

REST API 南

← 回主文档

本指南提供了 OmniSS7 **REST API** 和 **Swagger UI** 的全面文档。

目录

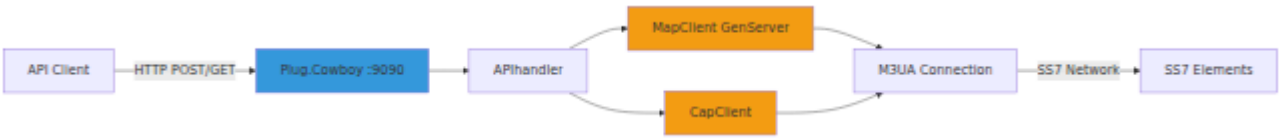
1. 概述
2. HTTP 服务器配置
3. Swagger UI
4. API 端点
5. 示例
6. 认证
7. 响应格式
8. 错误处理
9. 指标 (Prometheus)
10. 示例请求

概述

OmniSS7 在端口 **9090** 上公开了一个 REST API，用于通过 SS7 网络驱动 MAP（移动应用部分）和 CAP（CAMEL 应用部分）操作，以及用于测试的原始 SCCP/M3UA 注入。该 API 定义了 **47** 个 `POST /api/*` 操作端点，以及指标和 Swagger 端点。它允许您：

- 触发 MAP 请求（移动性、短信、认证、用户数据、补充服务、位置服务、GPRS、设备）
- 驱动 CAP/CAMEL 呼叫控制对话
- 注入原始 SCCP/M3UA 有效负载
- 通过 Prometheus 监控系统指标

有关每个操作的请求/响应模式和 MAP 操作码，本指南交叉引用了 **MAP** 客户端南，该指南详细记录了每个操作。



端口和传输

OmniSS7 运行三个不同的 **HTTP** 服务器。本指南涵盖第一个（:9090）。其他服务器在 配置参考 中记录。

端口	传输	服务器	本南的范围
9090	HTTP (`Plug.Cowboy`)	`APIhandler`	是。 MAP/CAP REST `/api/*`、`/metrics`、`/`
8087	HTTPS (`Bandit`, Phoenix)	`ControlPanelWeb.Endpoint`	否: 网页控制面板 UI。请参见 We
8445	HTTPS / TLS	`:api_ex` REST 控制器	否: 状态/路由/链接 REST。请参[

API 架构

HTTP 服务器配置

服务器详情

`/api/*`、`/metrics` 和 `/swagger` 端点由 `Plug.Cowboy` HTTP 监听器提供服务。

参数	值	可配置
协议	HTTP	否
IP 地址	0.0.0.0（所有接口）	仅通过代码
端口	9090	是：`config :omniss7, api_port: <port>`（默认 `9090`）
传输	`Plug.Cowboy`	否

访问 **URL**：`http://[server-ip]:9090`

> 控制面板（:8087，Bandit）和 `:api_ex` REST 控制器（:8445）是独立的服务器。请参见 端口和传输。

启用/禁用 HTTP 服务器

控制 :9090 HTTP 服务器的启动：

```
config :omniss7,
  start_http_server: true  # 置为 false 以禁用
```

默认：`true`（启用）

禁用时：:9090 HTTP 服务器将不会启动，REST API / Swagger UI / 指标端点将不可用。控制面板（:8087）和 `:api_ex`（:8445）服务器不受影响（它们由 `start_web_interfaces` 控制）。

Swagger UI

该 API 包含一个 **Swagger UI** 用于交互式 API 文档和测试。

访问 Swagger UI

URL：`http://[server-ip]:9090/swagger`

功能：

- 交互式 API 文档
- 测试端点的尝试功能
- 请求/响应模式
- 示例有效负载

Swagger JSON

OpenAPI 规范可在以下位置获取：

URL: `http://[server-ip]:9090/swagger.json`

用例：

- 导入到 Postman 或其他 API 客户端
- 生成客户端库
- API 文档自动化

API 端点

每个操作端点遵循模式 `POST /api/{operation}` 并接受 JSON 请求体。请求是 同步 的：处理程序会阻塞，直到 SS7 响应到达或操作超时（默认 **10** 秒 用于等待的 MAP 操作；请参见 错误处理）。

下面的表格列出了所有 **47** 个操作端点，按类别分组。有关每个操作的请求字段、响应模式和 MAP 操作码，请参见相应的 **MAP** 客户端指南，该指南与本指南一起更新，并详细记录了每个操作。

> 响应代码（所有端点）： `200` = 成功，`400` = 错误/无效请求（缺少或类型错误的字段），`504` = 超时（在操作的超时内没有 SS7 响应），`500` = 操作错误（非超时），`503` = 功能未启用（USSD），`404` = 未知路径。请参见 错误处理。

移动管理

操作	端点	描述
更新位置	`POST /api/updateLocation`	将用户的服务 VLR 注册到 HLR。在 HLR 模式下，这会触发 InsertSubscriberData (ISD) 序列。
取消位置	`POST /api/cancelLocation`	指示 VLR 删除用户记录。
���供漫游号码	`POST /api/prn`	从服务 MSC 获取 MSRN。
清除 MS	`POST /api/purgeMS`	将用户标记为在 HLR 中已清除。
发送识别	`POST /api/sendIdentification`	从之前的 VLR 检索 IMSI/认证数据（TMSI 以十六进制提供）。
重置	`POST /api/reset`	HLR 发起的重置到 VLR（未确认: □回 `{"status":"sent"}`）。
恢复数据	`POST /api/restoreData`	VLR 请求从 HLR 恢复用户数据。



短信

操作	端点	描述
发送 SM 路由信息	`POST /api/sri-for-sm`	查询 HLR 以获取 SMS 交付的服务节点。
MT-转发 SM	`POST /api/MT-forwardSM`	将移动终止的 SM 交付到服务 MSC/SGSN（`smsPDU` 以十六进制表示）。
MO-转发 SM	`POST /api/forwardSM`	提交移动发起的 SM（`smsPDU` 以十六进制表示）。
发送 SM	`POST /api/sendSM`	从 GSM-7 文本构建并交付 SMS-DELIVER。
构建交付 PDU	`POST /api/deliverPDU`	构建 SMS-DELIVER PDU 并□回十六进制（不发送 SS7）。
报告 SM 交付状态	`POST /api/reportSM-DeliveryStatus`	向 HLR 报告 SM 交付结果。
准备发送 SM	`POST /api/readyForSM`	通知 HLR 用户可接收 SMS。
警告服务中心	`POST /api/alertServiceCentre`	警告 SMSC 用��可用。

认证

操作	端点	描述
发送认证信息	`POST /api/send-auth-info`	从 HLR 检索认证向量。
发送 IMSI	`POST /api/sendIMSI`	将 MSISDN 解析为 HLR 中的 IMSI。

用户数据

操作	端点	描述
发送路由信息	`POST /api/sri`	查询 HLR 以获取语音呼叫路由信息。
提供用户信息	`POST /api/provideSubscriberInfo`	从 VLR 请求用户状态/位置。
删除用户数据	`POST /api/deleteSubscriberData`	在 VLR 中删除用户数据（可选的 `withdraw` 标志）。
任意时间查询	`POST /api/anyTimeInterrogation`	gsmSCF 查询 HLR 中的用户信息/位置。
任意时间订阅查询	`POST /api/anyTimeSubscriptionInterrogation`	查询 HLR 中的订阅数据。
任意时间修改	`POST /api/anyTimeModification`	在 HLR 中修改订阅数据。
注意用户数据已修改	`POST /api/noteSubscriberDataModified`	HLR 通知 gsmSCF 数据已更改。
注意 MM 事件	`POST /api/noteMM-Event`	向 gsmSCF 报告移动管理事件。

补充服务

操作	端点	描述
查询 SS	`POST /api/interrogateSS`	查询补充服务的状态。
注册 SS	`POST /api/registerSS`	注册补充服务（例如呼叫转移 / CFU）。
删除 SS	`POST /api/eraseSS`	删除补充服务。
激活 SS	`POST /api/activateSS`	激活补充服务。
停用 SS	`POST /api/deactivateSS`	停用补充服务。
注册密码	`POST /api/registerPassword`	注册 SS 密码（`ss_code` 整数）。
获取密码	`POST /api/getPassword`	检索/验证 SS 密码。
转发检查 SS 指示	`POST /api/forwardCheckSS-Indication`	向 VLR 进行未确认检查（回 `{"status":"sent"}`）。

五个核心 SS 操作（`interrogateSS`、`registerSS`、`eraseSS`、`activateSS`、`deactivateSS`）共享请求形状：`ss_code`（整数）和 `hlr_gt`，可选的 `forwarded_to`、`no_reply_time` 和 `basic_service`。

位置服务（LCS）

操作	端点	描述
发送 LCS 路由信息	`POST /api/sendRoutingInfoForLCS`	GMLC 查询 HLR 以获取 LCS 路由。
提供用户位置	`POST /api/provideSubscriberLocation`	从服务节点请求用户的位置。
用户位置报告	`POST /api/subscriberLocationReport`	向 GMLC 报告用户的位置。

GPRS

操作	端点	描述
更新 GPRS 位置	`POST /api/updateGprsLocation`	将用户的服务 SGSN 注册到 HLR。
发送 GPRS 路由信息	`POST /api/sendRoutingInfoForGprs`	查询 HLR 以获取 GPRS 路由（GGSN 选择）。

设备

操作	端点	描述
检查 IMEI	`POST /api/checkIMEI`	查询 EIR 中的设备状态。

USSD

操作	端点	描述
USSD 发送（网络发起）	`POST /api/ussd/send`	向用户推送 USSD 消息。需要 `ussd_gateway_enabled: true`（否则回 `503`）。请参见 USSD 网关指南。

CAP / CAMEL

操作	端点	描述
CAP InitialDP	`POST /api/cap/initialDP`	向 gsmSCF 触发 CAMEL 对话。回一个 `otid` 以供后续操作。
CAP 连接	`POST /api/cap/connect`	将呼叫路由到目的地（`destination_number` + `otid`）。
CAP 继续	`POST /api/cap/continue`	继续呼叫处理（`otid`）。
CAP 释放呼叫	`POST /api/cap/releaseCall`	释放呼叫并附带 `cause`（`otid`）。
CAP 应用计费	`POST /api/cap/applyCharging`	应用计费窗口（`duration` + `otid`）。

CAP 后续操作携带对话 `otid`（十六进制，由 `initialDP` 回）并响应 `{"status":"sent","otid":"<hex>"}`。请参见 CAMEL 网关指南。

原始注入（测试/诊断）

操作	端点	描述
原始 SCCP	`POST /api/raw/sccp`	注入手工构建的 SCCP 有效负载（十六进制）。可选的 SCCP 覆盖（被叫/主叫方、SSNs、OPC/DPC）。
原始 M3UA	`POST /api/raw/m3ua`	注入手工构建的 M3UA 有效负载（十六进制）。可选的 OPC/DPC/路由上下文覆盖。

实用端点

端点	方法	目的
`/metrics`	GET	Prometheus 指标（文本格式）。
`/swagger`	GET	Swagger UI。
`/swagger.json`	GET	OpenAPI 规范。

示例

以下示例说明了代表性操作的请求/响应形状。有关每个操作的完整字段参考，请参见 MAP 客户端指南。

SendRoutingInfo (SRI)

检索建立与移动用户呼叫的路由信息。

端点：`POST /api/sri`

请求体：

```
{
  "msisdn": "1234567890",
  "gmsc": "5551234567"
}
```

□□	□□	□□	□□
`msisdn`	□□□	□	□□□ MSISDN
`gmsc`	□□□	□	□□ MSC □□□□

□□□200 OK□□

```
{
  "result": {
    "imsi": "001001234567890",
    "msrn": "5551234999",
    "vlr_number": "5551234800"
  }
}
```

cURL □□□

```
curl -X POST http://localhost:9090/api/sri \
  -H "Content-Type: application/json" \
  -d '{"msisdn": "1234567890", "gmsc": "5551234567"}'
```

SendRoutingInfoForSM (SRI-for-SM)

□□□ SMS □□□□□□□□□□□□□□□

□□□`POST /api/sri-for-sm`

□□□□

```
{
  "msisdn": "1234567890",
  "service_center": "5551234567"
}
```

□□	□□	□□	□□
`msisdn`	□□□	□	□□□ MSISDN
`service_center`	□□□	□	□□□□□□□□

cURL □□□

```
curl -X POST http://localhost:9090/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{"msisdn": "1234567890", "service_center": "5551234567"}'
```

UpdateLocation

□□ HLR □□□□□□□VLR □□□□□ HLR □□□□□□□□ InsertSubscriberData (ISD) □□□

□□□`POST /api/updateLocation`

□□□□

```
{
  "imsi": "001001234567890",
  "vlr": "5551234800"
}
```

字段	类型	必需	描述
`imsi`	字符串	是	用户 IMSI
`vlr`	字符串	是	VLR 全球标题地址

USSD 发送（网络发起）

向用户推送 USSD 消息。需要 `ussd_gateway_enabled: true`。有关完整的回调协议和会话生命周期，请参见 USSD 网关指南。

端点：`POST /api/ussd/send`

请求体：

```
{
  "msisdn": "+254712345678",
  "text": "您有一笔待付款项。回复 1 进行支付。",
  "callback_url": "http://billing-app:9000/ussd"
}
```

字段	类型	必需	描述
`msisdn`	字符串	是	目的地用户 MSISDN
`text`	字符串	是	要显示的 USSD 文本（GSM 7 位）
`callback_url`	字符串	是	接收用户回复的 URL，通过 HTTP POST

响应（200 OK）：

```
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "status": "sent"
}
```

错误（503，网关禁用）：

```
{
  "error": "USSD gateway not enabled"
}
```

认证

当前状态：该 API 不需要认证。

安全考虑：

- API 旨在用于内部/受信网络
- 考虑使用防火墙规则限制访问
- 对于生产部署，考虑实现认证中间件

响应格式

所有响应使用 **JSON** 格式。

成功响应

HTTP 状态：200 OK

结构：

```
{
  "result": {
    // 作特定的响应数据
  }
}
```

错误响应

HTTP 状态:

- 400 Bad Request - 无效或缺失请求字段
- 500 Internal Server Error - 操作失败（SS7 栈□回的非超时错误）
- 503 Service Unavailable - 功能未启用（USSD 网关）
- 504 Gateway Timeout - 操作超时（默认 10 秒）
- 404 Not Found - 无效端点

结构:

```
{
  "error": "timeout"
}
```

或

```
{
  "error": "invalid request"
}
```

错误处理

常见错误

错误	HTTP 代码	描述	解决方案
无效 JSON	400	请求体不是有效的 JSON	检查 JSON 语法
缺失/无效字段	400	缺少或类型错误的必需字段（响应通常包含 `"required"` 列表）	包含所有必需参数并确保类型正确
操作错误	500	SS7 栈回了非超时错误	检查 `"error"` 消息；检查 SS7 事件日志
功能禁用	503	USSD 网关未启用	设置 `ussd_gateway_enabled: true`
超时	504	操作超出其超时（默认 10 秒）	检查 M3UA 连接，HLR/VLR 可用性
未找到	404	无效端点	检查端点 URL

超时行为

等待的操作默认使用 **10** 秒超时：

1. 请求发送到 MapClient GenServer
2. 等待响应最长 10 秒
3. 如果没有响应 → 回 504 Gateway Timeout
4. 如果收到响应 → 回 200 OK 和结果

故障排除超时：

- 检查 M3UA 连接状态（Web UI → M3UA 页面）
- 验证网络元素（HLR/VLR/MSC）是否可达
- 检查路由配置

- 查看 SS7 事件日志以获取错误

□标 (Prometheus)

该 API 暴露 Prometheus 指标以供监控。

□标端点

URL: `http://[server-ip]:9090/metrics`

格式: Prometheus 文本格式

示例输出:

```
# HELP map_requests_total Total MAP requests
# TYPE map_requests_total counter
map_requests_total{operation="sri"} 42
map_requests_total{operation="sri_for_sm"} 158
map_requests_total{operation="updateLocation"} 23

# HELP cap_requests_total Total CAP requests
# TYPE cap_requests_total counter
cap_requests_total{operation="initialDP"} 87
cap_requests_total{operation="requestReportBCSMEvent"} 91

# HELP map_request_duration_milliseconds Duration of MAP
request/responses in ms
# TYPE map_request_duration_milliseconds histogram
map_request_duration_milliseconds_bucket{operation="sri",le="10"}
5
map_request_duration_milliseconds_bucket{operation="sri",le="50"}
12
map_request_duration_milliseconds_bucket{operation="sri",le="100"}
35
...

# HELP map_pending_requests Number of pending MAP TID waiters
# TYPE map_pending_requests gauge
map_pending_requests 3

# HELP omniss7_license_status Current license status (1 = valid, 0
= invalid)
# TYPE omniss7_license_status gauge
omniss7_license_status 1
```

可用□标

指标按协议层/子系统分组。所有指标在启动时声明，并使用受保护的助手发出，因此未注册的指标不会干扰协议热路径。

应用（**MAP / CAP / USSD /** 许可证）

□标	类型	标签	描述
`map_requests_total`	计数器	`operation`	按操作类型统计的总 MAP 请求数
`map_request_errors_total`	计数器	`operation`	按操作类型统计的总 MAP 请求错误数
`cap_requests_total`	计数器	`operation`	按操作类型统计的总 CAP 请求数
`cap_request_errors_total`	计数器	`operation`	按操作类型统计的总 CAP 请求错误数
`map_request_duration_milliseconds`	直方图	`operation`	请求持续时间（毫秒）
`map_pending_requests`	计数器	-	待处理的 MAP 事务数
`ussd_requests_total`	计数器	`direction`	总 USSD 请求数（入站/出站）
`ussd_active_sessions`	计数器	-	当前活动的 USSD 会话数
`omniss7_license_status`	计数器	-	当前许可证状态（1 = 有效，0 = 无效）

SCCP

□□	□□	□□	
`sccp_messages_total`	□ □□□ □	`message_type`, `direction`	□□□ □`udt`/`xudt`/ □□□□`sent`/`r`
`sccp_decode_errors_total`	□□□	-	□□□□□ SCCP
`sccp_segments_received_total`	□□□	-	□□□ XUDT/LU
`sccp_reassembly_completed_total`	□□□	-	□□□□□□□ SC
`sccp_reassembly_timeouts_total`	□□□	-	□□□□□□□□□□□

TCAP

□□	□ □	□□	
`tcap_dialogues_total`	□ □ □	`type`	□□□□□□`begin`/`continue`/`end`/`abo`
`tcap_aborts_total`	□ □ □	`reason`	□□□□□□ TCAP □□□□□P-Abort → `unrecognized_message_type`/`un` □□ `p_abort_other`□□U-Abort → `u`
`tcap_decode_errors_total`	□ □ □	-	□□/□□□□□□□ TCAP □□
`tcap_active_dialogues`	□ □ □	-	□□□□□ MAP □□□□□

SCTP

□□	□□	□□	
`sctp_association_state`	□□□	`remote_ip`, `remote_port`	□□□□□1 = up□
`sctp_association_state_changes_total`	□□□	`event`	□□□□ □`comm_up`/`co`
`sctp_send_errors_total`	□□□	`reason`	□□□ SCTP □□□
`sctp_heartbeat_sent_total`	□□□	`type`	□□□ M3UA □□
`sctp_heartbeat_received_total`	□□□	`type`	□□□ M3UA □□

□□ / SMSc

□□	□□	□□	□□
`sms_messages_total`	□□□	`direction`, `operation`	□□□□`mo`/`mt`□□□□□□`submit`/`deliver`□□□□ SMS
`sms_delivery_failures_total`	□□□	`reason`	□□□□□□ SMS □□□□
`sms_tracked_subscribers`	□□□	-	□□□ SubscriberTracker □□□□□

M2PA□MTP2 □□□□□

□□	□□	□□	□□
`m2pa_messages_sent_total`	□□□	`link_name`, `point_code`, `adjacent_pc`	□□ M2PA □□ MTP3 □□
`m2pa_messages_received_total`	□□□	`link_name`, `point_code`, `adjacent_pc`	□□ M2PA □□ MTP3 □□
`m2pa_bytes_sent_total`	□□□	`link_name`, `point_code`, `adjacent_pc`	□□ M2PA □□□□□
`m2pa_bytes_received_total`	□□□	`link_name`, `point_code`, `adjacent_pc`	□□ M2PA □□□□□
`m2pa_link_state_changes_total`	□□□	`link_name`, `from_state`, `to_state`	M2PA □□□□□□
`m2pa_errors_total`	□□□	`link_name`, `error_type`	□□□□□□□ M2PA □□

M3UA STP

□□	□□	□□	□□
`m3ua_stp_messages_received_total`	□□□	`peer_name`, `point_code`	□□□□□□□ M3UA □□
`m3ua_stp_messages_sent_total`	□□□	`peer_name`, `point_code`	□□□□□□□ M3UA □□
`m3ua_stp_routing_failures_total`	□□□	`reason`	STP □□□□ □`no_route`/`no_gt_`

Prometheus □□

□□□□□ `prometheus.yml`□

```
scrape_configs:
  - job_name: 'omniss7'
    static_configs:
      - targets: ['server-ip:9090']
    metrics_path: '/metrics'
    scrape_interval: 15s
```

□□□□

Python □□

```python
import requests
import json

# SRI-for-SM □□
url = "http://localhost:9090/api/sri-for-sm"
payload = {
    "msisdn": "1234567890",
    "service_center": "5551234567"
}

response = requests.post(url, json=payload, timeout=15)

if response.status_code == 200:
    result = response.json()
    print(f"□□: {result}")
elif response.status_code == 504:
    print("□□ - □□□□□□□")
else:
    print(f"□□: {response.status_code} - {response.text}")
```

JavaScript □□

```
const axios = require('axios');

async function sendSRI() {
  try {
    const response = await
axios.post('http://localhost:9090/api/sri', {
      msisdn: '1234567890',
      gmsc: '5551234567'
    }, {
      timeout: 15000
    });

    console.log('□□:', response.data);
  } catch (error) {
    if (error.code === 'ECONNABORTED') {
      console.error('□□ - □□□□□□');
    } else {
      console.error('□□:', error.response?.data || error.message);
    }
  }
}

sendSRI();
```

Bash/cURL □□

```
#!/bin/bash

# UpdateLocation □□
response=$(curl -s -w "\n%{http_code}" -X POST
http://localhost:9090/api/updateLocation \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "vlr": "5551234800"
  }')

http_code=$(echo "$response" | tail -n 1)
body=$(echo "$response" | sed '$d')

if [ "$http_code" -eq 200 ]; then
  echo "□□: $body"
elif [ "$http_code" -eq 504 ]; then
  echo "□□ - □□□□□□"
else
  echo "□□ $http_code: $body"
fi
```

流程图

API 请求流程

总结

OmniSS7 REST API 提供:

- **47** 个操作端点，涵盖移动性、短信、认证、用户数据、补充服务、位置服务、GPRS、设备、USSD、CAP/CAMEL 和原始注入。
- **Swagger UI** 在 `/swagger` 提供交互式文档和测试。
- **Prometheus** 指标在 `/metrics` 提供监控和可观察性。
- **10** 秒默认超时用于等待的操作（超时回 `504`）。
- 一个 `Plug.Cowboy` **HTTP** 服务器在端口 **9090**，由 `start_http_server` 切换。该服务器与控制面板（:8087）和 `:api_ex` REST 控制器（:8445）分开。

有关每个操作的请求/响应模式和 MAP 操作码，请参见 MAP 客户指南。

有关 Web UI 访问，请参见 Web UI 指南。

技术参考（附录）

← 回主文档

SS7协议和OmniSS7实现的技术参考。

SS7协议栈

MAP操作码

下面的操作码是根据3GPP TS 29.002的MAP `local` 操作码。此表涵盖了OmniSS7生成和/或处理的操作；有关完整的客户端触发操作及其请求参数，请参见**MAP客户端指南**。

操作	操作码	目的
updateLocation	2	注册用户位置（CS）
cancelLocation	3	从VLR/SGSN注销用户
provideRoamingNumber	4	请求MSRN
noteSubscriberDataModified	5	通知gsmSCF用户数据更改
insertSubscriberData	7	将用户资料推送到VLR/SGSN
deleteSubscriberData	8	从VLR/SGSN撤回用户数据
registerSS	10	注册附加服务（例如CFU）
eraseSS	11	删除附加服�?
activateSS	12	激活附加服务
deactivateSS	13	停用附加服务
interrogateSS	14	查询附加服务状态
registerPassword	17	注册SS密码
getPassword	18	请求用户的SS密码
sendRoutingInfo	22	查询呼叫路由（SRI）
updateGprsLocation	23	注册分组交换（SGSN）位置
sendRoutingInfoForGprs	24	查询指向GGSN的GPRS路由
reset	37	HLR重置到VLR/SGSN
forwardCheckSS-Indication	38	指示SS数据应重新检查

操作	操作码	目的
checkIMEI	43	查询EIR以获取设备状态
mt-forwardSM	44	将SMS发送给用户（MT）
sendRoutingInfoForSM	45	查询SMS路由（SRI-for-SM）
mo-forwardSM	46	从用户转发SMS（MO）
reportSM-DeliveryStatus	47	向HLR报告SMS交付结果
sendIdentification	55	从先前的VLR检索IMSI/auth
sendAuthenticationInfo	56	请求认证向量
restoreData	57	在VLR故障后恢复用户数据
sendIMSI	58	在HLR中将MSISDN解析为IMSI
processUnstructuredSS-Request	59	移动发起的USSD请求（入站）
unstructuredSS-Request	60	网络发起的USSD请求（出站）
unstructuredSS-Notify	61	网络发起的USSD通知
anyTimeSubscriptionInterrogation	62	gsmSCF查询订阅数据
alertServiceCentre	64	通知SMSc用���可达
anyTimeModification	65	gsmSCF修改用户数据
readyForSM	66	通知HLR用户准备好接收SMS
purgeMS	67	从VLR/SGSN中清除用户记录
provideSubscriberInfo	70	请求用户状态/位置

操作	操作码	目的
anyTimeInterrogation	71	gsmSCF查询用户状态/位置
provideSubscriberLocation	83	请求服务节点的位置信息（LCS）
sendRoutingInfoForLCS	85	GMLC查询用户的服务节点（LCS）
subscriberLocationReport	86	服务节点向GMLC报告位置（LCS）
noteMM-Event	89	向gsmSCF报告移动管理事件

TCAP消息类型

- **BEGIN** - 开始事务
- **CONTINUE** - 事务中
- **END** - 最终响应
- **ABORT** - 取消事务

SCCP寻址

全球标题格式

- **E.164** - 国际电话号码（例如，447712345678）
- **E.212** - IMSI格式（例如，234509876543210）
- **E.214** - 点码格式

子系统编号（**SSN**）

- **SSN 6**: HLR
- **SSN 7**: VLR

SSN 8: MSC/SMSC
SSN 9: GMLC
SSN 10: SGSN

SMS TPDU

消息类型

SMS-DELIVER (MT) - 网络到移动 SMS-SUBMIT (MO) - 移动到网络 SMS-STATUS-REPORT - 交付状态 SMS-COMMAND - 远程命令

字符编码

GSM7 - 7位GSM字母表（每条SMS 160个字符） UCS2 - 16位Unicode（每条SMS 70个字符） 8-bit - 二进制数据（每条SMS 140字节）

M3UA状态

DOWN - 无SCTP连接
CONNECTING - SCTP连接中 ASPUP_SENT - 等待ASPUP确认 INACTIVE - ASP已启动但未激活
ASPAC_SENT - 等待ASPAC确认 ACTIVE - 准备处理流量

常见SS7点码

点码通常是14位（ITU）或24位（ANSI）值。

示例格式（ITU）：

网络：3位
集群：8位
成员：3位

SCCP错误代码

0 - 地址没有翻译
1 - 特定地址没有翻译
2 - 子系统拥塞
3 - 子系统故障
4 - 用户未装备
5 - MTP故障
6 - 网络拥塞
7 - 不合格
8 - 消息传输错误

MAP错误代码

代码	错误	描述
1	unknownSubscriber	用户不在HLR中
27	absentSubscriber	用户不可达
34	systemFailure	网络故障
35	dataMissing	所需数据不可用
36	unexpectedDataValue	参数值无效

相关文档

- ← 回主文档
- STP指南
- MAP客户端指南
- 短信中心指南
- HLR指南
- 常见功能

CAMEL 网关配置□南

概述

CAMEL 网关 **(CAMELGW)** 模式将 OmniSS7 转变为一个智能网络 (IN) 平台，通过 CAMEL 应用部分 (CAP) 协议提供实时呼叫控制和计费。OmniSS7 充当 **gsmSCF** (服务控制功能)，接收来自 MSC/VLR/GMSC 中 **gsmSSF** 的触发，并通过 CGrateS 驱动计费决策。

什么是 **CAMEL**?

