ETSI CC □□
  - LIMF □□ HI3 □ CC □□□ LEMF

□□□□□



SMSc □□□□□ LI □□□

SMSc 数据字段	HI2 (IRI)	HI3 (CC)	CDR 表列
消息 ID	□ □□ ID	□ □□	`message_id`
呼叫号码	□ □□□ A	-	`calling_number`
被叫号码	□ □□□ B	-	`called_number`
提交时间	□ □□□	-	`submission_time`
交付时间	□ □□	-	`delivery_time`
状态	□ □□	-	`status`
消息正文	-	□ □□	`message_body`
原始 PDU	-	□ □□□	(Mnesia/CDR)
源 SMSC	□ □□□□	-	`source_smsc`
目的 SMSC	□ □□□□	-	`dest_smsc`
IMSI	□ □□□ ID	-	(通过前端)

LIMF 集成选项:

选项 **1**: 轮询架构

- LIMF 定期查询 CDR 数据库（每 1-60 秒）
- SQL 查询根据 HI1 授权列表中的目标电话号码进行过滤
- 复杂度低，易于实施
- 消息交付与 LI 交付之间有轻微延迟

选项 **2**: 实时馈送架构

- SMSc PubSub 发布消息事件
- LIMF 订阅实时消息流

- LIMF 根据目标列表进行过滤
- 合法拦截的近零延迟
- 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
- 延迟和可靠性的最佳平衡

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���图
- 按目标标识符的 API 过滤

于事件的拦截：

- 所有到/来自特定号码的消息
- 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 密码能力概述

OmniMessage SMSc 实施了加密机制，以保护通信和敏感数据。本节根据 ANSSI 要求记录所有密码能力。

2.2 传输层加密

2.2.1 TLS/SSL 实施

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: 已弃用（不推荐）

实施：

Erlang/OTP SSL/TLS 库（经过密码学验证）
支持 TLS 的 Cowboy Web 服务器 Phoenix 框架 HTTPS 端点

密码套件：

系统使用 Erlang/OTP 的默认安全密码套件选择，包括：

首选 - TLS 1.3：

TLS_AES_256_GCM_SHA384 TLS_AES_128_GCM_SHA256 TLS_CHACHA20_POLY1305_SHA256

支持 - TLS 1.2：

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

安全特性：

通过 ECDHE/DHE 密钥交换实现完美前向保密（PFS）
强大的 Diffie-Hellman 组（2048 位最小）
支持椭圆曲线密码学
支持服务器名称指示（SNI）

证书管理：

支持 X.509 证书
RSA 密钥大小：2048 位最小，4096 位推荐
支持 ECDSA
证书链验证

- 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"
  }
```

 完整配置参考见 **CONFIGURATION.md**

应用：

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

2.3 数据静态加密

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

- 文件系统级加密
- 列级加密（pgcrypto 扩展）

2.3.2 Mnesia 磁盘存储

Mnesia 数据库：

- 磁盘副本存储以实现消息持久性
- 推荐文件系统级加密（LUKS，dm-crypt）
- 通过 Erlang VM 隔离实现内存保护

2.3.3 文件系统加密

敏感数据存储：

- 配置文件：推荐文件系统加密
- 私钥：文件权限（0600）+ 文件系统加密
- 日志文件：可配置的归档日志加密
- CDR 导出：敏感导出的加密存储

密钥存储：

- TLS 证书和密钥存储在 `priv/cert/`
- 具有受限权限的基于文件的密钥库
- 安全的密钥轮换程序

2.4 身份验证和访问控制

2.4.1 API 身份验证

REST API 安全性：

- 强制使用 HTTPS/TLS 传输加密
- 基于头部的身份验证（SMSc 头部用于前端识别）
- 基于 IP 的访问控制（防火���级别）
- 基于证书的客户端身份验证（可选）

前端注册：

- 唯一的前端识别（名称、类型、IP、主机名）
- 基于心跳的身份验证
- 基于过期的会话管理（90 秒超时）
- 前端跟踪和监控

2.4.2 数据库身份验证

数据库访问控制：

- 用户名/密码身份验证
- TLS/SSL 连接支持
- 基于 IP 的连接限制
- 基于角色的访问控制（RBAC）

配置：

```
# config/runtime.exs
config :sms_c, SmsC.Repo,
  username: "omnitouch",
  password: "omnitouch2024",  # 生产环境中应使用强密码
  hostname: "localhost",
  ssl: true  # 启用 TLS 进行数据库连接
```

访问控制建议：

```
-- 为执法访问创建只读用户
CREATE USER 'li_readonly'@'%' IDENTIFIED BY 'secure_password';
GRANT SELECT ON sms_c.cdrs TO 'li_readonly'@'%';

-- 创建没有消息正文访问权限的有限用户
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 密码算法详细信息

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）

使用：

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）

使用场景：

TLS 会话加密
数据库静态加密
可选的消息体加密

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）

使用场景：

TLS 证书认证
数字签名
密钥交换

2.6 SMS 协议安全

2.6.1 SMS 消息编码

字符编码支持：

GSM 7 位（标准 SMS 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 SMS 加密：

SMS 协议不提供端到端加密
消息内容在 SMSc 层可访问
根据需要启用合法拦截

2.6.2 协议安全考虑

SMPP 协议（���部前端）：

- SMPP 层的用户名/密码认证
- 支持 TLS（SMPP over TLS）
- 绑定认证

IMS 协议（外部前端）：

- 基于 SIP 的消息传递
- SIP 认证机制
- 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 协议认证
- 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
- 消息流追踪
- 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）
- 网络分析的 PCAP 捕获支持

- 法医的数据库查询日志
- Prometheus 指标导出

2.7.2 漏洞评估考虑

已知限制：

- SMS 协议本质上未加密（设计使然，支持合法拦截）
- 配置文件中的数据库凭据（应使用秘密管理）
- 自签名证书支持（仅限开发/测试）

安全加固建议：

- 使用强大的 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）
- 定期更新 Erlang/OTP 和依赖项的安全性
- 对 API 端口实施防火墙限制
- 前端访问的 IP 白名单

安全测��：

- 定期依赖项漏洞扫描
- 渗透测试支持
- 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

# 生产环境：从受信任的 CA 获取证书
```

随机数生成：

- Erlang/OTP CSPRNG（密码安全伪随机数生成器）
- 系统熵池（/dev/urandom）
- 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：

- 权限受限的文件系统（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：

- 数据库凭据的带外密钥交换
- 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

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

2.9.1 SMS 通过 SS7/MAP 协议的合规性

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**：字母表和特定语言信息 - 字符编码（GSM7、UCS-2）
- **3GPP TS 29.002**：移动应用部分（MAP）规范 - SMS 的 SS7 信令
- **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）；电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 SMS 通过 IMS 协议的合规性

3GPP IMS 标准：

- **3GPP TS 23.228**：IP 多媒体子系统（IMS） - 第 2 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**：支持通过 IP 网络的 SMS - 第 3 阶段规范
- **3GPP TS 23.204**：支持通过通用 3GPP IP 访问的短消息服务（SMS） - 第 2 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 的服务的访问安全（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）；认证框架（AF）
- **ETSI TS 133 203**：基于 IP 的服务的访问安全

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 的私有扩展以确保身份
- **RFC 5765**：点对点系统中的安全问题和解决方案

IMS 的合法拦截：

- **ETSI TS 102 232-5**：合法拦截（LI）；交接规范 - 第 5 部分：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**：增强功能的扩展 SMPP 协议

SMPP 安全指南：

- **SMPP** 上的 **TLS**：SMPP 连接的传输层安全（SMPP over TLS）
- **SMPP** 绑定认证：系统 ID 和密码认证
- 基于 **IP** 的访问控制：对 SMPP 连接的网络级限制

互操作性标准：

- **GSM 03.40**（**ETSI TS 100 901**）：SMS 点对点（PP）的技术实现
- **GSM 03.38**（**ETSI TS 100 900**）：字母表和特定语言信息
- **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 No. 5（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**：传输层安全（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 抗密码分析能力

2.10.1 设计原则

抵御密码分析：

不使用自定义/专有加密算法
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
停用弱算法
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 TLS 会话）
数据库凭据轮换政策

监控和检测：

失败的认证日志
证书到期监控
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 访问由 IP/认证限制
所有访问的审计日志

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 到 SQL 的自动归档
自动清除旧 CDR（基于 cron）

隐私保护：

交付后删除消息体的选项
消息体从 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 消息保留在归档前的时间
  message_retention_hours: 24,

  # 为隐私在交付后删除消息体
  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
```

 请参阅 **CONFIGURATION.md** 以获取所有保留设置

3.3 执法的交接接口

标准接口:

1. REST API 访问:

- 用于消息检索的 HTTPS 端点
- JSON 格式数据交换
- 认证和授权
- 按目标标准进行查询过滤

2. 直接数据库访问:

- 只读 SQL 凭据
- 标准 SQL 查询
- CDR 表访问
- 索引搜索功能

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）：

消息体（文本内容）
原始 PDU 数据
编码信息
多部分消息组装

导出示例：

```
# 执法的 CSV 导出
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. 系统安全和完整性

4.1 应用安全

Elixir/Erlang 安全:

- 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：

最小暴露端口：
8443（HTTPS REST API）
8086（HTTPS 控制面板）
建议进行防火墙配置
前端访问的 IP 白名单
面向互联网的服务的 DMZ 部署

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：

失败的认证尝试
异常消息模式
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

 完整的监控文档请参阅 **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 在节点之间的复制
- 自动故障转移
- 节点发现和加入

数据冗余：

- 所有集群节点上的 Mnesia 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. 文档参考

5.1 技术手册

项目库中的可用文档：

README.md - 系统概述、架构和功能 CONFIGURATION.md - 完整配置参考 API_REFERENCE.md - REST API 文档 OPERATIONS_GUIDE.md - 操作程序和监控 CDR_SCHEMA.md - 通话详细记录数据库架构 sms_routing_guide.md - SMS 路由配置 number_translation_guide.md - 号码标准化 METRICS.md - Prometheus 指标和监控 TROUBLESHOOTING.md - 常见问题和解决方案

5.2 安全认证

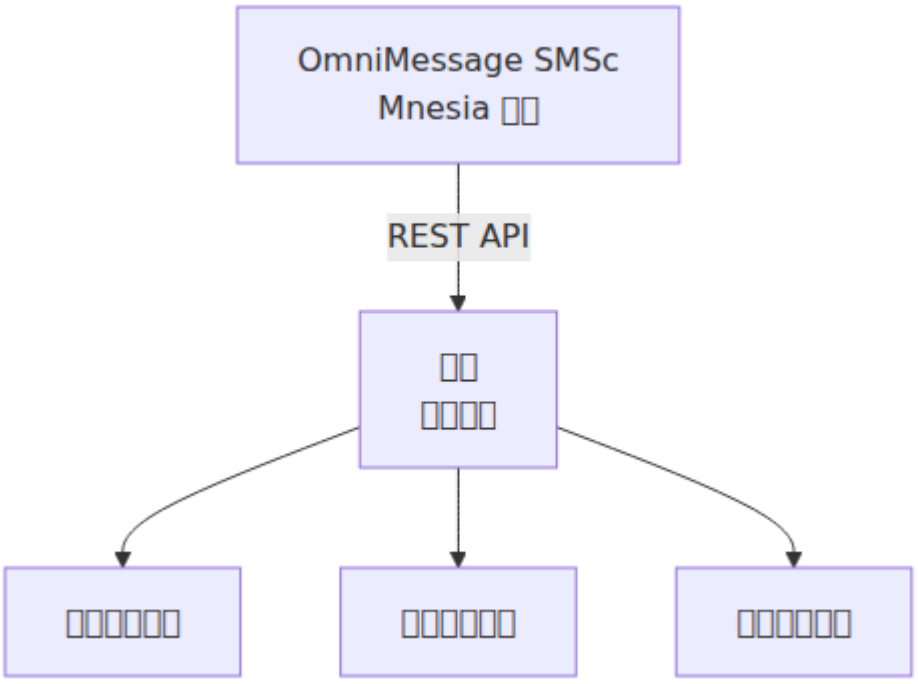
渗透测试报告： [根据请求提供]
安全审计报告： [根据请求提供]
漏洞评估： [根据请求提供]
Erlang/OTP 加密验证：行业标准加密库

5.3 合规性文档

ANSSI R226 授权请求：本文件
合法拦截合规性：根据法国电信法规的要求
数据保护合规性：消息数据的 GDPR 考虑

6. 联系信息

供应商/运营商信息：



- 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

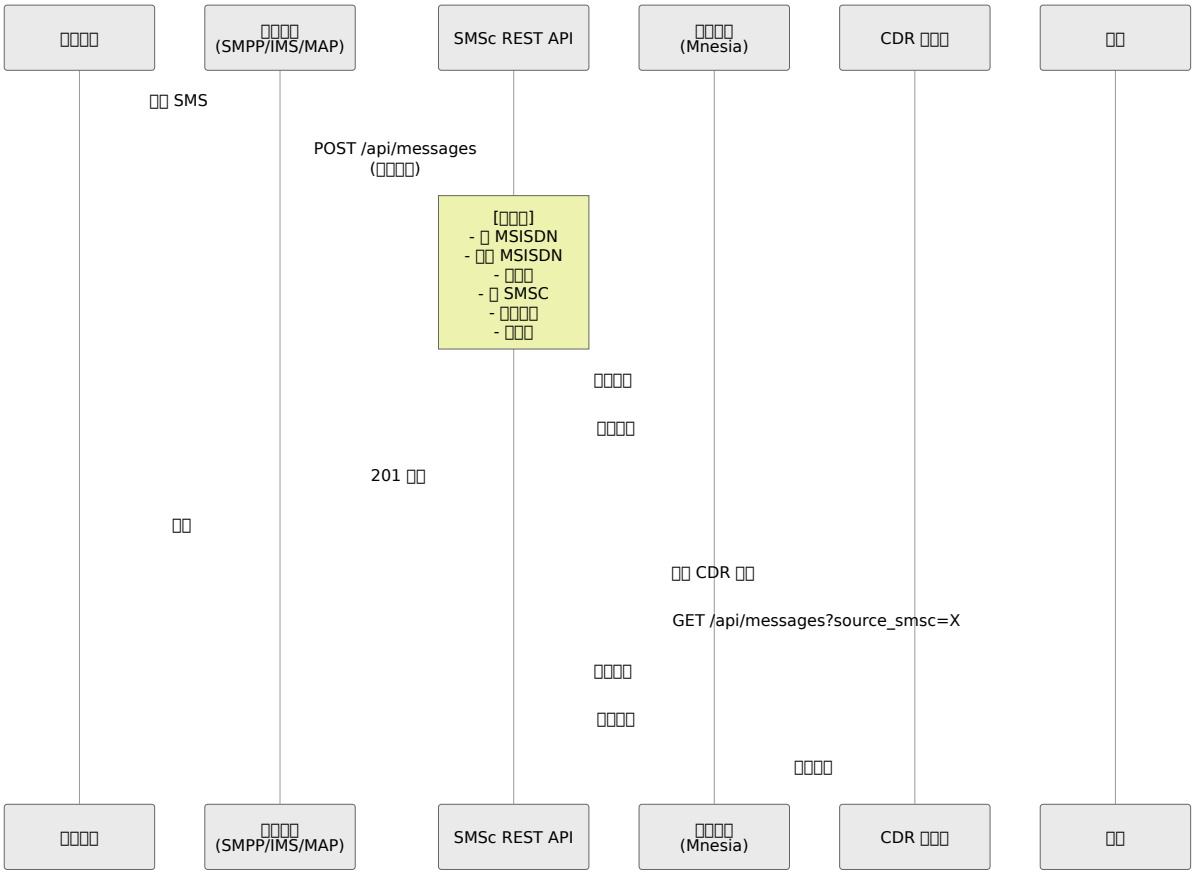
附录

附录 A: 带有拦截点的 SMS 消息流

A.1 出站 SMS 流（移动终止）

图例： [拦截点] = 捕获和存储合法拦截数据的点

A.2 □□ SMS □□□□□□□



□□ B□□□□□□□ CDR □□

OmniMessage SMSc □ SQL □□□□□MySQL □ PostgreSQL□□□□□□□

字段名称	类型	描述	拦截相关性
`id`	BIGINT	自增主键	记录标识符
`message_id`	BIGINT	唯一消息标识符	会话关联
`calling_number`	VARCHAR(255)	源 MSISDN	目标跟踪的主要标识符
`called_number`	VARCHAR(255)	目标 MSISDN	目标目的地跟踪
`source_smsc`	VARCHAR(255)	源网关标识符	协议/网关跟踪
`dest_smsc`	VARCHAR(255)	目标网关标识符	路由信息
`origin_node`	VARCHAR(255)	Erlang 集群节点（发起）	系统跟踪
`destination_node`	VARCHAR(255)	Erlang 集群节点（交付）	集群分布
`submission_time`	DATETIME	消息提交时间戳	事件时间线
`delivery_time`	DATETIME	消息交付时间戳	交付确认
`expiry_time`	DATETIME	消息过期时间戳	失败交付跟踪
`status`	VARCHAR(50)	消息状态（已交付/过期/失败/拒绝）	最终处置
`delivery_attempts`	INT	交付尝试次数	重试分析
`message_parts`	INT	SMS 段数	拼接消息跟踪
`deadletter`	BOOLEAN	死信队列标志	永久失败指示
`message_body`	TEXT	SMS 消息内容	消息内容（可选）
`inserted_at`	DATETIME	CDR 创建时间戳	审计跟踪
`updated_at`	DATETIME	CDR 更新时间戳	修改跟踪

 完整的模式文档和 **SQL** 示例在 **CDR_SCHEMA.md**

B.2 CDR 查询示例用于合法拦截

查询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;
```

查询时间窗口内的消息：

```
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;
```

导出到 **CSV** 供执法使用：

```
.mode csv
.output /tmp/interception_report.csv
SELECT message_id, calling_number, called_number, message_body,
       submission_time, delivery_time, status
FROM cdrs
WHERE calling_number = '+33612345678'
ORDER BY submission_time DESC;
```

B.3 CDR 数据库访问方法

1. 直接 **SQL** 访问：

- 只读数据库凭据
- 标准 SQL 查询
- JDBC/ODBC 连接
- 数据库客户端工具（MySQL Workbench，pgAdmin）

2. REST API 访问:

- 未来增强: 用于 CDR 查询的 REST API
- JSON 格式响应
- 认证和授权
- 查询参数过滤

3. 批量导出:

- 通过 mysql/psql 命令行导出 CSV
- 自动化导出脚本
- 通过 cron 定时导出

B.4 CDR 保留和隐私

保留配置:

```
# config/runtime.exs
config :sms_c,
  # 交付后删除消息正文（隐私模式）
  delete_message_body_after_delivery: false,  # true 表示隐私

  # 启用/禁用 CDR 写入
  cdr_enabled: true,

  # Mnesia 到 CDR 存档设置
  message_retention_hours: 24
```

隐私选项:

- 消息正文可以在交付后设置为 NULL
- 数据库表加密（MySQL ENCRYPTION='Y'）
- 列级访问限制
- 用于分析的掩码导出

附录 C: 拦截的 REST API 参考

C.1 消息检索端点

获取所有消息：

```
GET /api/messages
Authorization: Bearer <token>
```

按 **SMSC** 获取消息：

```
GET /api/messages/get_by_smsc?smsc=gateway-name
```

获取特定消息：

```
GET /api/messages/{id}
```

响应格式：

```
{
  "status": "success",
  "data": [
    {
      "id": 12345,
      "message_id": 12345,
      "source_msisdn": "+33612345678",
      "destination_msisdn": "+33687654321",
      "message_body": "消息内容在这里",
      "source_smsc": "ims.gateway",
      "dest_smsc": "smpp.provider",
      "status": "delivered",
      "delivery_attempts": 1,
      "inserted_at": "2025-11-29T10:30:00Z",
      "deliver_time": "2025-11-29T10:30:05Z",
      "expires": "2025-11-30T10:30:00Z"
    }
  ]
}
```

 完整的 **API** 文档在 **API_REFERENCE.md**

C.2 过滤和搜索

查询参数：

- `source_smsc` - 按源网关过滤
- `dest_smsc` - 按目的网关过滤
- 时间范围过滤（通过 CDR 数据库查询）
- 状态过滤（通过 CDR 数据库查询）

未来增强：

- 用于 CDR 查询的 REST API
- 按电话号码的高级过滤
- 日期范围过滤
- 对消息正文的全文搜索

附录 D：配置示例

D.1 TLS 证书配置

生成 TLS 证书：

```
# 生成 4096 位 RSA 私钥
openssl genrsa -out priv/cert/omnitouch.pem 4096

# 生成证书签名请求
openssl req -new -key priv/cert/omnitouch.pem -out
priv/cert/omnitouch.csr \
  -subj "/C=FR/ST=IDF/L=Paris/O=Omnitouch/CN=smsc.example.com"

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

# 生产：将 CSR 提交给受信任的 CA 进行签名
```

在应用中配置：

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

D.2 数据库加密配置

MySQL 表加密:

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

-- 验证加密状态
SELECT TABLE_NAME, CREATE_OPTIONS
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'sms_c' AND TABLE_NAME = 'cdrs';
```

PostgreSQL 连接加密:

```
# config/runtime.exs
config :sms_c, SmsC.Repo,
  username: "omnitouch",
  password: "secure_password",
  hostname: "localhost",
  database: "sms_c",
  ssl: true,
  ssl_opts: [
    verify: :verify_peer,
    cacertfile: "/path/to/ca.crt",
    certfile: "/path/to/client.crt",
    keyfile: "/path/to/client.key"
  ]
```

D.3 隐私配置

启用消息正文删除:

```
# config/runtime.exs
config :sms_c,
  # 在成功交付后删除消息正文
  delete_message_body_after_delivery: true,

  # 在 UI 中隐藏消息正文
  hide_message_body_in_ui: true,

  # 在导出中隐藏消息正文
  hide_message_body_in_export: true
```

创建隐私保护的数据库视图：

```
-- 创建不包含消息正文的视图用于一般分析
CREATE VIEW cdrs_metadata AS
SELECT
  id, message_id, calling_number, called_number,
  source_smsc, dest_smsc, origin_node, destination_node,
  submission_time, delivery_time, expiry_time,
  status, delivery_attempts, message_parts, deadletter,
  inserted_at, updated_at
FROM cdrs;

-- 授予分析用户访问权限
GRANT SELECT ON cdrs_metadata TO 'analytics'@'%';
```

附录 E：术语表

监管和标准机构

- **ANSSI:** 法国国家信息系统安全局
- **ETSI:** 欧洲电信标准化协会
- **3GPP:** 第三代合作伙伴计划 - 移动电信标准
- **IETF:** 互联网工程任务组 - 互联网标准机构

电信术语

- **SMSc:** 短信服务中心 - 短信消息路由和交付的中央系统
- **SMPP:** 短消息对等协议 - 短信交换的行业标准

ESME: 外部短消息实体 - SMPP 客户端应用
IMS: IP 多媒体子系统 - 用于多媒体服务的全 IP 网络架构 SIP: 会话发起协议 - IMS 消息的信令协议
P-CSCF: 代理呼叫会话控制功能 - IMS 网络入口点 S-CSCF: 服务呼叫会话控制功能 - IMS 会话控制 HSS: 家庭用户服务器 - IMS 用户数据库 SS7/MAP: 信令系统 7 / 移动应用部分 - 传统移动信令协议 MSC: 移动交换中心 - 电路交换网络元素
VLR: 游客位置注册 - 用户位置数据库
GT: 全球标题 - SS7 地址方案
SCCP: 信令连接控制部分 - SS7 网络层
TCAP: 事务能力应用部分 - SS7 应用层
MSISDN: 移动站国际用户目录号码 - 电话号码
IMSI: 国际移动用户身份 - 唯一用户标识符
E.164: 电话号码的国际编号计划
ENUM: E.164 数字映射 - 基于 DNS 的电话号码到 URI 映射 PDU: 协议数据单元 - 二进制编码的短信消息
TP-DCS: 转移协议数据编码方案 - 短信编码规范
TP-DU: 转移协议数据单元 - 短信特定的 PDU 格式
UDH: 用户数据头 - 用于连接/特殊短信的头部
TON/NPI: 号码类型 / 编号计划指示符 - 号码格式分类
GSM7: GSM 7 位默认字母表 - 短信字符编码标准 UCS-2: 通用字符集 2 字节 - 短信的 Unicode 编码

系统组件

Mnesia: Erlang 分布式数据库系统 - 内存/磁盘存储 CDR: 通话详细记录 - 消息的计费和分析记录
REST API: 表现层状态转移 - 基于 HTTP 的 API 架构 Phoenix: Elixir 网络框架
Cowboy: Erlang HTTP 服务器
Ecto: Elixir 数据库封装和查询语言
PubSub: 发布-订阅消息模式

合法拦截

- **LI:** 合法拦截 - 对电信的法律监控
- **LIMF:** 合法拦截中介功能 - 在电信网络和执法之间接口的系统
- **LEMF:** 执法监控设施 - 接收拦截数据的执法系统
- **IRI:** 拦截相关信息 - 用于执法的通话/消��元数据
- **CC:** 通信内容 - 实际消息内容
- **HI1** 接口**:** ETSI LI 行政交接接口 - 逮捕令提供和目标激活
- **HI2** 接口**:** ETSI LI 交接接口用于 IRI 交付 - 向执法交接元数据
- **HI3** 接口**:** ETSI LI 交接接口用于 CC 交付 - 向执法交接内容
- **R226:** 法国刑法第 R226-3 和 R226-7 条款，管理拦截设备
- **ETSI:** 欧洲电信标准化协会 - 定义 LI 标准
- **ETSI TS 102 232:** 合法拦截交接接口的技术规范

消息处理

- **MT:** 移动终止 - 发往移动用户的出站消息
- **MO:** 移动发起 - 从移动用户发来的入站消息
- **DLR:** 交付回执 - 消息交付确认
- 死信**:** 在所有重试尝试后未能交付的消息
- 指数退避**:** 增加重试延迟（2分钟，4分钟，8分钟等）

安全和加密

- **TLS:** 传输层安全性 - 加密协议
- **PFS:** 完美前向保密 - 会话密钥安全的密码学属性
- **AES:** 高级加密标准
- **RSA:** Rivest-Shamir-Adleman - 公钥密码学
- **ECDSA:** 椭圆曲线数字签名算法
- **SHA:** 安全哈希算法
- **X.509:** 证书标准
- **CA:** 证书颁发机构
- **CSPRNG:** 密码安全伪随机数生成器

数据库术语

MySQL: 开源关系数据库
PostgreSQL: 开源对象关系数据库 TDE: 透明数据加密
RBAC: 基于角色的访问控制

文档版本： 1.0
日期： 2025-11-29
准备对象： ANSSI R226 授权申请
文档分类：监管合规 - 机密

批量消息 / 活动

← 返回文档索引

概述

活动功能向多个接收者发送单条纯文本消息，同时保护 SMSC 避免过载。您只需上传一次命名的 接收者列表，然后针对该列表运行一个或多个 活动。每个活动以可配置的速率将消息滴入正常消息队列，并报告实时进度和投递统计信息。

两个可重用对象：

对象	描述
活动列表	一组命名的可重用接收者 MSISDN，每个接收者最多可包含四个可选变量。上传一次（JSON 数组或 CSV），在活动中重复使用。
活动	一次发送运行：一个列表（或 □□□□□□□）+ 一个消息模板，滴入队列。

所有内容都通过 HTTP API 公开。所有状态存储在 Mnesia 中，与 SMSC 的其余部分一起。

定向：列表和活跃订阅者默认值

活动的接收者来自两个来源之一：

- 接收者列表（`list_id`）- 您上传 MSISDN。
- 根本没有列表 - 如果您创建一个 没有 `list_id` 的活动，它将针对 所有当前活跃（注册）订阅者 - 每个在活动开始时在位置存储中有未过期注册的 MSISDN。这是接触“当前网络上的所有人”的最简单方法。

每个接收者的变量和模板

每个列表接收者可以携带最多四个变量 - **`var1`**, **`var2`**, **`var3`**, **`var4`** - 用于诸如名字、姓氏或计划名称等信息。活动消息是一个模板：`{{ var1 }}`...`{{ var4 }}`（和 `{{ msisdn }}`）在消息发送时为每个接收者拼接。大括号内的空格是可选的，任何没有值的占位符将呈现为空字符串。

> 模板设计故意简约 - 仅变量替换（没有条件或过滤器），因□没有额外的依赖关系，也没有接收者收到的意外内容。

通过 **CSV** 上传变量 - MSISDN 列是命名为 `msisdn`/`destination_msisdn`/`number`/`phone` 的列（如果没有命名，则为第一列）；其他每一列按列顺序成为 `var1`…`var4`：

```
msisdn,first,last,plan
12025550101,Alice,Smith,Gold
12025550102,Bob,Jones,Silver
```

通过 **JSON** 上传变量 - 接收者对象可以直接设置变量：

```
{ "name": "VIPs", "recipients": [
  { "msisdn": "12025550101", "var1": "Alice", "var2": "Gold" }
] }
```

示例模板：

```
Hi {{ var1 }}, your {{ var2 }} plan renews soon. Reply STOP to opt
out.
```

对于上述接收者 `12025550101`（`var1=Alice`，`var2=Gold`），这将发送：`Hi Alice, your Gold plan renews soon. Reply STOP to opt out.`

受众过滤器

当活动使用列表时，`audience` 标志决定谁实际上会收到消息：

受众	行为
`active_only`（默认）	仅列表成员在活动开始时 注册（未过期位置）。其他所有人被记录为 `skipped_inactive`，并且不会发送。
`all`	列表中的每个成员，无论注册状态。

注册检查是 在开始时拍摄的快照 - 后来注册的订阅者不会被选中。当活动 没有列表 时，接收者集已经是活跃订阅者基础，因 `audience` 隐式为 `active_only`，任何提供的值将被忽略。

生命周期

1. **draft** - 创建，尚无目标。
2. **start** - 活动快照其目标（来自其列表或活跃订阅者基础），然移动到 **running**。如果没有要发送的内容，它将直接进入 **completed**。
3. **running** - 滴水工作者将消息提交到队列。
4. **pause / resume** - 暂停/继续滴水。
5. **completed** - 每个目标都已滴入队列（投递随后继续异步进行，投递统计信息不断更新）。
6. **cancelled** - 永久停止。

删除活动

删除活动 停止它但保留记录 - 它被取消而不是清除，因其历史和最终统计信息保持可见。停止和完成的活动在超过保留 TTL（`campaign_retention_hours`，默认 7 天）后会自动删除。

滴水速率（过载保护）

每个活动都有一个 `drip_tps`（每秒提交到队列的消息数，默认来自配置）。后台工作者在固定间隔唤醒，并在每个滴水周期内提交最多 `drip_tps × interval` 消息 - 因一百万个接收者的活动不会淹没队列。速率是 按活动；多个正在运行的活动各自以自己的速率滴水。

进度和投递统计信息

每个活动对象包括一个实时的 `stats` 块（轮询 `GET /api/campaigns/:id`）：

字段	意义
`total`	在开始时快照的目标
`pending`	尚未滴入队列
`queued`	已提交，等待投递报告
`delivered`	确认已投递
`failed`	投��失败/死信
`skipped_inactive`	因为在开始时未注册而被跳过
`dispatched`	`queued + delivered + failed`
`progress_percent`	不再 `pending` 的目标百分比
`delivery_percent`	已投递消息的百分比

投递和失败计数会随着投递报告通过正常的 SMSC 投递路径流回而自动更新。每个接收者的详细信息（以及按状态过滤，例如所有失败的接收者）可以通过 `GET /api/campaign_targets/:id?state=failed` 获取。

配置

```
config :sms_c, :campaigns,
  # 启动滴水工作者（设置为 false 以暂停所有活动投递）
  enabled: true,
  # 滴水周期之间的间隔（毫秒）
  drip_tick_ms: 1000,
  # 当活动未设置自己的 drip_tps 时，每个活动的每秒消息数
  default_drip_tps: 50,
  # 当活动未设置 validity_hours 时的默认消息有效期
  message_validity_hours: 72,
  # 停止/完成的活动在被清除之前持续的时间（小时）
  campaign_retention_hours: 168
```

有效性和定时投递

活动可以在创建时设置：

- **`validity_hours`** - 消息有效期。每条提交的消息都带有 `expires` 标记，因□未投递的消息会过期，而不是在队列中停留。默认为配置中的 `message_validity_hours`。
- **`deliver_after`** - 一个可选的 ISO8601 ��间（必须在未来）用于定时发送。

定时投递如何与滴水交互。活动仍然在 `drip_tps` 下 □□ 开始滴水，但每条消息的时钟会向前移动一个固定的偏移量 `delta = deliver_after − started_at`。因□，在实时 t 下滴入的消息标记为 `send_time`/`deliver_after = t + delta` 和 `expires = t + delta + validity_hours`。由于消息以 `drip_tps` 的速度实时插入，它们的每条消息 `deliver_after` 值将在 **`deliver_after`** 开始时以 **`drip_tps`** 交错 - 投递是速率限制的，而不是在定时瞬间的洪流。如果没有 `deliver_after`，`delta = 0`，消息将立即滴入和投递。

典型工作流程

```
# 1. 上传一个命名的接收者列表，带有每个接收者的变量（CSV 或 JSON）
curl -k -X POST https://smsc:8443/api/campaign_lists \
  -H 'Content-Type: application/json' \
  -d '{"name":"VIP customers",

"csv":"msisdn,first,plan\n12025550101,Alice,Gold\n12025550102,Bob,Sil
# => {"id": 1, "recipient_count": 2, ...}

# 2. 针对该列表创建一个活动，模板中包含变量
curl -k -X POST https://smsc:8443/api/campaigns \
  -H 'Content-Type: application/json' \
  -d '{"name":"Renewal notice","message":"Hi {{ var1 }}, your {{ var2
renews soon.",
       "source_msisdn":"12345","source_smsc":"IMS_SMSC",
       "list_id":1,"audience":"active_only","drip_tps":50}'
# => {"id": 7, "status": "draft", ...}

# 3. 启动它
curl -k -X POST https://smsc:8443/api/campaigns/control \
  -H 'Content-Type: application/json' \
  -d '{"campaign_id":7,"action":"start"}'

# 4. 轮询进度 / 投递统计
curl -k https://smsc:8443/api/campaigns/7
# => {... "stats":
{"total":..,"delivered":..,"failed":..,"progress_percent":..}}
```

要向 所有活跃订阅者 发送消息，请在步骤 2 中省略 `list_id`（并完全跳过步骤 1）。

另请参阅

- 活动 API 参考
- SMS 路由指南 - 如何路由/投递滴入的消息
- 配置参考

CDR（通话详单）架构参考
← 返回文档索引

用于长期消息存储、计费和分析的CDR数据库表的完整参考。

目录

概述
表架构
字段描述
SQL示例
索引
按数据库的数据类型
隐私考虑
保留和归档
计费集成

概述

cdrs 表存储系统处理的所有SMS消息的通话详单。系统在消息达到终态时写入CDR：delivered、expired、 failed或 rejected。系统将不平衡的balance_rejected记录为 rejected。保留时间清理作为任何在 Mnesia中老化而没有实时写入的后备。有关完整写入路径和幂等性保证，请参见 CDR何时被写入。

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
  UNIQUE KEY cdrs_origin_node_message_id_index (origin_node,
```

```
message_id),
  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)
);
```

> `(origin_node, message_id)` 唯一键为每个CDR提供了一个在集群中全局唯一的身份。它还使CDR写入具有幂
> ���性。近实时写入和保留后备都可以尝试插入。第二次插入通过 `ON CONFLICT DO NOTHING` 是一个无操作。

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 UNIQUE INDEX cdrs_origin_node_message_id_index ON
cdrs(origin_node, message_id);
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消息队列的消息标识符。由每个节点计数器生成（持久化在Mnesia中，因它可以在重启后存活），因仅在单个节点内唯一。对于跨多节点集群的全局唯一键，使用`(origin_node, message_id)`组合 - 这一对由唯一索引强制执行，并且是计费系统推荐的连接/去重键。

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	传递（���尝试传递）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”[对于故障排除和负载分配分析非常有用。
`destination_node`	VARCHAR(255)	YES	消息传递的Erlang节点名称（如果与源不同）。对于单点部署或消息从未传递的情况为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	消息成功传递的时间。如果消息过期、失败或被拒绝，则为NULL。
`expiry_time`	DATETIME	YES	消息过期（变得不可传递）的时间。如果消息已传递或仍在待处理状态，则为NULL。

传递持续时间计算：

```
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传递尝试次数。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或更多。如果未知则为NULL。
`deadletter`	BOOLEAN	NO	系统是否将消息移至死信队列。TRUE表示系统无法传递消息并已耗尽所有重试。默认值：FALSE

状态值：

状态	描述	可计费	传递时间
`delivered`	成功传递给接收者	是	已设置
`expired`	超过有效期未交付	取决于计费政策	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` 以保护隐私，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中以便合规/归��
- 隐私模式：当 `delete_message_body_after_delivery: true` 时，消息体设置为NULL
- 合规模式：以加密或哈希形式存储消息体（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;
```

已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;
```

客户的可计费消息（按拨打号码前缀）：

```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：

```
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**）：

```
-- 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;
```

索引

创建以下索引以优化常见查询：

索引名称	列	目的
`PRIMARY`	`id`	主键，确保唯一记录
`cdrs_origin_node_message_id_index`	`origin_node, message_id`	唯一。在集群中全局唯一的CDR身份；使写入具有幂等性
`idx_cdrs_message_id`	`message_id`	按原始消息ID查找CDR
`idx_cdrs_calling_number`	`calling_number`	查找特定发送者的消息
`idx_cdrs_called_number`	`called_number`	查找特定接收者的消息
`idx_cdrs_status`	`status`	按交付状态过滤
`idx_cdrs_submission_time`	`submission_time`	基于时间的查询，计费周期
`idx_cdrs_dest_smsc`	`dest_smsc`	路由性能分析

附加索引建议

对于高流量部署，考虑以下附加索引：

计费查询的复合索引：

```
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
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���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,
  # 成功交付后删除消息体
  delete_message_body_after_delivery: true,

  # 在UI中隐藏消息体
  hide_message_body_in_ui: true,

  # 在导出中隐藏消息体
  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）或文件系统级别的加密。

4. 访问控制

限制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限:

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

为计费创建一个单独的费率表:

```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;
```

导出到计费系统

内置**CSV**导出（推荐）：

SMS-C可以直接将CDR呈现为RFC 4180 CSV文件，而无需通过SQL客户端。在发布中，通过远程控制驱动：

```
bin/omnimessage rpc
'SmsC.Messaging.CDR.export_csv_to_file("/tmp/cdrs-202607.csv",
since: ~U[2026-07-01 00:00:00Z], until: ~U[2026-08-01 00:00:00Z],
status: :delivered)'
```

选项：`:since`、`:until`（均为`DateTime`）、`:status`（原子/字符串或列表，例如`[:delivered, :failed]`）和`:limit`。返回值为`{:ok, {path, row_count}}`。`message_body`列遵循

hide_message_body_in_export 设置 - 将其设置为 true 以发出 [REDACTED] 而不是消息内容。一个配套的导出函数返回相同的CSV作为字符串，如果你想将其流式传输到其他地方。

直接SQL客户端导出（替代方案）：

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

所有 SMS-C 配置选项的完整参考，包含常见部署场景的示例。

目录

配置文件
数据库配置
API 配置
Web UI 配置
联邦配置
消息队列配置
计费配置
Diameter Sh (HSS) 配置
ENUM 配置
号码翻译配置
合法拦截 (X1/X2/X3)
路由配置
性能调优配置
日志配置
常见配置场景

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
特定环境设置

config/prod.exs (可选)

特定于生产的覆盖。

最佳实践：在 runtime.exs 中使用环境变量来存储敏感值，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（通话详单）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	与 MySQL 兼容，开源
PostgreSQL	13+	`Ecto.Adapters.Postgres`	5432	高级功能，支持 JSON

注意：Mnesia 会自动用于操作数据（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。

CDR 写入时机

CDR 在两个时点写入 SQL 数据库：

1. 近实时，在终端��态时 - 当消息达到最终结果（`delivered`、`expired`、`failed`、`rejected` 或 `balance_rejected`）时，CDR 会异步写入。这是主要路径，保持计费的实时性，而无需等待消息保留。
2. 保留时间后备 - 当保留工作者从 Mnesia 中删除过期消息时，首先会导出 CDR。对于任何未达到终端状态的消息，记录为 `expired`。工作者仅在 CDR 导出成功后才删除消息。如果 CDR 数据库短暂不可用，则不会丢失任何数据：工作者将在下次清理时重试导出。

这两条路径都是幂等的。每个 CDR 使用唯一的 `(origin_node, message_id)` 对作为其键。写入使用 `ON CONFLICT DO NOTHING`。因□，后备不会重复近实时写入已覆盖的消息。

每个节点计数器生成 `message_id`（持久化在 Mnesia 中，因□在重启后仍然有效）。因□，`message_id` 仅在单个节点内唯一。当多个 SMS-C 节点写入共享 CDR 数据库时，`(origin_node, message_id)` 一起构成全局唯一键。请参见 CDR 架构参考。设置 `cdr_enabled: false` 可完全禁用 CDR 写入。

CSV / 文本文件导出

CDR 可以从 SQL 数据库导出到 RFC 4180 CSV 文件。在发布版本中（`mix` 不可用时），通过远程控制台驱动：

```
bin/omnimessage rpc
'SmsC.Messaging.CDR.export_csv_to_file("/tmp/cdrs-202607.csv",
since: ~U[2026-07-01 00:00:00Z], until: ~U[2026-08-01 00:00:00Z])'
```

支持的选项：`:since`、`:until`（均为 `DateTime`）、`:status`（原子/字符串或列表）和 `:limit`。`message_body` 列遵循 `hide_message_body_in_export` 设置（启用时显示为 `[REDACTED]`）。

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")
```

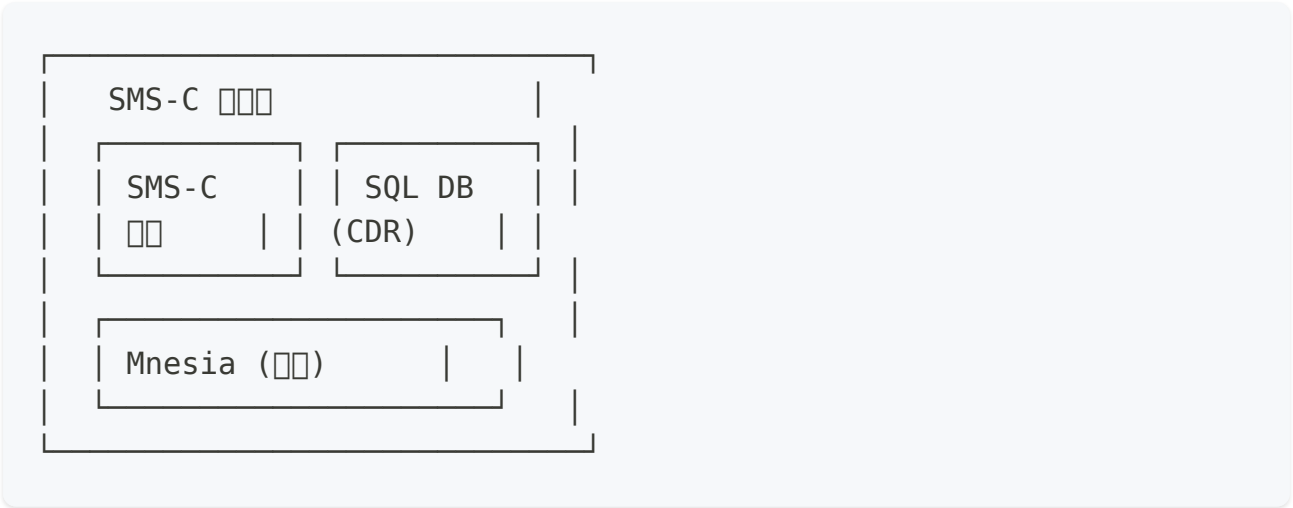
PostgreSQL 配置

```
# 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
- 易于设置复制



PostgreSQL - 如果您需要:

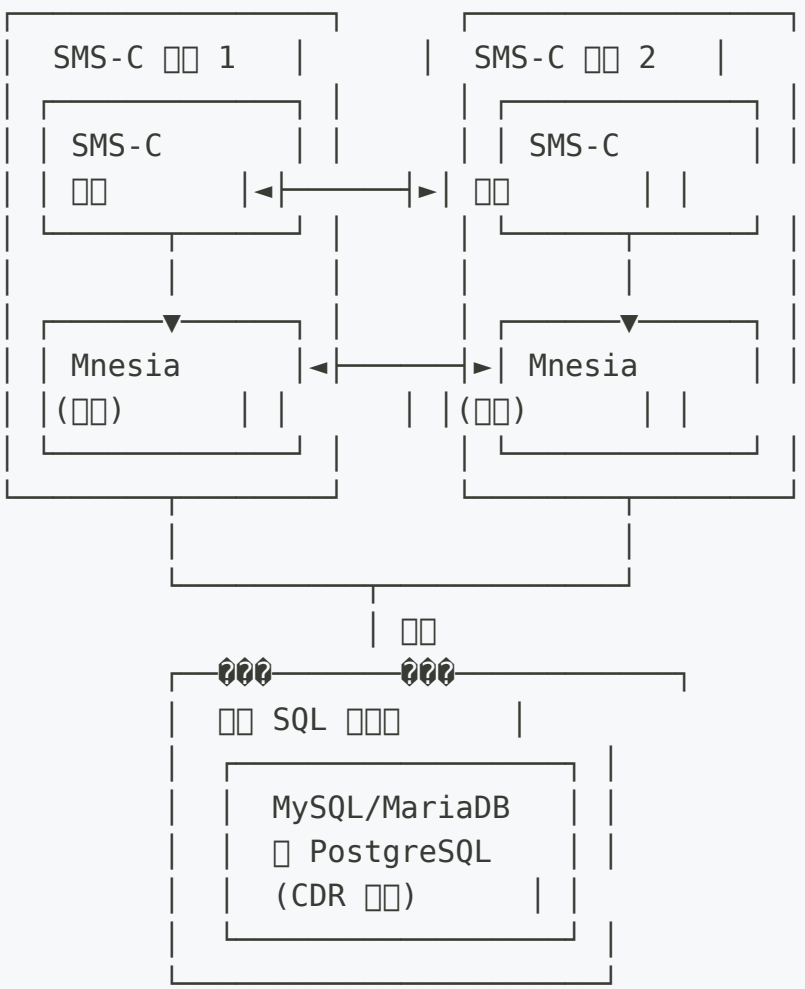
- 用于分析的高级 JSON/JSONB 功能
- 对 CDR 数据的复杂查询
- 现有的 PostgreSQL 基础设施
- 用于地理分析的 PostGIS

部署拓扑

重要: SQL CDR 数据库可以与您的 SMS-C 实例在 不同的服务器 上运行。这是生产部署的推荐方法。

单服务器部署（开发/测试）：

分布式部署（生产 - 推荐）：



独立 **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

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，则发出警报 - 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,
  api: %{
    port: 8443,
    listen_ip: "0.0.0.0",
    enable_tls: true,
    tls_cert_path: "priv/cert/omnitouch.crt",
    tls_key_path: "priv/cert/omnitouch.pem"
  }
```

API 设置位于 `:api_ex` 应用程序的 `:api` 映射下。路由表（`routes:`）在 `config/config.exs` 中编译时定义，□处不进行编辑。

□□	□□	□□	□□	□□
`port`	□□	□	`8443`	REST API □ TCP □□□
`listen_ip`	□□□	□	`0.0.0.0`	API □□□□ □□□□ `0.0.0.0` □□□□□□□
`enable_tls`	□□	□	`true`	□□ HTTPS □ □ API□
`tls_cert_path`	□□□	□□ `enable_tls`	`priv/cert/omnitouch.crt`	TLS □□□ □□□
`tls_key_path`	□□□	□□ `enable_tls`	`priv/cert/omnitouch.pem`	TLS □□□ □□□

TLS/SSL □□

□ **TLS** □□□□□□□□□□□

```
config :api_ex,
  api: %{
    port: 8443,
    listen_ip: "0.0.0.0",
    enable_tls: true,
    tls_cert_path: "priv/cert/omnitouch.crt",
    tls_key_path: "priv/cert/omnitouch.pem"
  }
```

不带 **TLS** 的开发设置：

```
config :api_ex,
  api: %{
    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）颁发的证书。

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，请参见操作指南。

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, ControlPanelWeb.Endpoint,
  https: [
    port: 443,
    keyfile: "priv/cert/omnitouch.pem",
    certfile: "priv/cert/omnitouch.crt"
  ]
```

Web UI 是一个 Phoenix 端点。TLS 通过端点的 `https:` 关键字列表配置（而不是 `tls_cert_path`/`tls_key_path`），并通过 `config/config.exs` 中的 `force_ssl` 强制使用 HTTPS。

参数	类型	必需	默认	描述
`port`	整数	否	`443`	控制��板的 HTTPS 端口。
`keyfile`	字符串	是	`priv/cert/omnitouch.pem`	TLS 私钥的路径。
`certfile`	字符串	是	`priv/cert/omnitouch.crt`	TLS 证书的路径。

生产 **Web UI** 设置

```
config :control_panel, ControlPanelWeb.Endpoint,
  url: [host: "sms-admin.example.com", path: "/"],
  https: [
    port: 443,
    keyfile: "priv/cert/web.pem",
    certfile: "priv/cert/web.crt"
  ]
```

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;
  }

  # □□ WebSocket □□□ LiveView
  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;
  }
}
```

联邦配置

OmniMessage 使用基于 HTTP 的联邦机制进行多控制器部署。控制器通过 DNS SRV 记录或静态对等列表发现彼，通过 HTTPS 交换健康和前端注册信息，并在需要时将消息转发到远程控制器。

请参阅 地理联邦指南 以获取完整架构、部署示例和故障排除信息。

快速开始

```
# config/runtime.exs
config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "_smsc._tcp.smsc.example.com"
```

完整参数参考

参数	类型	默认	描述
`enabled`	布尔	`false`	联邦的主开关。
`dns_srv_domain`	字符串	`""`	用于对等发现的 DNS SRV 域。
`dns_poll_interval_ms`	整数	`30000`	DNS 重新解析间隔。
`health_check_interval_ms`	整数	`10000`	对等健康检查间隔。
`registry_sync_interval_ms`	整数	`15000`	前端注册交换间隔。
`forward_retry_interval_ms`	整数	`5000`	排队消息重试间隔。
`forward_max_retries`	整数	`50`	标记失败之前的最大转发尝试次数。
`http_timeout_ms`	整数	`5000`	所有对等 HTTP 调用的超时。
`api_port`	整数	`8443`	用于构建对等 URL 的 API 端口。
`static_peers`	列表	`[]`	静态对等列表（替代 DNS SRV）。

网络要求

联邦仅需要在站点之间开放端口 **8443**（HTTPS），相比之下，Erlang 集群需要开放端口 4369 + 9100-9200。

```
# 允许来自对等站点的联邦流量
iptables -A INPUT -p tcp -s 10.0.1.0/24 --dport 8443 -j ACCEPT
iptables -A INPUT -p tcp -s 10.0.2.0/24 --dport 8443 -j ACCEPT
```

消息队列配置

控制消息的保留和过期行为。

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投递，并被标记为清理。

投递重试配置

重试行为使用指数退避:

```
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

最大尝试次数在死信之前：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**：清理工作者运行的频率（默认：10）
- **fingerprint_ttl_minutes**：重复检测窗口（默认：5）
- **event_ttl_days**：事件日志保留（默认：7）

计费配置

在线计费通过 **Diameter Ro** 接口（信用控制应用，RFC 4006）进行。对于每条计费消息，SMS-C 向 OCS 发送信用控制请求（事件请求，`CC-Request-Type 4`），请求消息的 发送者；OCS 根据订阅者的余额进行授权或拒绝。这重用 Diameter 堆栈（请参见下面的 Diameter Sh） - `:ro` 应用程序必须注册。

启用计费

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

config :sms_c, :charging,
  # 当发送者余额不足时的处理方式:
  #   :reject     - 将消息标为 :balance_rejected 并不交付
  #   :auto_reply - 额外将 SMS out_of_balance_reply_text 回复给发送者
  out_of_balance_action: :reject,
  out_of_balance_reply_text: "由于余额不足，您的消息无法发送。",
  # 当 OCS 出错或 Diameter 堆栈未配置时的失败模式
  # （不是明确的余额不足答案）:
  #   :allow  - 失败开放，交付消息（默认）
  #   :reject - 失败关闭，视为余额不足
  on_charging_error: :allow,
  # 计费配置文件，选择 OCS 如何对消息进行计费。作为 CCR
  # Service-Context-Id 发送。
  rating_profile: "sms@sms-c"
```

计费还需要启用 Diameter 堆栈，并注册 `:ro` 应用程序（信用控制是 IETF 基础，供应商 0）：

```
config :sms_c,
  diameter_enabled: true

config :diameter_ex,
  diameter: %{
    service_name: :omnimessage,
    host: "smsc01",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    # ... 请参见 Diameter Sh 部分以获取完整的堆栈/对等 ...
    applications: [
      %{
        application_name: :ro,
        application_dictionary: :diameter_gen_3gpp_ro,
        vendor_specific_application_ids: [
          %{vendor_id: 0, auth_application_id: 4,
acct_application_id: nil}
        ]
      }
    ]
  }
```

每条路由计费

每条路由的 `charged` 策略独立于全局默认值控制计费：

`charged`	行为
`:yes`	始终通过 OCS 进行余额检查
`:no`	从不计费
`:default`	使用 `default_charging_enabled`

禁用计费

```
config :sms_c,
  default_charging_enabled: false
```

禁用时（默认），：default 路由在没有任何 OCS 余额检查的情况下路由。个别路由仍然可以选择使用 charged::yes。

默认值与优雅降级

config:sms_c,:charging 块是可选的 - 每个键都有一个合理的默认值（out_of_balance_action: :reject，内置回复文本，以及 rating_profile: "sms@sms-c"），而 default_charging_enabled 默认值为 false。

默认情况下，充电 失败开放（on_charging_error::allow），与之前的 HTTP 充电路径相匹配，该路径记录失败但从未阻止交付：

如果一个路由请求充电，但 Diameter 堆栈 未 启用/配置，则消息 被允许（记录为警告）。
如果 OCS 无法访问或返回协议错误，则消息 被允许。

将 on_charging_error::reject 设置为 失败关闭 - OCS 错误和未配置的堆栈将被视为超出平衡， out_of_balance_action 将适用。

无论哪种方式，来自 OCS 的 明确超出平衡的答案 始终触发配置的 out_of_balance_action。被拒绝的消息标记为状态 :balance_rejected。

Diameter Sh (HSS) 配置

OmniMessage 可以通过 Diameter Sh 接口查询 HSS，以检测暂时离线的在线用户，从而防止消息被路由到默认网关。有关完整的操作文档，请参见 HSS 订阅者查找指南。

```
# □□□□□□□□□ HSS dip
config :sms_c,
  diameter_enabled: true

# Diameter □□□□
config :diameter_ex,
  diameter: %{
    service_name: :omnimessage,
    listen_ip: "0.0.0.0",
    listen_port: 3868,
    decode_format: :map,
    host: "smsc01",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    product_name: "OmniMessage",
    request_timeout: 5000,
    control_module: SmsC.Diameter.Control,
    processor_module: DiameterEx.Processor,
    vendor_id: 10415,
    supported_vendor_ids: [10415],
    applications: [
      %{
        application_name: :sh,
        application_dictionary: :diameter_gen_3gpp_sh,
        vendor_specific_application_ids: [
          %{vendor_id: 10415, auth_application_id: 16_777_217,
acct_application_id: nil}
        ]
      }
    ],
    peers: [
      %{
        host: "dra01.epc.mnc001.mcc001.3gppnetwork.org",
        ip: "10.0.0.1",
        port: 3868,
        realm: "epc.mnc001.mcc001.3gppnetwork.org",
        tls: false,
        transport: :diameter_tcp,
        initiate_connection: true
      }
    ]
  }
```

参数	类型	必需	默认	描述
`diameter_enabled`	布尔	否	`false`	在路由匹配期间启用 HSS 订阅者 dip（对于具有 `:on_net`/`:off_net` 标准的路由）。当为 `false` 时，这些标准失败关闭
`hss_cache_ttl_seconds`	整数	否	`7200`	缓存 HSS 查找结果的 TTL（秒）
`mock_sh`	布尔	否	`false`	使用模拟 HSS 响应进行测试（未启动 Diameter 堆栈）
`mock_sh_on_net_numbers`	列表	否	`[]`	在模拟模式下视为在线的 MSISDN

有关完整的 Diameter 堆栈参数参考（主机、领域、对等体、应用程序），请参见 HSS 订阅者查找指南。

ENUM 配置

基于 DNS 的 E.164 号码查找，用于智能路由。

禁用 ENUM（默认）

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

使用默认 DNS 启用 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
```

使用自定义 DNS 服务器启用 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/卫星链路：

```
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

在路由之前应用基于正则表达式的号码规范化。

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})$",            # 匹配 10 位数字
      calling_replace: "+1\1",                 # 前缀 +1
      called_match: "^(\d{10})$",
      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: "□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

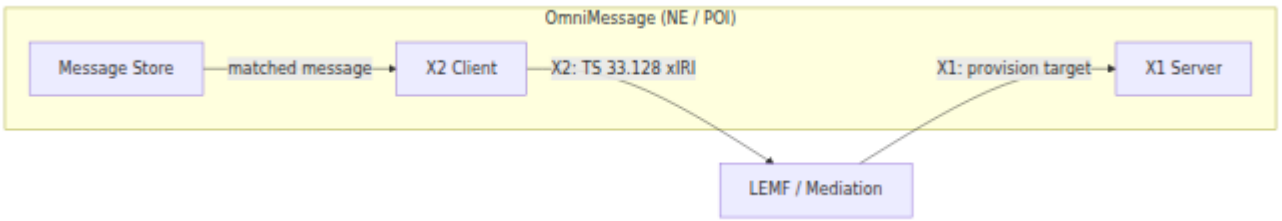
规则按优先级顺序评估（数字越小，优先级越高）：

```
config :sms_c,
  translation_rules: [
    # 优先级 1: 最具体的□则优先
    %{
      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: "高级号码□范化",
      enabled: true
    },

    # 优先级 50: 一般□则
    %{
      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: "一般 10 位数字□范化",
      enabled: true
    }
  ]
```

合法拦截 (X1/X2/X3)

OmniMessage 作为短消息服务拦截点 (POI)。OmniMessage 实现了 ETSI TS 103 221 拦截接口。OmniMessage 作为网络元素 (NE) 运行。执法监控设施 (LEMF) 通过 X1 配置拦截目标。当消息匹配已配置的目标时，OmniMessage 将消息编码为 3GPP TS 33.128 xIRI 记录。然后，OmniMessage 通过 X2 将记录发送到 LEMF。



接口概述

接口	目的	**OmniMessage** 角色	传输	使用
X1	配置和移除拦截任务和目标	服务器	HTTPS 及双向 TLS	是
X2	传递拦截相关信息（每条拦截的 SMS 作为 TS 33.128 xIRI）	客户端	TCP，选用 TLS	是
X3	传递通信内容（媒体）	客户端	TCP	否。SMS 没有媒体平面。

标准参考：

参考	标题
ETSI TS 103 221-1	内部拦截接口 X1（配置）
ETSI TS 103 221-2	内部拦截接口 X2/X3（交付）
3GPP TS 33.128	合法拦截有效载荷格式

架构

配置

在 `config :sms_c, :li` 下配置拦截。为每个部署在 `config/runtime.exs` 中设置值。默认情况下，拦截是禁用的。在您设置 `enabled: true` 之前，不会启动任何拦截进程，也不会发生任何拦截。

```
config :sms_c, :li,
  enabled: true,
  ne_identifier: "smsc-poi-01",
  x1: [
    enabled: true,
    port: 8443,
    tls: [
      certfile: "/etc/omnimessage/li/x1-cert.pem",
      keyfile: "/etc/omnimessage/li/x1-key.pem",
      cacertfile: "/etc/omnimessage/li/lemf-ca.pem"
    ]
  ],
  x2: [enabled: true, address: "192.0.2.10", port: 12345, tls:
false]
```

顶层参数

参数	类型	必需	默认	描述
`enabled`	布尔	否	`false`	主开关。当为 `false` 时，不会启动任何拦截进程。
`ne_identifier`	字符串	否	`"smsc-01"`	网络功能标识符（TS 103 221-2 属性 18）。OmniMessage 在每个 X2 记录中发送□值。LEMF 使用它按名称标记连接，而不是按 IP 地址。
`x1`	关键字列表	否	`[]`	X1 配置服务器。请参阅 X1 参数。
`x2`	关键字列表	否	`[]`	X2 交付客户端。请参阅 X2 参数。
`x3`	关键字列表	否	`[]`	为了与拦截库的对称性。OmniMessage 不使用 X3。将其保留为 `[enabled: false]`。

X1 参数

OmniMessage 运行 X1 服务器。LEMF 连接到服务器以配置目标。TLS 是强制性的（TS 103 221-1 条款 7.2.2.1）。除非您设置 `allow_insecure: true`，否则服务器不会以明文模式启动。

参数	类型	必需	默认	描述
`enabled`	布尔	否	`false`	启动 X1 服务器。
`port`	整数	否	`8443`	X1 服务器的 TCP 端口。
`tls`	关��字列表	是	-	X1 服务器的 TLS 设置。请参阅 X1 TLS 参数。
`allow_insecure`	布尔	否	`false`	通过明文 HTTP 运行 X1 服务器。仅在实验室中使用□选项。请勿在生产中使用□选项。

X1 TLS 参数

参数	类型	必需	默认	描述
`certfile`	字符串	是	-	服务器证书的路径，PEM 格式。
`keyfile`	字符串	是	-	服务器私钥的路径，PEM 格式。
`cacertfile`	字符串	否	-	验证客户端证书的证书颁发机构的路径。设置□值以启用双向 TLS。生产部署需要双向 TLS。

X2 参数

OmniMessage 运行 X2 客户端。客户端连接到 LEMF。它将每条拦截的 SMS 作为 TS 33.128 xIRI 记录发送。该记录携带 SMS 消息和匹配目标标识符。

参数	类型	必需	默认	描述
`enabled`	布尔	否	`false`	启动 X2 客户端。
`address`	字符串	是	-	LEMF X2 端点的 IP 地址或主机名。
`port`	整数	是	-	LEMF X2 端点的 TCP 端口。
`tls`	布尔或关键字列表	否	`false`	设置为 `true` 或 TLS 选项的关键字列表，以通过 TLS 交付 X2。设置为 `false` 以进行明文 TCP。

持久性

OmniMessage 默认将配置的拦截任务和目标保存在内存中。要在重启后保持配置的目标，请在部署配置中启用持久拦截存储。OmniMessage 将任务和目标写入磁盘。OmniMessage 在下次启动时重新加载它们。

合规性

有关拦截合规性概述，请参见 ANSSI R226 拦截合规性。

路由配置

首次启动时加载初始路由规则。有关完整的路由文档，请参见 SMS 路由指南。

从配置加载路由

```
# config/runtime.exs
config :sms_c,
  sms_routes: [
    # 地理路由示例
    %{
      calling_regex: nil,
      called_regex: ~r/^\+1/,
      source_smsc: nil,
      dest_smsc: "north_america_gateway",
      source_type: nil,
      enum_domain: nil,
      enum_result_domain: nil,
      action: :route,
      auto_reply_message: nil,
      charged: :default,
      originating_network: :any,
      terminating_network: :any,
      weight: 100,
      priority: 50,
      description: "北美路由",
      enabled: true
    },

    # 负载均衡路由示例
    %{
      calling_regex: nil,
      called_regex: ~r/^\+44/,
      source_smsc: nil,
      dest_smsc: "uk_gateway_1",
      source_type: nil,
      enum_domain: nil,
      enum_result_domain: nil,
      action: :route,
      auto_reply_message: nil,
      charged: :default,
      originating_network: :any,
      terminating_network: :any,
      weight: 70,
      priority: 50,
      description: "英国主网关（70%）",
      enabled: true
```

```
    },
    %{
      calling_regex: nil,
      called_regex: ~r/^\+44/,
      source_smsc: nil,
      dest_smsc: "uk_gateway_2",
      source_type: nil,
      enum_domain: nil,
      enum_result_domain: nil,
      action: :route,
      auto_reply_message: nil,
      charged: :default,
      originating_network: :any,
      terminating_network: :any,
      weight: 30,
      priority: 50,
      description: "英国备份网关（30%）",
      enabled: true
    },

    # 仅限在线 - 将 SMPP 绑定限制为仅在线目标
    %{
      calling_regex: nil,
      called_regex: nil,
      source_smsc: "carrier_smpp_bind",
      dest_smsc: "local_msc",
      source_type: :smpp,
      enum_domain: nil,
      enum_result_domain: nil,
      action: :route,
      auto_reply_message: nil,
      charged: :default,
      originating_network: :any,
      terminating_network: :on_net,
      weight: 100,
      priority: 50,
      description: "运营商 X - 仅限在线终止",
      enabled: true
    }
  ]
```

⚠️ 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: []
```

仅当路由表为空（首次启动）时，才会加载配置中定义的路由。

性能调优配置

批量插入工作者

```
# config/config.exs
config :sms_c,
  batch_insert_batch_size: 100,            # 每批消息数
  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
高容量	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 omnimessage
    endscript
}
```

常见配置场景

高容量聚合器

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

# 批量工作者
config :sms_c,
  batch_insert_batch_size: 200,
  batch_insert_flush_interval_ms: 200

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

# 充电（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
```

企业实时消息传递

优化以实现低延迟（< 20ms）：

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

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

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

# 充电（启用） - 通过 Diameter Ro 进行余额检查
# （请参见充电配置部分的 :charging + :diameter_ex 块）
config :sms_c,
  default_charging_enabled: true
```

多租户服务提供商

每个租户单独配置：

```
# 租户 1 环境
export DB_NAME=sms_c_tenant1
export NODE_NAME=sms_tenant1@node1.example.com
# 每个租户的 OCS 计费通过 :charging rating_profile 配置选择

# 租户 2 环境
export DB_NAME=sms_c_tenant2
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 DR
  ],
  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

# 检查 Diameter Ro 对 OCS 的对等体（如果启用了充电），在远程控制台中：
#   :diameter.service_info(:omnimessage, :connections)

# 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`

配置最佳实践

1. 使用环境变量 来存储敏感值（密码、API 密钥）
2. 在生产之前测试配置更改 在暂存环境中
3. 在部署说明中记录自定义设置
4. 版本控制配置文件（不包括秘密）
5. 更改后监控 性能回归
6. 保留工作配置的备份
7. 重启前验证 以避免启动失败
8. 在各个环境中使用一致的命名
9. 设置适合硬件的资源限制

10. 定期审查 以移除未使用的功能

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	凭据/主机错误	验证 DB_* 环境变量
API 无法访问	端口/IP 绑定错误	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/DRA 无法访问	检查 Diameter Ro 对等（计费失败开放）
ENUM 查找失败	DNS 服务器无法访问	测试 DNS 连接，检查超时
性能差	批处理设置错误	查看上面的性能调优配置部分
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功能。

```
config :sms_c,
  # 在 Mnesia 中保留消息的时间（小时）。
  # 如果未设置，内置回退为 24；随附的 runtime.exs 设置为 168（7 天）。
  message_retention_hours: 168,

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

建议：

- 生产环境： 24-72 小时（平衡操作需求与内存）
- 开发环境： 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

当消息被送达或过期时，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（内置默认：false）
  cdr_enabled: true
```

CDR 记录包括：

- 消息 ID，呼叫/被呼叫号码
- 源/目的地 SMSC
- 源/目的地节点（对于集群）
- 提交、交付、过期时间戳
- 状态，交付尝试
- 可选消息体（见隐私控制）

何时禁用：

- 测试环境中不需要 CDR
- 临时故障排除以减少数据库负载

隐私控制

配置消息体的可见性和保留以符合隐私合规性。

```
config :sms_c,
  # 成功交付后从 Mnesia 中删除消息体
  delete_message_body_after_delivery: false,

  # 在 Web UI 中隐藏消息体
  hide_message_body_in_ui: false,

  # 在 CSV 导出中隐藏消息体
  hide_message_body_in_export: false
```