CAMEL (定制应用程序用于移动网络增强逻辑) 是一组标准，可以在 GSM 核心网络或 UMTS 网络上运行。它允许运营商提供需要实时控制呼叫的服务，例如:

- 预付费呼叫 - 实时余额检查和计费
- 增值服务 - 价值增加服务的特殊计费
- 呼叫路由控制 - 基于时间/位置的动态目的地路由
- 虚拟专用网络 - 企业编号计划
- 呼叫筛选 - 根据标准允许/阻止呼叫

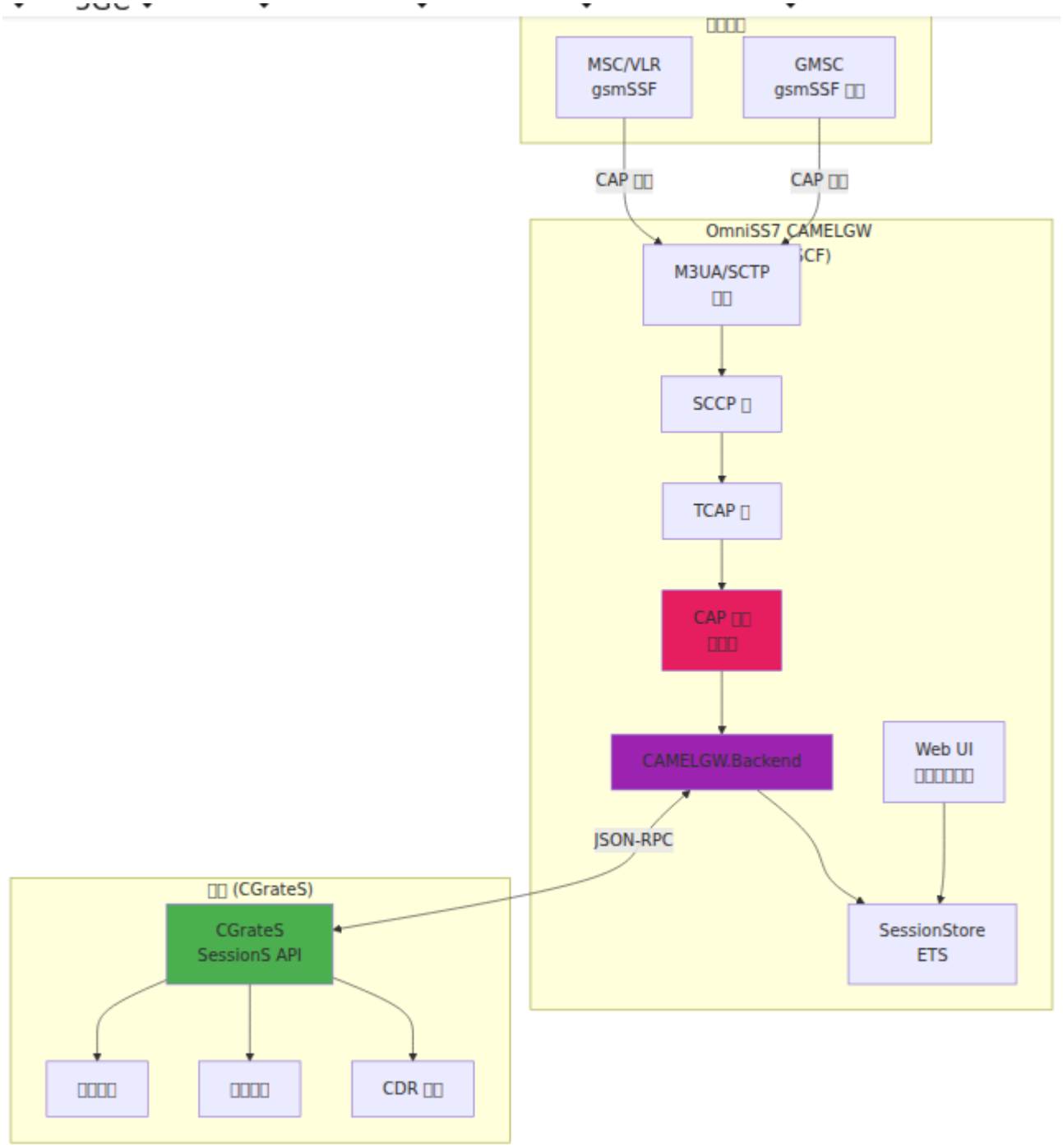
CAP 协议版本

OmniSS7 CAMELGW 支持多个 CAP 版本。版本控制在 TCAP 对话中携带的应用上下文 OID:

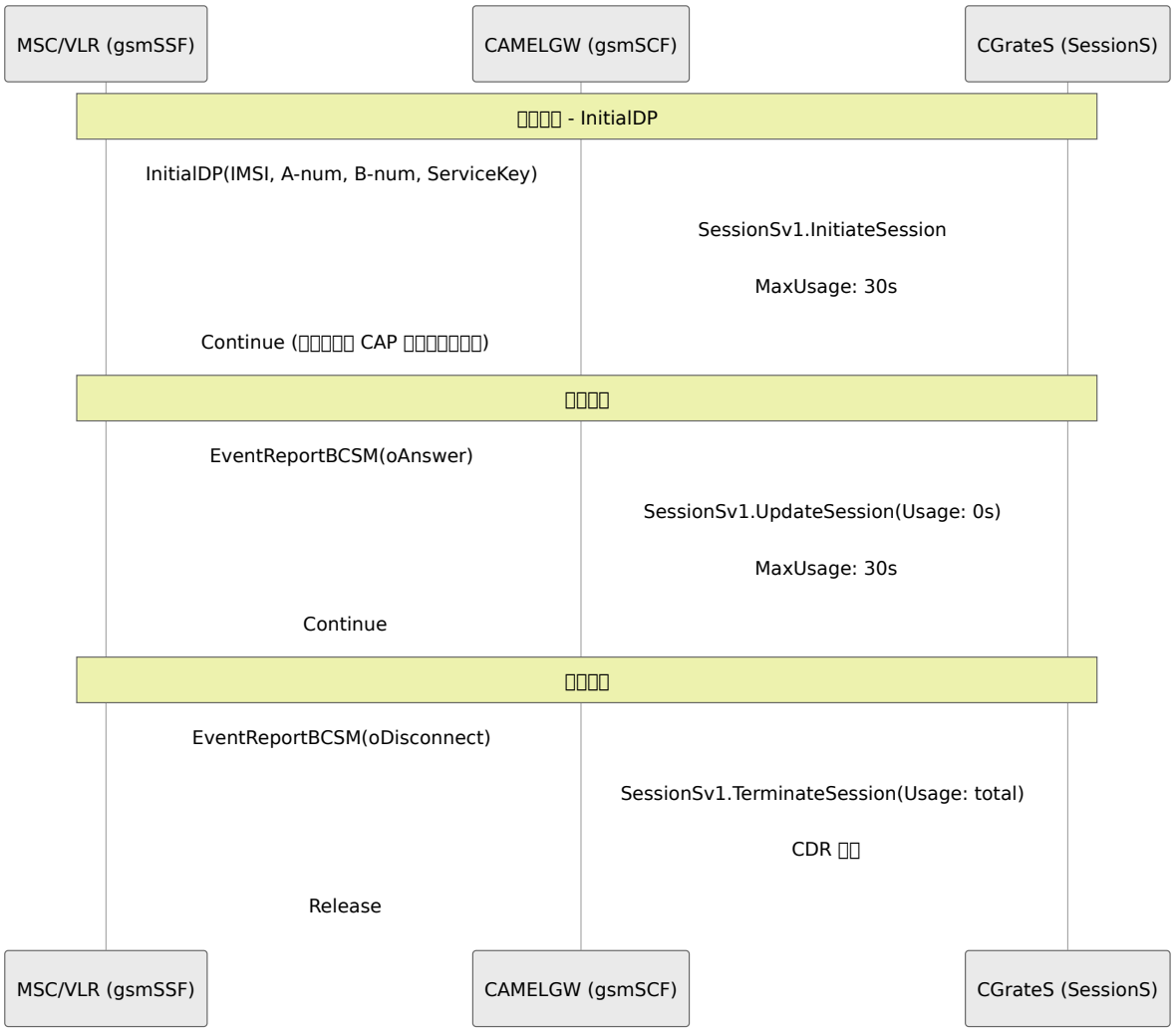
版本	阶段	应用上下文 **OID**
CAP v1	CAMEL 阶段 1	`0.4.0.0.1.0.50.0`
CAP v2	CAMEL 阶段 2	`0.4.0.0.1.0.50.1`
CAP v3	CAMEL 阶段 3	`0.4.0.0.1.21.3.4`
CAP v4	CAMEL 阶段 4	`0.4.0.0.1.23.3.4`

默认**:** CAP v2 (最广泛部署)。传入版本从 InitialDP 对话部分自动检测，并在生成响应时重用，以便响应与每个 MSC 的 CAP 版本匹配。

□□


MSC/VLR
gsmSSF
GMSC
gsmSSF □□
CAP □□
CAP □□
OmniSS7 CAMELGW
M3UA/SCTP
□□
SCCP □
TCAP □
CAP □□
□□□
CAMELGW.Backend
Web UI
□□□□□□□
JSON-RPC
SessionStore
ETS
□□ (CGrateS)
CGrateS
SessionS API
□□□□
□□□□
CDR □□

□□□□□□



□□

□□□□

- □□□□□□ OmniSS7
- □ MSC/GMSC (gsmSSF) □ M3UA □□
- □□□ JSON-RPC □□□ CGrateS □□ (□□□□□□□□□□)

启用 **CAMEL** 网关模式

CAMEL 处理由两个独立的标志控制。`cap_client_enabled` 启动 CAP/M3UA 客户端和 CAMEL 会话组件；`cgrates_enabled` 启用后端的 CGrateS 计费集成。在 `config/runtime.exs` 中配置这些：

```
config :omniss7,
  # 启动 CAP/CAMEL 客户端 (M3UA ASP, SessionStore, 请求处理)
  cap_client_enabled: true,

  # 其他独立模式 (根据需要为您的部署□置)
  map_client_enabled: false,
  smsc_mode_enabled: false,

  # 用于外发请求/对话的 CAP 版本 (:v1 | :v2 | :v3 | :v4)
  cap_version: :v2,

  # CGrateS 计费集成
  cgrates_enabled: true,

  # CAP 的 M3UA 连接 (作为 ASP 连接到 MSC/GMSC gsmSSF)
  cap_client_m3ua: %{
    routing_context: 1,
    opc: 5013,
    dpc: 5011
  }
```

> 关于 **`cap_client_m3ua`** 的说明**:** CAMEL 客户端仅从此映射中读取 `:routing_context`、`:opc` 和 `:dpc`。任何其他键将被 CAP 客户端忽略。M3UA ASP 进程名称在内部固定为 `:camelgw_client_asp`。

> 没有 `camelgw_mode_enabled`、`hlr_mode_enabled` 或 `smsc_mode_enabled`-for-CAMEL 标志。控制 CAMEL 行为的唯一标志是 `cap_client_enabled` 和 `cgrates_enabled`。

模式标志

□	□□	□□
`cap_client_enabled`	`false`	□□ CAP/CAMEL □□□ (M3UA ASP, SessionStore, CAP □□□□)
`cgrates_enabled`	`false`	□□ `CAMELGW.Backend` □□ CGrateS □□□□
`cap_version`	`:v2`	□□□□□□□ CAP □□/□□ OID
`map_client_enabled`	`false`	□□□ MAP □□□□□
`smsc_mode_enabled`	`false`	□□□ SMSc □□ (□□ MAP □□□)

CGrateS □□□□

□	□□	□□
`cgrates_enabled`	`false`	CGrateS □□□ □□□
`cgrates_url`	`http://localhost:2080/jsonrpc`	CGrateS JSON-RPC □□
`cgrates_tenant`	`cgrates.org`	□□ SessionS □□□□□□□
`cgrates_request_type`	`*prepaid`	CGrateS □□□ □ (□□ `*prepaid`□ `*postaid`)
`cgrates_timeout`	`5000`	HTTP □□□□□□ □□□□□□

配置 Web UI 页面

Web UI 通过控制面板公开 CAMEL 页面。这些在 `config/config.exs` 中配置:

```
config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 事件"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "对等体"},
    {SS7.Web.CAMELSessionsLive, "/camel_sessions", "CAMEL 会话"},
    {SS7.Web.CAMELRequestLive, "/camel_request", "CAMEL 请求构建器"}
  ],
  page_order: [
    "/events", "/m3ua", "/camel_sessions", "/camel_request",
    "/application", "/configuration"
  ]
```