用例：

配置	用例
`delete_message_body_after_delivery: true`	节省 Mnesia 空间，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  # 保留没有主体的 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，配置状态被记录：

```
[info] 消息存储：Mnesia（保留：24h）
[info] CDR 导出：已启用
[info] 交付后删除消息体：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。

地理联邦

← 返回文档索引 | 配置参考 | 操作指南

概述

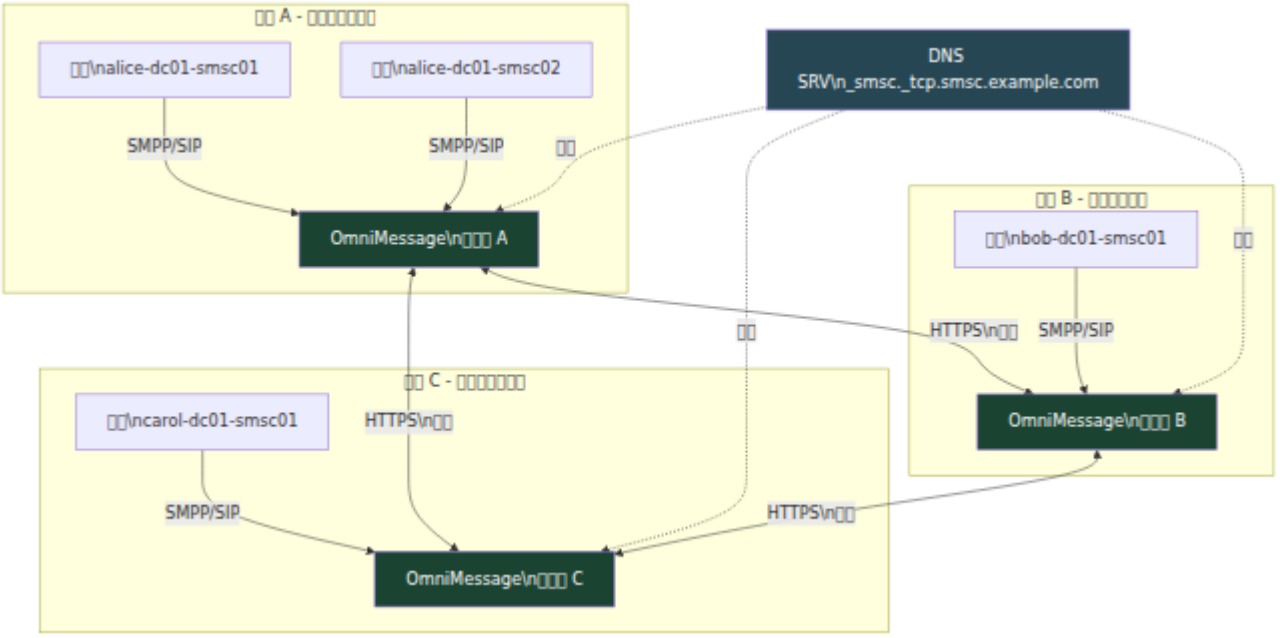
OmniMessage 使用 于 **HTTP** 的联邦 和 拉取模型 进行跨数据中心或区域的多控制器部署。每个控制器独立操作 - 管理自己的消息队列、路由表和连接的前端集。控制器通过 DNS SRV 记录（或静态配置）发现彼，通过 HTTPS 交换健康和前端注册信息。

当路由确定消息属于远程站点时，消息 保留在源控制器的 **Mnesia** 中。会向目标控制器发送轻量通知，触发立即轮询。目标控制器的 FederationPoller 定期（并在通知时）查询源对等体以获取目的地为其本地前端的消息，缓存这些消息，并使其可供本地前端使用。交付后，目标控制器将交付状态报告回源。

关键特性

方面	详情
网络	在任何 WAN 上工作，包括不可靠的链接
共享状态	��� - 每个控制器是独立的
消息所有权	消息在交付之前保留在源控制器上
安全性	使用 HTTPS 和 TLS
端口	单– HTTPS 端口（8443）
规模目标	5-20 个控制器
分区处理	轮询器重试 - 无分裂脑

架构



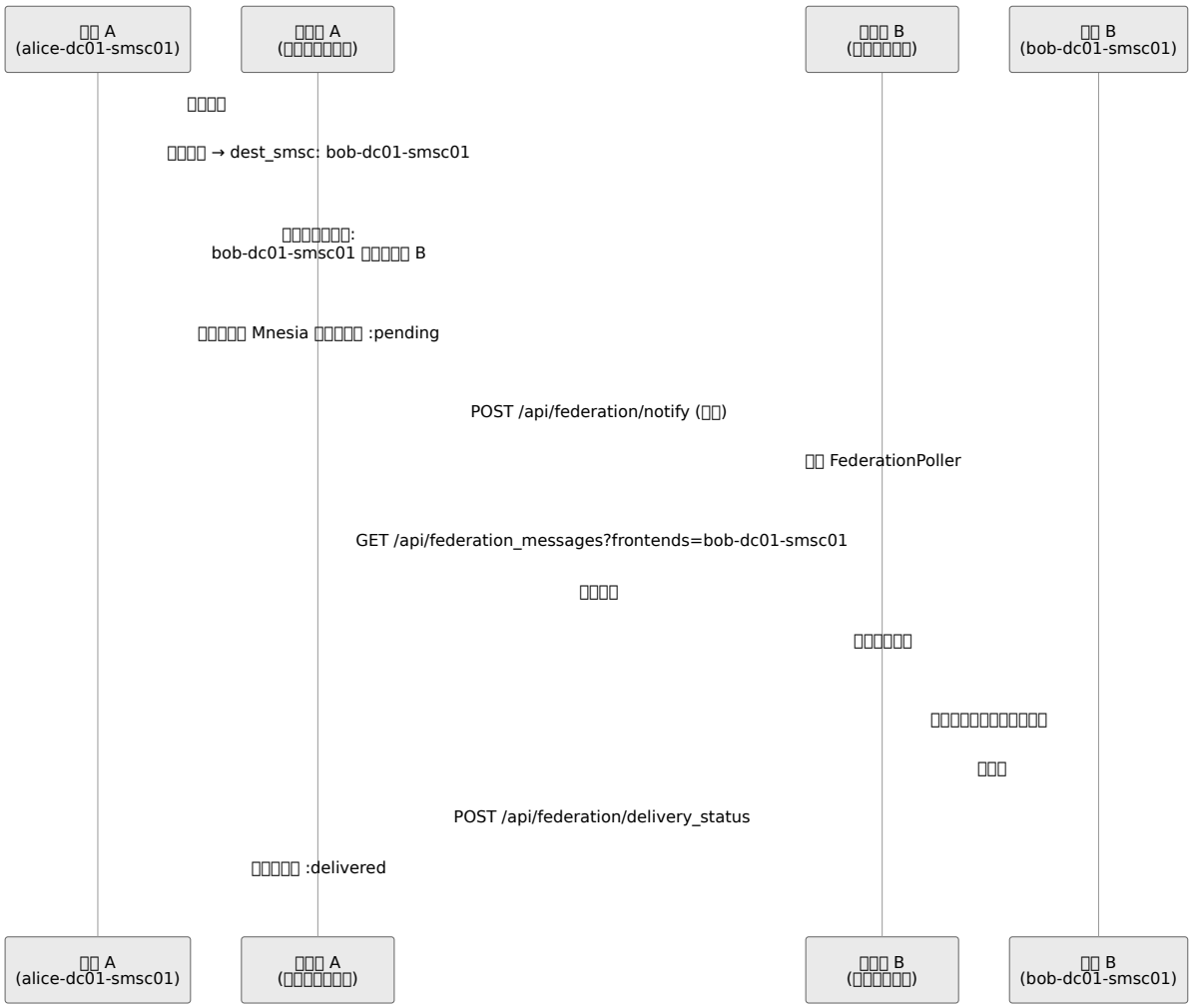
关键设计决策

消息流（拉取模型）：消息保留在源控制器上。目标控制器通过定期轮询和按需通知拉取它们。前端仅与其主控制器通信。
分区处理：轮询器自动重试。消息在成功交付之前保持安全在源控制器上。
配置同步：无。每个控制器独立管理自己的路由表。
对等网：全网格。所有控制器与所有其他控制器交换健康和前端注册信息。
发现：DNS SRV 记录（推荐）或静态对等体列表（用于没有动态 DNS 的环境）。

消息流（拉取模型）

当消息到达控制器并且路由确定目标前端位于远程控制器上时：

1. 消息保留在源控制器的 Mnesia 中，状态为:pending
2. 向目标控制器发送轻量通知（火忘即忘）
3. 目标控制器的 FederationPoller 通过 GET /api/federation_messages 获取消息
4. 消息在本地缓存并提供给本地前端
5. 交付后，目标控制器将状态报告回源

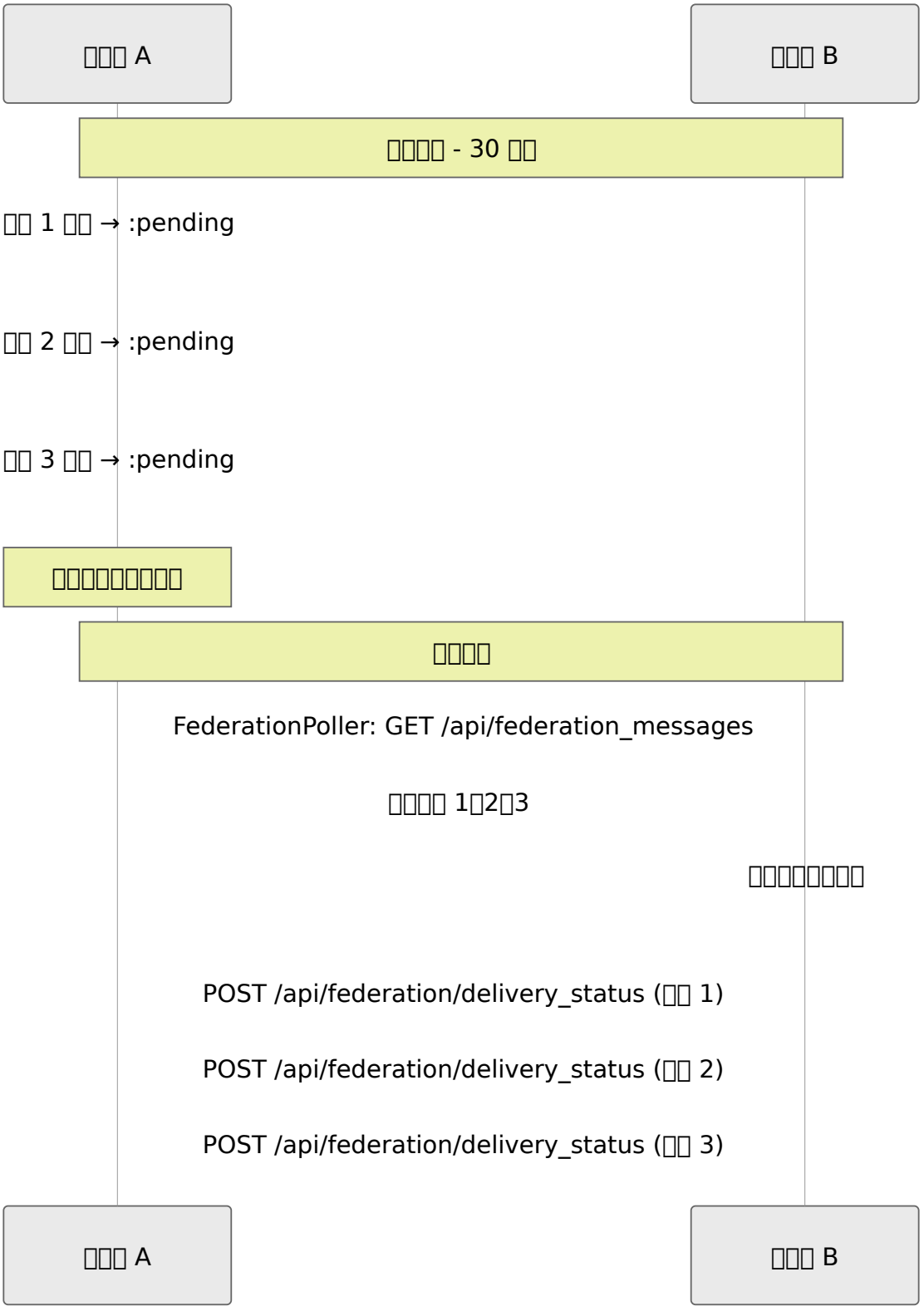

□□ A
(alice-dc01-smsc01)
□□□ A
(□□□□□□□□)
□□□ B
(□□□□□□□)
□□ B
(bob-dc01-smsc01)
□□□□
□□□□ → dest_smsc: bob-dc01-smsc01
□□□□□□□:
bob-dc01-smsc01 □□□□□ B
□□□□□ Mnesia □□□□□ :pending
POST /api/federation/notify (□□)
□□ FederationPoller
GET /api/federation_messages?frontends=bob-dc01-smsc01
□□□□
□□□□□□
□□□□□□□□□□□□□□
□□□
POST /api/federation/delivery_status
□□□□□ :delivered
□□ A
(alice-dc01-smsc01)
□□□ A
(□□□□□□□□)
□□□ B
(□□□□□□□)
□□ B
(bob-dc01-smsc01)

□□□□□□ - □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `:pending` □□□□□□□□□□□□□□□□□□□□

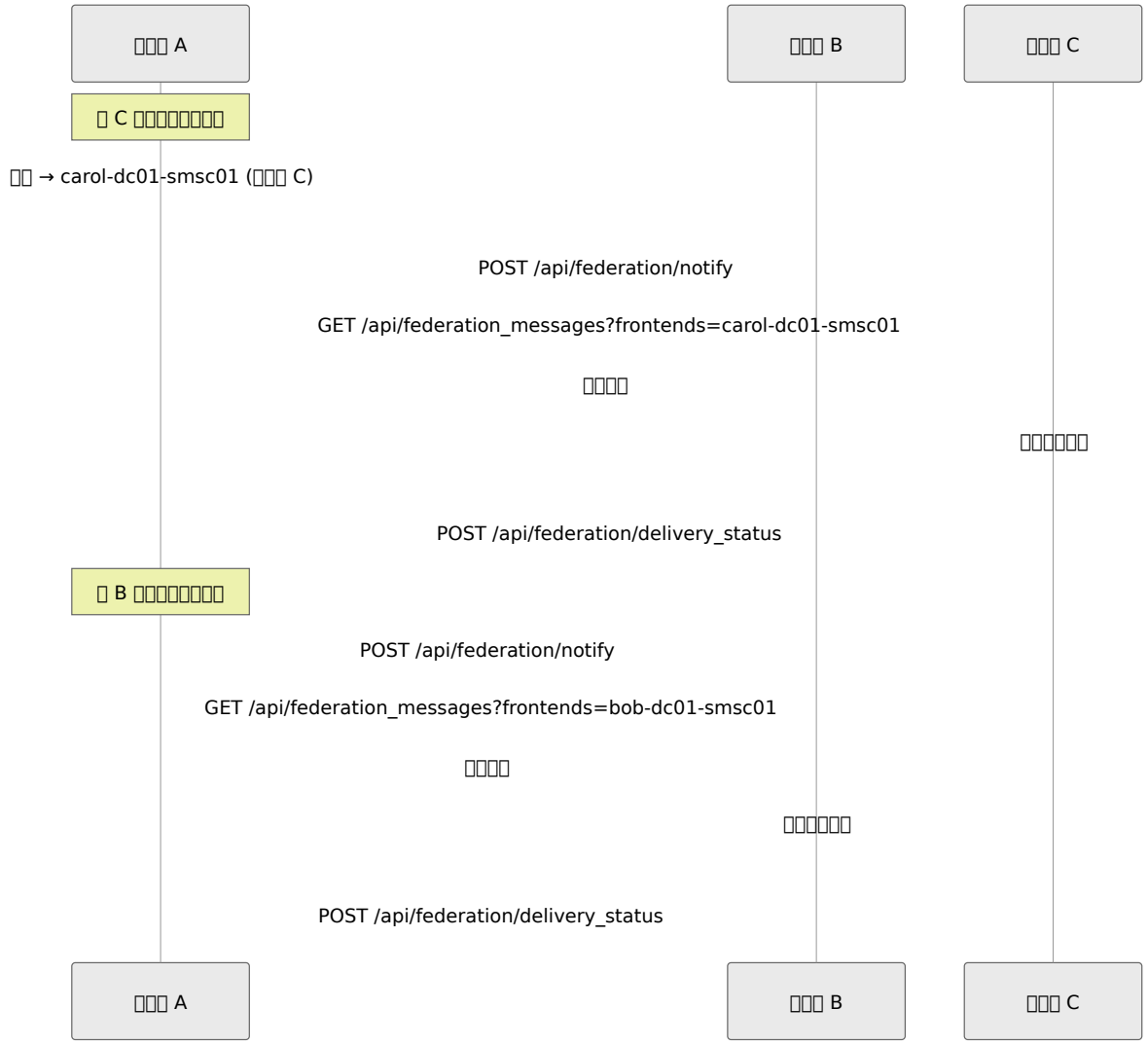
分区恢复 - 多个排队消息

在长时间的分区期间，消息在源控制器上累积。当链接恢复时，FederationPoller 会自动赶上 - 所有待处理的消息在一次查询中返回。


□□□ A
□□□ B
□□□□ - 30 □□
□□ 1 □□ → :pending
□□ 2 □□ → :pending
□□ 3 □□ → :pending
□□□□□□□□□□
□□□□
FederationPoller: GET /api/federation_messages
□□□□ 1□2□3
□□□□□□□□□
POST /api/federation/delivery_status (□□ 1)
POST /api/federation/delivery_status (□□ 2)
POST /api/federation/delivery_status (□□ 3)
□□□ A
□□□ B

□□□□□□

□□□□□□□□□□



□□□□□□□□□□

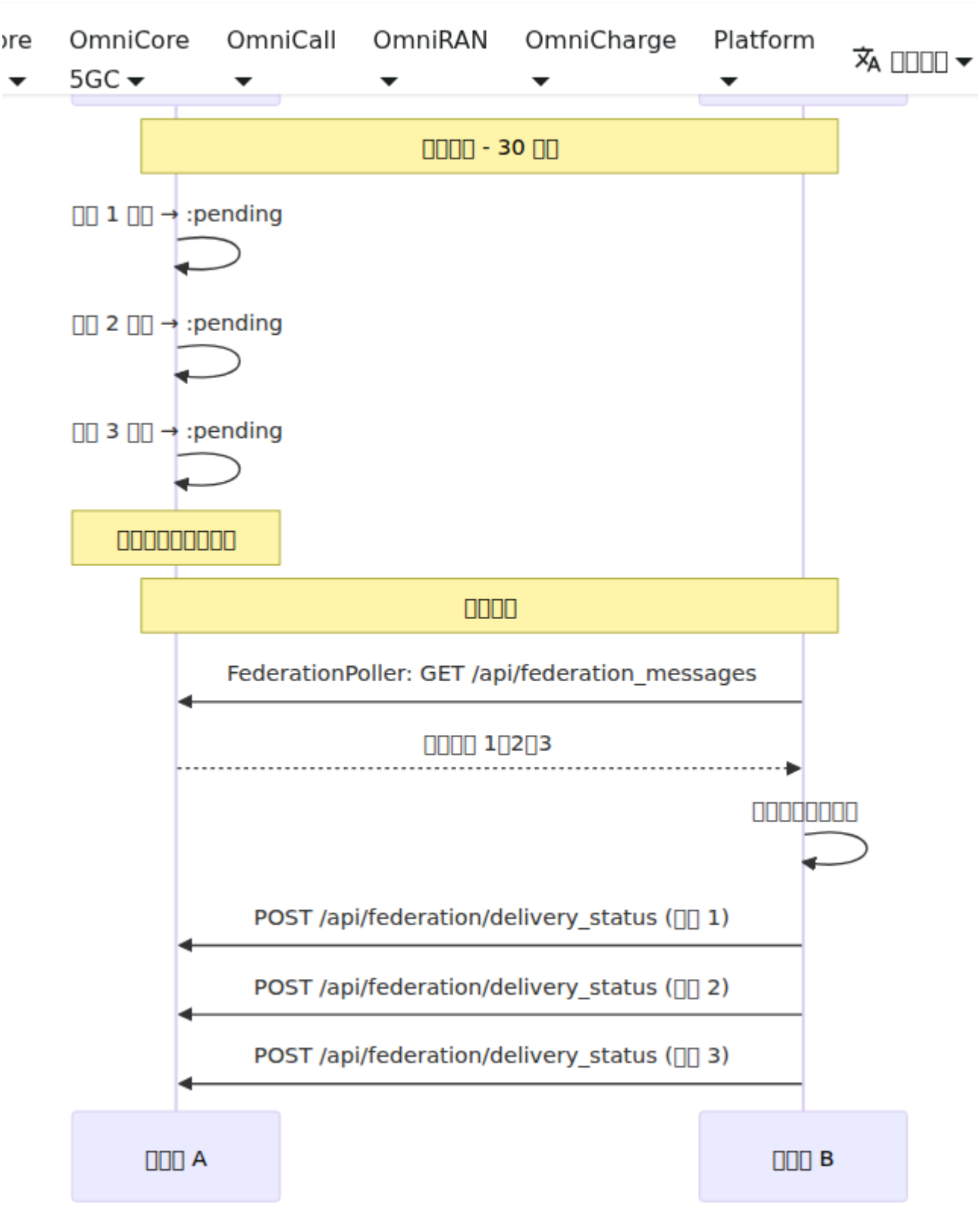
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□

□ 10 □□□□□□□□□□□□□□□□□□□□□□□□□□□□ `POST /api/federation/health`□□□□□□□□□ - □□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 3 □□□□□□□□□ □□□□

□□□□□□

□ 15 □□□□□□□□□□□□□□□□□□□□□□□□□□□□ `POST /api/federation/registry`□□□□□□□□□□ - □□□□□□□□□□□□□□□□□□□□


ore
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□□□□
□□□□ - 30 □□
□□ 1 □□ → :pending
□□ 2 □□ → :pending
□□ 3 □□ → :pending
□□□□□□□□□□
□□□□
FederationPoller: GET /api/federation_messages
□□□□ 1□2□3
□□□□□□□□□
POST /api/federation/delivery_status (□□ 1)
POST /api/federation/delivery_status (□□ 2)
POST /api/federation/delivery_status (□□ 3)
□□□ A
□□□ B

□□□□□□□□□□□□□□□□□□

状态	健康检查	**FederationPoller**	注册同步
未知	进行中	不轮询	不尝试
健康	通过	主动轮询	活跃
降级	1-2 次失败	不轮询	不尝试
不健康	3 次以上失败	不轮询	不尝试

配置

联邦在 `config/runtime.exs` 中的 `:federation` 键下配置。默认情况下 禁用，必须显式启用。

最小配置（**DNS SRV** 发现）

```
# config/runtime.exs
config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "_smsc._tcp.smsc.example.com"
```

完整配置参考

```
# config/runtime.exs
config :sms_c, :federation,
  # 主开关 - 当为 false 时，联邦完全不活动
  enabled: true,

  # 用于对等体发现的 DNS SRV 域（留空以使用 static_peers）
  dns_srv_domain: "_smsc._tcp.smsc.example.com",

  # 重新解析 DNS SRV 记录的频率（毫秒）
  dns_poll_interval_ms: 30_000,

  # 与所有已知对等体交换健康状态的频率（毫秒）
  health_check_interval_ms: 10_000,

  # 与健康对等体交换前端注册的频率（毫秒）
  registry_sync_interval_ms: 15_000,

  # FederationPoller 查询健康对等体以获取消息的频率（毫秒）
  poll_interval_ms: 5_000,

  # 所有对等体 API 调用的 HTTP 超时（毫秒）
  http_timeout_ms: 5_000,

  # 从 DNS SRV 记录构造对等体 URL 时使用的 API 端口
  api_port: 8443,

  # 静态对等体列表 - 当 dns_srv_domain 为空时使用
  static_peers: []
```

联邦参数

参数	类型	必需	默认	描述
`enabled`	布尔	是	`false`	主开关。当为 `false` 时，联邦服务启动但不发现或联系对等体。
`dns_srv_domain`	字符串	否	`""`	用于自动对等体发现的 DNS SRV 域。当为空时，使用 `static_peers`。
`dns_poll_interval_ms`	整数	否	`30000`	DNS SRV 重新解析尝试之间的间隔。较低的值更快检测新对等体，但增加 DNS 负载。
`health_check_interval_ms`	整数	否	`10000`	健康检查轮次之间的间隔。每轮联系每个已知对等体。
`registry_sync_interval_ms`	整数	否	`15000`	前端注册同步轮次之间的间隔。每轮联系每个健康对等体。
`poll_interval_ms`	整数	否	`5000`	FederationPoller 查询健康对等体以获取消息的频率。通知触发即时轮询。
`http_timeout_ms`	整数	否	`5000`	对每个单独 HTTPS 调用到对等体的超时。适用于健康检查、注册同步和消息获取。
`api_port`	整数	否	`8443`	从 DNS SRV 记录构造对等体 URL 时使用的 API 端口。必须与 `config :api_ex` 中配置的 API 端口匹配。
`static_peers`	列表	否	`[]`	在没有 DNS SRV 的环境中使用的对等体配置列表。每个条目是一个包含 `host` 和 `port` 键的映射。

静态对等体配置

对于没有 DNS SRV 可用的环境，显式配置对等体：

```
config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "",
  static_peers: [
    %{host: "10.0.1.2", port: 8443},
    %{host: "10.0.2.2", port: 8443},
    %{host: "10.0.3.2", port: 8443}
  ]
```

参数	类型	必需	默认	描述
`host`	字符串	是	-	远程控制器的 IP 地址或主机名。
`port`	整数	否	`8443`	远程控制器的 HTTPS API 端口。

DNS SRV 记录设置

DNS SRV 记录允许控制器自动发现彼□。每个控制器解析配置的域名，并使用结果的主机/端口对作为对等体。

DNS 区域配置：

```
; 优先级 权重 端口 目标
_smsc._tcp.smsc.example.com. 86400 IN SRV 10 100 8443 smsc-
alice.smsc.example.com.
_smsc._tcp.smsc.example.com. 86400 IN SRV 10 100 8443 smsc-
bob.smsc.example.com.
_smsc._tcp.smsc.example.com. 86400 IN SRV 10 100 8443 smsc-
carol.smsc.example.com.

; 目标的 A 记录
smsc-alice.smsc.example.com.    86400 IN A 10.0.1.2
smsc-bob.smsc.example.com.    86400 IN A 10.0.2.2
smsc-carol.smsc.example.com.  86400 IN A 10.0.3.2
```

添加新站点：向 DNS 添加新的 SRV 记录。所有现有控制器将在一个 `dns_poll_interval_ms` 周期内（默认 30 秒）发现新对等体。

移除站点：从 DNS 中移除 SRV 记录。现有控制器将在一个轮询周期内移除离开的对等体。

网络要求

联邦只需在站点之间使用一个 HTTPS 端口。

端口	协议	方向	目的
8443	TCP/TLS	双向	所有联邦流量（健康、注册、通知、消息、交付状态）
53	UDP/TCP	出站	DNS SRV 解析（如果使用 DNS 发现）

防火墙规则（每个站点）：

```
# 允许来自对等控制器 IP 的联邦流量
iptables -A INPUT -p tcp -s 10.0.1.0/24 --dport 8443 -j ACCEPT
iptables -A INPUT -p tcp -s 10.0.2.0/24 --dport 8443 -j ACCEPT
iptables -A INPUT -p tcp -s 10.0.3.0/24 --dport 8443 -j ACCEPT
```

联邦 API 端点

控制器在标准 API 端口（8443）下的 `/api/federation/` 上提供所有联邦端点。对等控制器调用这些端点，而不是外部客户端或前端。

端点	方法	目的
`/api/federation/identity`	GET	返回□控制器的节点名称和协议版本
`/api/federation/health`	POST	双向健康交换 - 调用者发送状态，被调用者响应自己的状态
`/api/federation/registry`	POST	双向前端注册交换
`/api/federation/notify`	POST	接收轻量通知，表明消息可用于本地前端
`/api/federation/delivery_status`	POST	接收来自目标控制器的交付确认
`/api/federation_messages`	GET	返回指定前端的待处理消息（由 FederationPoller 使用）

所有请求和响应使用 JSON。对等体标识在 `X-Federation-Node` 头中传递。

部署示例

示例 1：使用 DNS SRV 的三站点部署

三个数��中心，每个数据中心都有本地前端服务于其订阅者基础。DNS SRV 提供发现。

```
# 站点 A - 数据中心爱丽丝 (config/runtime.exs)
config :sms_c,
  smsc_node_name: "alice-dc01-smsc01"

config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "_smsc._tcp.smsc.example.com"
```

```
# 站点 B - 数据中心鲍勃 (config/runtime.exs)
config :sms_c,
  smsc_node_name: "bob-dc01-smsc01"

config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "_smsc._tcp.smsc.example.com"
```

```
# 站点 C - 数据中心卡罗尔 (config/runtime.exs)
config :sms_c,
  smsc_node_name: "carol-dc01-smsc01"

config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "_smsc._tcp.smsc.example.com"
```

工作原理：所有三个控制器解析相同的 SRV 域并自动发现彼□。每个控制器注册其本地前端。当到达数据中心爱丽丝的消息目标是由数据中心鲍勃提供的订阅者时，路由表选择 `bob-dc01-smsc01` 作为目标。联邦层检测到该前端属于控制器 B，并发送通知。控制器 B 的 FederationPoller 查询控制器 A，并缓存消息以供本地交付。

示例 2：使用静态对等体的双站点主动-主动

两个通过专用链接连接的数据中心。没有 DNS SRV 可用。

```
# 主数据中心 (config/runtime.exs)
config :sms_c,
  smsc_node_name: "primary-smsc01"

config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "",
  static_peers: [
    %{host: "10.200.1.5", port: 8443}
  ]
```

```
# 次要数据中心 (config/runtime.exs)
config :sms_c,
  smsc_node_name: "secondary-smsc01"

config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "",
  static_peers: [
    %{host: "10.100.1.5", port: 8443}
  ]
```

工作原理：每个控制器都配置了对方的 IP 地址。健康检查和注册同步通过专用链接进行。消息保留在接收它们的控制器上。每个控制器的 FederationPoller 查询对等体以获取目的地为本地前端的消息。

示例 3：针对低延迟链接的积极轮询

对于通过可靠、低延迟链接连接的站点，快速联邦很重要：

```
config :sms_c, :federation,
  enabled: true,
  dns_srv_domain: "_smsc._tcp.cluster.internal",
  health_check_interval_ms: 5_000,
  registry_sync_interval_ms: 5_000,
  poll_interval_ms: 1_000,
  http_timeout_ms: 2_000
```

用例：校园或城市区域部署，站点之间的 RTT 链接小于 10 毫秒，快速故障转移比最小化控制流量更重要。

指标

联邦公开以下遥测事件，可以通过 Prometheus 在 9568 端口观察。

指标：`sms_c_federation_message_received_count` 类型：计数器 描述：从对等体接收到的联邦通知数量 标签：

- `origin_node` - 发送通知的对等体

指标：`sms_c_federation_health_check_count` 类型：计数器 描述：每个对等体的健康检查结果 标签：

- `peer` - 对等体标识
- `result` - `ok` 或 `failed`

示例查询：

```
# 每分钟通知率
rate(sms_c_federation_message_received_count[5m]) * 60

# 按对等体统计的健康检查失败
sum by (peer)
(rate(sms_c_federation_health_check_count{result="failed"}[5m]))
```

关键指标监控

监控内容	位置	警报阈值
缓存的联邦消息	`GET /api/federation/status` 或 Web UI	增长计数（对等体未交付）
对等体健康状态	`GET /api/federation/identity` 在每个对等体上	任何对等体不健康 > 5 分钟
健康检查延迟	应用程序日志	> 2 秒（链接降级）

故障排除

对等体未发现

症状：控制器日志显示没有发现对等体；`cluster_status` 返回为空。

可能原因：

- DNS SRV 域不正确或无法解析
- DNS 服务器无法从控制器主机访问
- SRV 记录尚未传播
- 联邦未启用（`enabled: false`）

解决方案:

1. 验证控制器主机上的 DNS SRV 解析：dig SRV _smsc._tcp.smsc.example.com
2. 检查联邦配置中是否设置了 enabled: true
3. 检查应用程序日志中是否有“联邦 DNS 发现”消息
4. 如果使用静态对等体，验证 static_peers 列表是否包含正确的主机/端口条目

对等体标记为不健康

症状: 向特定站点发送的消息未被轮询。日志显示“健康检查失败”消息。

可能原因:

远程控制器已关闭
防火墙阻止站点之间的 8443 端口
TLS 证书问题
站点之间的网络分区

解决方案:

1. 验证远程控制器是否正在运行：curl -k https://<peer-host>:8443/api/federation/identity
2. 检查防火墙规则是否允许来自本地控制器 IP 的 8443 端口流量
3. 检查两侧的应用程序日志以获取 TLS 或连接错误
4. 检查站点之间的网络连接

消息未交付到远程站点

症状: 消息在源控制器上保持:pending 状态。目标上的缓存联邦消息计数为 0。

可能原因:

目标控制器不健康（FederationPoller 仅查询健康对等体） FederationPoller 未运行
前端注册未同步（目标不知道哪些前端是本地的）

解决方案:

1. 检查目标控制器上的对等体状态
2. 验证 FederationPoller 是否在监督树中（检查应用程序日志中是否有“Federation poller started”）
3. 验证前端注册是否已同步（GET /api/federation/status - 检查前端列表）
4. 手动测试 federation_messages 端点：curl -k 'https://<origin>:8443/api/federation_messages?frontends= <frontend_name>&include_unrouted=false'

HSS 订阅者查找（Diameter Sh）

← 返回文档索引 | SMS 路由指南

概述

OmniMessage 可以通过 Diameter Sh 接口查询家庭订阅者服务器 (HSS)，以确定一个 MSISDN 是否属于网络（在网）或不属于网络（离网）。这个答案被用作 路由匹配标准：路由可以要求发起或终止号码在网或离网，然后才能应用。

该查询是 懒惰的 - 它发生在 □□□□□□□，仅在正在评估的路由声明 `originating_network` 或 `terminating_network` 要求为 `:on_net` 或 `:off_net` 时触发。将两个都留为 `:any`（默认值）的路由从不触发查询。在路由之前不会执行全局 HSS 查找。

OmniMessage 发送一个 用户数据请求 **(UDR)** 以获取该号码。如果 HSS 确认订阅者存在（结果代码 2001），则该号码为在网；如果 HSS 返回 □□□□（结果代码 5001），则该号码为 离网。有关这些标准如何与路由的其���部分结合的信息，请参见 SMS 路由指南 → 在网 / 离网匹配。

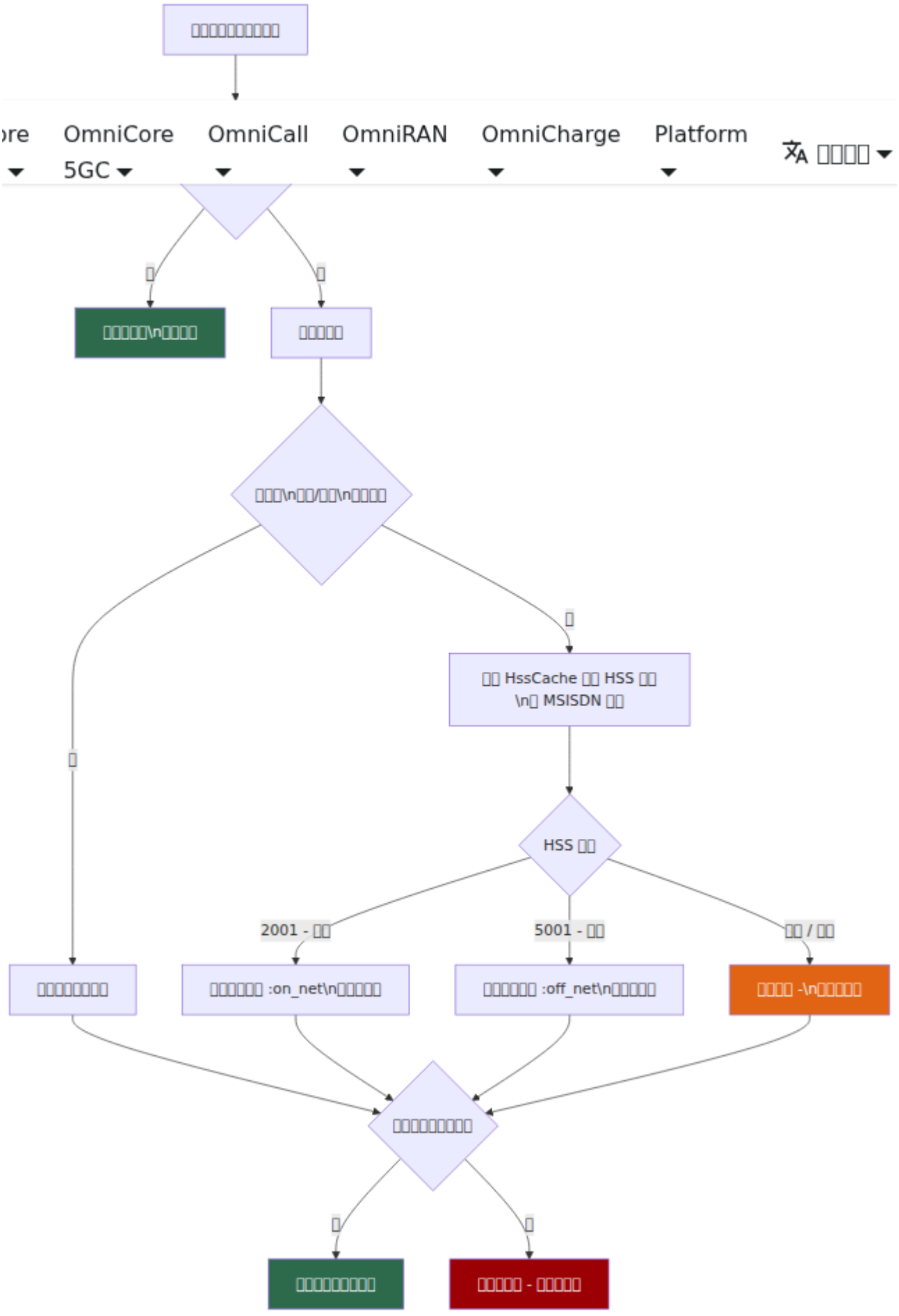
> 在网与离网 - **HSS** 还是 **ENUM**？ HSS (Sh) 回答“这个号码是 □□ HSS 中的订阅者吗？”并给出明确的在网/离网结果。ENUM (DNS NAPTR) 回答“这个号码解析到哪个运营商域？”并通过匹配您自己的域来推导在网/离网。当您的 HSS 是权威时使用 Sh；对于可移植性/跨运营商路由使用 ENUM。请参见 SMS 路由指南 → ENUM 查找。

相关规范

规范	标题
3GPP TS 29.329	基于 Diameter 的 Sh 接口
3GPP TS 29.328	IMS Sh 接口 - 信令流程和消息内容
RFC 6733	Diameter 基础协议

HSS 查询的适用位置

OmniMessage 按以下顺序评估消息：


OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
HSS
2001 -
5001 -
:on_net
:off_net

□于位置的路由（优先检查）

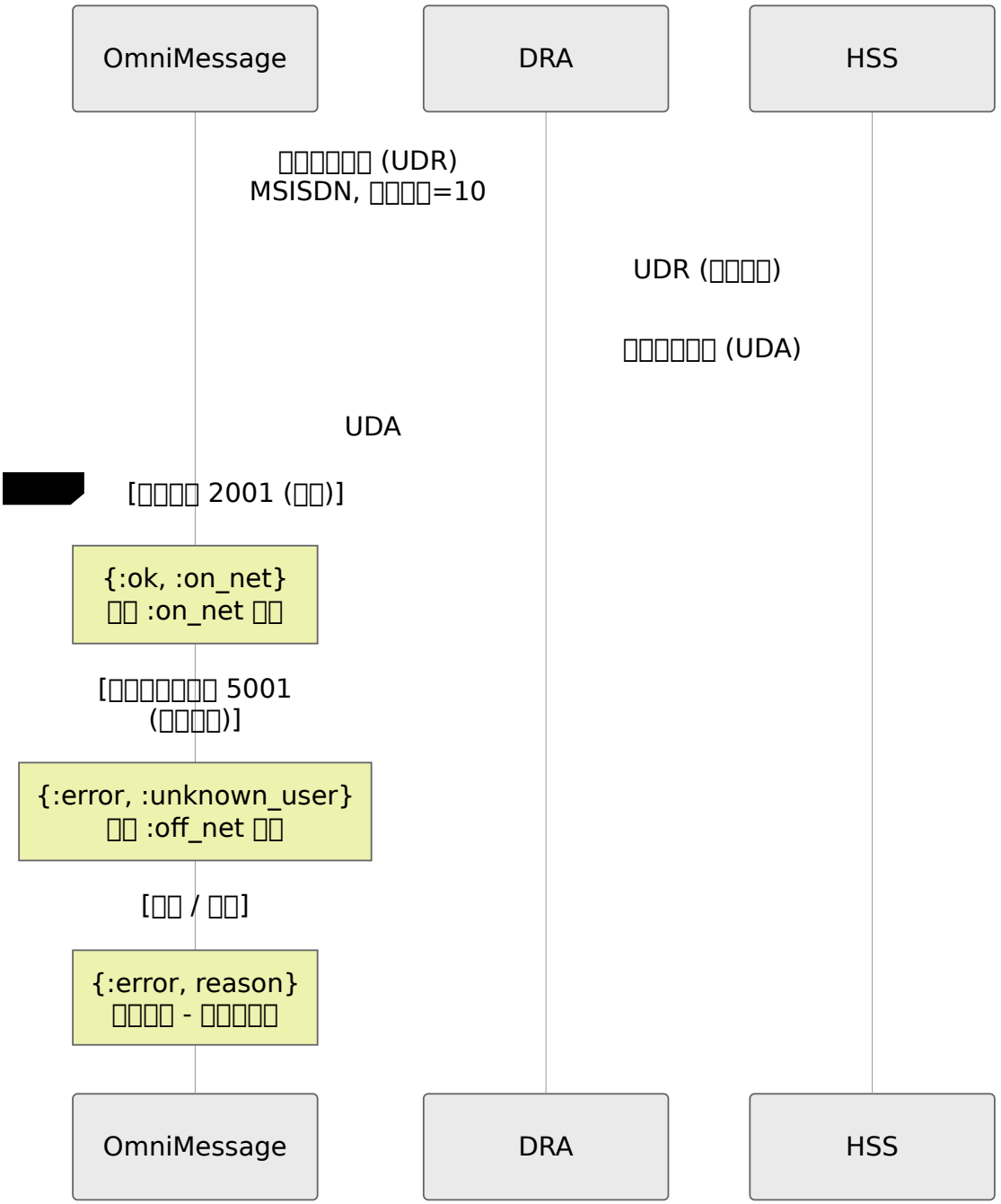
如果在注册/位置表中找到目标 MSISDN，则消息直接路由到服务该订阅者的前端。不会进行 HSS 查找或路由表评估。（请参见 位置过期。）

路由表 + 懒惰 **HSS** 查询

如果没有活动注册，路由器将评估路由表。对于每个声明 `originating_network`/`terminating_network` 为 `:on_net` 或 `:off_net` 的路由，路由器在匹配该路由时会执行 HSS 查询（通过缓存）。只有在满足其网络标准及所有其他标准的情况下，路由才会匹配。如果 Diameter 被禁用或 HSS 没有给出明确答案，则标准失败（失败关闭），路由不匹配。

持有消息以进行 存储转发 是一个单��的概念：`:saf` ���由 □□ 触发它，而不是 HSS 查询。请参见 存储转发交付。

Diameter Sh □□□


OmniMessage
DRA
HSS
□□□□□□□ (UDR)
MSISDN, □□□□=10
UDR (□□□□)
□□□□□□□ (UDA)
UDA
[□□□□ 2001 (□□)]
{:ok, :on_net}
□□ :on_net □□
[□□□□□□□□ 5001
(□□□□)]
{:error, :unknown_user}
□□ :off_net □□
[□□ / □□]
{:error, reason}
□□□□ - □□□□□
OmniMessage
DRA
HSS

UDR 参数

AVP	值	描述
Session-Id	自动生成	唯一会话标识符
Auth-Session-State	1 (NO_STATE_MAINTAINED)	查询不需要会话状态
Data-Reference	10	请求所有用户数据
Origin-Host	配置的主机 + 域	OmniMessage 的 Diameter 身份
Origin-Realm	配置的域	OmniMessage 的 Diameter 域
Destination-Realm	配置的域	HSS 域（通过 DRA 路由）
User-Identity	TBCD 编码的 MSISDN	订阅者的号码
Server-Name	"OmniMessage"	应用服务器身份
Vendor-Specific-Application-Id	Auth: 16777217, Vendor: 10415	根据 3GPP TS 29.329 的 Sh 应用标识符

UDA 结果代码

订阅者存在性检查将 UDA 映射到三种结果之一：

代码	类型	结果	对路由匹配的含义
2001	结果代码	`{:ok, :on_net}`	在网 - 满足 `:on_net`，不满足 `:off_net`
5001	实验性结果代码	`{:error, :unknown_user}`	离网 - 满足 `:off_net`，不满足 `:on_net`
其他 / 超时 / 无对等方	-	`{:error, reason}`	失败关闭 - 两者都不满足；路由不匹配

配置

启用 **HSS** 查找

`config/runtime.exs` 中的两个配置块控制 HSS 查找：

1. `:sms_c` 下的 `diameter_enabled` 标志 - 控制在路由匹配期间是否执行 HSS 查询。当为 `false` 时，任何具有 `:on_net`/`:off_net` 标准的路由都将失败匹配（失败关闭）。
2. `:diameter_ex` 配置 - 配置 Diameter 堆栈本身（身份、对等方、应用程序）

□□□□

```
# □□□□□□□□□□ HSS □□
config :sms_c,
  diameter_enabled: true

# Diameter □□□□
config :diameter_ex,
  diameter: %{
    service_name: :omnimessage,
    listen_ip: "0.0.0.0",
    listen_port: 3868,
    decode_format: :map,
    host: "smsc01",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    product_name: "OmniMessage",
    request_timeout: 5000,
    peer_selection_algorithm: :random,
    allow_undefined_peers_to_connect: true,
    log_unauthorized_peer_connection_attempts: true,
    control_module: SmsC.Diameter.Control,
    processor_module: DiameterEx.Processor,
    auth_application_ids: [],
    acct_application_ids: [],
    vendor_id: 10415,
    supported_vendor_ids: [10415],
    applications: [
      %{
        application_name: :sh,
        application_dictionary: :diameter_gen_3gpp_sh,
        vendor_specific_application_ids: [
          %{
            vendor_id: 10415,
            auth_application_id: 16_777_217,
            acct_application_id: nil
          }
        ]
      }
    ],
    peers: [
      %{
        port: 3868,
        host: "dra01.epc.mnc001.mcc001.3gppnetwork.org",
```

```
        ip: "10.0.0.1",
        realm: "epc.mnc001.mcc001.3gppnetwork.org",
        tls: false,
        transport: :diameter_tcp,
        initiate_connection: true
      }
    ]
  }
```

OmniMessage 参数

参数	类型	必需	默认	描述
`diameter_enabled`	布尔	否	`false`	在路由匹配期间��用 HSS 查询。当为 `false` 时，具有 `:on_net`/`:off_net` 标准的路由将失败关闭。
`hss_cache_ttl_seconds`	整数	否	`7200`	缓存的 HSS 查找结果的 TTL（秒）。在 `runtime.exs` 中设置（在操作员 YAML 架构中未公开）。
`mock_sh`	布尔	否	`false`	使用模拟 HSS 响应而不是实际的 Diameter 查询。仅用于测试。
`mock_sh_on_net_numbers`	列表	否	`[]`	当 `mock_sh` 为 `true` 时视为在网的 MSISDN 列表。

Diameter □□□□

□□	□□	□□	□□	
`service_name`	□□□	□	-	Diam
`listen_ip`	□□□□	□	`"0.0.0.0"`	□□□□□
`listen_port`	□□□	□	3868	□□□□ 6733
`decode_format`	□□□	□	-	Diam
`host`	□□□□	□	-	Omni □□□□□
`realm`	□□□□	□	-	Omni 3GPP `epc.r`
`product_name`	□□□□	□	-	□ Dia
`request_timeout`	□□□	□	5000	□□ HS
`peer_selection_algorithm`	□□□	□	`:random`	□□□□□ `:rou`

□□	□□	□□	□□	
`allow_undefined_peers_to_connect`	□□	□	`true`	□□□□□□□□□□
`log_unauthorized_peer_connection_attempts`	□□	□	`true`	□□□□□□□□□□
`control_module`	□□	□	-	Diam `SmsC`
`processor_module`	□□	□	-	Diam `Diame`
`auth_application_ids`	□□	□	`[]`	□□□□□□□□□□ ID□
`acct_application_ids`	□□	□	`[]`	□□□□□□□□□□
`vendor_id`	□□	□	-	3GPP
`supported_vendor_ids`	□□	□	-	□□□□□□□□□□

□□□□□□□

`applications` ��□□□□□□□ Diameter □□□□□□□□ HSS □□□□□□□ Sh □□□□□□

参数	类型	必需	默认	描述
`application_name`	原子	是	-	应用程序标识符。对于 HSS 查找，必须为 `:sh`。
`application_dictionary`	原子	是	-	Erlang Diameter 字典模块。必须为 `:diameter_gen_3gpp_sh`。
`vendor_specific_application_ids`	列表	是	-	供应商特定应用程序标识符。见下文。

供应商特定应用程序 **ID (Sh)**:

参数	类型	值	描述
`vendor_id`	整数	`10415`	3GPP 供应商 ID
`auth_application_id`	整数	`16_777_217`	根据 3GPP TS 29.329 的 Sh 应用 ID
`acct_application_id`	-	`nil`	对于 Sh 不使用

对等方参数

`peers` 列表定义 OmniMessage 连接的 Diameter 对等方（通常是 DRA 或 HSS 直接）。

参数	类型	必需	默认	描述
`host`	字符串	是	-	对等方的 Diameter 主机身份（FQDN）。必须与对等方的配置身份完全匹配。
`ip`	字符串	是	-	对等方的 TCP 连接 IP 地址。
`port`	×××	否	3868	对等方的 Diameter 端口。
`realm`	字符串	是	-	对等方的 Diameter 域。
`tls`	布尔	否	`false`	启用□对等方连接的 TLS。
`transport`	原子	否	`:diameter_tcp`	传输协议：`:diameter_tcp` 或 `:diameter_sctp`。
`initiate_connection`	布尔	否	`true`	如果为 `true`，OmniMessage 将主动发起与对等方的 TCP 连接。如果为 `false`，OmniMessage 将等待对等方连接。

多个对等方

配置多个对等方以实现冗余。OmniMessage 使用 `peer_selection_algorithm` 在连接的对等方中进行选择：

```
peers: [
  %{
    port: 3868,
    host: "dra01.epc.mnc001.mcc001.3gppnetwork.org",
    ip: "10.0.0.1",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    tls: false,
    transport: :diameter_tcp,
    initiate_connection: true
  },
  %{
    port: 3868,
    host: "dra02.epc.mnc001.mcc001.3gppnetwork.org",
    ip: "10.0.0.2",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    tls: false,
    transport: :diameter_tcp,
    initiate_connection: true
  }
]
```

模拟模式

对于没有实时 HSS 的测试环境，启用模拟模式：

```
config :sms_c,
  diameter_enabled: true,
  mock_sh: true,
  mock_sh_on_net_numbers: [
    "15555550001",
    "15555550002"
  ]
```

在模拟模式下，`mock_sh_on_net_numbers` 列表中的任何 MSISDN 将解析为 `{:ok, :on_net}`（相当于结果代码 2001）。所有其他 MSISDN 将解析为 `{:error, :unknown_user}`（相当于结果代码 5001）。不会启动 Diameter 堆栈，也不会进行网络连接。

HSS 结果缓存

HSS 查找通过 `SmsC.Diameter.HssCache` 按 MSISDN 缓存，以避免冗余的 Diameter 往返：

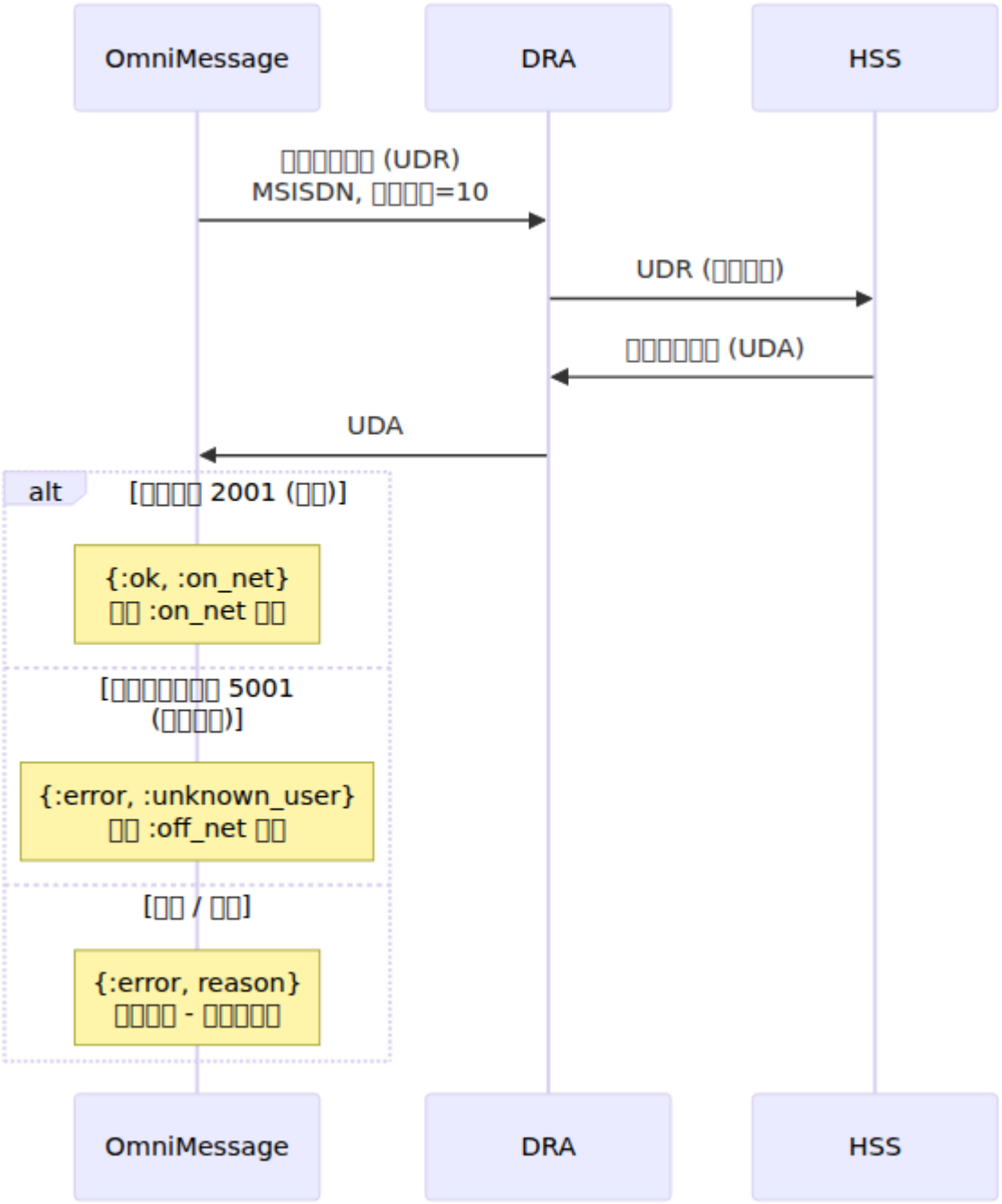
- **TTL**：`hss_cache_ttl_seconds`（默认 7200 = 2 小时）
- 共享：在一次评估中需要相同 MSISDN 状态的多个路由共享单个查询
- 强制刷新：当消息被显式重新路由时，缓存被绕过（进行新的查询）
- 谓词：结果仅在 `{:ok, :on_net}` 时视为在网，仅在 `{:error, :unknown_user}` 时视为离网；所有其他结果（包括 Diameter 被禁用时的 `nil`）都视为既不是 - 这就是产生失败关闭匹配的原因。

存储转发交付

`:saf` 路由 操作 驱动存储转发，而不是 HSS 查询。具有 `action: :saf` 的路由将消息保留以便稍后交付。要仅对在网订阅者进行保留，请将 `:saf` 与 `terminating_network: :on_net` 结合，并提供一个离网回退路由（见 SMS 路由指南 → `saf`）。

当消息被保留以进行 SAF 时，它将保留在 Mnesia ���息队列中，`dest_smsc` 未设置。当订阅者注册并到达位置更新时，交付循环会提取该 MSISDN 的待处理消息并将其路由到服务前端。

队列中的消息受标准保留策略的约束。如果订阅者在消息过期之前未注册，则消息将根据配置的 `dead_letter_time_minutes` 移动到死信队列。



操作注意事项

启动日志

OmniMessage 在启动时记录 Diameter Sh 状态:

- 启用: `Diameter Sh (HSS): ENABLED (host: smsc01, realm: epc.mnc001.mcc001.3gppnetwork.org, peers: 2)`

- 禁用：`Diameter Sh (HSS): DISABLED`

失败关闭行为

如果 HSS 查询未返回明确答案 - 超时、没有连接的对等方、未处理的响应或意外的结果代码 - `subscriber_exists?/1` 返回 `{:error, reason}`。对于需要 `:on_net` 或 `:off_net` 的路由，该标准 不 满足，因□路由 不匹配，评估将继续进行下一个路由。

这是一个 失败关闭 的设计：在没有明确答案的情况下，OmniMessage 不会将号码视为在网或离网，因□仅在网的路由不会触发，仅离网的网关路由也不会触发。如果您希望在 HSS 中断期间路由消息，请提供一个兜底回退路由（`terminating_network: :any`）。

当 **Diameter** 被显式禁用（`diameter_enabled: false`）时，缓存查找返回 `nil`，并且同样的失败关闭匹配适用于任何具有 `:on_net`/`:off_net` 标准的路由。将两个网络字段都留为 `:any` 的路由不受影响。

可观察性

HSS 查询可以通过以下方式观察：

- 每条消息的路由日志 - 每个路由的 `route_evaluated` 事件包括网络检查，例如 `term_network: ✓ (terminating requires on_net, +15551234 is on-net)` 或 `✗ (... is off-net)`。在控制面板消息活动日志和事件 API 中可见。
- **Diameter Sh ���控** - `SmsC.Telemetry` 发出请求、响应（按结果标记）、持续时间和错误指标（见 METRICS.md）。
- 应用日志 - `HSS dip: <msisdn> is ON-NET (2001)` / `is NOT in HSS (5001)` 行来自 `SmsC.Diameter.Sh`。

位置过期

基于位置的路由检查订阅者注册的 `expires` 时间戳。如果订阅者的注册已过期，例如因为他们进入飞行模式，路由器将该位置视为不活动。然后评估将继续进行路由表，包括任何匹配路由所需的懒惰 HSS 查询。

这可以防止消息发送到无法再到达订阅者的前端，避免不必要的交付失败和重试周期。

路由级别的在网 / 离网强制执行

路由通过两个字段强制执行在网/离网要求 - 不是 布尔标志，也 不是 路由操作：

字段	值	检查	行为
`originating_network`	`:any` | `:on_net` | `:off_net`	呼叫号码	仅在源 MSISDN 的 HSS 状态匹配时路由匹配（失败关闭）
`terminating_network`	`:any` | `:on_net` | `:off_net`	被叫号码	仅在目标 MSISDN 的 HSS 状态匹配时路由匹配（失败关闭）

两者的默认值均为 `:any`（无 HSS 检查）。`:on_net` 的值要求结果代码 2001；`:off_net` 要求 5001；任何不明确的案都将失败该标准。

配置示例

```
config :sms_c, :sms_routes, [
  # 仅在网目的地 - 交付到本地网关
  %{
    calling_regex: nil,
    called_regex: ~r/^1555/,
    source_smsc: nil,
    dest_smsc: "local-gateway",
    source_type: nil,
    action: :route,
    charged: :default,
    weight: 100,
    priority: 10,
    description: "本地网关 - 仅在网目的地",
    originating_network: :any,
    terminating_network: :on_net,
    enabled: true
  },

  # 离网回退 - 捕获所有未被在网路由处理的内容
  %{
    calling_regex: nil,
    called_regex: nil,
    source_smsc: nil,
    dest_smsc: "external-gateway",
    source_type: :ims,
    action: :route,
    charged: :default,
    weight: 100,
    priority: 255,
    description: "外部网关 - 回退",
    originating_network: :any,
    terminating_network: :any,
    enabled: true
  }
]
```

工作原理：对于发送给未注册订阅者的消息：

1. 位置检查 - 未找到活动注册
2. 路由评估 - 按优先级顺序检查路由

3. P10 "local-gateway" 路由具有 `terminating_network: :on_net`，因□对被叫号码执行 HSS 查询。如果 HSS 返回 2001（在网），则路由匹配，消息将交付给 `local-gateway`。如果返回 5001（离网）或失败（失败关闭），则路由不匹配。
4. P255 "external-gateway" 路由具有 `terminating_network: :any`，因□不需要 HSS 答复并作为回退匹配。

用例：在本地基础设施上保持本网流量，同时将离网（或HSS不确定）流量发送到外部运营商。为了□□本网用户进行存储转发，而不是将其发送到网关，将P10路由的`action`设置为`:saf`。

故障排除

本网用户被路由到备用网关

症状：发送给本网用户的消息走了离网/备用路线，而不是本网路线。

可能原因：

- `diameter_enabled`为`false` - `:on_net`标准失败关闭，因□本网路由永远不匹配
- 没有Diameter对等体连接（检查应用程序日志以获取对等体连接状态）
- HSS返回错误或超时（检查Sh错误遥测和“HSS dip”日志行）
- 号码转换生成的号码格式与HSS期望的不一致

解决方案：

1. 在`runtime.exs`中验证`diameter_enabled: true`
2. 检查启动时的Diameter对等体连接状态日志（`Diameter Sh (HSS): ENABLED (... peers: N)`）
3. 查找`HSS dip:`日志行和Sh错误指标
4. 确认转换后的号码格式与HSS期望的格式一致（E.164格式，不带`+`）

消息被保持��从未送达

症状：消息被正确保持（`:saf`）给本网用户，但从未送达，即使在用户注册后也是如□。

可能原因：

- 位置更新未到达OmniMessage

位置注册中的MSISDN与排队消息的destination_msisdn不匹配
消息已过期并被移动到死信队列

解决方案：

1. 验证位置/注册更新是否被接收（检查控制面板中的注册页面）
2. 确认消息队列和注册表中的MSISDN格式一致
3. 检查dead_letter_time_minutes - 如果用户可能离线较长时间，请增加该值

Diameter对等体未连接

症状：日志显示启动时没有Diameter对等体连接。

可能原因：

配置中的对等体IP或端口不正确
防火墙阻止端口3868
对等体主机身份不匹配（配置的host必须与对等体的Diameter身份完全匹配） OmniMessage与对等体之间的域不匹配

解决方案：

1. 验证对等体IP地址和端口是否正确
2. 确认防火墙规则允许在配置的端口上进行TCP流量
3. 检查对等体的host值是否与对等体在其能力交换中广告的内容完全匹配
4. 确保OmniMessage与对等体配置之间的 realm匹配

SMS-C Prometheus 指标文档

← 返回文档索引

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
```

该端点以 Prometheus 文本格式暴露指标，Prometheus 服务器可以抓取这些指标。系统在处理消息时实时更新指标。

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` - 消息交付指标
- `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。

��:

- `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，仍然可以接收消息，但将路由到目的地 "NOLICENCE"，而不是正常路由。

许可证无效时的行为:

- 接收的消息被 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

警报:

```
- 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 - 消息被路由到 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 的消息数量（指示许可证问题）
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[illegible] SMSC 名称
- `source_type`: 源连接类型 (ims, circuit_switched, smpp)
- `message_type`: 消息类型 (sms, mms)

用例: 按来源和类型监控传入消息量。用于检测流量模式、识别繁忙时段和发现消息流中的异常。

警报: 对于突然下降（潜在的源连接问题）或激增（潜在的攻击/垃圾邮件）设置警报。

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。高失败率可能指示格式错误的消息或集成问题。
警报: 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花费的时间（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。
警报: 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：匹配并选择特定���由进行消息路由的总次数。

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：了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非常有用。

警报：如果高优先级路由很少匹配（可能指示路由配置错误），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。

警报：对任何路由失败发出警报，因为它们指示消息无法送达。

---

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。

警报：对丢弃操作的意外激增发出警报（可能指示垃圾邮件攻击）。

---

sms_c_routing_stop_duration

类型：Histogram

描述：评估所有路由并选择最佳匹配所���费的时间。

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。

警报：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：从缓存中提供的 ENUM 查找总数（不需要 DNS 查询）。

标签：

- `domain`：查询的 ENUM 域

用例：监控缓存有效性。高缓存命中率减少 DNS 负载并提高性能。

警报：如果缓存命中率低于阈值发出警报（可能指示缓存问题或异常流量）。

---

sms_c_enum_cache_miss_count

类型：Counter

描述：需要 DNS 查询的 ENUM 查找总数（不在缓存中）。

标签：

- `domain`：查询的 ENUM 域

用例：跟踪缓存未命中以了解缓存有效性。与命中计数一起使用以计算命中率。

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

---

sms_c_enum_cache_size_size

类型：Gauge

描述：ENUM 缓存中当前条目的数量。

用例：监控缓存大小以确保其不会无限增长。帮助调整缓存 TTL 设置。

警报：如果缓存大���超过预期范围发出警报（可能指示内存泄漏）。

---

sms_c_enum_lookup_stop_duration

类型：Histogram

描述：完成 ENUM 查找所需的时间（包括未缓存时的 DNS 查询）。

单位：毫秒

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。
警报: 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: 成功的 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: 了解 ENUM 记录分布。大多数查找应返回 1-3 条记录。
警报: 如果频繁返回 0 条记录发出警报（DNS 配置问题）。

交付指标

sms_c_delivery_queued_count

类型: Counter
描述: 排队等待交付到目标 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。对容量规划非常有用。
警报: 与交付成功/失败计数进行比较以检测积累。

sms_c_delivery_attempted_count

类型: Counter
描述: 进行的交付尝试总数（包括重试）。
标签:
dest_smsc: 目标 SMSC 名称
用例: 跟踪交付尝试量。相对于排队计数的高尝试计数指示重试行为。

sms_c_delivery_succeeded_count

类型: Counter
描述: 成功交付到目标 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交付。主要成功指标。
警报: 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交付的消息总数。
标签:
dest_smsc: 目标 SMSC 名称
reason: 失败原因
用例: 跟踪交付失败以识别问题目标或失败模式。
警报: 对于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（无法交付）的消息总数。
标签:
reason: 死信原因（例如，max_retries_exceeded, expired）
用例: 监控需要人工干预的无法交付的消息。
警报: 对任何死信事件发出警报，因为它们表示完全交付失败。

sms_c_delivery_succeeded_attempt_count

类型: Histogram
描述: 成功交付之前所需的交付尝试次数。
桶: 1, 2, 3, 5, 10
标签:
dest_smsc: 目标 SMSC 名称
用例: 了解重试行为。大多数交付应在第一次尝试时成功。

警报: 如果平均尝试计数超过 2 发出警报（指示目标可靠性问题）。

sms_c_delivery_failed_attempt_count

类型: Histogram

描述: 在最终失败之前进行的交付尝试次数。

桶: 1, 2, 3, 5, 10

标签:

dest_smsc: 目标 SMSC 名称

用例: 了解放弃之前发生了多少次重试。

sms_c_delivery_time_delta_delta_ms

类型: Histogram

描述: 从消息提交到交付确认的时间。该指标捕获从消息进入 SMS-C 到系统确认交付的完整端到端延迟。它提供详细的标签，以便按来源、目标和重试行为进行分析。

单位: 毫秒

桶: 50, 100, 250, 500, 1000, 2000, 5000, 10000, 20000, 60000, 600000 ms (50ms 到 10 分钟)

标签:

source_smsc: 提交消息的源 SMSC
dest_smsc: 交付消息的目标 SMSC
delivery_attempts: 所需的交付尝试次数（0 = 第一次尝试成功）

用例: 按源-目标对分析端到端交付性能。识别缓慢路径、问题 SMSC 组合，并将延迟与重试行为相关联。对 SLA 监控和容量规划至关重要。

警报:

当 p95 超过特定源/目标组合的 SLA 阈值时发出警报
当消息持续需要多次尝试时发出警报

示例 Prometheus 查询:

```
# 按源和目标 SMSC 的 P95 交付时间
histogram_quantile(0.95,
  sum by (source_smsc, dest_smsc, le) (
    rate(sms_c_delivery_time_delta_delta_ms_bucket[5m])
  )
)

# P99 交付时间总计
histogram_quantile(0.99,
  sum by (le) (
    rate(sms_c_delivery_time_delta_delta_ms_bucket[5m])
  )
)

# 按源 SMSC 的平均交付时间
sum by (source_smsc) (rate(sms_c_delivery_time_delta_delta_ms_sum[5m]
/
sum by (source_smsc)
(rate(sms_c_delivery_time_delta_delta_ms_count[5m]))

# 需要重试的消息的交付时间与第一次尝试成功的交付时间
histogram_quantile(0.95,
  sum by (le)
(rate(sms_c_delivery_time_delta_delta_ms_bucket{delivery_attempts="0"
[5m]))
)
histogram_quantile(0.95,
  sum by (le)
(rate(sms_c_delivery_time_delta_delta_ms_bucket{delivery_attempts!="0
[5m]))
)

# 最慢的源-目标对（按 p95）
topk(10,
  histogram_quantile(0.95,
    sum by (source_smsc, dest_smsc, le) (
      rate(sms_c_delivery_time_delta_delta_ms_bucket[1h])
    )
  )
)

# 在 2 秒内交付的消息百分比
sum(rate(sms_c_delivery_time_delta_delta_ms_bucket{le="2000"}[5m])) /
```

```
sum(rate(sms_c_delivery_time_delta_delta_ms_count[5m])) * 100

# 按目标交付的 2 秒内交付的消息
sum by (dest_smsc)
(rate(sms_c_delivery_time_delta_delta_ms_bucket{le="2000"}[5m])) /
sum by (dest_smsc) (rate(sms_c_delivery_time_delta_delta_ms_count[5m]
* 100
```

---

sms_c_delivery_time_by_smsc_delta_ms

类型: Histogram

描述: 从消息提交到交付确认的时间，仅按源和目标 SMSC 标记。这是 `sms_c_delivery_time_delta_delta_ms` 的低基数伴随指标 - 它省略了 `delivery_attempts` 标签，因□可以直接读取每个源/目标对的平均交付时间，而无需跨尝试计数进行聚合。从相同的 `[:sms_c, :delivery, :time_delta]` 事件中发出。

单位: 毫秒

桶: 50, 100, 250, 500, 1000, 2000, 5000, 10000, 20000, 60000, 600000 ms (50ms 到 10 分钟)

标签:

- `source_smsc`: 提交消息的源 SMSC
- `dest_smsc`: 交付消息的目标 SMSC

用例: 仪表板显示每个源/目标对的平均端到端交付时间。在平均/SLA 面板中优先使用□指标，而不是 `sms_c_delivery_time_delta_delta_ms`，因为不需要尝试计数的细分；当需要按重试行为切片时使用 `_delta` 变体。

示例 **Prometheus** 查询:

```
# 按源/目标 SMSC 的平均交付时间（毫秒）
sum by (source_smsc, dest_smsc)
(rate(sms_c_delivery_time_by_smsc_delta_ms_sum[5m])) /
sum by (source_smsc, dest_smsc)
(rate(sms_c_delivery_time_by_smsc_delta_ms_count[5m]))

# 总体平均交付时间
sum(rate(sms_c_delivery_time_by_smsc_delta_ms_sum[5m])) /
sum(rate(sms_c_delivery_time_by_smsc_delta_ms_count[5m]))

# 按源/目标 SMSC 的 P95 交付时间
histogram_quantile(0.95,
  sum by (source_smsc, dest_smsc, le) (
    rate(sms_c_delivery_time_by_smsc_delta_ms_bucket[5m])
  )
)
```

队列指标

sms_c_queue_size_size

类型: Gauge

描述：当前队列中的消息总数（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。

警报：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: 当前待交付的消息数量（尚未尝试）。
标签:
queue_type: 队列类型
用例: 监控待处理消息计数。高待处理计数指示处理延迟。
警报: 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: 队列类型
用例: 监控失败消息的积累。指示交付问题。
警报: 对于高失败计数发出警报，因为这会影响交付率。

sms_c_queue_size_delivered

类型: Gauge
描述: 当前等待清理/从队列中移除的已交付消息数量。
标签:
queue_type: 队列类型
用例: 监控清理滞后。高计数表明清理过程滞后。
警报: 如果已交付消息显著积累，则发出警报。

sms_c_queue_oldest_message_age_seconds

类型：Gauge

描述：当前处于待处理状态的最旧消息的年龄（以秒为单位）。

标签：

- `queue_type`：队列类型

用例：检测消息老化和处理停滞。对 SLA 监控至关重要。

警报：当最旧消息的年龄超过 SLA 阈值时发出警报（例如，> 300 秒）。

计费指标

计费通过 **Diameter Ro** 接口（信用控制）进行。`account` 标签携带被收费用户的 MSISDN（消息发送者）。对于明确的余额不足拒绝（`reason: :no_credit`）和 OCS 错误（传输/协议）��都会发出 `charging_failed` 事件。只有 `:no_credit` 拒绝会拒绝消息（状态 `:balance_rejected`）；OCS 错误会失败并且消息会被交付。

sms_c_charging_requested_count

类型：Counter

描述：发送到 OCS 的计费请求总数（Diameter Ro CCR 事件请求）。

标签：

- `account`：被收费用户的 MSISDN（消息发送者）

用例：跟踪每个账户的计费量。对账务对账非常有用。

---

sms_c_charging_succeeded_count

类型：Counter

描述：成功计费操作的总数。

标签：

- `account`：被收费的账户标识符

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`：失败原因

用例：识别可能影响收入或需要账户干预的计费失败。

警报：对于高计费失败率发出警报。

---

sms_c_charging_succeeded_duration

类型：Histogram

描述：完成成功计费请求所需的时间。

单位：毫秒

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

标签：

- `account`：账户标识符

用例：监控计费系统性能。缓慢的费可能会延迟消息交付。

警报: 当 p95 计费时间超过阈值时发出警报。

系统健康指标

sms_c_mnesia_table_size_record_count

类型: Gauge

描述: 每个 Mnesia 数据库表中的当前记录数量。每 10 秒轮询一次。

标签:

table: 表名

跟踪的表:

sms_route - SMS 路由规则
message_store - 消息队列（待处理、已交付、失败的消息） location_store - 用户位置/注册数据 frontend_store - 前端/SMSC 注册 translation_rule - 数字翻译规则 cell_tower_store - 基站位置数据 message_events - 消息事件日志

用例: 监控数据库增长。检测意外数据积累。容量规划。

警报: 对于意外的表增长率或接近容量限制时发出警报。

示例 Prometheus 查询:

```
# 所有表的大小
sms_c_mnesia_table_size_record_count

# 消息队列大小
sms_c_mnesia_table_size_record_count{table="message_store"}

# 活跃用户注册
sms_c_mnesia_table_size_record_count{table="location_store"}

# 路由□则计数
sms_c_mnesia_table_size_record_count{table="sms_route"}
```

---

sms_c_frontend_status_count

类型: Gauge

描述: 每个连接状态的前端数量。

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。

警报: 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。

警报：设置警报以监控：

- 注册率下降（可能指示网络问题）
- 注册的异常激增
- 非 IMS 注册的高比例（遗留设备涌入）

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]))
```