页面标签	路由	模块
对等体	`/m3ua`	`SS7.Web.M3UAStatusLive`
CAMEL 会话	`/camel_sessions`	`SS7.Web.CAMELSessionsLive`
CAMEL 请求构建器	`/camel_request`	`SS7.Web.CAMELRequestLive`

支持的 CAP 操作

CAP 操作通过名称和方向进行识别。请注意，一些 CAP 操作码在内部不一致（事件日志操作码映射和请求生成器在某些操作上不一致），因此下面的操作通过 名称和端点 描述。引用的操作码是请求生成器实际发出的操作码。

传入操作 (gsmSSF → gsmSCF)

根据解码的 TCAP 组件操作码分派:

操作	备注
InitialDP	呼叫设置通知；触发 CGrateS `InitiateSession`。在操作码 0 和操作码 60（视为 InitialDP/InitialDPSMS）时接受
AssistRequestInstructions	已确认；在 TCAP 开始时在操作码 1 分派
EventReportBCSM	BCSM 事件（oAnswer, oDisconnect, …）；驱动 CGrateS 更新/终止
Continue	已确认
ConnectToResource	已确认
RequestReportBCSMEvent	已确认
ApplyCharging	参数 BER 解码并已确认
Connect	参数 BER 解码并已确认
ReleaseCall	已确认

未处理的操作码将以 TCAP 错误响应（不支持的设施 / 意外数据值）。

外发操作 **(gsmSCF → gsmSSF)**

发送到 gsmSSF。下面的操作码是 OmniSS7 发出的值：

□□	□□□
InitialDP	0
Connect	20
ReleaseCall	22
RequestReportBCSMEvent	23
Continue	31
ApplyCharging	35
InitialDPSMS	60
NoteMM-Event	89 (MAP)

> **InitialDPSMS** (□□□ 60) □□ MO-SMS CAMEL □□□□ smsSSF □ SMS-CSI / mo-sms-CSI □□□□□□□ gsmSCF□□□□□□□□□ SMSC □□□□□□□□□□ CAP ASN.1 □□□□□□□ □□ `id-ac-cap3-sms-AC` □□□□□□
>
> **NoteMM-Event** □□□ MAP □□ (□□□ 89□□□ CAP)□□□□ VLR □ gsmSCF □ CAMEL M-CSI □□□□□□□□ MAP ASN.1 □□□□ (`:TCAPMessages.encode(:MapSpecificPDUs, …)`) □□□ `mm-EventReporting-v3` □□□□□□

□□ CAP TCAP □□□□□ `:GenericTCAP.encode(:GenericPDUs, …)` □□□□□□□□ MAP □ NoteMM-Event □□ `:TCAPMessages`□

Web UI □□

CAMEL □□□□

□□**:** CAMEL □□ · □□**:** `/camel_sessions` (HTTPS, □□ 8087)

实时监控活动的 CAMEL 呼叫会话：

实时会话列表，自动刷新
会话详情 - OTID、呼叫 ID、状态、持续时间
CAP 版本从 InitialDP 检测并按会话存储
呼叫信息 - IMSI、A 号码、B 号码、服务密钥
状态跟踪 - initiated、answered、terminated

表列:呼叫 ID、状态、版本、IMSI、主叫号码、被叫号码、服务密钥、持续时间、开始时间、OTID

相同的会话数据可以通过 REST API 在 /camel-sessions 访问 (见监控与操作)。

CAMEL 请求构建器

标题: CAMEL 请求构建器 · 路由: /camel_request (HTTPS, 端口 8087)

用于构建和发送 CAP 请求以进行测试的交互式工具。

表单提供的请求类型: InitialDP、Connect、ReleaseCall、RequestReportBCSMEvent、 ApplyCharging、Continue。

CAP 版本选择器:表单具有每个请求的 CAP 版本下拉菜单 (v1/v2/v3/v4 及其 OID)。选择会暂时覆盖该请求的配置默认值，并在表单更改之间保持；请求历史显示在CAP Ver列中使用的版本。

高级 SCCP/M3UA 选项:

字段	默认	备注
被叫方 GT	`99988415011`	目标全局标题
主叫方 GT	(可配置)	源全局标题
被叫 SSN	`146`	gsmSSF
主叫 SSN	`146`	gsmSSF
OPC	`5013`	源点代码
DPC	`5011`	目标点代码

与 **CGrateS** 的集成

OmniSS7 通过 CGrateS **SessionS** JSON-RPC API 对呼叫进行计费。
`CGrateS.SessionsClient` 发布到配置的 `cgrates_url` 并将 CAP ��作映射到 SessionS 方法:

CAP 触发	**SessionS** 方法
InitialDP	`SessionSv1.InitiateSession`
EventReportBCSM (oAnswer)	`SessionSv1.UpdateSession`
EventReportBCSM (oDisconnect)	`SessionSv1.TerminateSession`

带计费的呼叫生命周期

1. 呼叫发起 **(InitialDP)**

当 MSC 发送 InitialDP 时, CAMELGW:

1. 从对话部分检测 **CAP** 版本 (回退到配置的默认值)

2. 解码 **InitialIDP** 参数（IMSI、主叫/被叫号码、服务密钥）
3. 调用 **CGrateS** `SessionSv1.InitiateSession`
4. 从响应中读取 **`MaxUsage`**（允许的呼叫持续时间，以秒为单位）
5. 在 **`CAMELGW.SessionStore`** 中存储会话（ETS），以 OTID 为键，包括 CAP 版本和形式为 `CAMEL-<hex OTID>` 的呼叫 ID
6. 使用检测到的 **CAP** 版本响应 **MSC**

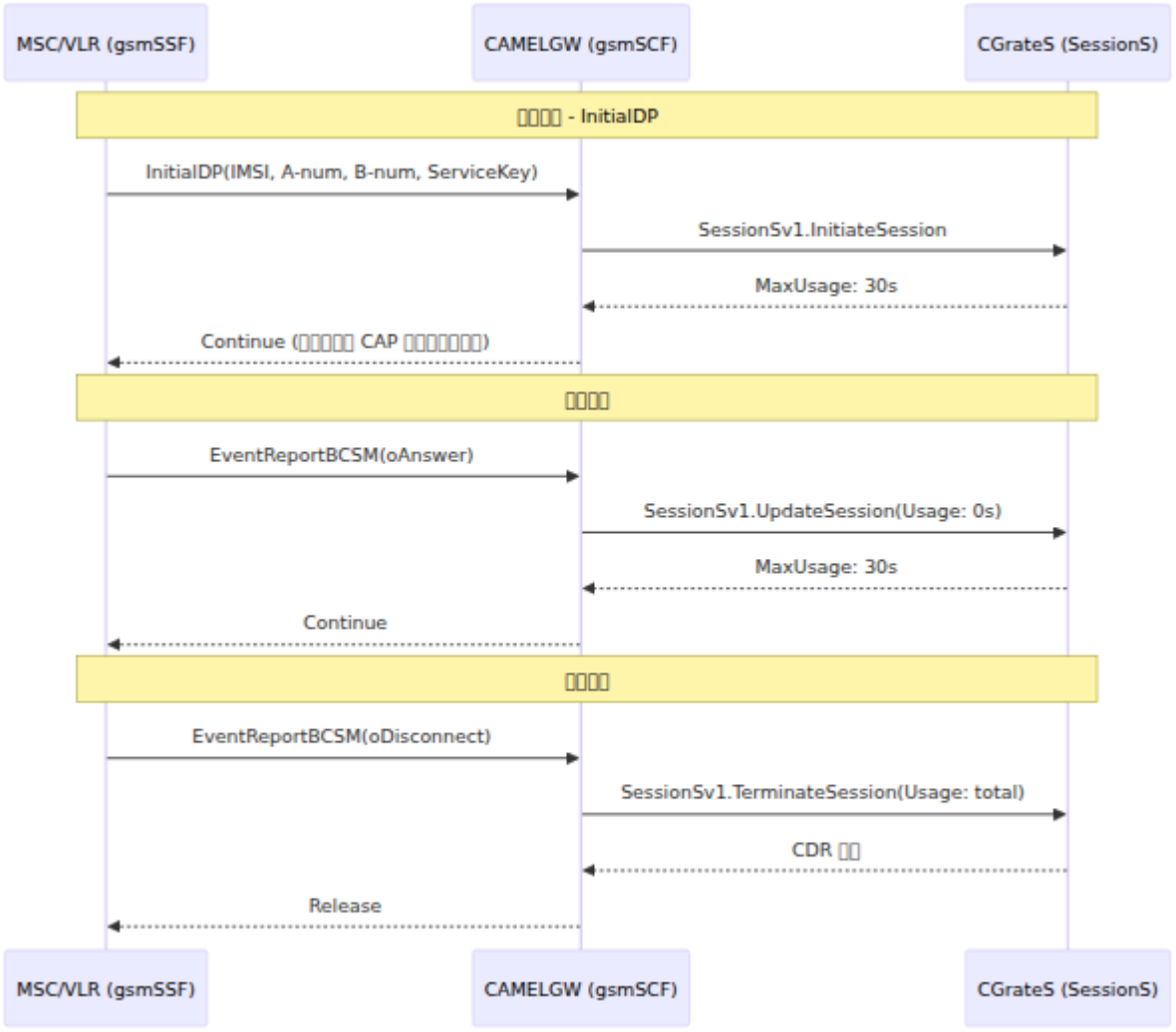
SessionS 事件上的账户是订阅者 **IMSI**（如果存在），否则为主叫号码。事件还携带 `Destination`、`ServiceKey`、`MSCAddress`、`CallingPartyNumber`、`CalledPartyNumber` 和 `OriginID`（呼叫 ID）。

2. 呼叫应答 (EventReportBCSM - oAnswer)

在 `oAnswer` 时，后端查找会话，将其状态设置为 `answered`，并以使用量 0 调用 `SessionSv1.UpdateSession` 以启动扣费循环。如果 CGrateS 失败，后端 不会 终止呼叫，后端记录错误并继续。

3. 呼叫终止 (EventReportBCSM - oDisconnect)

在 `oDisconnect` 时，后端根据会话开始时间计算总持续时间，将状态设置为 `terminated`，以最终使用量调用 `SessionSv1.TerminateSession`，并从存储中删除会话。CDR 由 CGrateS 生成。即使终止呼叫出现错误，仍会清理会话。



CAMEL □ CDR □□

CDR □ CGrateS □□□□□□□□

- `Account` - IMSI (□□□□□□□)
- `Destination` - □□□□□
- `OriginID` - □□□□□ (`CAMEL-<hex OTID>`)
- `Usage` - □□□□□□□ (□)
- `IMSI`□`CallingPartyNumber`□`CalledPartyNumber`
- `MSCAddress` - □□ MSC
- `ServiceKey` - CAMEL □□□

测试

使用请求构建器进行手动测试

1. 打开 `https://your-server:8087/camel_request`
2. 选择 **InitialDP**，设置服务密钥/主叫/被叫号码，选择 CAP 版本（可选），并发送。注意生成的 OTID。
3. 打开 `https://your-server:8087/camel_sessions` 查看状态为 `initiated` 的会话。
4. 发送 **EventReportBCSM**，使用 `oAnswer` 将会话移动到 `answered`。
5. 发送 **ReleaseCall**，原因 `16`（正常）以终止。

使用真实 **MSC** 进行测试

在您的 MSC/VLR 上配置 CAMEL 服务（服务密钥、gsmSCF 全局标题），以便触发向 OmniSS7 的 InitialDP。华为 MSC 的示例：

```
ADD CAMELSERVICE:
  SERVICEID=1,
  SERVICEKEY=100,
  GSMSCFADDR="55512341234",   # CAMELGW 全局标题
  DEFAULTCALLHANDLING=CONTINUE;

ADD CAMELSUBSCRIBER:
  IMSI="310150123456789",
  SERVICEID=1,
  TRIGGERTYPE=TERMCALL;
```

监控日志

OmniSS7 发出结构化 JSON 日志（每行一个对象，通过 `OmniLogger.JsonFormatter`），因此您可以根据元数据字段进行过滤：

```
# 实时监控并过滤 CAP 事件，按 map_event 字段
tail -f /var/log/omniss7/omniss7.log | jq 'select(.map_event |
tostring | startswith("CAP:"))'
```

□□□□□

Prometheus □□

CAMELGW □ `http://localhost:9090/metrics` □□□□□

CAP □□:

- `cap_requests_total{operation}` - □□□□□ CAP □□□□□□□ snake_case□□□ `initial_dp`□`connect`□`release_call`□`continue`□`apply_charging`□
- `cap_request_errors_total{operation}` - □□□□ CAP □□□□/□□□□□

MAP / API □□:

- `map_requests_total{operation}` - □□□□□ MAP □□
- `map_pending_requests` - □□□□ MAP TID □□□□□

M3UA STP □□ (□□□□□ STP □□):

- `m3ua_stp_routing_failures_total{reason}` - □□□□□□□□

```
# □□ CAP □□□□□
curl http://localhost:9090/metrics | grep cap_requests_total

# □ InitialDP (□□ snake_case □□)
curl http://localhost:9090/metrics | grep
'cap_requests_total{operation="initial_dp"}'
```

□□□□

```
# M3UA □□□□□ (TLS API □□□ 8445)
curl -k https://localhost:8445/m3ua-status

# □□ CAMEL □□ (□□ session_count + □□□□□□□□)
curl -k https://localhost:8445/camel-sessions
```

没有 `/api/ocs-status`、`/api/events` 或 `/api/camel/sessions/count` 端点。会话计数来自 `/camel-sessions`。

日志配置

在 `config/runtime.exs` 中调整日志级别：

```
config :logger, level: :info   # :debug | :info | :warning |
:error
```

格式化程序在 `config/config.exs` 中设置为结构化 JSON：

```
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all,
  truncate: :infinity
```