---

sms_c_location_active_registrations_count

类型：Gauge

描述：当前活跃用户注册的数量，按位置（前端/SMSC）和 IMS 能力分组。该指标每 10 秒轮询一次，反映位置存储的当前状态。

标签：

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

用例：监控当���用户在前端之间的分布。跟踪 IMS 采用情况。识别前端之间的负载不平衡。为每个 SMSC 实例进行容量规划。

警报：设置警报以监控：

- 某个位置的注册突然下降（可能指示前端问题）
- 前端之间的分布不平衡
- 总注册接近容量限制

示例 **Prometheus** 查询：

```
# 总活跃注册
sum(sms_c_location_active_registrations_count)

# 按位置的活跃注册
sum by (location) (sms_c_location_active_registrations_count)

# 按位置的活跃 IMS 能力注册
sum by (location)
(sms_c_location_active_registrations_count{ims_capable="true"})

# 每个位置的 IMS 采用率
sum by (location)
(sms_c_location_active_registrations_count{ims_capable="true"}) /
sum by (location) (sms_c_location_active_registrations_count) *
100

# 总 IMS 与非 IMS 注册
sum by (ims_capable) (sms_c_location_active_registrations_count)

# 注册最多的位置
topk(10, sum by (location)
(sms_c_location_active_registrations_count))

# 注册分布占总数的百分比
sum by (location) (sms_c_location_active_registrations_count) /
sum(sms_c_location_active_registrations_count) * 100
```

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（毫秒）从请求开始到响应完成。

标签:

- `method`: HTTP 方法（例如，`GET`，`POST`）

- `path`: 请求路径，高基数段（UUID、数字 ID）折叠为 `/:id`（例如，`/api/messages`，`/api/messages/:id`）

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

用例: 监控 API 性能。识别慢端点。跟踪响应时间 SLA。

警报: 设置警报以监控:

- 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]))

# 慢于 1 秒的请求
sum(rate(phoenix_endpoint_stop_duration_bucket{le="1000"}[5m]))
```

---

phoenix_endpoint_stop_count

类型: Counter

描述: 按方法、路径和 HTTP 状态码分类的完成的 HTTP 请求总数。

标签:

- `method`: HTTP 方法（例如，`GET`，`POST`）
- `path`: 请求路径（高基数段折叠为 `/:id`）
- `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。

警报: 设置警报以监控:

- 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：

```
# 每个路径的请求率
sum by (path) (rate(phoenix_endpoint_stop_count[5m]))

# 按路径的错误率
sum by (path) (rate(phoenix_endpoint_stop_count{status=~"5.."}
[5m])) /
sum by (path) (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。

警报：对于关键端点的任何非零值设置警报。

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])
```

Diameter Sh 指标

SMS-C 通过 **Diameter Sh** 接口（3GPP TS 29.329）查询 HSS，以在路由评估期间将用户分类为在网或离网。这些指标仅在启用 Diameter 时发出。

sms_c_diameter_peer_status

类型: Gauge

描述: Diameter 对等体是否连接（`1`）或未连接（`0`）。

标签:

- `origin_host`: 对等体的配置 Diameter Origin-Host
- `ip`: 对等体 IP 地址

示例查询:

```
# 当前宕机的对等体
sms_c_diameter_peer_status == 0
```

---

sms_c_diameter_sh_request_count

类型: Counter

描述: 发送的 Diameter Sh（HSS）查找请求总数（UDR）。

---

sms_c_diameter_sh_response_count

类型: Counter

描述: 按结果分类的 Diameter Sh 查找响应总数（UDA）。

标签:

- `result`: 查找结果（例如，`on_net`, `unknown_user`, `error`）
- `result_code`: Diameter Experimental-Result / Result-Code（例如，`2001`, `5001`）

示例查询:

```
# Sh 查找错误比率
sum(rate(sms_c_diameter_sh_response_count{result="error"}[5m]))
  / sum(rate(sms_c_diameter_sh_response_count[5m]))
```

---

sms_c_diameter_sh_response_duration_ms

类型: Distribution (Histogram)

描述: Diameter Sh 查找持续时间（毫秒）。

标签:

- `result`: 查找结果

桶: 5, 10, 25, 50, 100, 250, 500, 1000, 2500 ms

示例查询:

```
# P95 Sh 查找延迟
histogram_quantile(0.95,
rate(sms_c_diameter_sh_response_duration_ms_bucket[5m]))
```

---

sms_c_diameter_sh_error_count

类型: Counter
描述: Diameter Sh 查找错误总数（传输/超时/解码失败，与负 UDA 不同）。
标签:
reason: 错误原因

联邦指标

联邦在 SMS-C 控制器之间通过拉取式对等模型转发消息。请参见 联邦。

sms_c_federation_forward_attempted_count

类型: Counter
描述: 转发到远程控制器的消息总数。
标签:
dest_controller: 目标控制器身份

sms_c_federation_forward_completed_count

类型: Counter
描述: 成功交付到远程控制器的消息总数。
标签:
dest_controller: 目标控制器身份

sms_c_federation_forward_queued_count

类型: Counter
描述: 因为对等体无法访问而排队的消息总数。
标签:
dest_controller: 目标控制器身份

sms_c_federation_forward_failed_count

类型: Counter
描述: 超过最大重试次数的消息总数。
标签:
dest_controller: 目标控制器身份

sms_c_federation_message_received_count

类型: Counter
描述: 从远程控制器接收到的消息总数。
标签:
origin_node: 源控制器节点

sms_c_federation_health_check_count

类型: Counter
描述: 按对等体和结果的健康检查尝试总数。
标签:

- `peer`：对等体身份
- `result`：健康检���结果（例如，`healthy`，`unhealthy`）

---

sms_c_federation_peer_status_count

类型：Gauge

描述：按状态分类的当前联邦对等体数量。

标签：

- `status`：对等体状态（例如，`healthy`，`degraded`，`unhealthy`）

---

sms_c_federation_queue_depth_count

类型：Gauge

描述：当前联邦转发队列的深度。

示例查询：

```
# 如果转发队列积压，则发出警报
sms_c_federation_queue_depth_count > 1000
```

Erlang VM 指标

vm_memory_total

类型：Gauge

描述：Erlang VM 分配的总内存（以字节为单位）。

用例：监控整体内存使用情况。检测内存泄漏。规划容量。

警报: 当内存使用量 > 80% 可用系统内存时发出警报。

vm_memory_processes
类型: Gauge
描述: Erlang 进程使用的内存（以字节为单位）。
用例: 跟踪进程内存消耗。最常见的内存增长来源。
警报: 对持续的高增长率发出警报。

vm_total_run_queue_lengths_total

类型: Gauge
描述: 所有 CPU 调度器中等待调度的进程总数。
用例: 测量系统负载。高值表示 CPU 饱和。
警报: 当持续 > 10 * CPU 核心数量时发出警报。

vm_system_counts_process_count

类型: 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。
警报: 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

按目标的交付成功率

跟踪每个目标 SMSC 的消息交付成功率：
公式: (sms_c_delivery_succeeded_count) / (sms_c_delivery_queued_count) 解释: 应该 > 95% 对于健康的目标。较低的比率指示交付问题。

端到端消息延迟

监控从接收消息到交付的总时间：
指标:
sms_c_message_processing_stop_duration (处理) sms_c_delivery_time_delta_delta_ms (交付)
解释: 总和代表用户面向的延迟。

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。较低的比率可能指示短 TTL 或高流量方差。

路由利用率

识别处理最多流量的路由：

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（追赶）：

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

了解交付重试的频率：

指标：`sms_c_delivery_succeeded_attempt_count` 直方图百分位数

解释：如果 p95 > 1，大多数消息需要重试。指示目标可靠性问题。

推荐警报

警报	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
交付失败激增	`delivery_failed_count` 激增	High	目标问题
死信事件	`delivery_dead_letter_count` > 0	High	无法交付的消息
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/交付）
2. 队列大小（待处理、失败、已交付）
3. 按目标的交付成功率
4. p95 处理和交付延迟
5. 活跃前端状态
6. 当前警报

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 查找持续时间直方图
4. 计费持续时间直方图
5. 交付尝试分布
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: 消息未交付

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` - 交付尝试是否失败?
5. 检查 `dest_smsc` 标签以识别问题目标

场景：消息处理缓慢

调查步骤：

1. 检查 `sms_c_message_processing_stop_duration` 直方图 - 整体处理时间
2. 检查 `sms_c_routing_stop_duration` - 路由是否缓慢？
3. 检查 `sms_c_enum_lookup_stop_duration` - ENUM 查找是否缓慢？
4. 检查 `sms_c_charging_succeeded_duration` - 计费是否缓慢？
5. 确定瓶颈并调查特定组件

场景：消息队列增长

调查步骤：

1. 检查 `sms_c_queue_size_pending` 趋势 - 它是否在增长？
2. 检查 `sms_c_delivery_attempted_count` - 是否发生了交付尝试？
3. 检查 `sms_c_delivery_failed_count` - 它们是否失败？
4. 检查 `sms_c_delivery_time_delta_delta_ms` - 交付是否花费太长时间？
5. 检查 `dest_smsc` 标签以识别缓慢的目标

Prometheus 查询示例

消息吞吐量

每秒接收的消息（**5** 分钟平均）：

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

每分钟接收的消息（**1** 小时平均）：

```
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]))
```

交付性能

交付成功率（百分比）：

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

交付失败率（百分比）：

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

平均交付尝试（**p95**）：

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

按目标的交付成功：

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

交付失败原因：

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

交付时间（**p95**）：

```
histogram_quantile(0.95,
sms_c_delivery_time_delta_delta_ms_bucket)
```

交付时间（**p99**）：

```
histogram_quantile(0.99,
sms_c_delivery_time_delta_delta_ms_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（每小时消息）：

```
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 **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 **NAPTR** 记录的平均数量：

```
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（图表）
    - 查询：`rate(sms_c_message_received_count[5m])`
    - 查询：`rate(sms_c_delivery_succeeded_count[5m])`
    - 单位：消息/秒
    - 图例：{{source_type}}

2. 交付成功率（仪表）

查询: (rate(sms_c_delivery_succeeded_count[5m]) / rate(sms_c_delivery_queued_count[5m])) * 100
单位: 百分比（0-100）
阈值:
红色: < 90
黄色: 90-95
绿色: > 95

3. 队列深度（图表）

查询: sms_c_queue_size_pending
查询: sms_c_queue_size_failed
单位: 消息
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: 分钟
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: 计数
颜色: 蓝色

6. 路由失败（图表）

查询: rate(sms_c_routing_failed_count[5m]) * 60
单位: 失败/分钟
警报阈值: > 0

仪表板 2: ���能分析

目的：分析系统性能并识别瓶颈。

面板：

1. 端到端延迟（图表）
    - 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：毫秒
    - 图例：百分位数
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)`
    - 交付：`histogram_quantile(0.95, sms_c_delivery_time_delta_delta_ms_bucket)`
    - 单位：毫秒
    - 水平条
3. 交付尝试分布（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（图表）

- 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：百分比
   - 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 与 CS 与 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

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. 按目标的交付（图表）

查询: sum by (dest_smsc)
(rate(sms_c_delivery_succeeded_count[5m]))
单位: 消息/秒
堆叠区域图
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（图表）

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（图表）

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（图表）

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（图表）

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 = 追赶

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: 消息/秒

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: 百分比
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）

交付成功: (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 内交付的消息（状态）

查询: count(sms_c_delivery_time_delta_delta_ms_bucket{le="5000"})
/ count(sms_c_delivery_time_delta_delta_ms_bucket)
显示在 5 秒内交付的百分比
单位: 百分比

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 = 停止
显示当前状态

5. 每日成功率趋势（图表）

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%

警报规则示例

关键警报

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: "消息无法路由。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: "消息队列有 {{ $value }} 条待处理消息"
  description: "队列正在积压。检查交付性能。"
```

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: "最旧消息的年龄为 {{ $value }} 秒"
  description: "消息未被交付。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: "没有可用的交付路径。检查前端连接性。"
```

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 }} 消息移动到死信队列"
  description: "消息变得无法交付。调查失败。"
```

警告警报

低交付成功率：

```
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: "交付成功率为 {{ $value | humanizePercentage }}"
  description: "成功率低于 95%。调查交付失败。"
```

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 百分位交付尝试次数: {{ $value }}"
  description: "消息需要多次尝试。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: "消息处理缓慢。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。可能指示独特号码流量。"
```

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 连接性。"
```

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
Histogram 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，请使用适当的时间范围：5m 用于实时，1h 用于趋势，24h+ 用于容量规划
在 Prometheus 中设置记录规则以提高仪表板性能，以便频繁使用复杂查询
在 Grafana 中使用变量模板以实现动态仪表板（选择 dest_smsc、source_smsc 等）

MMS 中心 (MMSc)

目录

- 概述
- 架构
- 消息提交处理
- 交付流程
- MMS 通知
- 在网/离网判定
- 配置
- 存储
- 运行时行为
- 可观察性
- 故障排除

概述

OmniMessage 包含一个本地的 **MMS** 中心 **(MMSc)**，用于接收、存储和发送多媒体消息。它实现了 3GPP TS 23.140 定义的两个 MMS 互通接口：**MM1** 用于直接手机通信 (HTTP/WAP) 和 **MM4** 用于运营商间中继 (SMTP)。

MMSc 内置于 OmniMessage 核心中。它不作为单独的服务运行，也没有外部数据库或 Web 服务器依赖。当用户发送或接收 MMS 时，MMSc：

1. 将原始 MMS 内容存储在本地文件系统中，并将其元数据存储在 Mnesia 中。
2. 构建一个 **MMS** 通知，并通过用于���条 SMS 的相同进程管道将其作为普通的移动终止 SMS 提交，然后进行翻译、路由和交付。
3. 当检索手机获取通知 URL 时，通过 MM1 将存储的 MMS 返回给该手机。

由于通知作为普通 SMS 注入，MMS 重用了所有现有的 SMS 路由、翻译和交付机制，没有自导的 HTTP 调用。

相关规范

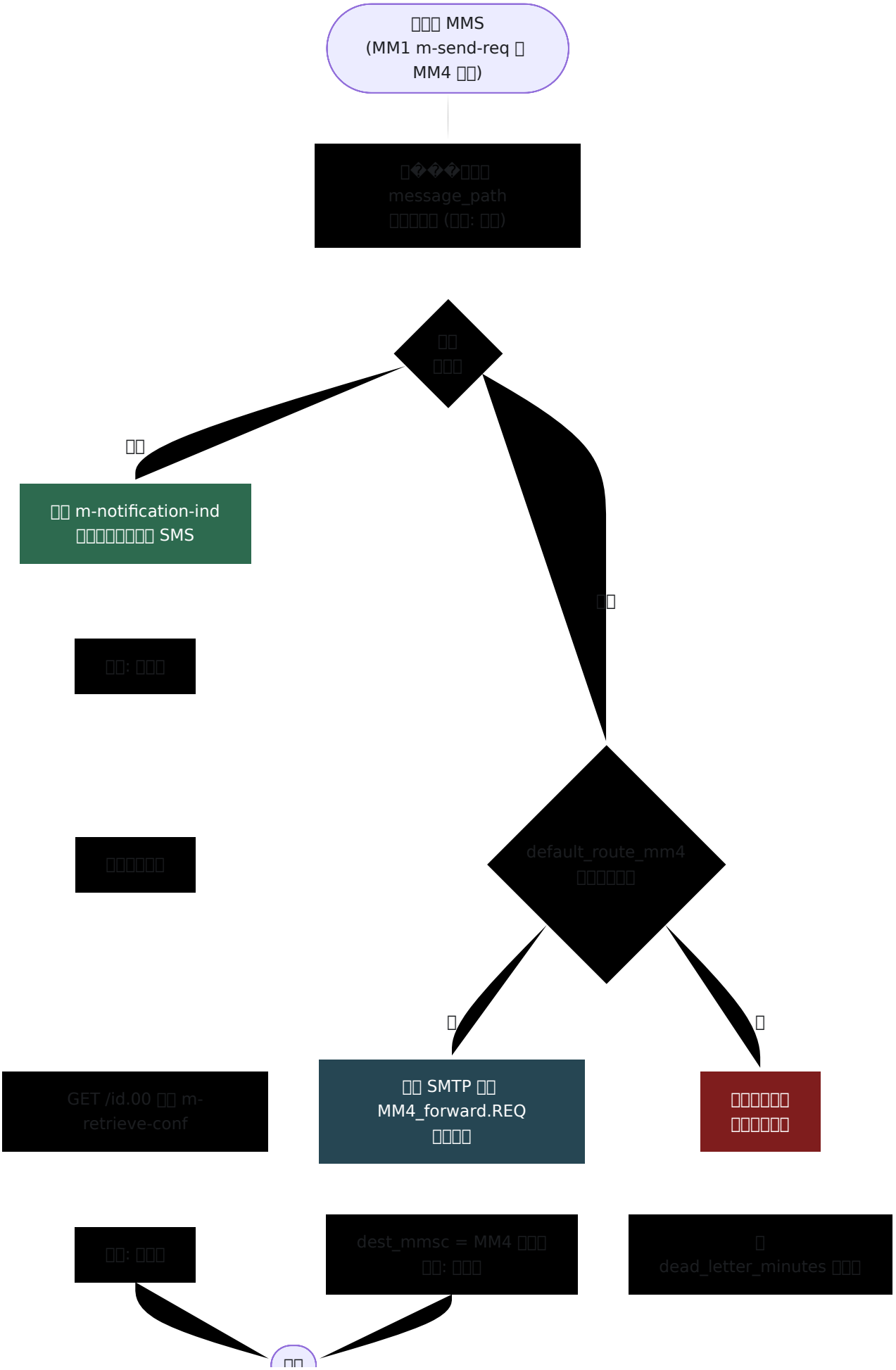
规范	标题
3GPP TS 23.140	多媒体消息服务 (MMS): 功能描述
3GPP TS 26.140	MMS: 媒体格式和编解码器
OMA MMS 封装协议	MM1 WAP 二进制编码

架构

MM1 (手机 HTTP) 和 MM4 (运营商间 SMTP) 接口各自监听自己的专用端口，与 OmniMessage API 分开。MMS 元数据存储在 Mnesia 中，与 SMS 消息存储一起。MMSc 将原始 MMS 内容作为文件保存在本地文件系统中。元数据记录仅存储文件路径。

消息提交处理

每个提交的 MMS（来自手机通过 MM1，或来自对等方通过 MM4）遵循相同的处理路径：存储内容、存储元数据、分类目标，然后要么本地通知，要么离网中继。

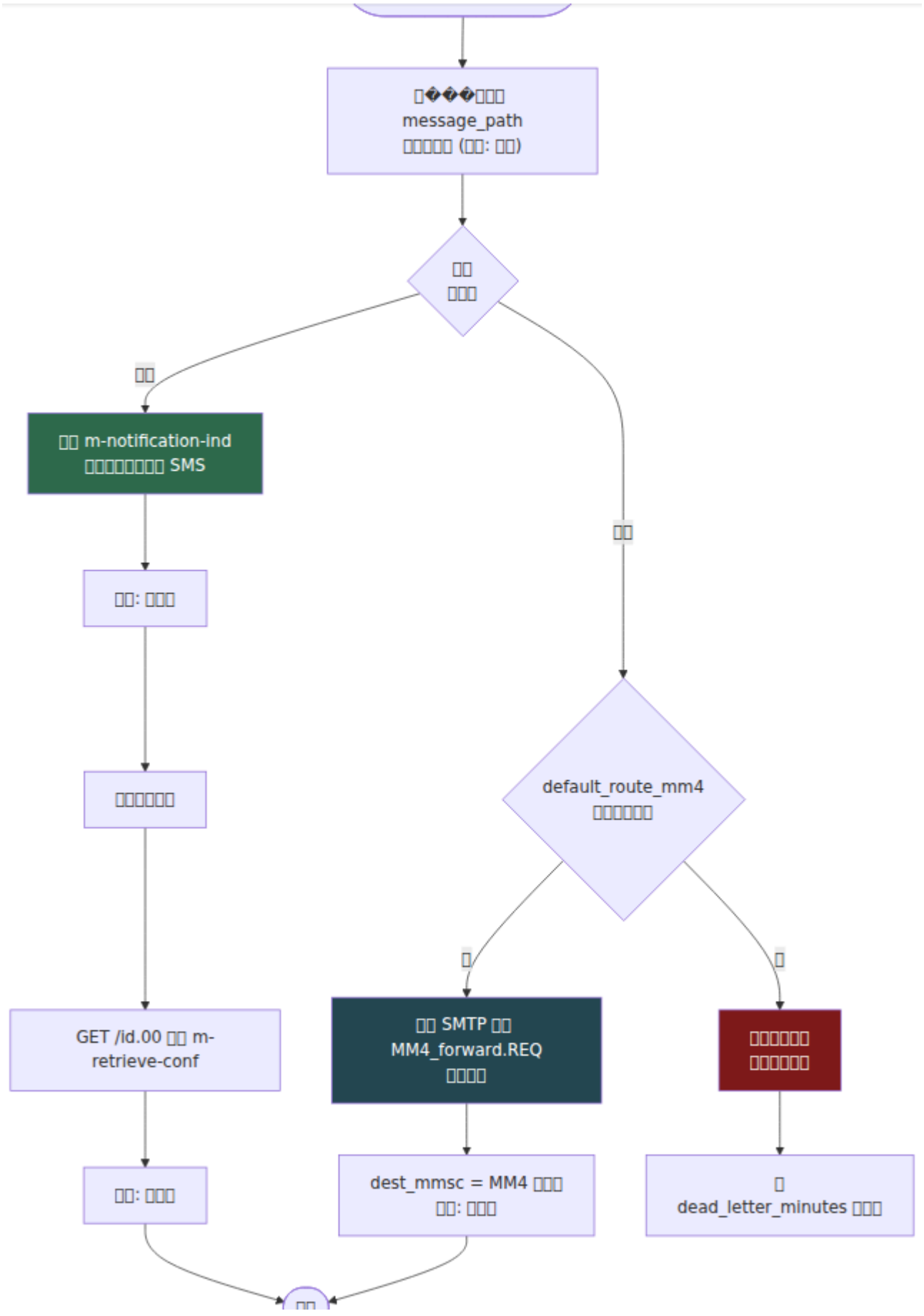


接收到 MMS
(MM1 m-send-req 或
MM4 入站)
□□□□□□□
message_path
□□□□□□ (□□: □□)
□□
□□□
在网
构建 m-notification-ind
通过管道提交通知 SMS
□□: □□□
□□□□□□□
□网
default_route_mm4
□□□□□□□
是
否
GET /id.00 □□ m-
retrieve-conf
通过 SMTP 转发
MM4_forward.REQ
到对等方
没有中继目标
状态保持存储
□□: □□□
dest_mmsc = MM4 □□□
□□: □□□
□
dead_letter_minutes □□□
□□

交付流程

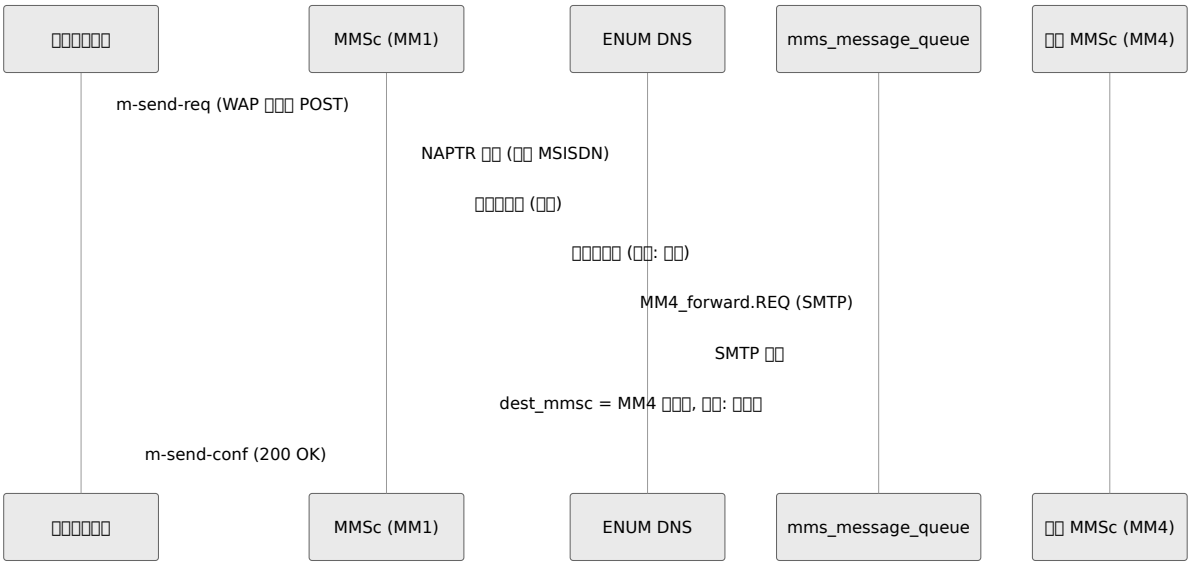
在网交付 **(MM1)**

用户向同一网络的接收者发送 MMS。


□□□□□□□□
message_path
□□□□□□ (□□: □□)
□□
□□□
□□
□□ m-notification-ind
□□□□□□□□□ SMS
□□: □□□
□□□□□□□
GET /id.00 □□ m-
retrieve-conf
□□: □□□
default_route_mm4
□□□□□□□
□
□□ SMTP □□
MM4_forward.REQ
□□□□
dest_mmsc = MM4 □□□
□□: □□□
□
□□□□□□□
□□□□□□□
□
dead_letter_minutes □□□

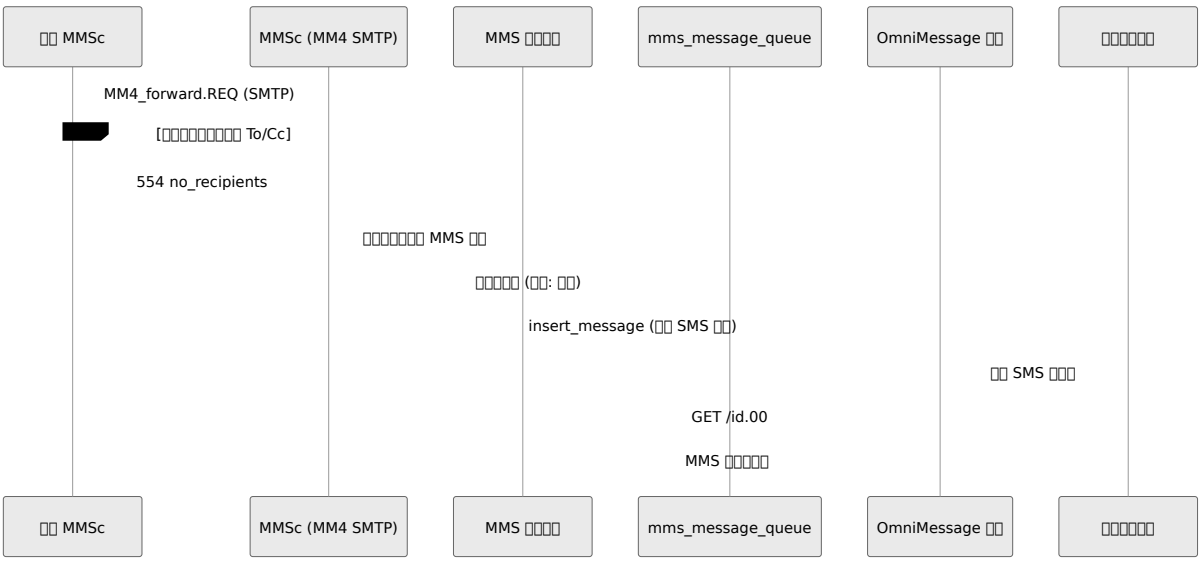
□□□□ (MM4)

□□□□□□□□□□□□□□□□ MMS□

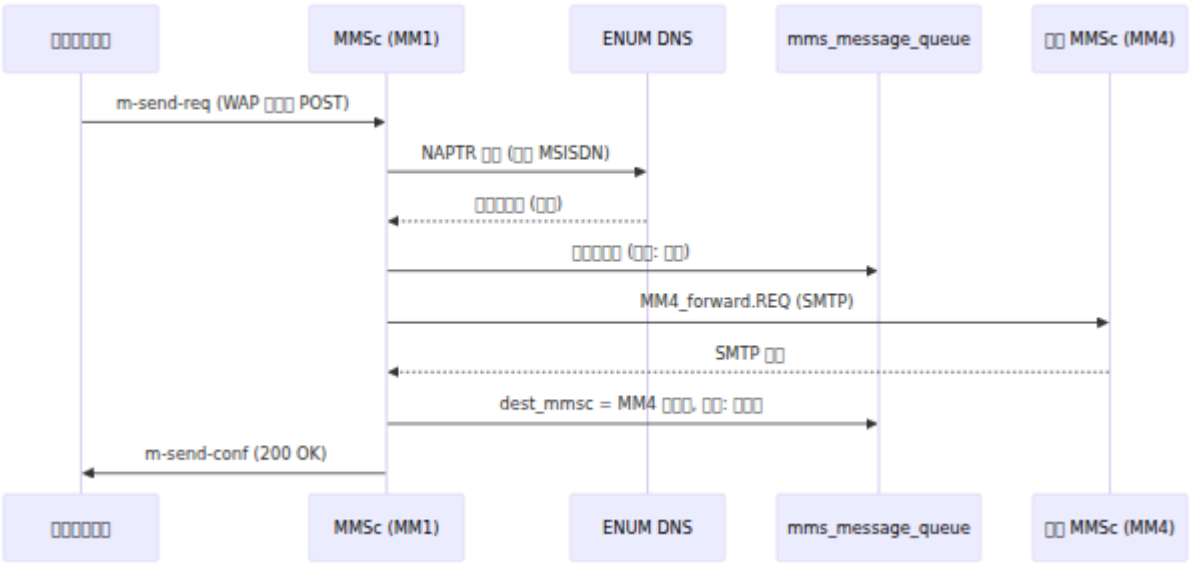


□□ MM4 (□□□□□□□□)

□□□□□□□□□□□□□□□□ MMS□



- □□ □□□ SMTP □□□□□ (`RCPT TO`); □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□`To`/`Cc` □□□ □□□
- □□□□`To` + `Cc` □□□□□□□□ `recipient_list` □□□□□□□□□□ `m-retrieve-conf` □□□□ □□□□□□□□□□□□□□□□□□ MMS□



- 媒体：MIME 树中的每个非 SMIL 部分（包括嵌套多部分）都被重新编码为存储的 WSP 内容。文本部分从其声明的字符集转换为 UTF-8。
- 在信封或 `To`/`Cc` 中没[illegible]收件人的消息被拒绝，返回 `554` 而不是接受并丢弃。

MMS 通知

MMS 通知是一个二进制 SMS，携带 **WSP** 编码的 `m-notification-ind` PDU。它使用 8 位数据编码（DCS `0x44`），被拆分为连接的 SMS 段，并作为普通 MT SMS 部分通过进程内管道提交到消息队列。因[illegible]，每个 SMS 的翻译、路由和交付规则都适用于它。

通知中编码的检索位置为：

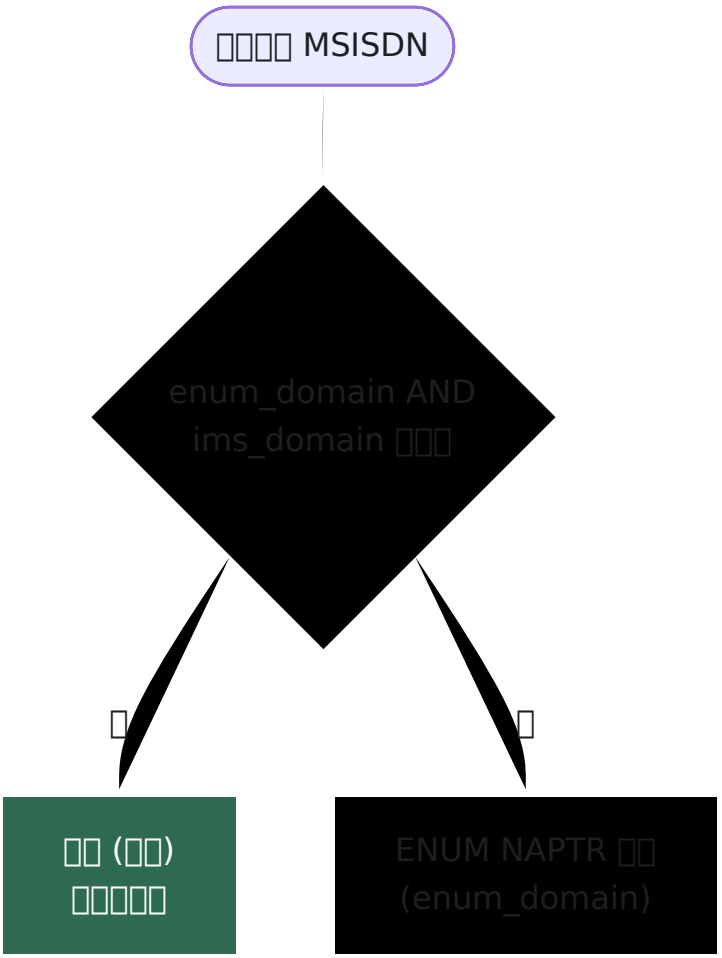
```
http://<mms_domain>/<unique_id>.00
```

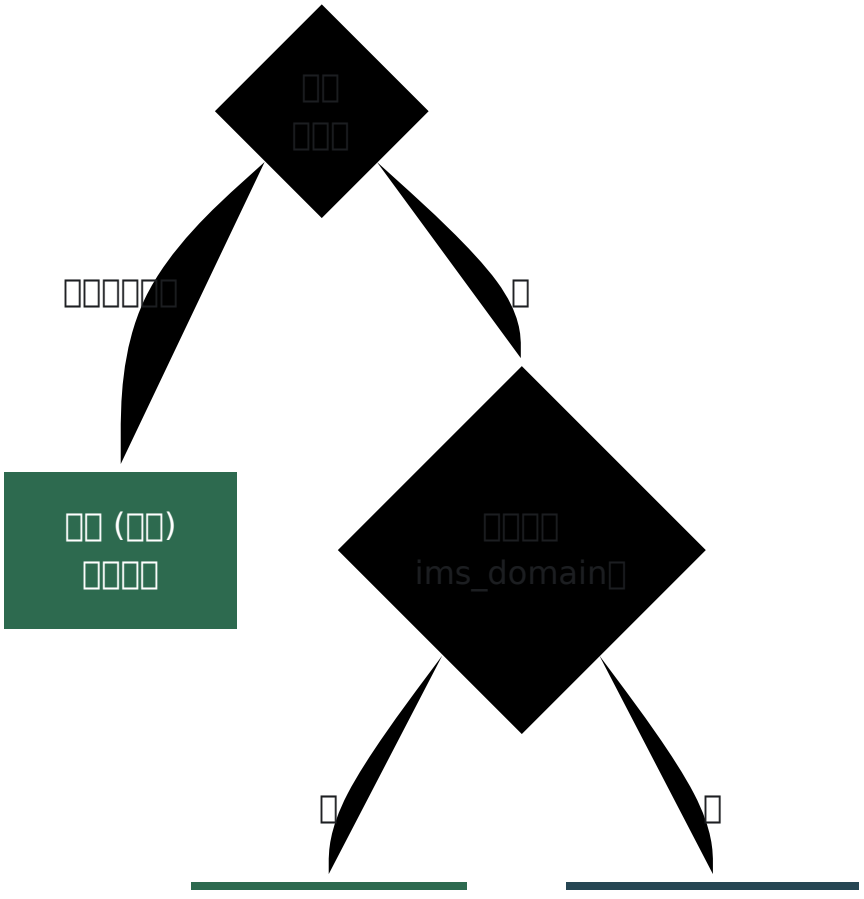
- `mms_domain` 是配置的 `mmsc.mms_domain`。
- `<unique_id>.00` 是检索密钥。MM1 处理程序去掉 `.00` 后缀以解析存储的消息。

接收手机通过 MM1 打开[illegible] URL 以获取 MMS 作为 `m-retrieve-conf`。

在网/离网判定

在 MMSc 决定是本地通知还是通过 MM4 中继之前，它重用 SMS-C [illegible] ENUM (NAPTR) 机制来分类目标。一个 ENUM 结果，如果其正则表达式/替换引用了配置的 `ims_domain`，则判定为在网。





分类目标 MSISDN
enum_domain AND ims_domain □□□
否
是
在网（默认）
存储并通知
ENUM NAPTR □□ (enum_domain)
□□ □□□
未命中或□□
是
在网（保守）
尝试通知
□□□□ ims_domain□
是
否

在网
注入通知 SMS

离网
通过 MM4 转发

条件	分类	动作
`ims_domain` 或 `enum_domain` 未配置	在网（默认）	本地存储并注入通知 SMS
ENUM NAPTR 结果引用 `ims_domain`	在网	本地存储并注入通知 SMS
ENUM NAPTR 结果不引用 `ims_domain`	离网	本地存储并通过 MM4 转发到对等方
ENUM 未命中或查找错误	在网（保守）	本地存储并尝试通[illegible]

当 ENUM 未配置时，所有目标都被视为在网（存储并通知）。只有在明确分类为离网的目标上才会尝试 MM4 中继。

配置

您可以在 `runtime.exs` 中的 `:sms_c` 应用程序的 `:mmsc` 键下配置 MMSc。它默认禁用。如果 `:mmsc` 块缺失，则该功能完全处于休眠状态：MMSc 不绑定任何端口，也不启动任何监听器。因[illegible]，现有部署在您明确启用之前不会受到影响。

配置结构

每个配置键驱动 MMSc 的特定部分。该图显示哪些键控制哪些过程。

配置示例