故障排除

问题：未接收到 CAP 消息

症状：请求构建器工作，但 MSC 从未发送 InitialDP。

检查：

1. M3UA 对等体状态：`curl -k https://localhost:8445/m3ua-status`
2. MSC CAMEL 服务配置（服务密钥、gsmSCF 地址）
3. SCCP 路由（gsmSCF 全局标题必须路由到 CAMELGW）
4. 防火墙规则（允许 SCTP 端口 2905）

问题：CGrateS 错误

症状：日志中出现 `not_configured`、`http_error` 或 `cgrates_error`。

检查：

1. `cgrates_enabled: true` 和 `cgrates_url` 指向可访问的 JSON-RPC 端点
2. 账户 (IMSI 或主叫号码) 存在并在 CGrateS 中有余额
3. `cgrates_tenant` 和 `cgrates_request_type` 与您的 CGrateS 计费设置匹配
4. `cgrates_timeout` 对于您的 CGrateS 延迟足够大

在呼叫期间发生计费失败时，CAMELGW 继续呼叫而不是终止；错误被记录。

问题：未找到会话

症状：EventReportBCSM 日志 "未找到会话"。

原因：OTID 不匹配或会话已终止。

检查：验证日志中的 OTID，并确保 DTID/OTID 在���话中相关。

问题：解码错误

症状：日志中出现 "解码 InitialDP 失败"。

原因：CAP 版本不匹配或消息格式错误。

检查：确认 CAP 版本配置与 MSC 匹配，并捕获 PCAP 进行分析：

```
tcpdump -i eth0 -w cap_trace.pcap sctp port 2905
```

总结

CAMEL 网关模式使 OmniSS7 能够充当 gsmSCF，具有：

- **CAP** 协议支持 (v1/v2/v3/v4) 及每请求版本覆盖
- 通过 **CGrateS SessionS JSON-RPC API** 的实时计费
- 呼叫控制操作 (Connect、Release、Continue、ApplyCharging)
- **ETS** 中的会话管理，以 **OTID** 为键

通过 Web UI 请求构建器的交互式测试
活动会话的实时监控 (Web UI 和 /camel-sessions API)
由 CGrateS 生成的 CDR 用于计费和分析

更多细节：

技术参考 - CAP 操作

产品: OmniSS7 CAMEL 网关
文档版本: 1.1

常见功能□南

← □回主文档

本指南涵盖所有 OmniSS7 操作模式的共同特性。

目录

1. Web UI 概述
2. API 文档
3. 监控和指标
4. 最佳实践
5. 用于网络冗余的 SCTP 多宿主

Web UI 概述

Web UI 通过 HTTPS 在端口 `8087` 提供服务（通过 `force_ssl` 强制使用 TLS），例如 `https://localhost:8087/`。

主导航

菜单由 config/config.exs 中启用的页面构建，并按 page_order排序。常见条目（菜单标签加粗）：

资源 (/application) - 应用状态和运行时信息
配置 (/configuration) - 系统配置查看器
SS7 事件 (/events) - 实时 SS7 信令事件和消息日志
系统日志 (/logs) - 应用日志查看器
SS7 客户端 (/client) - 交互式 MAP 请求测试客户端
对等体 (/m3ua) - M3UA/M2PA 对等连接
路由 (/routing) - 点码和全球标题路由
路由测试 (/routing_test) - 路由解析测试工具
SMSc 链接 (/smsc_links) - 短信后端链接 (SMSc)
SMSc 订阅者 (/smsc_subscribers) - 跟踪的短信订阅者 (SMSc)
HLR 链接 (/hlr_links) - HLR 后端链接 (HLR)
活动订阅者 (/subscribers) - 跟踪的 UpdateLocation 订阅者
CAMEL 会话 (/camel_sessions) - 实时 CAMEL 呼叫会话 (CAMEL GW) CAMEL 请求构建器 (/camel_request) - 交互式 CAP 请求构建器 (CAMEL GW)

- **MSISDN/IMSI** 测试 (`/msisdn_imsi_test`) - 号码转换测试工具

访问 Web UI

1. 打开您的网页浏览器
2. 导航到配置的主机端口 8087（例如，`https://localhost:8087/`）
3. 查看系统状态仪表板

Swagger API 文档

交互式 API 文档由 MAP 操作 API 在 HTTP 端口 9090 提供:

```
http://your-server:9090/swagger
```

Web UI 配置

Web UI 端点在 `config :control_panel, ControlPanelWeb.Endpoint` 中配置，导航菜单在 `config :control_panel` 下:

```
config :control_panel, ControlPanelWeb.Endpoint,
  force_ssl: [rewrite_on: [:x_forwarded_proto]],
  server: true,
  url: [host: "0.0.0.0", path: "/"],
  https: [
    port: 8087,
    keyfile: "priv/cert/omnitouch.pem",
    certfile: "priv/cert/omnitouch.crt"
  ]

config :control_panel,
  page_order: [
    "/application", "/configuration", "/events", "/logs",
"/client",
    "/m3ua", "/routing", "/routing_test", "/smsc_links",
"/smsc_subscribers",
    "/hlr_links", "/subscribers", "/camel_sessions",
"/camel_request",
    "/msisdn_imsi_test"
  ]
```

□□□□□

□□	□□	□□
`ControlPanelWeb.Endpoint.url.host`	□□□□	`"0.0.0.0"`
`ControlPanelWeb.Endpoint.https.port`	□□□	`8087`
`ControlPanelWeb.Endpoint.https.keyfile`	□□□□□	`"priv/cert/omnitouch.pe`
`ControlPanelWeb.Endpoint.https.certfile`	□□□□□	`"priv/cert/omnitouch.cr`
`ControlPanelWeb.Endpoint.force_ssl`	□□□□□	`[rewrite_on: [:x_forwarded_proto]]`
`page_order`	□□□□□□□□	□□□

> □□□ □□□□ □□□□□□□□□□□□□□

日志配置

日志由 `OmniLogger.JsonFormatter` 以结构化 JSON 格式（每行一个对象）发出，其中包含所有 `Logger.metadata`，因此每个进程上下文（STP 路由 OPC/DPC/GTs，短信网关到/从 MSISDNs，事务 otid/操作[illegible]会自动携带。此格式在 `config/config.exs` 中配置：

```
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all,
  truncate: :infinity
```

日志级别单独设置：

```
config :logger,
  level: :debug  # 选项: :debug, :info, :warning, :error
```

配置参数：

□□	□□	□□
`:logger.level`	□□	`:debug`
`:logger.default_formatter.format`	□□	`{OmniLogger.JsonFormatter, :format}`
`:logger.default_formatter.metadata`	□□/□□	`:all`
`:logger.default_formatter.truncate`	□□/□□	`:infinity`

API □□

OmniSS7 □□□□ HTTP □□□

□□	□□	□□	□□
MAP □□ API	`9090`	HTTP	MAP `/api/*` □□□`/swagger`□`/swagger.json`□`/met
JSON □□ API (`api_ex`)	`8445`	HTTPS	□□□□ (`/api/status`□`/api/m3ua-status`□`/api/routing`□`/api/subscribers`□`/api sessions`□...)

MAP API 基础 URL

```
http://your-server:9090/api
```

响应代码

- **200** - 成功
- **400** - 错误请求
- **504** - 网关超时

OpenAPI 规范

```
http://your-server:9090/swagger.json
```

监控和□标

Prometheus □标端点

```
http://your-server:9090/metrics
```

关键□标类别

M3UA/SCTP □标:

- SCTP 关联状态变化
- M3UA ASP 状态转换
- 发送/接收的协议数据单元

M2PA □标:

- 链路状态转换（DOWN → ALIGNMENT → PROVING → READY）
- 每条链路发送/接收的消息和字节

- 链路特定错误（解码、编码、SCTP）

STP 指标：

- 每个对等体接收/发送的消息
- 按原因的路由失败
- 在对等体之间的流量分布

MAP 客户端指标：

- 按操作类型的 MAP 请求
- 请求持续时间直方图
- 待处理事务计量

CAP 指标：

- 按操作类型的 CAP 请求
- CAMEL 网关操作

SMSc 指标：

- 队列深度
- 交付率
- 失败的消息

Grafana 集成

OmniSS7 指标与 Prometheus 和 Grafana 兼容。

最佳实践

安全建议

1. 网络隔离
    - 部署在专用 VLAN 中

防火墙规则限制访问
仅允许来自已知地址的 SCTP

2. Web UI 安全

为生产环境启用 TLS
使用带身份验证的反向代理
限制为管理 IP

3. API 安全

实施速率限制
使用 API 密钥或 OAuth
记录所有请求以备审计

性能调优

1. TPS 限制

配置适当的 TPS
监控系统负载
调整 SCTP 缓冲区

2. 短信后端调优

保持 SMS 后端 API 响应（请求在 5 秒后超时）
调整 auto_flush_tps 和 auto_flush_interval 以匹配后端容量
监控队列深度和交付率

3. M3UA 调优

调整 SCTP 心跳间隔
配置超时值
使用多个链路以实现冗余

用于网络冗余的 SCTP 多宿主

什么是 SCTP 多宿主?

SCTP 多宿主是 SCTP 协议的内置特性，允许单个 M3UA 连接绑定到同一网络接口或不同网络接口上的多个 IP 地址。这提供了传输层的自动故障转移和冗余。

主要好处：

- 自动故障转移：如果一条网络路径失败，SCTP 会自动切换到备用路径而不会中断连接
- 零配置故障转移：不需要应用级逻辑 - SCTP 处理路径监控和故障转移
- 提高可靠性：能够承受网络故障、交换机故障或 NIC 故障
- 负载均衡：SCTP 可以在多个路径之间分配流量（依赖于实现）

工作原理

当您为 M3UA ��接配置多个 IP 地址时，SCTP：

1. 绑定到所有 **IP**：套接字同时绑定到所有配置的 IP 地址
2. 监控路径：SCTP 持续在所有路径上发送心跳数据包以监控其健康状况
3. 检测故障：如果主路径上的心跳失败，SCTP 将其标记为不可达
4. 自动故障转移：流量立即切换到备用路径，无需应用干预
5. 路径恢复：当失败的路径恢复时，SCTP 会检测到并再次将其标记为可用

配置

SCTP 多宿主通过提供 **IP** 地址列表 而不是单个 IP 元组进行配置。

单个 **IP**（传统）

```
# 单个 IP - 无多宿主
local_ip: {10, 179, 4, 10}
```

多个 **IP**（启用多宿主）

```
# 多个 IP - 启用多宿主
# 第一个 IP 是主 IP，后续 IP 是备用路径
local_ip: [{10, 179, 4, 10}, {10, 179, 4, 11}]
```

配置示例

> 服务器角色的重要说明（入站连接）：
>
> 当配置具有 `role: :server`（接受来自多宿主对等体的入站连接）的对等体时，您必须将 `remote_ip` 指定为 单个元组 - 远程对等体用于发起 SCTP 连接（SCTP INIT）的 IP 地址。请勿使用列表。
>
> 为什么？ STP 根据 SCTP INIT 数据包的源 IP 匹配入站连接。SCTP 会在关联握手期间自动发现对等体的其他���宿主 IP 地址。`remote_ip` 的列表格式仅在 `role: :client`（出站连接）中有效。
>
> 示例：
>
> ```
> # ✓ 正确 - 服务器角色，单个 remote_ip
> %{
>   role: :server,
>   remote_ip: {10, 0, 2, 100},  # 单个元组
>   # ...
> }
>
> # ✗ 错误 - 服务器角色，列表将无法匹配入站连接
> %{
>   role: :server,
>   remote_ip: [{10, 0, 2, 100}, {10, 0, 2,
> 101}],  # 列表无效
>   # ...
> }
> ```

示例 **1**：具有多宿主的 **STP** 对等体（客户端角色 **-** 出站）

```
# STP □□□□□□□□□□□□□□□□
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "Partner_STP_Redundant",
      role: :client,  # □□ - □□□□□□□
      # □□□□□□□□□□□□ IP □□□□□□
      local_ip: [{213, 57, 23, 200}, {213, 57, 23, 201}],
      local_port: 0,
      # □□□□□□□□□□□□□ - □□□□□□□□□□□□□□□
      remote_ip: [{213, 57, 23, 100}, {213, 57, 23, 101}],
      remote_port: 2905,
      routing_context: 1,
      point_code: 100,
      network_indicator: :international
    }
  ]
```

□□ 2□□□□□□□ MAP □□□

```
# □□□□□□□ MAP □□□□□□
config :omniss7,
  map_client_enabled: true,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # □□□□□□□□□ IP □□□□□□□□
    local_ip: [{10, 0, 0, 100}, {10, 0, 0, 101}],
    local_port: 2905,
    # □□ STP □□□□□□□□
    remote_ip: [{10, 0, 0, 1}, {10, 0, 0, 2}],
    remote_port: 2905,
    routing_context: 1
  }
```

□□ 3□□□□□□□ STP □□□

```
# 具有多宿主的独立 STP 服务器
config :omniss7,
  sctp_handler: %{
    enabled: true,
    # 监听多个 IP 以接受入站连接
    local_ip: [{172, 16, 0, 10}, {172, 16, 0, 11}],
    local_port: 2905,
    point_code: 100
  }
```