```
# config/runtime.exs
config :sms_c, :mmsc,
  # 核心
  enabled: true,
  port: 8080,
  host: "0.0.0.0",
  mms_domain: "mms.example.com",
  message_path: "/var/lib/omnimessage/stored_mms",
  notify_source_msisdn: "6675",
  dead_letter_minutes: 1440,

  # 在/离网 (ENUM) 分类
  enum_domain: "e164.arpa",
  ims_domain: "ims.mnc001.mcc001.3gppnetwork.org",

  # 发送者解析 (MM1 客户端 IP 到 MSISDN)
  subscriber_ip_api_urls: ["https://10.0.0.10:8443",
"https://10.0.0.11:8443"],
  pcrf_api_url: nil,
  hss_api_url: nil,

  # 入站/出站 MM4 (运营商间)
  mm4_enabled: false,
  mm4_smtp_port: 25,
  mm4_domain: "mms.example.com",
  default_route_mm4: nil
```

`:mmsc` □□

□□	□□	□□	□□	
`enabled`	□□□□	□	`false`	□□
`port`	□□	□	`8080`	MM
`host`	□□□□	□	`0.0.0.0`	MM□
`mms_domain`	□□□□	□□□□□□	-	□s
`message_path`	□□□□	□	`/var/lib/omnimessage/stored_mms`	□□
`notify_source_msisdn`	□□□□	□	`6675`	□
`dead_letter_minutes`	□□	□	`1440`	□
`enum_domain`	□□□□	□	-	E□

□□	□□	□□	□□	
`ims_domain`	□□□□	□	`""`	□□
`subscriber_ip_api_urls`	□□□□□□□	□	`[]`	□□ □□ □
`pcrf_api_url`	□□□□	□	`nil`	□□ □
`hss_api_url`	□□□□	□	`nil`	□□
`mm4_enabled`	□□□□	□	`false`	□□
`mm4_smtp_port`	□□□	□	`25`	□
`mm4_domain`	□□□□	□	□□□ `mms_domain`	□
`default_route_mm4`	□□□□	□	`nil`	□□

MM4 对等方

出站 MM4 中继目标单独配置，因为它们是结构化的主机/端口/域记录，而不是标量设置：

```
# config/runtime.exs
config :sms_c, :mmsc_mm4_peers, %{
  "carrier-a" => [
    %{host: "mm4.carrier-a.example.net", port: 25, mms_domain:
"mms.carrier-a.example.net"}
  ]
}
```

`mmsc.default_route_mm4` 选择使用哪个对等组。每个组是一个列表；当它包含多个条目时，每次中继尝试选择一个，以便负载分散到对等方的入站主机。

MM4 对等参数：

参数	类型	必需	默认	描述
`host`	字符串	是	-	对等 MM4 SMTP 主机名或 IP。用作 SMTP 中继和 HELO 主机名。
`port`	整数	否	`25`	对等 MM4 SMTP 端口。
`mms_domain`	字符串	否	`""`	对等方的 MMS 域，用于构建 RCPT 地址（`+<msisdn>/TYPE=PLMN@<mms_domain>`）并记录为 `dest_mmsc`。

存储

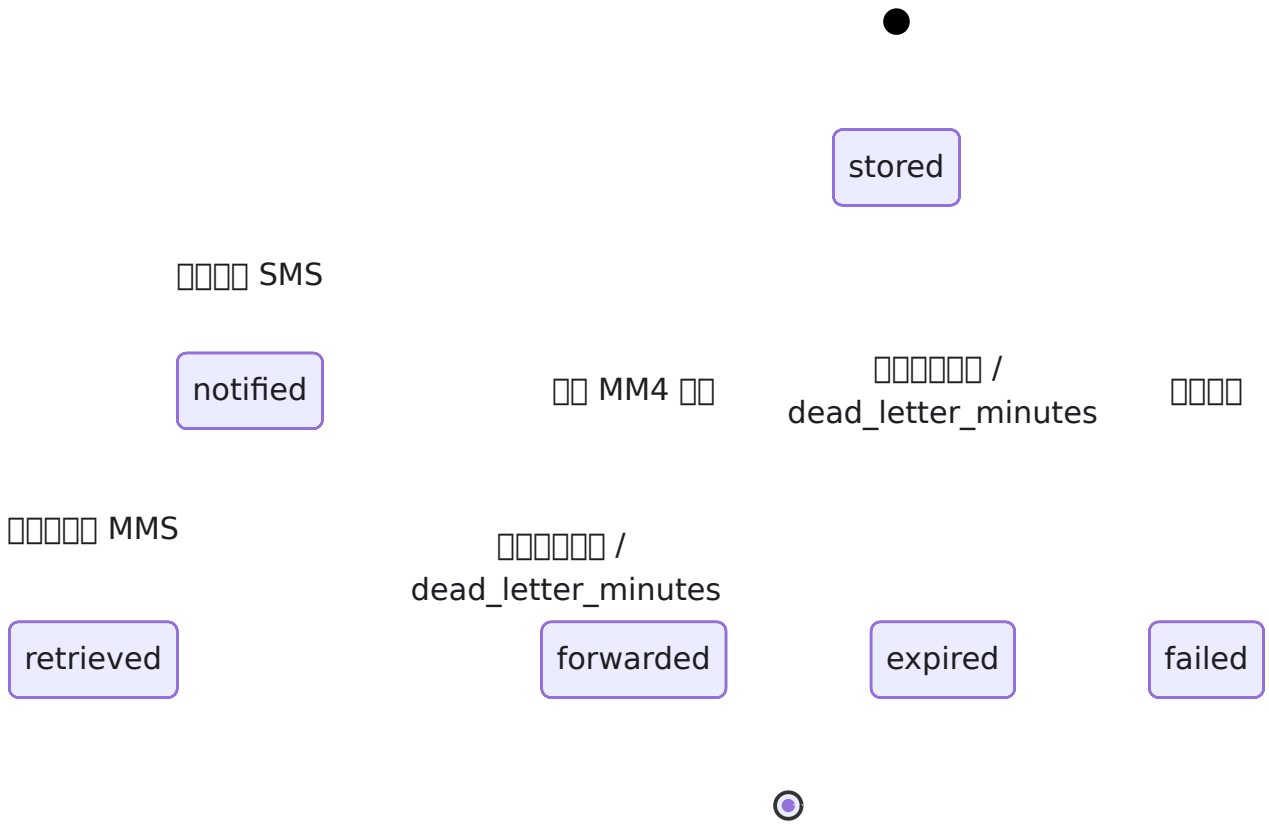
MMS 元数据存储在 Mnesia 表 `:mms_message_queue` 中，该表在启动时自动创建，与 SMS 消息存储一起。没有单独的数据库需要配置。原始 MMS 内容存储为 `.mms` 文件在 `message_path` 下；仅其路径保存在元数据记录中。

`mms_message_queue` 字段

字段	描述
`mms_id`	来自 Mnesia 计数器的整数主键。
`unique_id`	用于检索 URL 路径中的外部标识符（`<unique_id>.00`）。
`source_msisdn`	发送用户的 MSISDN。
`destination_msisdn`	主要接收者的 MSISDN。
`recipient_list`	组 MMS 和检索确认的完整 To/Cc 列表 (CSV)。
`subject`	MMS 主题行。
`file_path`	存储的 `.mms` 内容在 `message_path` 下的路径。
`source_mmsc`	发起消息的 MMSc 或 MM4 对等方的身份。
`dest_mmsc`	在出站中继后设置为 `MM4_<peer>`，或在本地交付后设置为交付目标。
`on_net`	MMSc 是否将目标分类为在网。
`notify_message_ids`	为通知创建的 SMS 消息 ID（链接到 SMS 消息存储）。
`send_time`	MMSc 接收 MMS 的时间。
`notify_time`	系统提交通知 SMS 的时间。
`retrieve_time`	订阅者检索 MMS 的时间。
`expires`	MMS 不再交付的过期时间。
`status`	生命周期状态（见下文）。
`deadletter`	当消息被标记为死信时设置。

□□	□□
`inserted_at` / `updated_at`	□□□□□□□□□□□

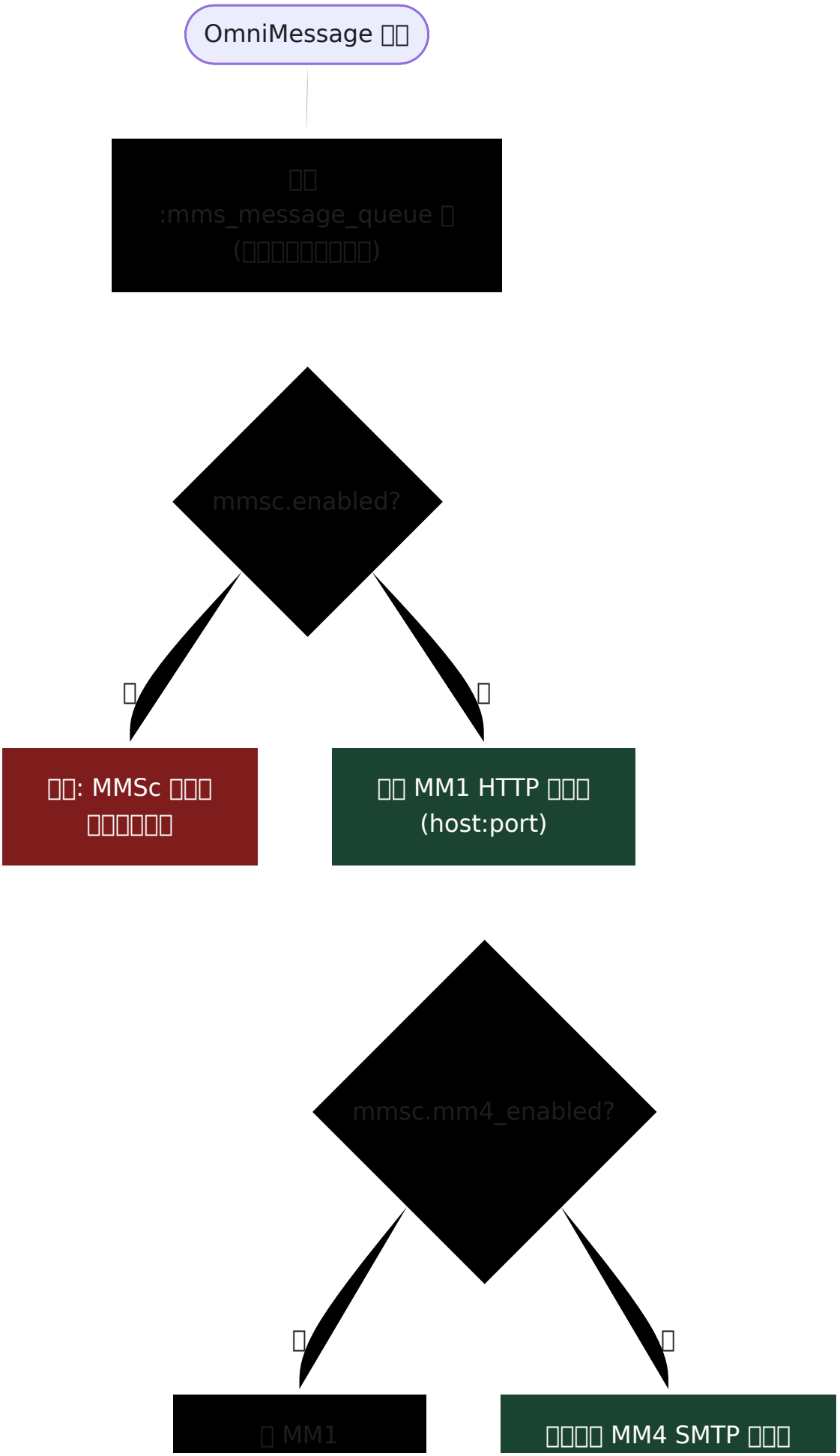
□□□□□□



□□	□□
`stored`	□□□□□□□□□□□□□□□□□□□□□□□□□□□□
`notified`	□□ SMS □□□□□□□□□□□□□□□□□□
`retrieved`	□□□□□ MM1 □□□ MMS□
`forwarded`	□□ MM4 □□□□□□□□□□
`expired`	□□□□□□□□□ `dead_letter_minutes` □□□□□
`failed`	□□□□□□□□□

运行时行为

MMSc 作为受监督的进程在 OmniMessage 发布中运行。没有单独部署或启动的内容。该图显示了在启动时每个监听器的配置控制。



OmniMessage 启动
□□
:mms_message_queue □
(□□□□□□□□□□)
mmsc.enabled?
否
是
日志: MMSc 已禁用
未启动监听器
启动 MM1 HTTP 监听器
(host:port)
mmsc.mm4_enabled?
否
是
□ MM1
启动入站 MM4 SMTP 服务器

- **MM1 HTTP** 监听器 仅在 `mmsc.enabled` 为 `true` 时启动。它绑定 `mmsc.host:mmsc.port` 并处理手机提交（`m-send-req`）、通知响应（`m-notifyresp-ind`）、确认（`m-acknowledge-ind`）和检索（`GET /<unique_id>.00`）。
- **MM4 SMTP** 服务器 仅在 `mmsc.enabled` 和 `mmsc.mm4_enabled` 均为 `true` 时启动。它绑定 `mmsc.mm4_smtp_port` 并接受来自对等运营商的入站 `MM4_forward.REQ` 消息。
- 出站 **MM4** 中继 在目标被分类为非网内时在进程内执行；它通过 SMTP 连接到配置的对等方。不需要外部邮件传输代理。
- 表创建 始终在启动时运行。当 MMSc 被禁用时，这只会创建一个空的 `:mms_message_queue` 表，这是无害的。

端口绑定：默认的 MM4 端口（25）是特权端口。OmniMessage 服务必须以能够绑定低端口的权限运行（`CAP_NET_BIND_SERVICE`），或者在端口转发后使用更高的 `mm4_smtp_port`。MM1 监听器默认使用非特权端口 8080。

TLS：MM1 监听器提供普通 HTTP。如果需要面向手机的加密，请在 `mmsc.port` 前的反向代理处终止 TLS。

可观察性

MMSc 目前不发出专用的 MMS Prometheus 指标。在网内交□□□会将通知 SMS 注入标准管道。因□，正常的 OmniMessage 消息指标显示 MMS 通知。这些通知在消息队列和 CDR 中的表现与其他 SMS 相同。请参见 指标参考。

对于每条消息的追踪，元数据记录的 `status`、`send_time`、`notify_time` 和 `retrieve_time` 字段跟踪消息的生命周期。

故障排除

MMSc 未启动

症状：配置端口上没有 MM1 监听器；启动日志显示 `MMSc: DISABLED`。

可能原因：

- `runtime.exs` 中缺少 `:mmsc` 块。
- `mmsc.enabled` 为 `false`。

解决方案：

1. 添加 `:mmsc` 配置块并设置 `enabled: true`。
2. 重启服务并确认启动日志报告 `MMSc: ENABLED`，并显示预期的端口和域。

手机无法检索 MMS

症状：通知 SMS 已送达，但手机无法获取内容。

可能原因：

- `mms_domain` 无法解析或无法从手机的 MMS APN 路由。
- 反向代理或防火墙阻止了 `mmsc.port`。
- 存储的 `.mms` 文件在 `message_path` 下缺失。

解决方案：

1. 确认通知 URL 主机（`mms_domain`）可以解析并且可以从 MMS APN 访问。
2. 验证对 `mmsc.host:mmsc.port` 的入站访问（以及任何 TLS 终止代理）。
3. 检查消息 `status` 并确认 `.mms` 文件存在于 `file_path`。

非网内 MMS 未中继

症状： MMS 到其他运营商被存储但从未转发；`status` 保持为 `stored`。

可能原因：

- `default_route_mm4` 未设置，因□未选择对等组。
- `:mmsc_mm4_peers` 中的对等条目具有错误的主机/端口。
- 由于 `ims_domain`/`enum_domain` 未设置，目标被错误分类为网内（所有内容都被视为网内）。

解决方案：

1. 将 `default_route_mm4` 设置为 `:mmsc_mm4_peers` 中存在的键。
2. 验证对等方的 `host`/`port` 并确认允许出站 SMTP。
3. 配置 `enum_domain` 和 `ims_domain` 以便正确分类非网内目标。

入站 MM4 被拒绝或未接收

症状：对等运营商无法通过 MM4 进行交付。

可能原因：

- `mm4_enabled` 为 `false`，因 SMTP 服务器未运行。
- 消息在 SMTP 信封或 `To`/`Cc` 头中没有指定收件人，因被拒绝，返回 `554 no_recipients`。
- 服务没有权限绑定特权端口 25。
- 防火墙阻止了入站 SMTP。

解决方案：

1. 设置 `mm4_enabled: true` 并重启。
2. 授予 `CAP_NET_BIND_SERVICE` 权限或设置非特权的 `mm4_smtp_port` 在端口转发后。
3. 允许来自对等方出口地址的入站流量到 `mm4_smtp_port`。

发件人地址解析失败

症状：来自非本地主机手机的提交因订阅者查找误而被拒绝。

可能原因：

- `subscriber_ip_api_urls` 和 `pcrf_api_url` / `hss_api_url` 均未设置，因只有本地主机可以解析为测试 MSISDN。
- 没有配置的 HSS 节点持有手机的 IP 绑定（UE 的数据会话尚未为 HSS 所知）。
- HSS/PCRF 端点无法访问或返回非 200 响应。

解决方案：

1. 配置 `subscriber_ip_api_urls`，包括每个 HSS 节点（或同时设置 `pcrf_api_url` 和 `hss_api_url`）以实现生产环境的发件人解析。
2. 验证端点可访问并返回预期的订阅者/MSISDN 映射。

SMS-C号码转换指南

← 返回文档索引

概述

SMS-C号码转换系统提供灵活的基于正则表达式的电话号码转换功能，适用于路由之前。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，并将多个转换链在一起。规则存储在Mnesia中以确保持久性，并且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：在应用转换之前按前缀匹配号码
- □于正则表达式的转换：强大的模式匹配和替换功能，支持捕获组
- 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：通过循环防止机制继续处理多个规则
- 独立的主叫/被叫转换：对主叫和被叫号码进行独立转换
- 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**：完整的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	自增唯一标识符	是（自动）
`calling_prefix`	字符串/nil	主叫号码的前缀匹配（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	匹配主叫号码的正则表达式模式。也作为网关的源TON/NPI覆盖	否
`calling_replace`	字符串/nil	主叫号码的替换模式	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
`source_ton`	字符串/nil	强制源地址的号码类型。空白/缺失 = 保持推断	否
`source_npi`	字符串/nil	强制源地址的编号计划指示符。空白/缺失 = 保持推断	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：规则按优先级顺序评估（数字越小越优先）。只有启用的规则才会被评估。

有关`source_ton`和`source_npi`字段的完整文档，请参见强制号码类型（TON）和编号计划指示符（NPI）。

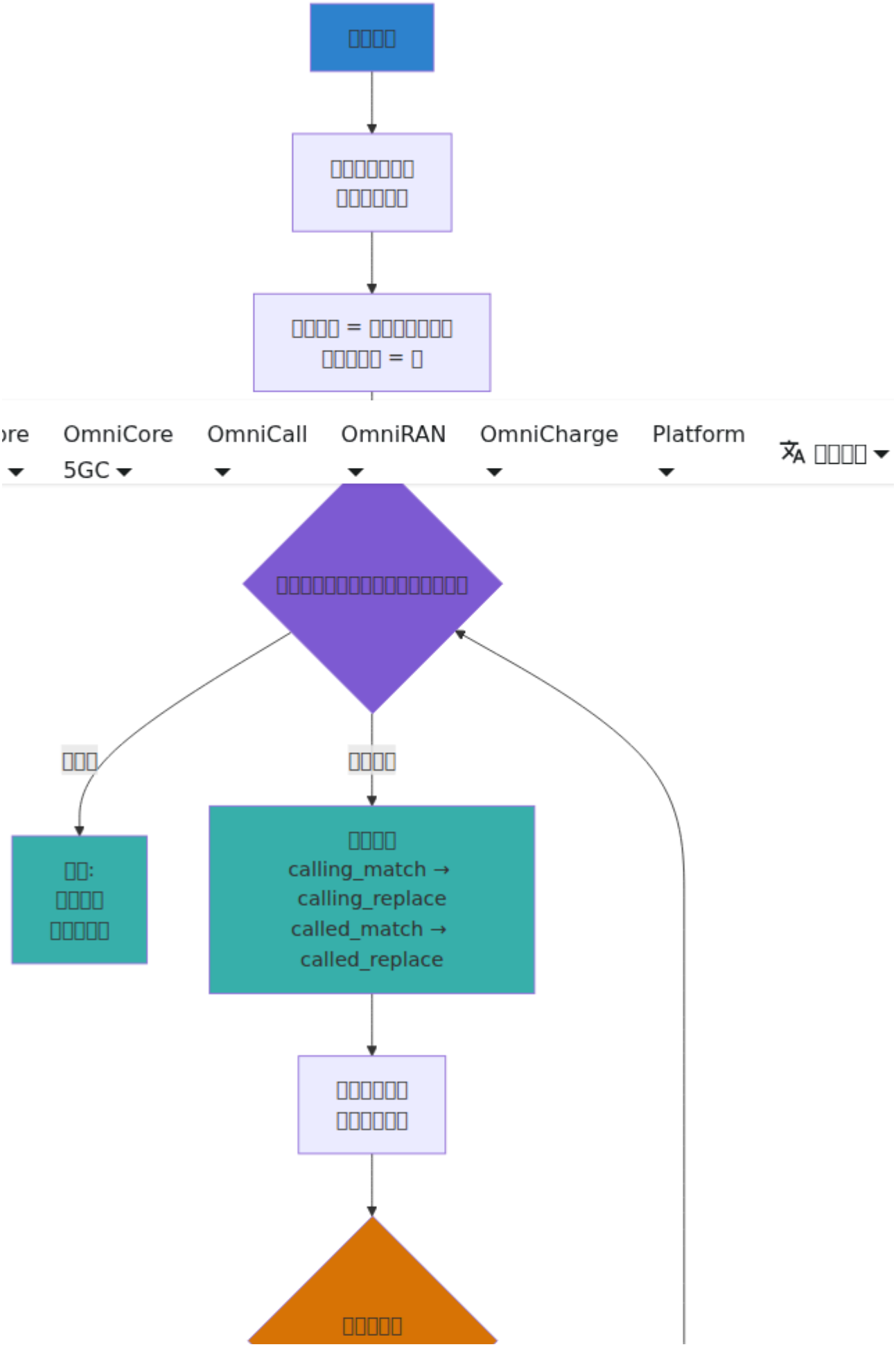
转换算法

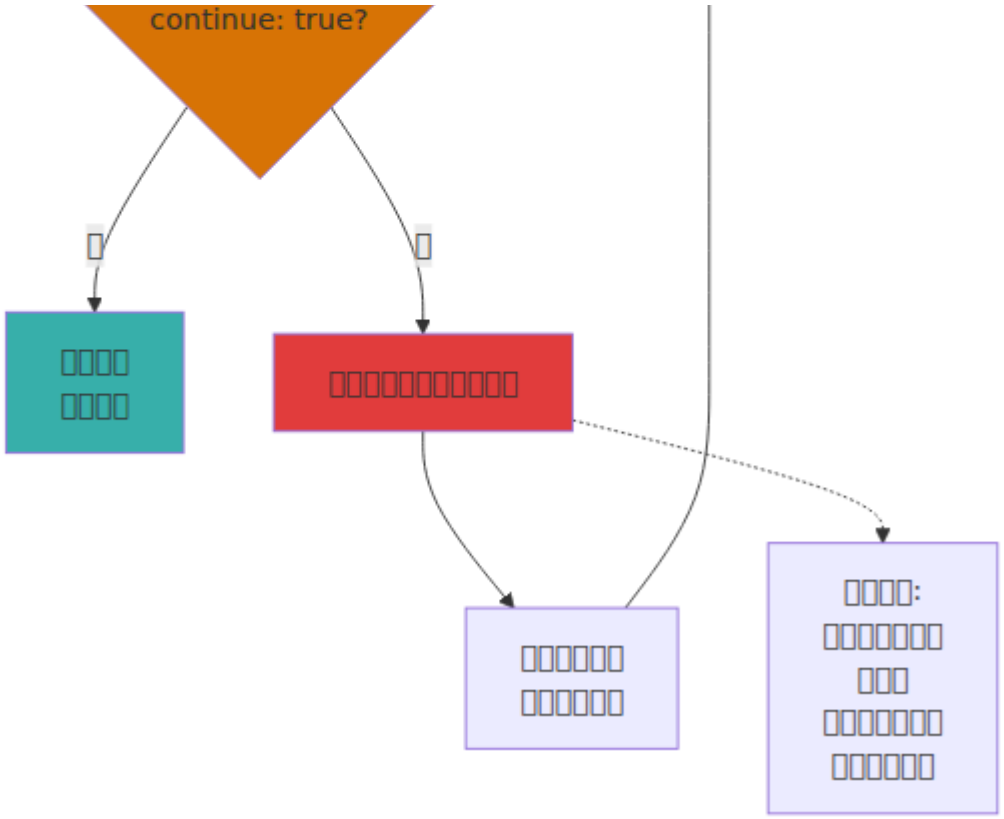
在转换号码时，系统：

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，根据消息参数：
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转换主叫号码使用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

规则链式处理与循环防止

continue标志启用强大的规则链式处理，同时防止无限循环：


OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
Platform
calling_match →
calling_replace
called_match →
called_replace



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

强制号码类型（**TON**）和编号计划指示符（**NPI**）

除了基于正则表达式的号码重写外，转换规则可以强制源（主叫）地址的号码类型（TON）和编号计划指示符（NPI）。当发送者的号码字符串本身无法传达正确的地址语义时，这一点非常有用。例如，短代码必须被标记为 `unknown`/`unknown`，而不是被视为国家或国际号码。

这些覆盖与号码重写**独立的：规则可以在不更改号码的情况下强制TON/NPI。

覆盖字段

字段	适用对象	效果
`source_ton`	源（主叫）地址	强制构建的SMS-DELIVER主叫地址（TPDU）的号码类型
`source_npi`	源（主叫）地址	强制构建的SMS-DELIVER主叫地址（TPDU）的编号计划指示符

空白或缺失 **=** 无覆盖。当字段留空（或规则完全缺失）时，地址值将被推断 **-** 请参见下面的推断回退。

> 注意： `source_ton`/`source_npi`直接嵌入编码在**TPDU**中的主叫地址（IMS/SS7前端传输的SMS-DELIVER）。该值存在于编码的TPDU中 **-** 消息记录上不存储任何额外信息。没有目标TON/NPI覆盖：SMS-DELIVER TPDU不携带目标地址，前端自行推导目标地址。

允许的值

号码类型（**TON**）：

值	含义
`unknown`	号码类型未知
`international`	国际号码（完整的E.164格式，带国家代码）
`national`	国家（显著）号码
`network_specific`	网络特定号码
`subscriber`	订��者号码
`alphanumeric`	字母数字地址（例如，品牌名称发送者ID）
`abbreviated`	缩略号码

编号计划指示符（**NPI**）：

值	含义
`unknown`	编号计划未知
`isdn`	ISDN/电信编号计划（E.164）
`data`	数据编号计划（X.121）
`telex`	Telex编号计划
`land_mobile`	陆地移动编号计划（E.212）
`national`	国家编号计划
`private`	私人编号计划
`ermes`	ERMES编号计划

任何超出这些集合的值在创建或更新规则时都会被拒绝。

匹配门控：将覆盖范围限制为发送者模式

覆盖是由规则的**主叫号码匹配（`calling_match`）**正则表达式门控的，这使得单个规则可以通过模式针对一部分发送者：

- 只有当规则的`calling_match`正则表达式匹配主叫号码时，源覆盖（`source_ton`/`source_npi`）才会被应用。
- 空白匹配是通配符 - 当规则在其他方面被选择时（通过前缀和/或源SMSC），覆盖会被应用。

规则可以携带一个`calling_match`，而没有相应的`calling_replace`。在这种配置中，匹配仅作为覆盖的门控，并不重写号码。这是将TON/NPI覆盖仅应用于符合模式的号码的推荐方式。

当`continue: true`链式处理多个规则时，TON/NPI覆盖会以最后写入优先的语义累积：后面的规则设置非空值会覆盖先前的值；空值永远不会清除先前设置的覆盖。

推断回退

当没有规则强制覆盖时，源地址值会在TPDU构建时从号码字符串中推断：

输入	推断**TON**	推断**NPI**
数字，≥ 5位	`international`	`isdn`
数字，< 5位	`national`	`isdn`
非数字（字母数字发送者，例如`Acme`）	`alphanumeric`	`unknown`

强制覆盖始终优先于推断值。仅强制TON（留空NPI）会覆盖TON，同时保持NPI推断，反之亦然。

设置**TON/NPI**覆盖

这些字段可以通过三种接口进行设置：

- **Web UI -** 号码转换规则编辑器提供源TON和源NPI下拉框。留空表示“无覆盖”。
- **REST API -** `/api/translations`端点接受`source_ton`和`source_npi`作为字符串值（见示例）。
- 配置 - `config :sms_c, :translation_rules`列表接受相同字段作为字符串值，在首次启动时加载。

示例：短代码发送者

一个3-6位的短代码发送者应标记为TON/NPI `unknown`/`unknown`，不重写号码。规则仅将`calling_match`作为门控（无`calling_replace`）：

配置（**`config/runtime.exs`**）：

```
config :sms_c, :translation_rules, [
  %{
    calling_match: "^\d{3,6}$",
    calling_replace: nil,
    source_ton: "unknown",
    source_npi: "unknown",
    priority: 5,
    continue: true,
    description: "短代码（3-6位）=> unknown/unknown TON/NPI",
    enabled: true
  }
]
```

工作原理：该规则会针对每条消息进行评估（无`calling_prefix`、`called_prefix`或`source_smsc`限制）。`calling_match`正则表达式`^\d{3,6}$`门控源覆盖，因此`source_ton`/`source_npi`仅在主叫号码为裸露的3到6位数字字符串时被强制为`unknown`/`unknown`。由于`calling_replace`为`nil`，号码本身保持不变。`continue: true`允许后续规则运行。

用例：来自短代码的应用程序发起的流量，否则会被错误推断为`national`（4位）或`international`（5-6位）号码。

等效的**REST API**请求 - `POST /api/translations`：

```
{
  "calling_match": "^\d{3,6}$",
  "calling_replace": null,
  "source_ton": "unknown",
  "source_npi": "unknown",
  "priority": 5,
  "continue": true,
  "description": "运营商短代码（3-6位）=> unknown/unknown TON/NPI",
  "enabled": true
}
```

API返回创建的规则（包括其分配的`rule_id`），TON/NPI值序列化为字符串。

示例：强制已知范围的国家

强制已知本地号码范围的源地址为`national`/`isdn`，不重写号码：

配置（`config/runtime.exs`）：

```
%{
  calling_prefix: "07",
  calling_match: "^07\d{8}$",
  calling_replace: nil,
  source_ton: "national",
  source_npi: "isdn",
  priority: 20,
  continue: false,
  description: "本地07xxxxxxx号码 => national/isdn",
  enabled: true
}
```

工作原理：`calling_prefix: "07"`将规则限制为以`07`开头的号码，`calling_match`正则表达式门控覆盖，精确匹配`07`后跟8位数字。当匹配时，源地址被标记为`national`/`isdn`，而不是推断的`international`/`isdn`（这将适用，因为号码有≥ 5位）。号码不被重写。

用例：一个网关或对等方要求国家格式的本地号码被标记为`national` TON，即使它们以原始拨号形式存储。

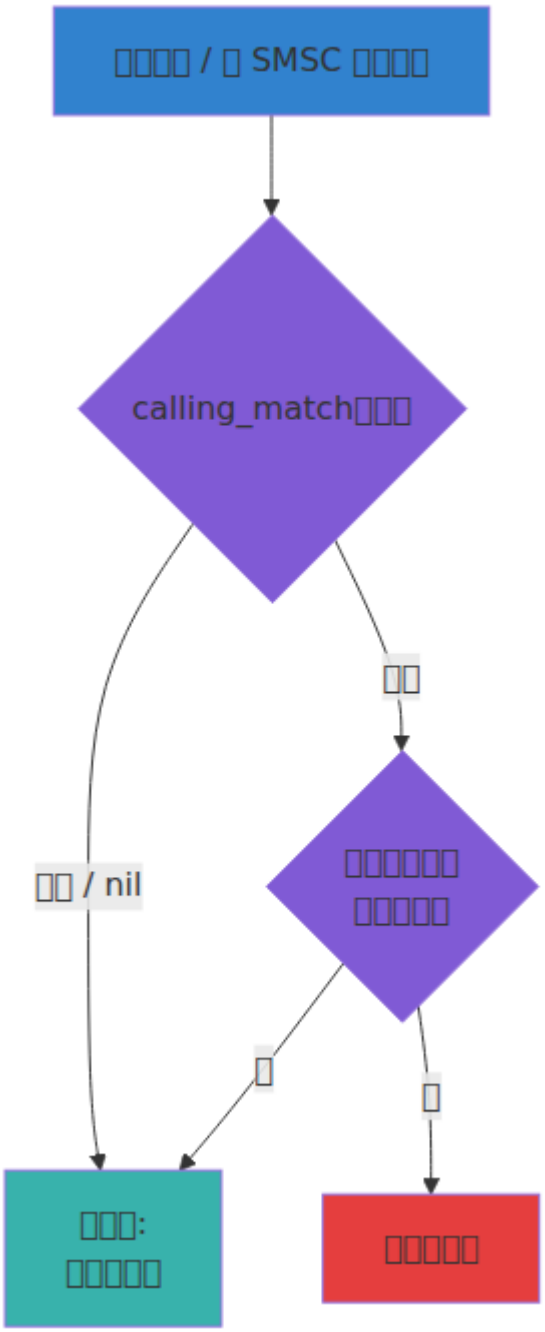
配置

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（首次启动）时加载配置中的规则。这保留了在运行时通过Web UI添加的规则，并防止重启时出现重复。

配置加载流程



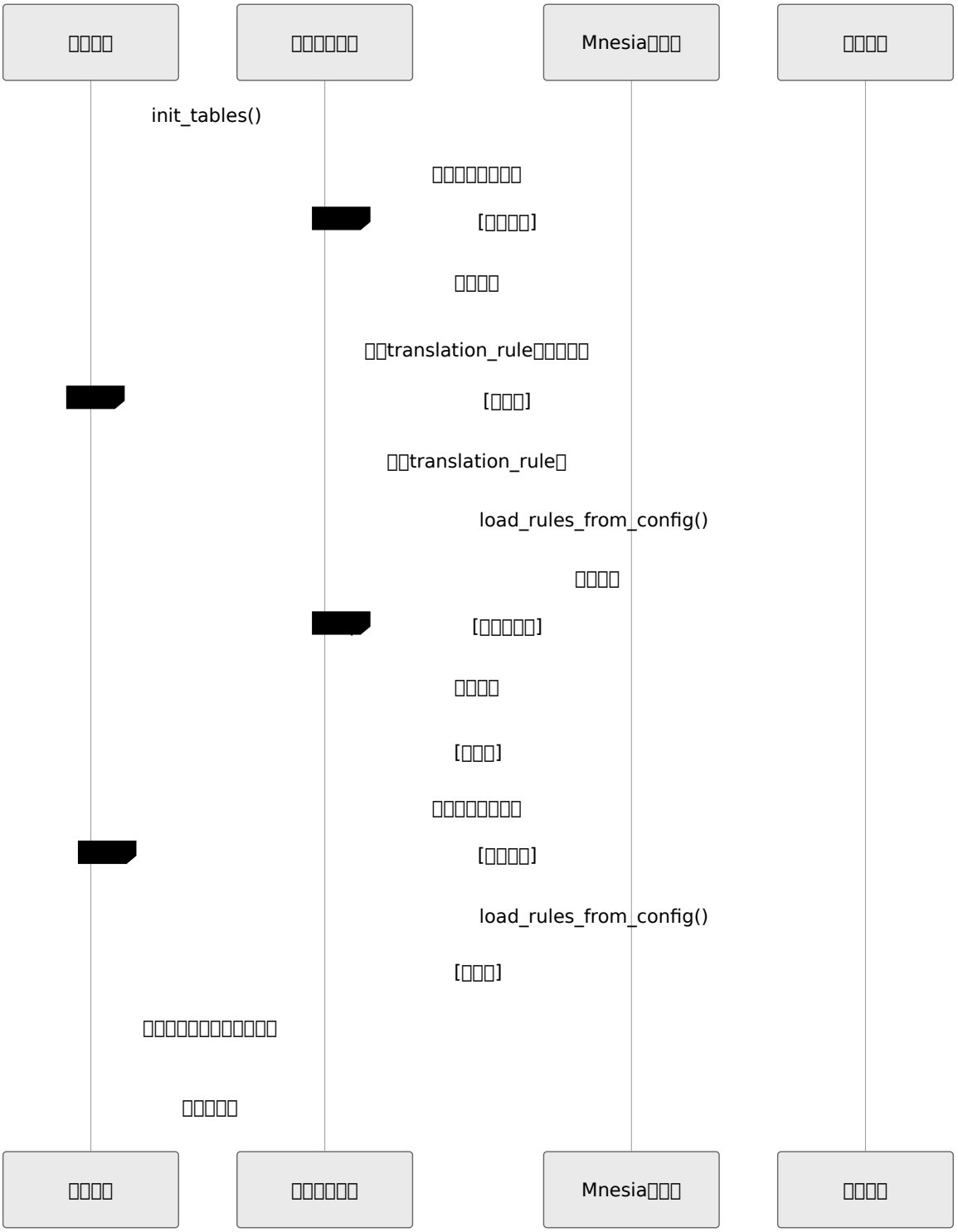
示例配置