示例 **4**：服务器角色 **-** 接受来自多宿主对等体的入站连接

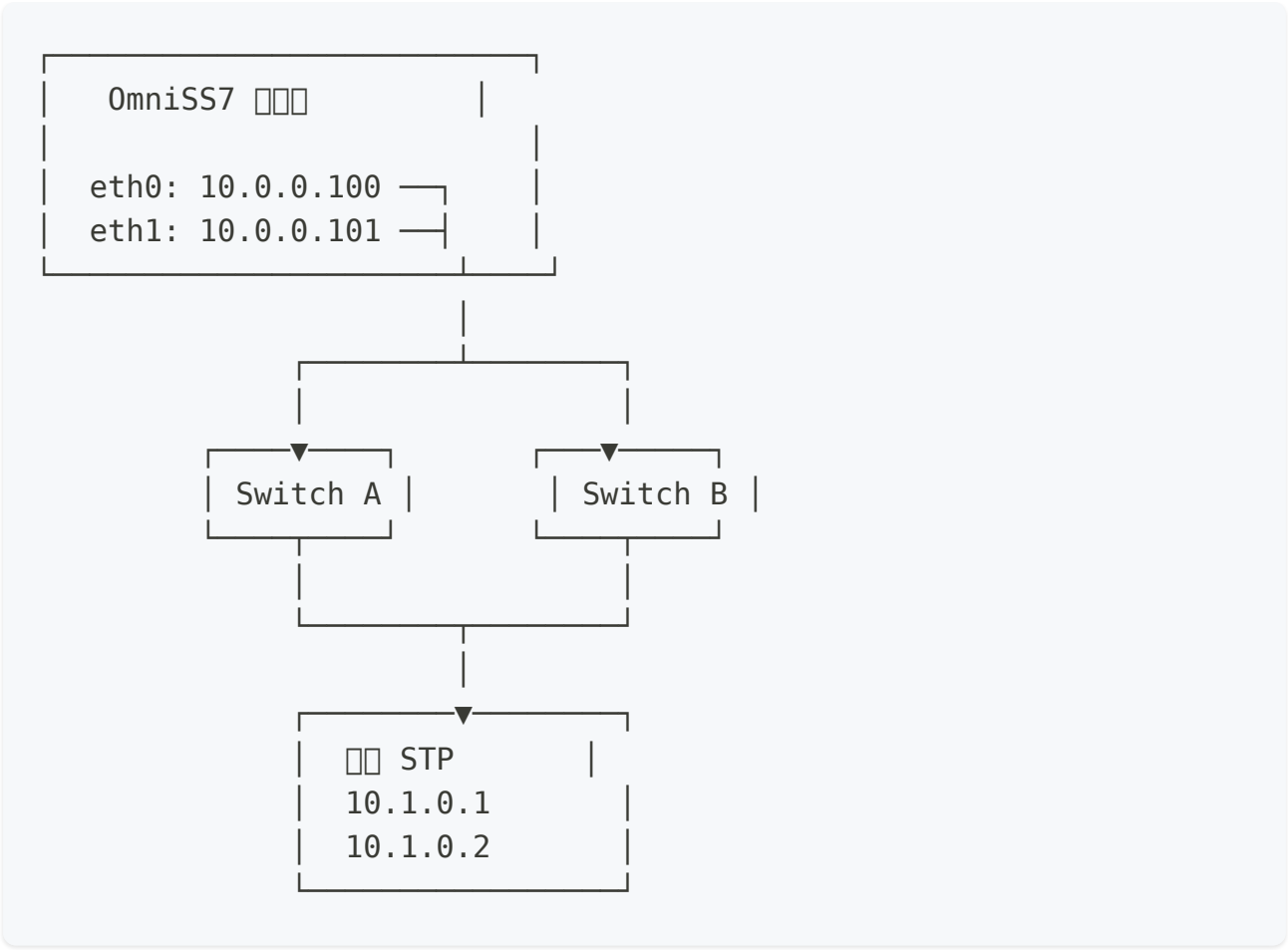
```
# 接受来自多宿主 HLR 的入站连接
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "HLR",
      role: :server,  # 入站 - HLR 连接到我们
      # 多宿主：我们的本地 IP（列表是可以的）
      local_ip: [{10, 0, 1, 10}, {10, 0, 1, 11}],
      local_port: 2905,
      # 重要：仅单个元组 - HLR 用于发起 SCTP INIT 的 IP
      # SCTP 会在握手期间自动发现 HLR 的其他 IP
      remote_ip: {10, 0, 2, 100},  # 仅主 IP（不是列表！）
      remote_port: 0,  # 接受来自任何源端口
      routing_context: 1,
      point_code: 100,
      network_indicator: :international
    }
  ]
```

示例 **5**：混合配置（向后兼容）

```
# □□□□□□□□□□□□□□□
config :omniss7,
  m3ua_peers: [
    # □□□□□□ - □□ IP
    %{
      peer_id: 1,
      name: "Legacy_STP",
      role: :client,
      local_ip: {10, 0, 0, 1},      # □□ IP □□
      local_port: 0,
      remote_ip: {10, 0, 0, 10},
      remote_port: 2905,
      routing_context: 1,
      point_code: 100
    },
    # □□□□□ - □□□
    %{
      peer_id: 2,
      name: "Redundant_STP",
      role: :client,
      local_ip: [{10, 0, 0, 2}, {10, 0, 0, 3}],  # IP □□
      local_port: 0,
      remote_ip: [{10, 0, 0, 20}, {10, 0, 0, 21}],
      remote_port: 2905,
      routing_context: 2,
      point_code: 200
    }
  ]
```

□□□□□□

□□ 1□□ NIC□□□□□□



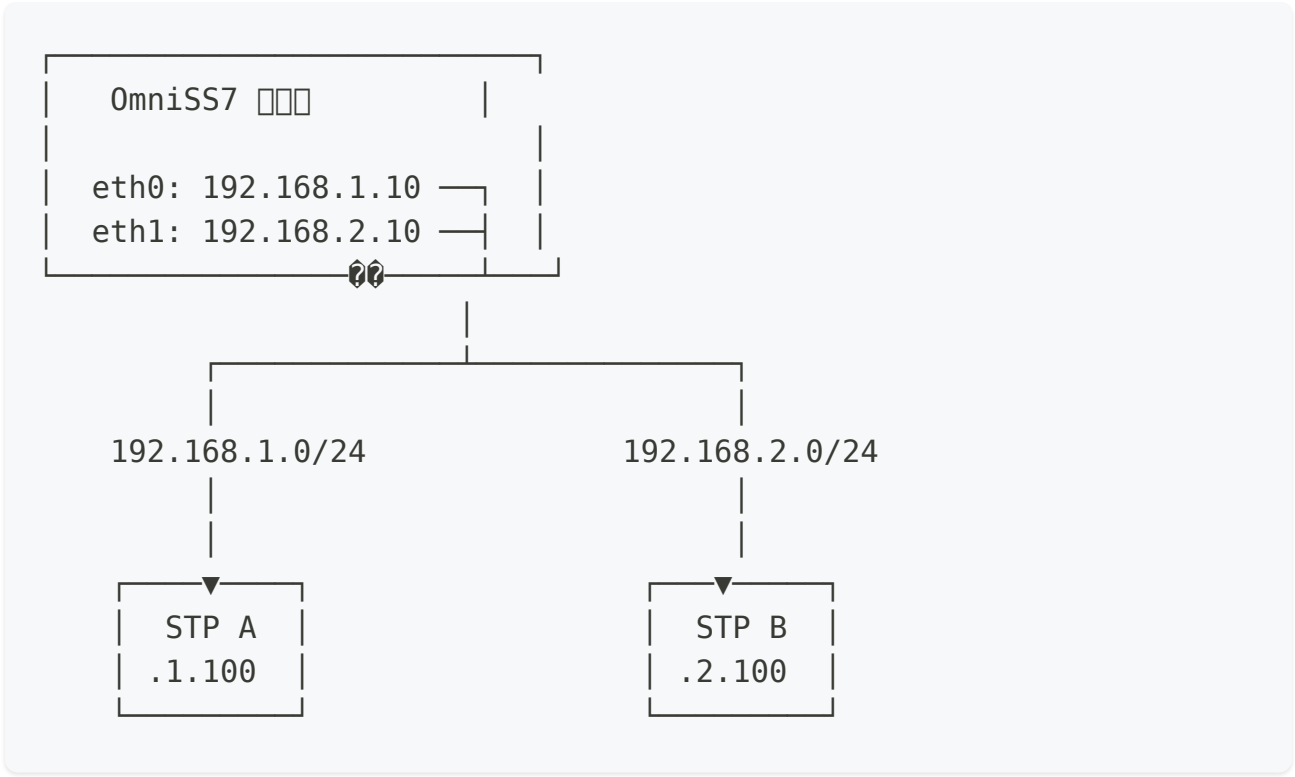
配置:

```
local_ip: [{10, 0, 0, 100}, {10, 0, 0, 101}]   # 两个 NIC
remote_ip: [{10, 1, 0, 1}, {10, 1, 0, 2}]      # 远程对等体
```

好处:

- 能够承受一个 NIC 的故障
- 能够承受一个交换机的故障
- 自动故障转移小于 1 秒

场景 **2**: 多个子网



配置：

```
local_ip: [{192, 168, 1, 10}, {192, 168, 2, 10}]
remote_ip: [{192, 168, 1, 100}, {192, 168, 2, 100}]
```

好处：

- 能够承受子网故障
- 可能实现地理冗余
- 独立的路由路径

监控和日志记录

启用多宿主时，您将看到指示配置的日志消息：

成功的多宿主

```
[info] SCTP 客户端多宿主：绑定 2 个本地 IP
[info] STP 监听器多宿主启用：绑定 2 个本地 IP
```

路径故障转移事件

```
[warning] [MULTIHOMING] 路径 10.0.0.100 对于对等体 Partner_STP
(assoc_id=1) 是不可达的
[info] [MULTIHOMING] 路径 10.0.0.101 现在是对等体 Partner_STP (assoc_id=1)
的主路径
[info] [MULTIHOMING] 路径 10.0.0.100 现在是对等体 Partner_STP (assoc_id=1)
的可用路径
```

Web UI 显示

Web UI 自动显示多宿主信息:

M3UA 状态页面:

- 单个 **IP**: 显示为 `10.0.0.100`
- 多个 **IP**: 显示为 `10.0.0.100 (+1)` 或 `10.0.0.100 (+2)`
- 详细视图: 显示所有 IP 及主/备标签

最佳实践

1. 网络设计

- 使用不同的 **NIC** 以实现最大冗余
- 不同的交换机 以承受交换机故障
- 尽可能使用不同的子网 以实现路由多样性
- 最初在同一数据中心 - 测试后再进行地理分离

2. IP 地址规划

- 第一个 **IP** 是主 **IP** - 确保它位于最可靠的路径上
- 顺序很重要 - 按优先级顺序列出 IP
- 一致的地址分配 - 使用相似的地址方案以便于故障排除

3. 测试故障转移

```
# 禁用主接口以测试故障转移
sudo ip link set eth0 down

# 监控日志以查看故障转移
tail -f /var/log/omniss7.log | grep MULTIHOMING

# 重新启用接口
sudo ip link set eth0 up
```

4. 双方都应支持多宿主

- 最佳：本地和远程都使用多个 IP
- 可接受：只有一方使用多宿主
- 注意：当两个端点都支持时，冗余效果最佳

5. 防火墙配置

```
# 允许所有多宿主 IP 的 SCTP
iptables -A INPUT -p sctp --dport 2905 -s 10.0.0.0/24 -j ACCEPT
iptables -A INPUT -p sctp --dport 2905 -s 10.1.0.0/24 -j ACCEPT
```

故障排除

问题：多宿主无法工作

症状：仅使用主 IP，无故障转移

检查：

1. 验证 Erlang SCTP 支持：`erl -eval 'gen_sctp:open(9999, [binary, {ip, {127,0,0,1}}]).'`
2. 检查内核 SCTP 模块：`lsmod | grep sctp`
3. 如有需要加载 SCTP：`sudo modprobe sctp`
4. 验证系统上配置了两个 IP：`ip addr show`

问题：路径未故障转移

症状：主路径标记为下线，但流量未切换

检查：

1. 检查 SCTP 心跳设置
2. 验证路由表中有所有路径的路由
3. 检查防火墙允许所有 IP 上的 SCTP
4. 查看 SCTP 路径监控日志

问题：频繁的路径波动

症状：路径在 UP 和 DOWN 之间不断切换

检查：

1. 网络不稳定 - 检查物理链路
2. SCTP 心跳过于激进 - 可能需要调优
3. 防火墙丢弃 SCTP 心跳
4. 一条路径上的 MTU 问题

性能考虑

- 最小开销：SCTP 心跳小且不频繁
- 无需应用更改：多宿主对应用层透明
- 快速故障转移：通常小于 1 秒的检测和故障转移
- 自动恢复：无需手动干预

兼容性

- 向后兼容：单个 IP 元组格式仍���有效
- 混合部署：可以混合单 IP 和多 IP 对等体
- 所有模式支持：在 STP、HLR、SMSc 和 MAP 客户端模式下均有效
- **Erlang** 要求：需要编译支持 SCTP 的 Erlang

监控和警报

关键□标：

- M3UA 连接状态

- MAP 请求成功率
- API 响应时间
- 消息队列深度

警报阈值：

- M3UA 下线 > 1 分钟
- MAP 超时率 > 10%
- 队列深度 > 1000
- API 错误率 > 5%

完整配置参考

所有配置参数

本节提供所有操作模式下可用配置参数的完整参考。

日志配置 (`:logger`)

```
config :logger, :default_formatter,
  format: {OmniLogger.JsonFormatter, :format},
  metadata: :all,
  truncate: :infinity

config :logger,
  level: :debug  # :debug | :info | :warning | :error
```

□□	□□	□□	□□
`:logger.level`	□□	□	`:debug`
`:logger.default_formatter.format`	□□	□	`{OmniLogger.JsonFormatt` `:format}`
`:logger.default_formatter.metadata`	□□/□□	□	`:all`
`:logger.default_formatter.truncate`	□□/□□	□	`:infinity`

Web UI □□ (`:control_panel`)

```
config :control_panel, ControlPanelWeb.Endpoint,
  force_ssl: [rewrite_on: [:x_forwarded_proto]],
  server: true,
  url: [host: "0.0.0.0", path: "/"],
  https: [
    port: 8087,
    keyfile: "priv/cert/omnitouch.pem",
    certfile: "priv/cert/omnitouch.crt"
  ]

config :control_panel,
  page_order: [
    "/application", "/configuration", "/events", "/logs",
"/client",
    "/m3ua", "/routing", "/routing_test", "/smsc_links",
"/smsc_subscribers",
    "/hlr_links", "/subscribers", "/camel_sessions",
"/camel_request",
    "/msisdn_imsi_test"
  ]
```