```
# 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
  },

  # 从国际格式中去掉前导零
  %{
    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
  },

  # 强制短代码发送者为unknown/unknown TON/NPI（仅门控，不重写）
  %{
    calling_match: "^\d{3,6}$",
    calling_replace: nil,
    source_ton: "unknown",
    source_npi: "unknown",
    priority: 6,
    description: "短代码（3-6位）=> unknown/unknown TON/NPI",
    enabled: true,
    continue: true
  },
```

```
  # 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: "为遗留网关格式化英国号码",
    enabled: true,
    continue: false
  }
]
```



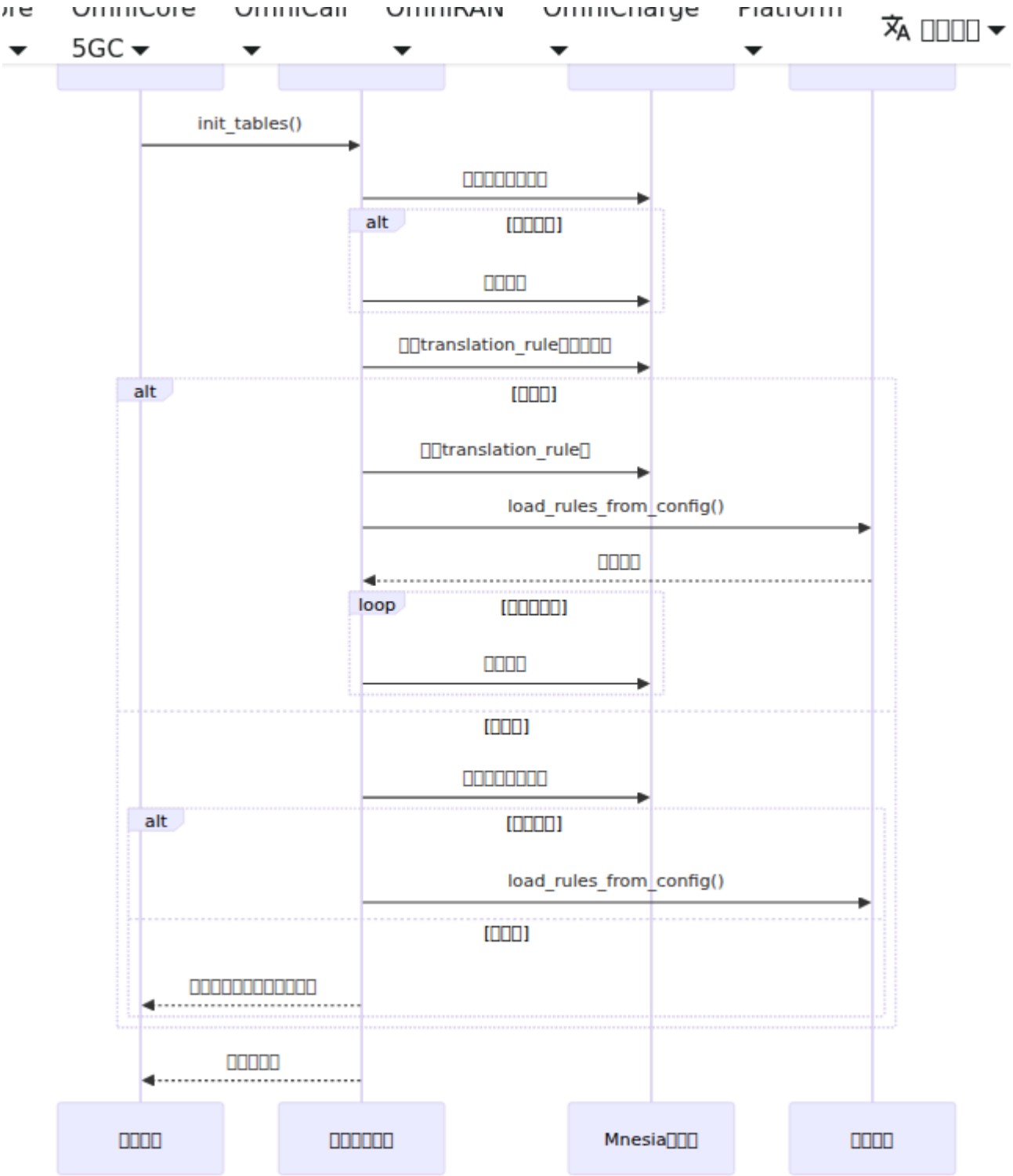
入门

初始化流程

应用程序
号码转换模块
Mnesia数据库
配置文件
init_tables()
检查模式是否存在
[模式缺失]
创建模式
检查translation_rule表是否存在
[表缺失]
创建translation_rule表
load_rules_from_config()
规则定义
[对每个规则]
插入规则
[表存在]
检查规则是否存在
[没有规则]
load_rules_from_config()
[有规则]
跳过配置加载（保留现有）
初始化完成
应用程序
号码转换模块
Mnesia数据库
配置文件

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启用的规则"
do_translate_numbers
(available_rules, matched_rules=[])
[对每个匹配规则]
查找第一个匹配的规则
[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
将规则添加到匹配规则中
[规则有 continue: true]
记录 "继续处理"
从可用规则中移除规则
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, overrides, matched_rules}
应用程序
转换引擎
规则数据库
事件记录器



常见用例

国际号码规范化

将各种国际格式规范化为E.164:

网关特定格式化

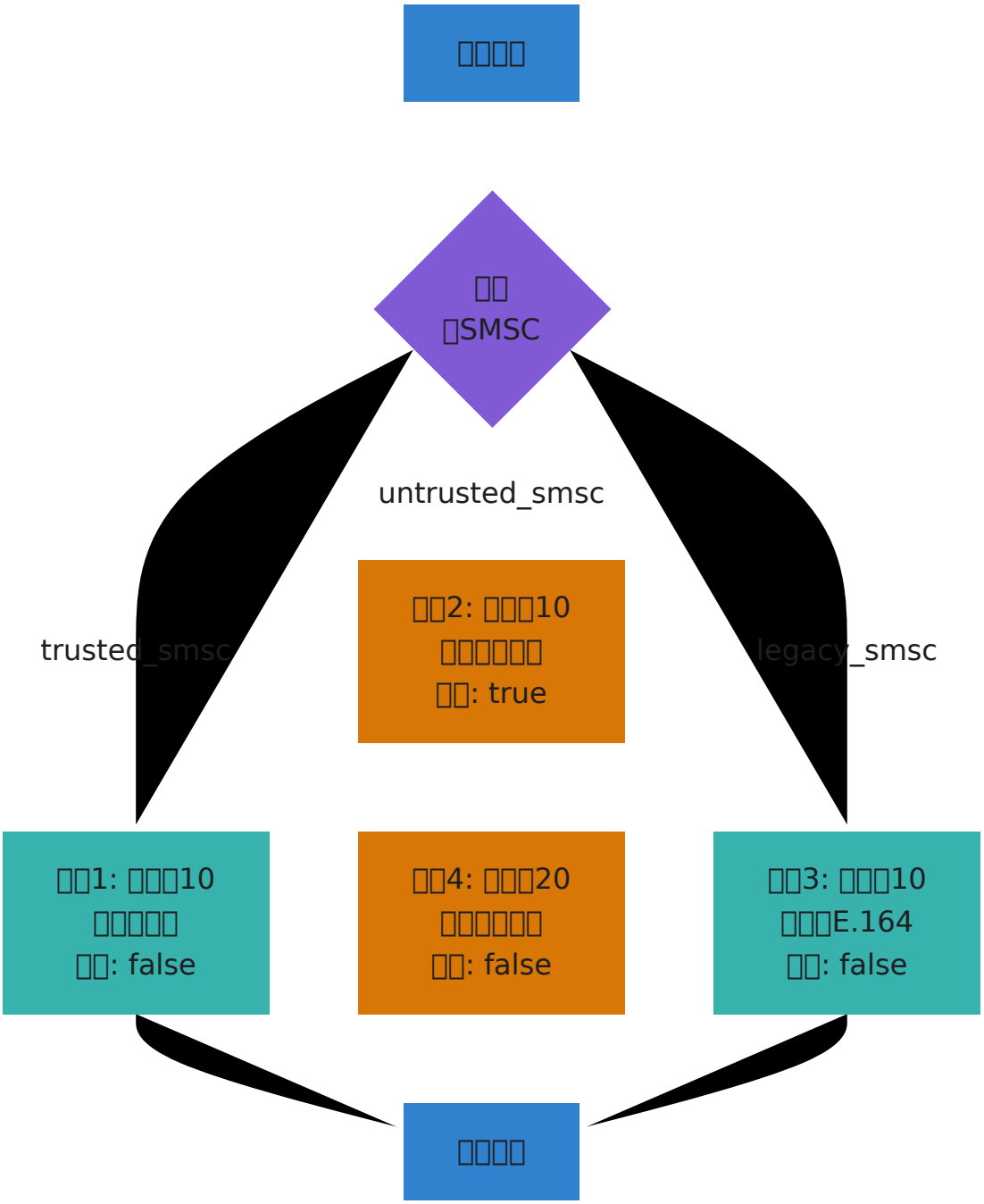
链式处理规则以满足特定网关要求的格式：

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

□试

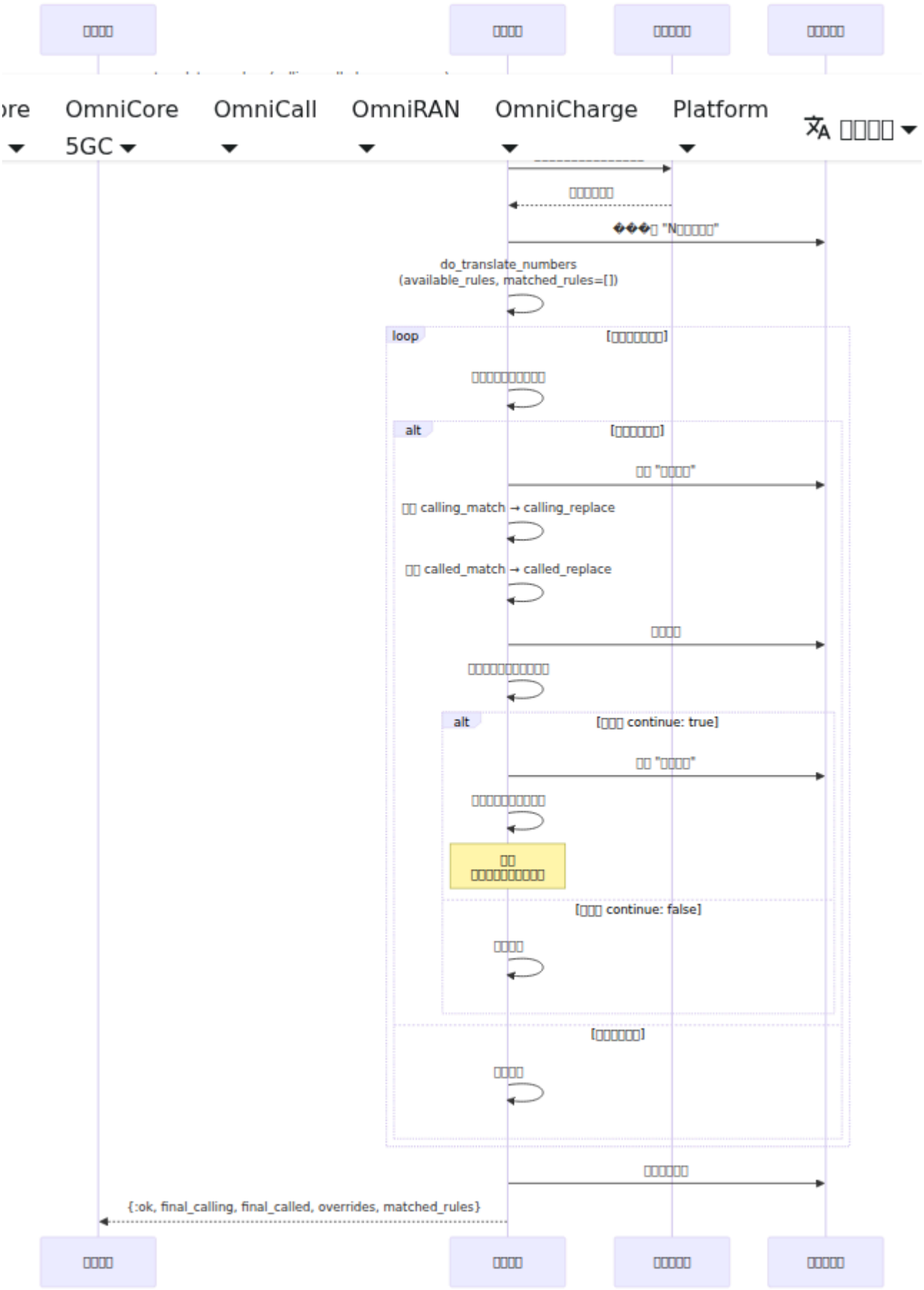
SMSC特定翻译

根据消息源应用不同的翻译：


□□□□
□□
□SMSC
untrusted_smsc
□□2: □□□10
□□□□□□□
□□: true
trusted_smsc
legacy_smsc
□□1: □□□10
□□□□□□
□□: false
□□4: □□□20
□□□□□□□
□□: false
□□3: □□□10
□□□E.164
□□: false
□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□


OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
do_translate_numbers (available_rules, matched_rules=[])
loop
alt
alt
{:ok, final_calling, final_called, overrides, matched_rules}

号码可携带性处理

处理需要前缀更改的移植号码：

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'

试

Web界面

转换规则管理UI

在`/number_translation`（通过导航菜单）访问规则管理界面：

功能：

- 在可排序的表格中查看所有规则，按优先级排序
- 使用表单验证添加新规则
- 编辑现有规则
- 启用/禁用规则而不删除
- 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）：
   - 主叫前缀（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）：
   - 主叫号码正则匹配和替换

    - 被叫号码正则匹配和替换
3. 设置源地址覆盖（可选）：
    - 源**TON / 源NPI**下拉框 - 强制主叫地址的号码类型/编号计划指示符（嵌入TPDU中）。由主叫号码匹配正则表达式门控。
    - 留下一个下拉框未设置以保持值推断。请参见强制号码类型（TON）和编号计划指示符（NPI）。
4. 设置优先级（1-255，数字越小优先级越高）
5. 设置状态：
    - 启用：规则处于活动状态
    - 继续处理：在□规则之后继续评估更多规则
6. 添加描述
7. 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
- ���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

在`/translation_simulator`（通过导航菜单）访问模拟器：

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
加载示例场景进行快速测试
查看测试历史（最近10次测试）

使用模拟器：

1. 输入测试参数：
主叫号码（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
主叫和被叫号码的前后对比
对于继续处理的规则，显示“↓ 继续”指示
转换以绿色突出显示
未更改的值标记为“通过”
4. 使用示例按钮加载预配置示例
5. 查看测试历史以比较不同场景

示例输出：

转换结果

═══

主叫号码: 5551234567 → +1-555-123-4567

被叫号码: 5558720155 → +1-555-872-0155

✓ 由3条□则翻译

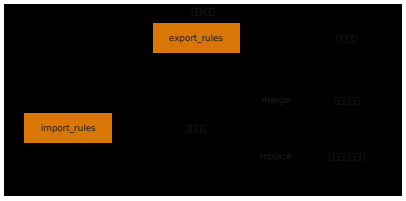
逐步转换

═══

```
┌─ 步骤1 ──────────────────────────────────────┐
│ □则 #1 (优先级10)          ↓ 继续 │
│ 为10位数字添加国家代码            │
│                                              │
│ 被叫: 5558720155 → +15558720155        │
└──────────────────────────────────────────────┘

┌─ 步骤2 ──────────────────────────────────────┐
│ □则 #2 (优先级20)          ↓ 继续 │
│ 用破折号格式化区号                │
│                                              │
│ 被叫: +15558720155 → +1-555-8720155    │
└──────────────────────────────────────────────┘

┌─ 步骤3 ──────────────────────────────────────┐
│ □则 #3 (优先级30)                      │
│ 网关的最终格式化                  │
│                                              │
│ 被叫: +1-555-8720155 → +1-555-872-0155│
└──────────────────────────────────────────────┘
```



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）：主叫电话号码
- `called_number`（可选）：被叫电话号码
- `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, overrides, [rules_applied]}` - 始终成功
- `overrides`是一个映射`%{source_ton:, source_npi:}`（当没有匹配规则设置时，每个值为`nil`；当多个规则设置值时，最后写入优先）
- 如果没有规则匹配，则返回原始号码和空覆盖
- 返回所有应用的规则列表（按顺序）

```
# 示例用法
{:ok, new_calling, new_called, _overrides, rules} =
  NumberTranslation.translate_numbers(
    calling_number: "5551234567",
    called_number: "5558720155",
    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□则")
  Enum.each(rules, fn rule ->
    Logger.info("  - □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: "测试时禁用"
})

# 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. 战略性使用继续：
    - 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
   - 避免回溯重的模式
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个规则
   - 使用Telemetry指标���控转换延迟

���作

1. 部署前测试：
   - 使用模拟器与真实示例

测试边缘情况（空号码、特殊字符）
验证继续标志行为

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：

启用消息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：

首先将新规则添加为禁用
使用模拟器进行测试
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：

- 前缀未匹配（检查是否完全匹配前缀）
- 源SMSC未匹配
- 正则模式未匹配输入格式
- 规则���禁用
- 优先级更高的规则首先匹配（且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. 验证正则模式捕获的组是否正确
3. 检查替换模式语法
4. 在在线正则测试器中测试正则
5. 审查规则优先级和继续标志

无限循环/性能下降

症状：转换耗时很长或似乎挂起

注意：由于循环防止机制，这不应该发生，但如果发生：

可能原因：

循环防止逻辑中的错误
极其复杂的正则评估
非常长的规则链

解决方案：

1. 检查应用程序日志中的错误
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. 如果循环防止失败，请报告错误

意外的规则链式处理

症状：应用的规则数量超过预期

可能原因：

规则在不应继续的情况下具有continue: true
优先级排序允许多个匹配
转换后的号码匹配其他规则

解决方案：

1. 使用模拟器查看确切的规则链
2. 审查所有规则的continue标志
3. 调整优先级以控制顺序
4. 在最终规则上设置continue: false

转换未在路由之前应用

症状：路由器看到未转换的号码

可能原因：

转换未集成在消息流中
转换发生在路由之后
应用程序代码绕过转换

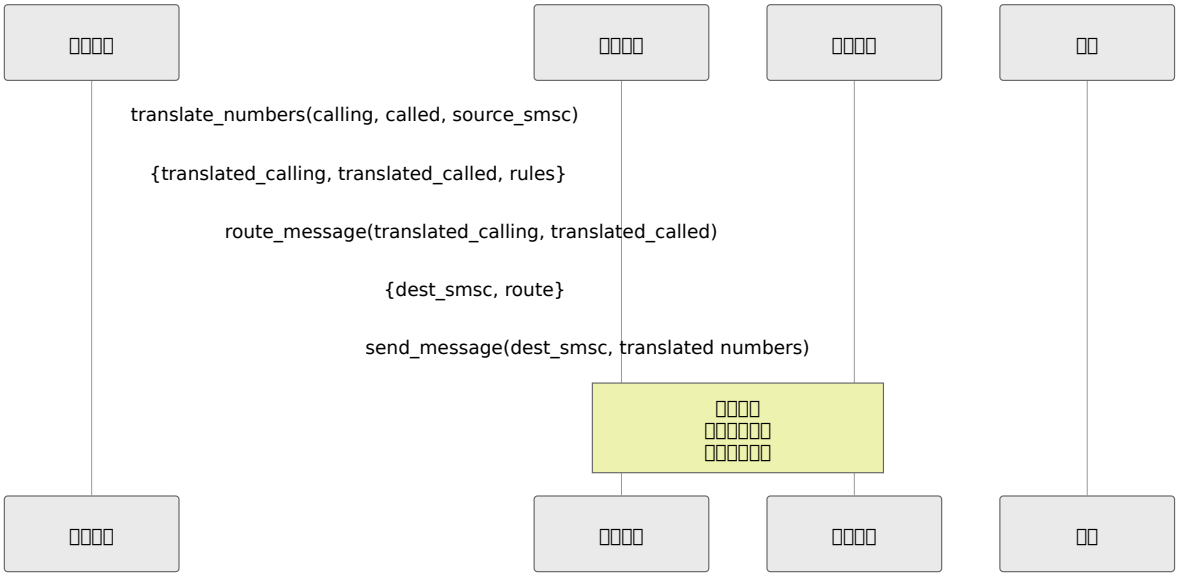
解决方案：

1. 验证应用程序集成：转换应在路由之前调用
2. 检查消息处理管道
3. 查看事件日志以获取转换事件
4. 确保在正确的顺序中调用translate_numbers

高级主题

与路由的集成

转换发生在路由之前，以确保一致的号码格式：



□□□□

□□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
)
```

关键指标监控：

- 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

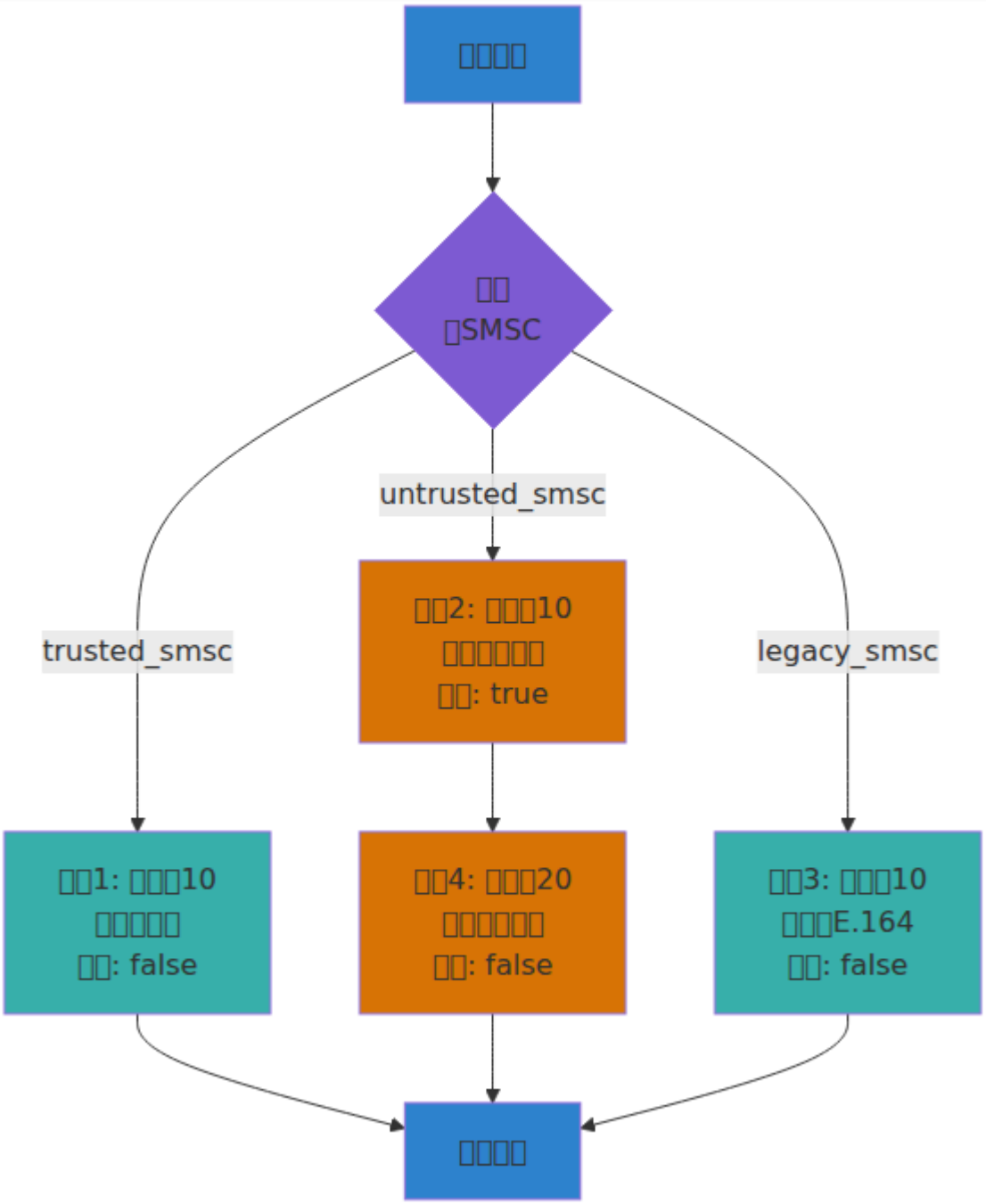
Mnesia表会自动分布在集群节点之间。转换规则会复制以确保高可用性。

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

迁移策略

在部署新转换规则时：



示例

示例**1**：美国号码规范化

需求：将各种美国号码格式转换为E.164 (+1XXXXXXXXXX)

```
# □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: "为裸露的10位数字添加+1",
  enabled: true,
  continue: false
}

# □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"（□则1）
# "15551234567" → "+15551234567"（□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

```
# □则1: 将00转换为+（继续到下一个□则）
%{
  calling_match: "^00(.+)$",
  calling_replace: "+\1",
  called_match: "^00(.+)$",
  called_replace: "+\1",
  priority: 5,
  description: "将00国际前缀转换为+",
  enabled: true,
  continue: true  # 继续格式化
}

# □则2: 为网关格式化（停止处理）
%{
  calling_match: "^\+(\d+)$",
  calling_replace: "00\1",
  called_match: "^\+(\d+)$",
  called_replace: "00\1",
  priority: 10,
  description: "将+号码格式化为00以供网关使用",
  enabled: true,
  continue: false  # 停止
}

# 测试用例:
# 步骤1: "00441234567890" → "+441234567890"（□则1，继续）
# 步骤2: "+441234567890" → "00441234567890"（□则2，停止）
# 结果: "00441234567890"
# 应用的□则: [□则1, □则2]
```

示例3: SMSC特定处理

需求: 根据源SMSC应用不同的规则

```
# □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: "通过受信任的网关的号码",
  enabled: true,
  continue: false
}

# □则2: 不受信任的SMSC - □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
}

# □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 → 用破折号格式化

```
# □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
}

# □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
}

# □则3: 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"（□则1，继续）
# 步骤2: "5551234567" → "+15551234567"（□则2，继续）
# 步骤3: "+15551234567" → "+1-555-123-4567"（□则3，停止）
```

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

支持

如有问题或疑问:

- 使用模拟器处理工作示例并调试转换逻辑
- 查看事件日志以了解转换决策
- 检查Mnesia表内容: `:mnesia.table_info(:translation_rule, :size)`
- 监控Telemetry指标以识别性能问题

SMS-C 操作指南

← 返回文档索引

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
- 监控
- 消息跟踪
- 路由管理
- 前端管理
- 号码翻译管理
- 系统维护
- 备份和恢复
- 容量规划
- 事件响应

日常操作

早晨健康检查

在每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. 审查 Prometheus 指标

访问 Prometheus 仪表板并检查：

消息吞吐量（过去 24 小时）
路由失败率（应 < 1%）
队列积压（应 < 1000 待处理）
投递成功率（应 > 95%）
前端连接状态（所有预期前端均处于活动状态）

3. 检查消息队列

访问 Web UI: https://sms-admin.example.com/message_queue

审查：

总待处理消息（应较少）
最旧消息的年龄（应 < 5 分钟）
高投递尝试的消息（如果 > 3 需调查）
死信消息（调查任何存在的消息）

4. 审查前端状态

访问 Web UI: https://sms-admin.example.com/frontend_status

验证：

所有预期前端均处于活动状态
无未过期的断开连接
过去 24 小时内无前端错误

5. 检查应用日志

访问 Web UI: https://sms-admin.example.com/logs 或检查日志文件

查找：

错误级别消息
路由失败

- 计费失败
- 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：

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

# 每小时投递的消息
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）
- ��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])

消息处理持续时间 (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：系统无法路由的消息
警报：任何失败（> 0）
查询： increase(sms_c_routing_failed_count[5m])

匹配路由 (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

警报：p95 > 2（重试次数过多）
查询： histogram_quantile(0.95, sms_c_delivery_succeeded_attempt_count)

队列指标

队列大小 (sms_c_queue_size_size)：

什么：队列中的总消息
警报：大小 > 10,000
查询： sms_c_queue_size_size

最旧消息年龄 (sms_c_queue_oldest_message_age_seconds)：

什么：最旧待处理消息的年龄
警报：年龄 > 300 秒
查询： sms_c_queue_oldest_message_age_seconds

仪表板设置

操作仪表板面板：

1. 消息吞吐量（图表）

接收的消息（5 分钟率）
投递的消息（5 分钟率）
时间范围：过去 24 小时

2. 队列状态（单一统计）

当前待处理消息
最旧消息年龄
失败消息计数

3. 投递性能（图表）

成功率随时间变化
失败率随时间变化

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（单一统计）

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
# ??不到路由 - 消息无法投递
- alert: RoutingFailures
  expr: increase(sms_c_routing_failed_count[5m]) > 0
  severity: critical
  description: "{{ $value }} 消息在过去 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 }} 条待处理消息"

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

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

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 - 消息已创建
   ↓
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 - 计费检查（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（2分钟延迟）
   ↓
5. delivery_attempt_3 - 第三次尝试失败（4分钟延迟）
   ↓
6. message_dead_letter - 超过重试限制
```

检查投递状态

等待投递：

- 状态：“queued”（现在准备发送）或“backoff”（等待重试）
- deliver_after：对于 `queued` 为 nil/过去，对于 `backoff` 为未来时间戳
- delivery_attempts：对于新鲜的 `queued` 消息为 0，对于 `backoff` 为更高的值

已投递消息：

- 状态：“delivered”（投递确认）或“sent”（已提交，尚无 DLR）
- deliver_time：投递时间戳
- dest_smsc：投递的前端

未投递/失败的消息：

- 状态：“backoff”且 delivery_attempts 较高 - 仍在重试（在 30 分钟间隔内上限，直到消息的 `expires` 被达到）
- 状态：“expired”、“dropped”、“balance_rejected”或“auto_replied” - 终端状态
- 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标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`（未显式路由）
   - `destination_msisdn` 有有效的位置记录
   - 位置的 `location` 字段与前端名称匹配
   - 位置尚未过期

示例场景：

一个 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显式路由时：

```
# 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
}
```

`uk_gateway` 前端将在轮询时自动接收该消息：

```
# 前端轮询消息
GET /api/messages/queue?smsc=uk_gateway

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

位置要求：

要使基于位置的路由正常工作：

- `locations` 表中必须有 `destination_msisdn` 的条目
- `location` ��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

重试失败的消息：

```
# 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：

```
# 每条路由路由的消息（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：目标号码前缀（地理路由所需）
    - 源 **SMSC**：源系统过滤器（可选）
    - 目标 **SMSC**：目标网关（除非自动回复/丢弃，否则必需）
    - 仅限内部网：仅将□路由限制为内部网目标（需要 Diameter/HSS）
    - 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: "英国主路由（70%）"

路由 2:

- 被叫前缀: `+44`
- 目标 SMSC: `uk_backup`
- 优先级: 50
- 权重: 30
- 描���: "英国备份路由（30%）"

示例：仅限内部网路由：

限制外部运营商或内部应用程序仅向内部网订阅者发送。发送到外部网目标的消息将被丢弃。

- 源 SMSC: `carrier_smpp_bind`
- 目标 SMSC: `local_msc`
- 仅限内部网：已勾选
- 优先级: 50
- 权重: 100
- 描述: "运营商 X - 仅限内部网"

这需要启用 Diameter/HSS。如果 HSS 不可用，消息将被丢弃（失败关闭）。

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：系统选择的路由
    - 所有匹配：符合条件的路由
    - 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��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。考虑禁用而不是删除。

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. 导航到 /sms_routing
2. 点击“导入路由”
3. 选择 JSON 文件
4. 选择导入模式：
合并：添加到现有路由
替换：删除所有并导入

用例：

在重大更改前备份
在环境之间复制路由
灾难恢复
配置版本控制

前端管理

监控前端连接

Web UI：导航到 /frontend_status

检查：

所有预期前端均为“活动”

- 最后见到的时间较近（< 90 秒）
- 无意外过期的前端

通过 **API**：

```
# 获取活动前端
curl https://api.example.com:8443/api/frontends/active

# 获取统计信息
curl https://api.example.com:8443/api/frontends/stats
```

调查断开连接

前端过期：

1. 检查前端日志是否有错误
2. 验证与 SMS-C 的网络连接
3. 确认前端正在运行
4. 检��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 日志是否有注册错误
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. 查看过去的注册记录

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。更改需要重新启动应用程序。

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: "从运营商 A 中剥离运营商代码",
  enabled: true
}
```

测试翻译规则

在配置更改后:

1. 重新启动应用程序以加载新规则
2. 提交测试消息，源/目标应匹配
3. 检查事件日志以获取 `number_translated` 事件
4. 验证号码是否正确转换

禁用翻译规则

在规则中设置 `enabled: false`:

```
%{
  ...
  enabled: false
}
```

重新启动应用程序。

系统维护

数据库维护

检查数据库大小:

使用您的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:

根据您的保留策略，CDR 记录应定期归档和清除:

- 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 autovacuum）

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 omnimessage || 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 omnimessage

# 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 omnimessage
```

重启后：

- 验证所有前端重新连接
- 检查 Prometheus 的指标连续性
- 监控日志以查找错误
- 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/omnimessage/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/omnimessage/config $BACKUP_DIR/

# 备份证书
cp -r /opt/omnimessage/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/omnimessage/scripts/backup_database.sh

BACKUP_DIR="/var/backups/sms_c/database"
DATE=$(date +%Y%m%d_%H%M%S)

mkdir -p $BACKUP_DIR

# 备份 SQL CDR 数据库
# MySQL/MariaDB: 使用 mysqldump 和 --single-transaction 以确保一致性
# PostgreSQL: 使用 pg_dump -F c 以自定义格式

# 示例结构（根据您的数据库进行调整）：
# - 使用适当的备份工具（mysqldump，pg_dump）
# - 启用事务安全备份以确保一致性
# - 压缩输出以节省空间
# - 配置保留期（例如，30 天）

# 删除旧备份
find $BACKUP_DIR -name "sms_c_*.gz" -mtime +30 -delete

echo "数据库备份完成: sms_c_${DATE}"
```

路由表备份：

```
#!/bin/bash
# /opt/omnimessage/scripts/backup_routes.sh

BACKUP_DIR="/var/backups/sms_c/routes"
DATE=$(date +%Y%m%d)

mkdir -p $BACKUP_DIR

# 通过 API 导出
curl https://api.example.com:8443/api/routes/export \
  > $BACKUP_DIR/routes_${DATE}.json

echo "路由备份完成: routes_${DATE}.json"
```

安排备份（crontab）：

```
# 每天凌晨 2 点
0 2 * * * /opt/omnimessage/scripts/backup_config.sh
0 2 * * * /opt/omnimessage/scripts/backup_database.sh
0 2 * * * /opt/omnimessage/scripts/backup_routes.sh
```

恢复程序

恢复配置：

```
# 停止应用程序
systemctl stop omnimessage

# 恢复配置文件
cp -r /var/backups/sms_c/20251030/config/*
/opt/omnimessage/config/

# 恢复证书
cp -r /var/backups/sms_c/20251030/cert/*
/opt/omnimessage/priv/cert/

# 启动应用程序
systemctl start omnimessage
```

恢复 **SQL CDR** 数据库：

使用适当的恢复工具进行恢复：

- **MySQL/MariaDB**：解压并通过 mysql 客户端进行管道
- **PostgreSQL**：使用 pg_restore 进行自定义格式的转储

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：

读取副本用于报告查询
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
- 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
- 路由失败 > 10%
- 队列积压增长

中（4 小时内响应）：

- 单个前端断开
- 投递成功�� 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：

最近有配置更改吗？
最近有部署吗？
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，则回滚最近的部署
如有需要，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
更新运行手册

常见事件

队列积压严重：

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. 检查是否缺少路由
4. 验证是否存在捕获所有路由
5. 检查事���日志以获取失败原因

前端断开：

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. 审查计费系统性能

有关详细的故障排除程序，请参见 故障排除指南.

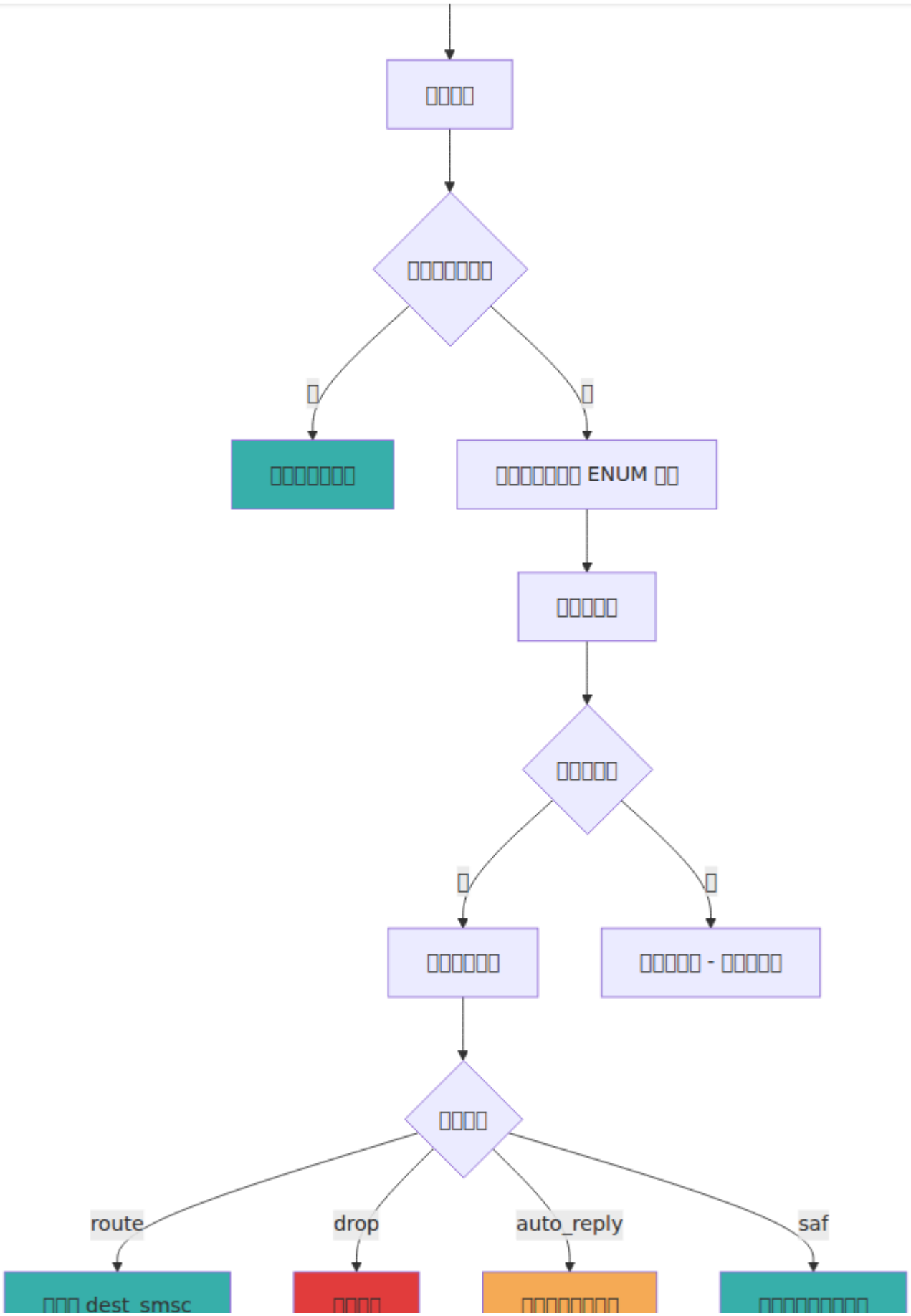
短信路由
← 返回文档索引

概述

SMS-C 路由引擎确定每条消息在进入系统后去向何处。Mnesia 存储路由。您可以通过 API 或控制面板在运行时修改它们。路由支持正则表达式匹配、优先级排序、加权负载均衡、ENUM（DNS NAPTR）结果匹配以及基于 HSS 的在网/离网匹配。

消息处理管道

每条消息遵循以下管道从提交到路由：


route
drop
auto_reply
saf

□□/□□□

□□□□□□

□□	□□	□□
`route_id`	□□	□□□□□□□□□□□
`calling_regex`	□□□□	□□□□□□□□□□□ `nil` = □□□□
`called_regex`	□□□□	□□□□□□□□□□□□ `nil` = □□□□
`source_smsc`	□□□□	□ SMSC □□□ `nil` = □□□□
`dest_smsc`	□□□□	□□ SMSC□□□ `route` □□□□□□□ `saf` □□□□□ `drop`/`auto_reply` □□□
`source_type`	□□□□	□□□□□□□ `ims`□`circuit_switched`□`smpp` □ `nil`
`enum_domain`	□□□□	□□□ ENUM □□□□ `nil` = □□□□□□□□□□□□□□□□□□□□□□□□□ DNS □□□□□□ `enum_domains` □□ - □□□ ENUM □□
`enum_result_domain`	□□□□	□□□□□□□□ ENUM NAPTR □□□□□□□□□□□□□□□□□ `nil` = □□□□

□□	□□	□□
`action`	□□□	□□□□□ `route`□ `drop`□ `auto_reply`□ `saf`
`auto_reply_message`	□□□	□□□□□□□□□□ `auto_reply` □□□□
`charged`	□□□	□□□ `yes`□ `no`□ `default`
`weight`	□□	□□□□□□□1–100□
`priority`	□□	□□□□1–255□□□□□□□□□□□□
`description`	□□□	□□□□□□□
`enabled`	□□□	□□/□□□□□
`originating_network`	□□	□ MSISDN HSS □□□□□□□□□ `:any` | `:on_net` | `:off_net`□□□ `:any`□
`terminating_network`	□□	□□ MSISDN HSS □□□□□□□□□ `:any` | `:on_net` | `:off_net`□□□ `:any`□

路由操作

`action` 字段有四个有效值：`:route`、`:drop`、`:auto_reply`、`:saf`。

`route`（默认）

将消息交付到 `dest_smsc`。这是标准的路由操作。

```
{
  "action": "route",
  "called_regex": "^61",
  "dest_smsc": "australia-gw",
  "priority": 100,
  "description": "将澳大利亚号码路由到 AU 网关"
}
```

`drop`

丢弃消息。无需 `dest_smsc`。

```
{
  "action": "drop",
  "called_regex": "^900",
  "priority": 5,
  "description": "阻止高收费号码"
}
```

`auto_reply`

将 `auto_reply_message` 发送回发件人。原始消息被标记为自动回复。

```
{
  "action": "auto_reply",
  "called_regex": "^911$",
  "auto_reply_message": "紧急：不支持发送短信到 911。请拨打 911。",
  "priority": 1,
  "description": "911 自动回复"
}
```

saf

存储转发。消息被保留并在目标订阅者注册时稍后交付。`dest_smsc` 是可选的（如果设置，则在最终交付被保留的消息时使用）。

`saf` 本身是无条件的 - 它不执行 HSS 查询。要仅对在网订阅者进行存储转发，请将 `saf` 与 `terminating_network::on_net` 匹配标准结合使用（见下文），并提供离网回退路由。

```
{
  "action": "saf",
  "terminating_network": "on_net",
  "priority": 200,
  "description": "对在网订阅者进行存储转发"
}
```

典型设置：在网保留 + 离网回退

P200 路由仅在目标确认在网时□□，因□在网订阅者被保留以进行 SAF。离网号码（和 HSS 错误 - 失败关闭）未能通过该路由并回落到 P255 回退。这确保在网订阅者不会泄漏到离网网关。

ENUM 查找

ENUM 允许路由决策依赖于目标号码的 DNS E.164 NAPTR 查找。这在号码可携带性或运营商间路由中是典型的，其中 DNS（而不是您的 HSS）是号码所属运营商的权威。

工作原理

1. 全局启用。只有在 `enum_enabled: true` 且 `enum_domains` 非空时，ENUM 才会运行。这是对每条消息执行的单次查找，在评估路由表之前进行 - 而不是每条路由。
2. 查询。被叫号码被转换为反向 ENUM 格式（`+15551234` → `4.3.2.1.5.5.5.1.e164.arpa`），并发出 NAPTR 查询。`enum_domains` 中的每个配置域按顺序尝试，停止在第一个返回记录的域。
3. 结果域提取。每个返回的 NAPTR 记录被解析并减少为 `result_domain` - `replacement` 字段的最后两个标签，或在 `sip:user@host` 重写的主机（`sip:+15551234@ims.example.com` → `example.com`）。
4. 匹配。设置了 `enum_result_domain` 的路由仅在任何返回的 NAPTR 记录的 `result_domain` 完全等于该字符串时匹配。`enum_result_domain: nil` 是通配符（始终匹配，即使没有运行 ENUM 查找）。
5. 缓存。结果在 ETS 中缓存 15 分钟，按 `(number, domain)`。失败也会被缓存，以避免在重复未命中时对 DNS 造成压力。

从 ENUM 推导在网/离网

ENUM 不返回布尔的在网/离网状态 - 它返回一个域。您通过将路由的 `enum_result_domain` 指向您自己的 IMS/运营商域来表示“在网”，并为其他所有内容提供回退路由。与 HSS 匹配（见下文）形成对比，它返回明确的在网/离网答案。

ENUM 不做的事情

- 原始 NAPTR 结果（`enum_result`）仅用作路由匹配键。它不用于重写目标 URI 或选择下一个跳点 - 匹配消息的去向完全由 `route.dest_smsc` 决定。实时交付管道在路由后丢弃原始 ENUM 结果。
- 路由上的 `enum_domain` 不用于限制 DNS 查询（查询始终使用全局 `enum_domains` 列表），并且不是匹配标准。它仅对 特异性评分 贡献。

配置

ENUM DNS 行为（`enum_enabled`、`enum_domains`、`enum_dns_servers`、`enum_timeout`）是全局配置的 - 请参见 CONFIGURATION.md → ENUM 配置。

在网 / 离网匹配

Diameter Sh dip 到 **HSS** 回答目标（或发件人）是否在网。路由将其作为两个匹配标准。这些不是布尔标志，也不是路由操作。它们是枚举值标准，路由器在与正则表达式/SMSC/类型检查同时进行评估。路由仅在满足其网络标准时匹配。

`originating_network` / `terminating_network`

每个取三个值之一：

值	含义
`:any`	无 HSS 检查；在网和离网均匹配（默认）
`:on_net`	仅当 MSISDN 确认在网时路由匹配（HSS 返回成功，结果代码 2001）
`:off_net`	仅当 MSISDN 确认离网时路由匹配（HSS 返回 □□□□，结果代码 5001）

- `originating_network` 检查呼叫号码；`terminating_network` 检查被叫号码。
- 用例：限制路由仅交付给在网订阅者（`terminating_network: :on_net`）；限制 SMPP 绑定或外部运营商仅接受在网发件人（`originating_network: :on_net`）；通过外部网关发送仅离网的回退（`terminating_network: :off_net`）。

失败关闭行为

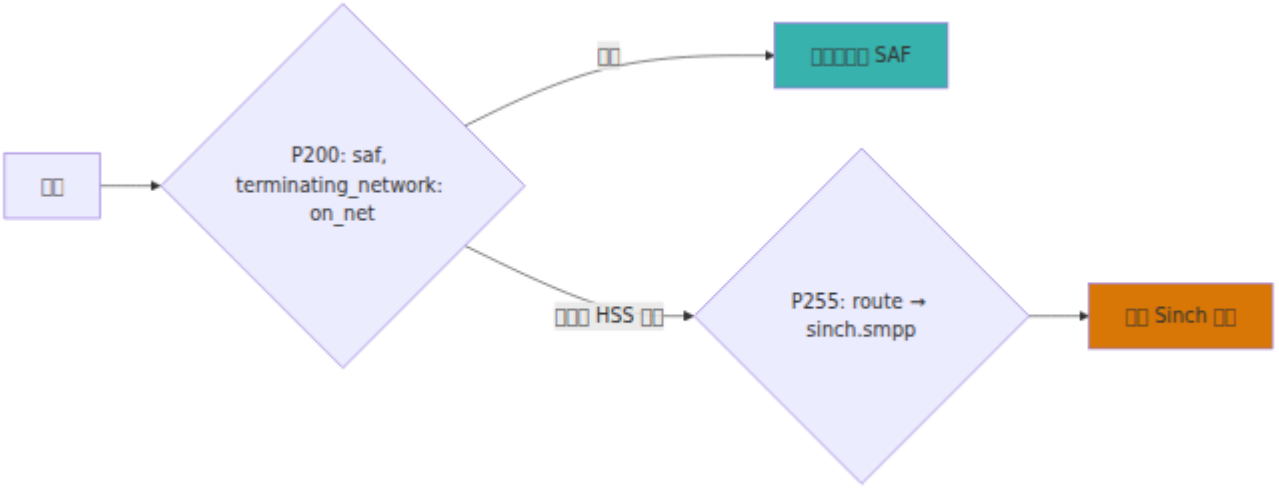
任何值不是 `:any` 都需要明确的 HSS 答复。如果答案不明确，则路由不匹配（引擎尝试下一个路由）：

- Diameter 禁用 → 路由不匹配
- HSS 无法访问 / 超时 / 传输错误 → 路由不匹配
- HSS 返回除 □□□□ 以外的错误 → 路由不匹配

请注意，即使是 `:off_net` 也会失败关闭：它仅在明确的 □□□□（5001）上匹配，而不是在传输失败时。这防止了在 HSS 仅仅关闭时将流量泄漏到离网网关。

HSS 结果缓存

HSS 查找是昂贵的（一个 Diameter 往返）。系统按 MSISDN 缓存结果以避免冗余查询。



- 缓存 **TTL**：2 小时（可通过 `hss_cache_ttl_seconds` 配置）
- 延迟查询：仅当正在评估的路由将 `originating_network` 或 `terminating_network` 设置为 `:on_net` 或 `:off_net` 时，才查询 HSS。将两者都留为 `:any` 的路由从不触发查询。
- 跨路由共享：如果多个路由需要相同 MSISDN 的状态，则仅发生一次 HSS 查询
- 强制重新路由：当消息被显式重新路由时，缓存被绕过（进行新的查询）

路由评估日志

每条为消息评估的路由都记录为该消息的 `route_evaluated` 事件。日志包括：

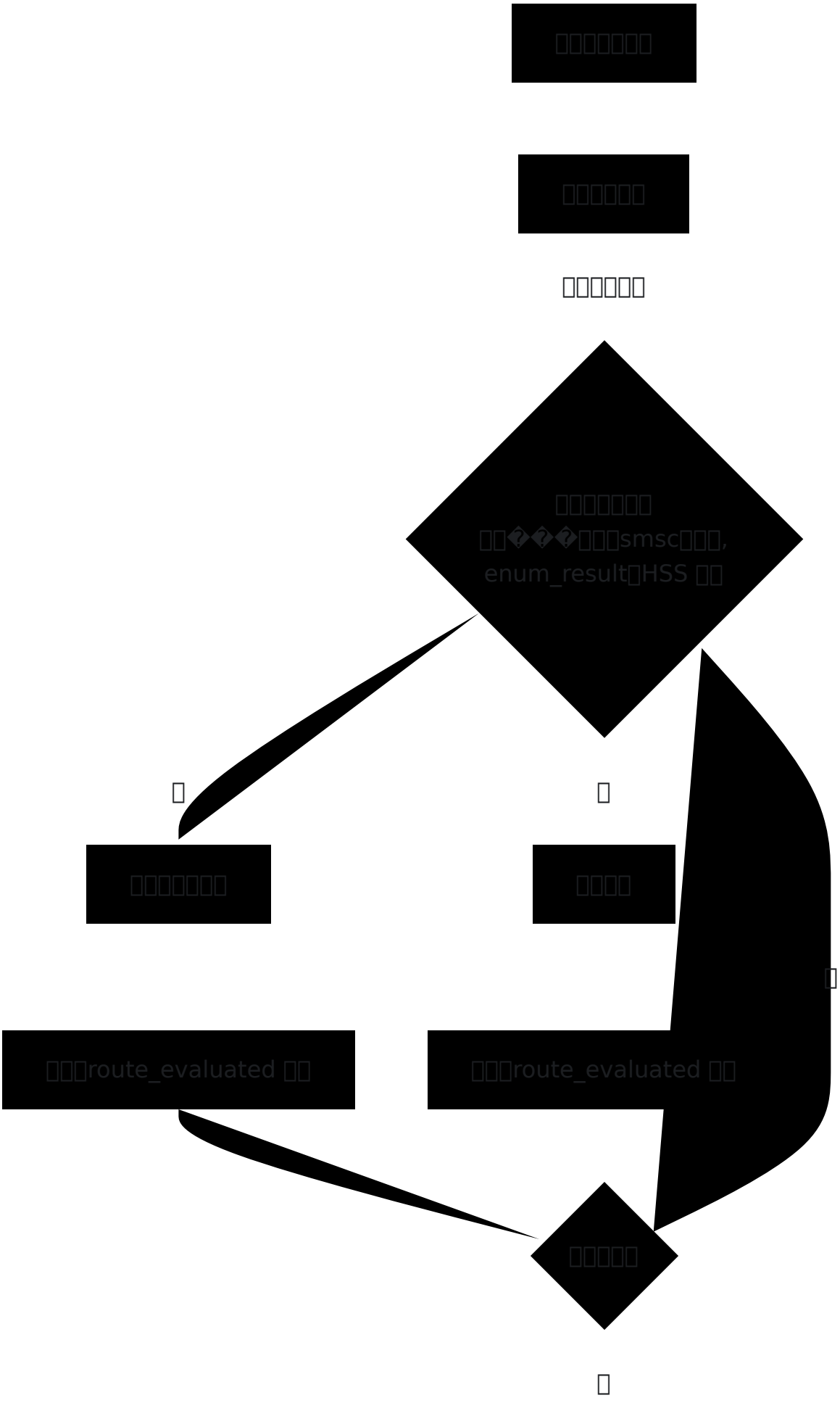
- 路由 ID、操作和优先级
- 每个匹配标准的通过/失败状态（包括 `enum_result` 和 `orig_network`/`term_network` HSS 检查）
- 比较的详细信息（正则表达式模式与号码、SMSC 名称与源、所需网络与实际 HSS 状态等）

示例事件日志条目：

```
路由 #4 [saf] 匹配: P200 "保留在网" |
  calling_regex: ✓ (通配符), called_regex: ✓ (通配符),
  source_smsc: ✓ (通配符), source_type: ✓ (通配符),
  enum_result: ✓ (通配符),
  term_network: ✓ (终止要求在网, +15551234 在网)

路由 #3 [route] 失败: P255 "Sinch 回退" → sinch.smpp |
  calling_regex: ✓ (通配符), called_regex: ✓ (通配符),
  source_smsc: ✓ (通配符), source_type: ✗ (ims vs circuit_switched),
  enum_result: ✓ (通配符), term_network: ✓ (任何)
```

这些事件在控制面板的消息活动日志和通过事件 API 可见。



对于每条路由
是
否
否

HSS □□/□□□

标准	分数
`called_regex` 模式长度	长度 × 100
`calling_regex` 模式长度	长度 × 50
`source_smsc` 指定	+25
`enum_result_domain` 指定	+15
`source_type` 指定	+10
`enum_domain` 指定	+5

更长、更具体的模式总是胜过通配符。

配置

从配置加载路由

在 `config/runtime.exs` 中定义的路由在路由表为空时首次启动时加载。后续重启保留通过 API 添加的路由。

```
config :sms_c, :sms_routes, [
  %{
    called_regex: ~r/^15555/,
    dest_smsc: "example-gw",
    action: :route,
    priority: 100,
    description: "示例国家路由"
  },
  %{
    action: :saf,
    terminating_network: :on_net,
    priority: 200,
    description: "对在网订阅者进行存储转发"
  },
  %{
    action: :auto_reply,
    called_regex: ~r/^911$/,
    auto_reply_message: "紧急：请拨打 911。",
    priority: 1,
    description: "911 自动回复"
  }
]
```

□□□□□□

□□	□□	□□	□□	□□
`action`	□□/□□□	□	`:route`	□□□ □□`:route`□`:drop`□`:auto_reply`
`calling_regex`	□□□□□	□	`nil`	□□□□□□□□□□`nil` □□□□
`called_regex`	□□□□□	□	`nil`	□□□□□□□□□□□`nil` □□□□
`source_smsc`	□□□	□	`nil`	□□ SMSC □□□□□`nil` □□□□
`dest_smsc`	□□□	□□□	`nil`	□□ `route` □□□□□□□ `saf` □□□□□ `drop`/`auto_reply` □□□
`source_type`	□□	□	`nil`	□□□□ □□`:ims`□`:circuit_switched`□
`enum_domain`	□□□	□	`nil`	□□□ ENUM □□□□□□□□□□□□□

□□	□□	□□	□□	□□
`enum_result_domain`	□□□□	□	`nil`	□□ ENUM NAPTR □□□□□□□□□□□□□r□□
`auto_reply_message`	□□□□	□□□	`nil`	□□□□ `:auto_reply` □□□
`charged`	□□	□	`:default`	□□□□□ `:yes` □ `:no` □ `:default`
`weight`	□□	□	`100`	□□□□□□□1–100□
`priority`	□□	□	`100`	□□□□□□1–255□□□□□□□□□□□□
`description`	□□□□	□	`""`	□□□□□□□
`enabled`	□□□□	□	`true`	□□□□□□□□□□
`originating_network`	□□	□	`:any`	□ MSISDN HSS □□□ □□ `:any` □ `:on_net` □ `:off_net`
`terminating_network`	□□	□	`:any`	□□ MSISDN HSS □□□ □□ `:any` □ `:on_net` □ `:off_net`

HSS 缓存配置

参数	类型	默认	描述
`hss_cache_ttl_seconds`	整数	`7200`	缓存 HSS 查找结果的生存时间（秒）。在 `runtime.exs` 中设置

API

创建路由

```
POST /api/routes
```

```
{
  "action": "saf",
  "terminating_network": "on_net",
  "priority": 200,
  "description": "对在网订阅者进行存储转发"
}
```

列��路由

```
GET /api/routes
```

更新路由

```
PUT /api/routes/:id
```

删除路由

```
DELETE /api/routes/:id
```

导入/导出

```
POST /api/routes/import
GET /api/routes/export
```

导入支持 `merge`（添加到现有）和 `replace`（清除并替换所有）模式。

故障排除

未找到路由

症状：消息保持待处理状态，`dest_smsc: null`，事件日志显示 `sms_routing_failed`

可能原因：

- 没有启用的路由匹配消息标准
- 路由正则表达式模式与号码格式不匹配（例如，缺少国家代码）
- 每个候选路由都有一个 HSS 未满足的 `terminating_network`/`originating_network` 要求（离网，或在失败关闭下的 HSS 错误），并且不存在 `:any` 回退路由

解决方案：

1. 检查消息的事件日志 - 每条评估的路由都记录了每个标准的通过/失败（包括 HSS 网络检查）
2. 使用控制面板中的路由模拟器测试相同的号码
3. 确保存在一个通用回退路由（`originating_network: :any`，`terminating_network: :any`），优先级设置为高（例如 255）

在网订阅者被路由到离网

症状：发送给在网订阅者的消息通过外部网关交付，而不是被保留/本地交付

可能原因：

- 没有配置 `terminating_network: :on_net` 的路由来捕获在网目标
- 在网路由的优先级□□（较低优先级）高于离网回退路由
- HSS 查找失败（检查 `hss_dip_error` / Sh 监控） - 在失败关闭下，在网路由不匹配，流量回落到回退
- 配置中未启用 Diameter

解决方案：

1. 添加一个优先级数字低于离网回退的 `terminating_network: :on_net` 路由
2. 通过控制面板验证 Diameter 是否启用，HSS 对等体是否已连接
3. 检查 HSS 缓存状态和事件日志中的最近查询结果

HSS 缓存过时

症状：订阅者已迁移或配置，但仍基于旧的在网/离网状态进行路由

解决方案：

1. 通过 API 重新路由消息 - 强制重新路由绕过 HSS 缓存
2. 等待缓存 TTL 过期（默认 2 小时）
3. 如果需要更快的更新，减少 `hss_cache_ttl_seconds`

SMS-C 故障排除指南
← 返回文档索引

全面的指南，用于诊断和解决常见的 SMS-C 问题。

目录

诊断工具
消息投递问题
路由问题
性能问题
数据库问题
前端连接问题
计费/收费问题
ENUM 查找问题
集群问题
API 问题
Web UI 问题
系统资源问题

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 omnimessage

# 5. 检查 SQL CDR 数据库连接 (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

查看最近的错误:

```
# 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:

```
# 每秒消息数
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

# 最旧消息的年龄（秒）
sms_c_queue_oldest_message_age_seconds
```

检查系统性能:

```
# 消息处理延迟 (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)
```

消息投递问题

消息未被投递

症状:

- 消息卡在 "queued"/"backoff" 状态
- 高未投递消息计数
- 没有投递通知

诊断步骤:

1. 检查前端连接:

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

预期: 活动前端列表

问题: 空列表或缺少预期的前端

2. 检查消息队列:

访问 Web UI: `/message_queue`

- 按状态过滤: "queued" 或 "backoff"
- 检查 `dest_smsc` 值
- 验证 `deliver_after` 不在未来

3. 检查路由:

访问 Web UI: `/simulator`

- 使用实际消息参数进行测试
- 验证路由匹配且目标正确

4. 检查前端轮询:

查看前端系统日志:

- 前端是否在查询 `/api/messages`?

- 前端是否正确发送 `smsc` 头?

解决方案:

没有连接的前端:

```
# 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**:

- 审查路由配置
- 检查路由优先级
- 在路由模拟器中测试
- 验证前端名称与消���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` 计数增加
- 尝试计数高的消息（> 3）
- 指数回退延迟

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：

- 前端为何未能投递？
- 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：

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

# 重置重试计数
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"}'
```

无效的目标号码：

- 验证号码格式
- 检查号码转换规则
- 如果确实无效，则删除消息

死信消息

症状：

- 消息中有 `deadletter: true`
- 消息超过过期时间
- 状态仍为 "queued"/"backoff"（失败的投递不会被标记为死信；一旦超过 `expires`，它们将停止服务）

诊断步骤：

1. 查找死信消息：

访问 Web UI: `/message_queue`

- 按过期状态过滤
- 检查过期时间戳

2. 检查为何过期：

- 审查事件日志
- 检查投递尝试历史
- 验证路由是否成功

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

未找到路由

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`

- 验证路由已配置
- 检查至少一个路由是否已启用

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）
- 审查评估结果
- 检查为何路由未匹配

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?
- 所有路由是否已禁用?

解决方案:

未配置路由:

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
```

所有路由均已禁用:

- 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
审查 "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：

数字越小 = 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 （高优先级）

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
```

自动回复未工作

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：

- `action: :auto_reply`
- `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：

- 验证路由是否被选中
- 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：

路由未匹配：

- 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： ✓
自动回复消息： "感谢您的消息。我们会尽快回复。"
```

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（路由数量多）:

- 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���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

# 连接池队列时间
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,  # 更大的批处理
  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

- 处理速度无法跟上输入速率

诊断步骤：

1. 检查输入与投递速率：

```
# 输入速率
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
- 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

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
- 如果频繁出现，考虑容量升级

数据库问题

连接失败

症状：

- 错误："unable to connect to database"
- 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 (MySQL/MariaDB)
mysql -u sms_user -p -h db.example.com -e "SELECT 1"

# 测试连接 (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 (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:

数据库宕机:

```
# 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 omnimessage
```

网络问题：

- 检查防火墙规则
- 验证安全组（云）
- 检查 VPN/网络连接

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
- 连接池队列积压

诊断步骤：

1. 检查慢查询日志：

```sql
-- 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:

```sql
-- 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:

```sql
-- 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：

缺失的索引：

```
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：

```
-- 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："Disk full"
- 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 {} + | sort -rh | head -20

# 查找最大文件 (PostgreSQL)
find /var/lib/postgresql -type f -exec du -h {} + | sort -rh |
head -20

# 检查日志文件
du -sh /var/log/sms_c/*
```

解决方��:

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:

```
# 强制日志轮换
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
- 将数据移动到更大的磁盘

前端连接问题

前端未显示为活动

症状:

- 前端状态显示 "expired"
- 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:

- 前端是否在调用 `/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
```

解决方案：

前端未注册：

测试手动注册：

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

如果成功，问题出在前端代码/配置中。

注册超时：

前端在 90 秒后过期。确保每 60 秒注册一次：

```
# 前端应每 60 秒调用注册
while True:
    register_with_smsc()
    time.sleep(60)
```

网络问题：

- 检查前端与 API 之间的防火墙
- 验证 DNS 解析
- 从前端服务器测试 curl

前端重复连接/断开

症状：

- 前端状态在活动/过期之间切换
- 历史注册计数高
- 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：

- 是否过于频繁？（每几秒）
- 是否过于不频繁？（> 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：

- 检查是否有间歇性连接
- 审查防火墙日志
- 检查负载均衡器健康检查

错误的注册间隔：

正确的间隔：

```
REGISTRATION_INTERVAL = 60  # 秒
```

充电通过 **Diameter Ro**（信用控制）进行。请注意充电 失败开放：OCS 错误或未配置的 Diameter 堆栈会记录失败，但仍会发送消息。只有明确的超出余额的答案会拒绝消息（状态 `:balance_rejected`）。

症状：

- `sms_c_charging_failed_count` 增加
- 事件日志显示 "charging_failed" 或 "balance_rejected"
- 消息标记为状态 `:balance_rejected`（超出余额），或日志警告关于 Diameter 堆栈未启用

诊断步骤：

1. 检查 **Diameter Ro** 对等体状态（CER/CEA 已建立到 DRA/OCS）：

```
# 在远程控制台中
:diameter.service_info(:omnimessage, :connections)
```

2. 确认堆栈已启用且 `:ro` 应用已注册：

```
grep -E "diameter_enabled|application_name: :ro"
config/runtime.exs
```

日志行 `Charging requested but the Diameter Ro stack is not enabled/configured` 意味着充电被跳过（允许消息） - 启用堆栈。

3. 检查评级配置文件是否与 OCS 上配置的评级上下文匹配：

```
grep rating_profile config/runtime.exs
```

解决方案：

OCS / DRA 无法访问：

- 验证 IP 可达性，并确保 DRA 已将□节点配置为 Diameter 对等体（Origin-Host/Realm）。
- 确认 CCR `Service-Context-Id`（`rating_profile`）与 OCS 的 SMS 评级配置文件匹配。

超出余额处理（`config :sms_c, :charging`）：

```
config :sms_c, :charging,
  out_of_balance_action: :reject,    # 或 :auto_reply
  out_of_balance_reply_text: "...",
  rating_profile: "sms@sms-c"
```

暂时禁用充电：

```
config :sms_c,
  default_charging_enabled: false
```

重启应用程序。

账户问题：

- 检查账户是否存在于 OCS
- 验证账户是否有余额
- 检查评级计划是否已配置

充电过慢

症状：

- `sms_c_charging_succeeded_duration` p95 > 500ms
- 启用充电时消息处理缓慢
- 禁用充电时快速

诊断步骤：

1. 检查充电延迟：

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

2. 检查 **OCS / DRA** 性能（Diameter Ro 往返）。高 `sms_c_charging_succeeded_duration` 通常指向 OCS 评级延迟或一个慢/不稳定的 DRA 对等体。

3. 检查到 **DRA** 的网络延迟：

```
# Ping DRA / OCS 主机
ping -c 10 dra01.example.com
```

解决方案：

OCS 缓慢��

- 优化 OCS 配置
- 增加 OCS 资源
- 使用更快的评级引擎

网络延迟：

- 将 OCS 部署得更靠近 SMS-C
- 使用直接网络路径
- 如果可能，避免 VPN/隧道

超时设置过低：

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
- 路由的 `enum_result_domain` 不匹配

诊断步骤:

1. 检查 **ENUM** 配置:

```
grep -A 10 "enum_" config/runtime.exs
```

2. 测试 **DNS** 连接:

```
# 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 服务器无法访问:

使用备用 DNS:

```
config :sms_c,
  enum_dns_servers: [
    &amp;#123;"8.8.8.8", 53&amp;#125;,  # Google 公共 DNS
    &amp;#123;"1.1.1.1", 53&amp;#125;   # Cloudflare DNS
  ]
```

ENUM 域错误：

更新域：

```
config :sms_c,
  enum_domains: ["e164.arpa"]  # 使用标准域
```

超时设置过短：

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：

```
# 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）：

- 针对唯一数字的流量（正常）
- 监控但如果 < 70% 则不报警

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果

- 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"
  ]
```

集群分裂脑

症状：

- 节点运行但断开连接
- 不同节点上的数据不同
- Mnesia 不一致

诊断步骤：

1. 检查节点连接性：

```
# 在每个节点上（IEx）
Node.list()
```

2. 检查 **Mnesia**：

```
:mnesia.system_info(:running_db_nodes)
```

解决方案：

重新连接节点：

```
# 停止所有节点
systemctl stop omnimessage

# 首先启动一个节点
systemctl start omnimessage  # 在 node1 上

# 等待其完全启动，然后启动其他节点
systemctl start omnimessage  # 在 node2 上
systemctl start omnimessage  # 在 node3 上
```

Mnesia 不一致：

- 从正确的节点导出路由
- 停止所有节点
- 删除 Mnesia 目录
- 启动节点
- 导入路由

API 问题

API 无法响应

症状：

- 连接超时

- 连接被拒绝
- 无响应

诊断步骤：

1. 检查 **API** 进程：

```
# 应用程序是否正在运行?
systemctl status omnimessage

# 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 omnimessage
```

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���误
- 500 状态代码
- 日志中出现错误

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 omnimessage
```

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：

端口错误：

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 omnimessage
```

防火墙：

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

LiveView 不更新

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，请确保支持 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 被阻塞：

- 配置代理以支持 WebSocket
- 检查防火墙
- 检查浏览器扩展

刷新页面：

- 强制刷新（Ctrl+F5）
- 清除浏览器缓存

系统资源问题

CPU 使用率高

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：

```
# 消息处理速率
rate(sms_c_message_received_count[5m])

# 路由操作
rate(sms_c_routing_route_matched_count[5m])
```

解决方案：

流量高：

- 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：

- ��查缓存命中率
- 如果不需要，考虑禁用

内存使用率高

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（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（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`