□□	□□	□□	□□
`ControlPanelWeb.Endpoint.url.host`	□□□□	□	`"0.0.0.0"`
`ControlPanelWeb.Endpoint.https.port`	□□□	□	`8087`
`ControlPanelWeb.Endpoint.https.keyfile`	□□□□	□	`"priv/cert/omnitouc`
`ControlPanelWeb.Endpoint.https.certfile`	□□□□	□	`"priv/cert/omnitouc`
`ControlPanelWeb.Endpoint.force_ssl`	□□□□	□	`[rewrite_on: [:x_forwarded_proto`
`page_order`	□□□□□□	□	□□□

SCTP SocketHandler □□ (`:omniss7`)

```
config :omniss7,
  sctp_handler: %{
    enabled: false,
    local_ip: {127, 0, 0, 1},
    local_port: 2905
  },
  enable_gt_routing: true,
  m3ua_peers: [...],
  m3ua_routes: [...],
  m3ua_gt_routes: [...]
```

□□	□□	□□	□□	□□
`sctp_handler.enabled`	□□	□	`false`	□□□□□ STP □□
`sctp_handler.local_ip`	□□	□	`{127, 0, 0, 1}`	□□□□ M3UA □ IP
`sctp_handler.local_port`	□□	□	`2905`	M3UA □ SCTP □□
`enable_gt_routing`	□□	□	`false`	□□□□□□□□□

M3UA □□□□□□

参数	类型	必需	描述
`peer_id`	整数	是	唯一的对等体标识符
`name`	字符串	是	描述性对等体名称
`role`	原子	是	`:client` 或 `:server`
`local_ip`	元组或列表	如果是 `:client`	本地 IP（单个：`{10, 0, 0, 1}` 或列表：`[{10, 0, 0, 1}, {10, 0, 0, 2}]`）
`local_port`	整数	如果是 `:client`	本地端口（0 表示动态）
`remote_ip`	元组或列表	是	远程对等体 IP（单个：`{10, 0, 0, 10}` 或列表：`[{10, 0, 0, 10}, {10, 0, 0, 11}]`）
`remote_port`	整数	如果是 `:client`	远程对等体端口
`routing_context`	整数	是	M3UA 路由上下文
`point_code`	整数	是	SS7 点码
`network_indicator`	原子	否	`:international` 或 `:national`

M3UA 路由参数:

参数	类型	必需	描述
`dest_pc`	整数	是	目标点码
`peer_id`	整数	是	通过其路由的对等体
`priority`	整数	是	路由优先级（数字越小优先级越高）
`network_indicator`	原子	否	`:international` 或 `:national`

M3UA GT 路由参数：

参数	类型	必需	描述
`gt_prefix`	字符串	是	要匹配的全球标题前缀
`peer_id`	整数	是	目标对等体
`priority`	整数	是	路由优先级
`description`	字符串	否	日志记录的路由描述
`source_ssn`	整数	否	仅在源 SSN 匹配时匹配
`dest_ssn`	整数	否	将目标 SSN 重写为此值

MAP 客户端配置 (`:omniss7`)

```
config :omniss7,
  map_client_enabled: false,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},
    remote_port: 2905,
    routing_context: 1
  }
```

□□	□□	□□	□□	□□
`map_client_enabled`	□□	□	`false`	□□ MAP □□□□□
`map_client_m3ua.mode`	□□□	□	`"ASP"`	M3UA □□□□ □`"ASP"` □ `"SGP"`□
`map_client_m3ua.callback`	□□	□	`{MapClient, :handle_payload, []}`	□□□□□□ □□
`map_client_m3ua.process_name`	□□	□	`:map_client_asp`	□□□□□□ □
`map_client_m3ua.local_ip`	□□	□	-	□□ IP □□
`map_client_m3ua.local_port`	□□	□	`2905`	□□ SCTP □□
`map_client_m3ua.remote_ip`	□□	□	-	□□ STP/SGP IP
`map_client_m3ua.remote_port`	□□	□	`2905`	□□ SCTP □□
`map_client_m3ua.routing_context`	□□	□	-	M3UA □ □□□□

□□□□□□ (`:omniss7`)

```
config :omniss7,
  auto_flush_enabled: false,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: nil,
  auto_flush_tps: 10
```

参数	类型	必需	默认	描述
`auto_flush_enabled`	布尔	否	`false`	启用短信队列的自动刷新
`auto_flush_interval`	整数	否	`10000`	队列轮询间隔（毫秒）
`auto_flush_dest_smsc`	字符串/nil	否	`nil`	按目标 SMSC 过滤（nil = 所有）
`auto_flush_tps`	整数	否	`10`	每秒最大事务数

HTTP API 配置（`:omniss7`）

短信后端现在使用 HTTP API 而不是直接的数据库连接。

```
config :omniss7,
  smsc_api_base_url: "https://omnimessage.example.com:8443",
  smsc_name: "omni-smsc01"   # 可选：默认为主机名派生名称
```

API 参数：

参数	类型	必需	默认	
`smsc_api_base_url`	字符串	是	`"https://omnimessage.example.com:8443"`	
`smsc_name`	字符串	否	主机名派生	

使用的 **API** 端点：

- `POST /api/frontends` - 将此前端实例注册到后端
- `POST /api/messages_raw` - 插入新的 SMS 消息
- `GET /api/messages` - 检索消息队列（带 `smsc` 头）
- `PATCH /api/messages/{id}` - 将消息标记为已交付
- `PUT /api/messages/{id}` - 更新消息状态
- `POST /api/events` - 添加事件跟踪
- `GET /api/status` - 健康检查端点

前端注册：

系统在启动时会自动使用后端 API 注册自己，并大约每 85 秒重新注册。注册包括���

- 前端名称和类型（报告为 `SS7`）
- 主机名
- 运行时间（秒）
- 配置细节（JSON 格式）

配置说明：

- 默认情况下，禁用自签名证书的 SSL 验证
- HTTP 请求在 5 秒后超时

- 所有时间戳均为 ISO 8601 格式
- API 使用 JSON 作为请求/响应主体

相关文档

- ← 回主文档
- STP 指南
- MAP 客户端指南
- 短信中心指南
- HLR 指南

OmniSS7 由 Omnitouch 网络服务提供

配置参考

← 回主文档

本文档提供了所有 OmniSS7 配置参数的全面参考。

目录

1. 概述
2. 配置位置
3. 操作模式标志
4. HLR 模式参数
5. SMSc 模式参数
6. STP 模式参数
7. CAMEL 网关模式参数
8. 全局标题 NAT 参数
9. M3UA 连接参数
10. SCCP 子系统状态
11. 网关筛选
12. USSD 网关参数
13. 运行时 / 监控标志
14. 基础设施参数
15. 数据库参数
16. 硬编码值

概述

OmniSS7 支持四种操作模式，完全通过配置选择：

- **STP** 模式 - 信号传输点用于路由
- **HLR** 模式 - 家庭位置寄存器用于用户管理

- **SMSc** 模式 - 短信中心用于消息传递
- **CAMEL GW** 模式 - CAMEL 网关用于智能呼叫控制

单个节点可以通过启用相关 操作模式标志 来组合角色（例如 HLR + SMSc）。

配置位置

OmniSS7 从两个层次读取其配置。了解哪个文件拥有一个值决定了您在已部署系统上更改它的位置。

文件	角色	部署时编辑?
`config/config.exs`	默认 / 编译时配置 随发布一起提供。定义控制面板页面、`:api_ex` REST 控制器、日志格式化程序和所有基线默认值。	否：构建的一部分。
`config/runtime.exs` (在仓库中)	仅为 **4** 行测试存根。在源树中，它没有设置超出测试环境的内容。	否。
`config/runtime.exs` (已部署)	每个部署的运行时配置，单独部署到每个服务器（通常从 Ansible/Jinja 模板呈现）。这是为给定站点设置模式标志、GT、点代码、M3UA 对等体和 API 后端的地方。	是的。这是面向操作员的文件。

> 简而言之：本文档中的示例使用 `config :omniss7, ...`。在实时节点中，这些内容放入 已部署的 `config/runtime.exs`。提交到仓库的 `config/runtime.exs` 故意是一个存根，仅供自动化测试套件使用。不要将其误认为生产配置。

以下所有键在运行时通过 `Application.get_env(:omniss7, ...)`（或相关应用程序）读取，因此所示的默认值是在省略键时使用的值。

操作模式标志

控制启用哪些功能。

参数	类型	默认	描述	模式
`map_client_enabled`	布尔	`false`	启用 MAP 客户端和 M3UA 连接	所有
`hlr_mode_enabled`	布尔	`false`	启用 HLR 特定功能	HLR
`smsc_mode_enabled`	布尔	`false`	启用 SMSc 特定功能	SMSc
`cap_client_enabled`	布尔	`false`	启用 CAMEL 操作的 CAP 客户端	CAMEL GW
`camelgw_mode_enabled`	布尔	`false`	启用 CAMEL 网关功能	CAMEL GW
`ussd_gateway_enabled`	布尔	`false`	启用 USSD 网关（HTTP/JSON 桥接）	USSD GW

示例：

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false
```

HLR 模式参数

HLR（家庭位置寄存器）模式的配置。

HLR API 配置

参数	类型	默认	必需	描述
`hlr_api_base_url`	字符串	-	是	后端 HLR API 端[
`hlr_api_status_path`	字符串	`"/api/status"`	否	附加到 `hlr_api_base` 的路径，用于 HLR [状态探测（由 **HLR** [和 `/hlr-links` REST 控制器使用）
`hlr_service_center_gt_address`	字符串	-	是	在 UpdateLoca 响应中回的 HLR [址
`smsc_service_center_gt_address`	字符串	-	是	在 SRI-for-SM [的 SMSC GT 地[

> **SSL** 验证：没有 `hlr_api_verify_ssl` 键。它在代码中没有被读取。后端 API 调用的 TLS 证书验证由通用 `:api` 映射的 `verify_ssl` 字段控制（见 API 后端 SSL 验证）。

示例：

```
config :omniss7,
  hlr_api_base_url: "https://omnihss.example.com:8443",
  hlr_api_status_path: "/api/status",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112"
```

AlertServiceCenter □□

□□□□□ UpdateLocation □□HLR □□□□ SMSc GT □□ alertServiceCenter □□□□□□□□□□ □□□□□

□□	□□	□□	□□	□□
`hlr_smsc_alert_gts`	□□□□	`[]`	□	□□ alertServiceCer □□□ SMSc □□□□
`hlr_alert_location_expiry_seconds`	□□	`172800`	□	□□□□□□□□□□□□4 □□□□□□□ 0 □□□□

□□□

```
config :omniss7,
  hlr_smsc_alert_gts: [
    "15559876543",
    "15559876544"
  ],
  hlr_alert_location_expiry_seconds: 172800  # 48 □□
```

MSISDN ↔ IMSI □□

□ MSISDN □□□□ IMSI □□□□□□□□□□□□□□□□□□□□□□ HLR □□□□ MSISDN ↔ IMSI □□□

参数	类型	默认	必需	描述
`hlr_imsi_plmn_prefix`	字符串	`"50557"`	否	合成 IMSI 生成的 PLMN 前缀（MCC+MNC）
`hlr_msisdn_country_code`	字符串	`"61"`	否	IMSI→MSISDN 反向映射的国家代码前缀
`hlr_msisdn_nsn_offset`	整数	`0`	否	MSISDN 中 NSN 开始的偏移量（通常是国家代码的长度）
`hlr_msisdn_nsn_length`	整数	`9`	否	从 MSISDN 中提取的国家用户号码的长度

示例（2 位国家代码）：

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",      # MCC 505 + MNC 57
  hlr_msisdn_country_code: "99",      # 示例 2 位国[illegible]代码
  hlr_msisdn_nsn_offset: 2,           # 跳过 2 位国家代码
  hlr_msisdn_nsn_length: 9            # 提取 9 位 NSN
```

示例（3 位国家代码）：

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",      # MCC 505 + MNC 57
  hlr_msisdn_country_code: "999",     # 示例 3 位国家代码
  hlr_msisdn_nsn_offset: 3,           # 跳过 3 位国家代码
  hlr_msisdn_nsn_length: 8            # 提取 8 位 NSN
```

重要：将 `nsn_offset` 设置为您的国家代码的长度以正确提取 NSN。例如：

- 国家代码 "9"（1 位）→ `nsn_offset: 1`
- 国家代码 "99"（2 位）→ `nsn_offset: 2`

- 国家代码 "999"（3 位） → `nsn_offset: 3`

InsertSubscriberData (ISD) 配置

在 UpdateLocation 期间发送到 VLR 的用户配置数据的配置。有关 ISD 序列和消息流的详细说明，请参见 HLR 指南中的 InsertSubscriberData 配置。

参数	类型	默认	必需	
`isd_network_access_mode`	原子	`:packetAndCircuit`	否	网络接入类型: `:packetAndCi` 或 `:circuitOnly`
`isd_send_ss_data`	布尔	`true`	否	发送带有补充服务数据的 ISD
`isd_send_call_barring`	布尔	`true`	否	发送带有呼叫限制数据的 ISD

示例:

```
config :omniss7,
  isd_network_access_mode: :packetAndCircuit,
  isd_send_ss_data: true,
  isd_send_call_barring: true
```

CAMEL 配置

用于基于 CAMEL 的智能呼叫路由的配置。有关 CAMEL 集成和服务密钥的详细说明，请参见 HLR 指南中的 CAMEL 集成。

参数	类型	默认	必需	
`camel_service_key`	整数	`11_110`	否	用于 S
`camel_trigger_detection_point`	原子	`:termAttemptAuthorized`	否	CAM 点: `:`
`camel_gsmscf_gt_address`	字符串	(使用被叫 GT)	否	CAM

示例:

```
config :omniss7,
  camel_service_key: 11_110,
  camel_trigger_detection_point: :termAttemptAuthorized
```

家庭 VLR 前缀

用于区分家庭用户与漫游用户的配置。有关家庭/漫游检测和 PRN 操作的详细说明，请参见 HLR 指南中的漫游用户处理。

参数	类型	默认	必需	描述
`home_vlr_prefixes`	列表	`["5551231"]`	否	被视为“家庭”网络的 VLR GT 前缀

示例:

```
config :omniss7,
  home_vlr_prefixes: ["5551231", "5551234"]
```

SMSc □□□□

□□□□□□□□□□

SMSc API □□

□□	□□	□□
`smsc_api_base_url`	□□□	`"https://omnimessage.example.com`
`smsc_api_status_path`	□□□	`"/api/status"`
`smsc_name`	□□□	`"{hostname}_SMSc"`
`smsc_service_center_gt_address`	□□□	-

> **SSL** □□□ □□ `smsc_api_verify_ssl` □□□□□□□□□□□□□□□□ API □□□ TLS □□□□□□□□ `:api` □□□ `verify_ssl` □□□□□□□ API □□ SSL □□□□

□□□

```
config :omniss7,
  smsc_api_base_url: "https://omnimessage.example.com:8443",
  smsc_api_status_path: "/api/status",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112"
```

注意：前端注册每 **5** 分钟（硬编码）通过 `SMS.FrontendRegistry` 模块进行。

自动刷新配置

参数	类型	默认	必需	描述
`auto_flush_enabled`	布尔	`true`	否	启用自动 SMS 队列处理
`auto_flush_interval`	整数	`10_000`	否	队列处理间隔（毫秒）
`auto_flush_dest_smsc`	字符串	-	是	自动刷新的目标 SMSC 名称
`auto_flush_tps`	整数	`10`	否	消息处理速率（事务/秒）

示例：

```
config :omniss7,
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10
```

STP 模式参数

M3UA 信号传输点模式的配置。有关详细的路由配置和示例，请参见 STP 配置指南。

□□ STP □□□

□□	□□	□□	□□	□□
`sctp_handler.enabled`	□□	`false`	□	□□□□ SCTP SocketHandler □□□
`sctp_handler.local_ip`	□□□□□□	`{127, 0, 0, 1}`	□	□□□□□ IP □□□□□ IP□ `{10, 0, 0, 1}` □ SCTP □□□ ��□□ IP□ `[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`sctp_handler.local_port`	□□	`2905`	□	□□□□□
`sctp_handler.point_code`	□□	-	□□□□	□ STP □ SS7 □□□

□□□□□ IP□□

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  }
```

□□□SCTP □□□□□

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    # 多个 IP 用于冗余
    local_ip: [{10, 179, 4, 10}, {10, 179, 4, 11}],
    local_port: 2905,
    point_code: 100
  }
```

> 注意：有关 SCTP 多宿主配置和好处的详细信息，请参见 常见指南中的 SCTP 多宿主。

全局标题路由

参数	类型	默认	必需	描述
`enable_gt_routing`	布尔	`false`	否	除 PC 路由外启用 GT 路由

示例：

```
config :omniss7,
  enable_gt_routing: true
```

M3UA/M2PA 对等体配置

对等体通过 `m3ua_peers` 列表配置（支持 M3UA 和 M2PA 协议）。有关完整的配置示例，请参见 STP 配置指南 和 M2PA 协议支持。

通用对等体参数：

□□	□□	□□	□□	□□
`peer_id`	□□	-	□	□□□□□□□□□□
`name`	□□□	-	□	□□□□□□□□□
`protocol`	□□	`:m3ua`	□	□□□□□`:m3ua` □ `:m2pa`
`role`	□□	`:client`	□	□□□ □□`:client`□`:server`□`:a □ `:sgp`
`local_ip`	□□□□□	-	□	□□□□□□□□ IP □□
`local_port`	□□	-	□	□□ SCTP □□□M3UA□2905□ M2PA□3565□
`remote_ip`	□□□□□	-	□	□□ IP □□
`remote_port`	□□	-	□	□□ SCTP □□
`routing_context`	□□	-	□	M3UA □□□□□□□□□□□ M3UA

参数	类型	默认	必需	描述
`point_code`	整数	-	是	本地点代码
`network_indicator`	原子	`:international`	否	网络指示符：`:international` [`:national`
`initiate_connection`	布尔	`true`	否	是否发起 SCTP 连接

M2PA 特定参数：

参数	类型	默认	必需	描述
`adjacent_point_code`	整数	-	是（M2PA）	邻近对等体的点代码

套接字管理：

M2PA 自动使用 SCTP.SocketHandler 进行共享套接字管理。所有 M2PA 对等体使用共享套接字，这允许多个对等体有效地共享同一 SCTP 端口。有关详细配置，请参见 M2PA 套接字要求。

示例（**M3UA** 对等体）：

```
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "HLR_East",
      protocol: :m3ua,
      role: :sgp,
      local_ip: {10, 179, 4, 10},
      local_port: 2905,
      remote_ip: {10, 179, 4, 20},
      remote_port: 2905,
      point_code: 100,
      network_indicator: :international
    }
  ]
```

□□□M2PA □□□□□

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 3565,
    point_code: 100
  },
  m3ua_peers: [
    %{
      peer_id: 2,
      name: "M2PA_Link_STP_West",
      protocol: :m2pa,
      role: :client,
      local_ip: {10, 179, 4, 10},
      local_port: 3565,
      remote_ip: {10, 179, 4, 30},
      remote_port: 3565,
      point_code: 100,
      adjacent_point_code: 200
    }
  ]
```

M3UA 点代码路由

点代码路由配置。路由定义用于到达特定目标点代码的对等体。

参数 类型 默认 必需 描述

m3ua_routes 映射列表 [] 否 点代码路由列表。如果未指定，则根据对等体点代码自动生成路由。

路由格式：每个 m3ua_routes 中的路由必须是一个映射，包含：

dest_pc：目标点代码（整数）

peer_id：通过的对等体 ID（整数）

priority：路由优先级 - 较低值 = 较高优先级（整数）

network_indicator：网络指示符（原子）：:international 或:national

示例：

config:omniss7,
m3ua_routes: [
通过对等体 1 路由到 PC 100（最高优先级）
%{dest_pc: 100, peer_id: 1, priority: 1, network_indicator: :international},
通过对等体 2 路由到 PC 200
%{dest_pc: 200, peer_id: 2, priority: 1, network_indicator: :international},
负载均衡：相同的 dest_pc 具有不同的优先级
%{dest_pc: 300, peer_id: 3, priority: 1, network_indicator: :international},
%{dest_pc: 300, peer_id: 4, priority: 2, network_indicator: :international}
]

M3UA 全局标题路由

基于全局标题前缀的路由，具有高级 SCCP 参数转换。使用最长前缀匹配优先。

参数	类型	默认	必需	描述
`m3ua_gt_routes`	映射列表	`[]`	???	带有可选转换的全局标题路由规则列表

路由格式：每个 `m3ua_gt_routes` 中的路由必须是一个映射，包含：

基本参数：

- `gt_prefix`：要匹配的全局标题前缀（字符串） - 空字符串匹配所有
- `peer_id`：通过的对等体 ID（整数） - 使用 0 来丢弃流量
- `priority`：路由优先级 - 较低值 = 较高优先级（整数）
- `description`：人类可读的描述（字符串）

可选匹配参数：

- `source_ssn`：匹配源子系统编号（整数）
- `source_tt`：匹配源转换类型（整数）
- `source_npi`：匹配源编号计划指示符（整数）
- `source_nai`：匹配源地址性质指示符（整数）

可选转换参数：

- `dest_ssn`：在转发消息中转换 SSN（整数）
- `dest_tt`：在转发消息中转换转换类型（整数）
- `dest_npi`：在转发消息中转换编号计划指示符（整数）
- `dest_nai`：在转发消息中转换地址性质指示符（整数）
- `route_on_ssn`：翻转被叫方路由指示符以路由到 SSN（布尔，默认 `false`）。对于中转 GT 路由，保持 `false` 以便下游节点执行 GTT（ITU-T Q.714 §2.4）；仅在此 STP 是最终翻译器并交付给仅 SSN 端点时设置为 `true`。请参见 STP 指南 → GTT 后的路由到 SSN。

常见值：

- 转换类型 **(TT)**：0=未知，1=国际，2=国家，3=网络特定
- 编号计划 **(NPI)**：0=未知，1=ISDN(E.164)，6=移动(E.212)
- 地址性质 **(NAI)**：0=未知，1=用户，3=国家，4=国际
- 子系统编号 **(SSN)**：6=HLR，7=VLR，8=MSC，9=EIR，等。

□□□

```
config :omniss7,
  m3ua_gt_routes: [
    # □□□□□□□
    %{gt_prefix: "1234", peer_id: 1, priority: 1, description: "□□
□□"},
    %{gt_prefix: "44", peer_id: 2, priority: 1, description: "□□□
□"},

    # □□□□□□□
    %{
      gt_prefix: "61",
      peer_id: 3,
      priority: 1,
      description: "□□□□□□□□TT 0→1 □□",
      source_tt: 0,  # □□ TT=0□□□□
      dest_tt: 1     # □□□ TT=1□□□□
    },

    # NPI □□
    %{
      gt_prefix: "49",
      peer_id: 1,
      priority: 1,
      description: "□□□□□□□□→ISDN NPI □□",
      source_npi: 6,  # □□ NPI=6□□□/E.212□
      dest_npi: 1     # □□□ NPI=1□ISDN/E.164□
    },

    # □□□□□□ SSN □□
    %{
      gt_prefix: "86",
      source_ssn: 8,   # □□ SSN=8□MSC□
      peer_id: 3,
      dest_ssn: 6,     # □□□ SSN=6□HLR□
      priority: 1,
      description: "□□□□□□□□□□□□□□",
      source_tt: 0,
      dest_tt: 2,
      source_npi: 6,
      dest_npi: 1,
      source_nai: 4,
      dest_nai: 3
    },
```

```
    # 默认/回退路由
    %{
      gt_prefix: "",
      peer_id: 1,
      priority: 99,
      description: "默认回退路由"
    }
  ]
```

CAMEL 网关模式参数

CAMEL 网关（CAP 协议）模式的配置。

CAMEL 模式标志

启用 CAMEL/CAP 功能（在 操作模式标志 中设置 `cap_client_enabled: true` 和 `camelgw_mode_enabled: true`）。

CAP 协议配置

参数	类型	默认	必需	描述
`cap_version`	原子	`:v2`	否	CAP 协议版本: `:v1`、`:v2`、`:v3` 或 `:v4`
`camel_gsmscf_gt_address`	字符串	(使用被叫 GT)	否	CAMEL 响应的默认 gsmSCF 全局标题

CAP 版本映射:

- `:v1` → 应用上下文 OID: 0.4.0.0.1.0.50.0
- `:v2` → 应用上下文 OID: 0.4.0.0.1.0.50.1（默认 - 最广泛支持）
- `:v3` → 应用上下文 OID: 0.4.0.0.1.21.3.4

- `:v4` → □□□□□ OID: 0.4.0.0.1.23.3.4

□□□ □□□□□□□□ OID □□□□□□□□□□□□□□□□□□□□□□□□

□□□

```
config :omniss7,
  cap_client_enabled: true,
  camelgw_mode_enabled: true,
  cap_version: :v2,
  camel_gsmscf_gt_address: "99988411553"
```

CGrateS □□

□ CGrateS □□□□□□□□□□□□□□/□□□□□□

□□	□□	□□	□□	□□
`cgrates_enabled`	□□	`false`	□	□□ CGrateS □□
`cgrates_url`	□□□	-	□□□□□	CGrateS JSON-RPC □□ U
`cgrates_tenant`	□□□	`"cgrates.org"`	□	CGrateS □□□□□
`cgrates_request_type`	□□□	`"*prepaid"`	□	□□□ □□`"*prepaid"`□`"*postp` □ `"*pseudoprepaid"`
`cgrates_timeout`	□□	`5000`	□	CGrateS □□□□□□□□

□□□

```
config :omniss7,
  cgrates_enabled: true,
  cgrates_url: "http://localhost:2080/jsonrpc",
  cgrates_tenant: "cgrates.org",
  cgrates_request_type: "*prepaid",
  cgrates_timeout: 5000
```

CAP M3UA □□

CAMEL □□□□□□□ M3UA □□□□ CAP □□□

□□	□□	□□	□□	□□
`cap_client_m3ua`	□□	-	□	CAP M3UA □□□□□□ `map_client_m3ua` □□□ □□□□□□□□ `opc` □ `dpc` □□□

□□□

```
config :omniss7,
  cap_client_m3ua: %{
    mode: "ASP",
    callback: {CapClient, :handle_payload, []},
    process_name: :camelgw_client_asp,
    local_ip: {10, 5, 198, 200},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 4,
    opc: 5013,    # □□□□□
    dpc: 5011     # □□□□□
  }
```

全局标题 NAT 参数

全局标题网络地址转换允许根据呼叫方前缀、被叫方前缀或两者的不同响应 GT。规则按权重匹配（较低 = 较高优先级），然后按前缀特异性（较长的组合前缀 = 更特异）。有关详细说明和示例，请参见 全局标题 NAT 指南。

参数	类型	默认	必需	描述
`gt_nat_enabled`	布尔	`false`	否	启用/禁用 GT NAT 功能
`gt_nat_rules`	映射列表	`[]`	是（如果启用）	带有前缀匹配的 GT NAT 规则列表

规则格式：每个 `gt_nat_rules` 中的规则必须是一个映射，包含：

- `calling_prefix`：要与呼叫 GT 匹配的字符串前缀（可选）
- `called_prefix`：要与被叫 GT 匹配的字符串前缀（可选）
- `weight`：整数优先级（较低 = 较高优先级） - 默认：100
- `response_gt`：在响应中使用的全局标题（必需）

匹配优先级：

1. 规则按 `weight` 匹配（较低值 = 较高优先级）
2. 如果权重相等，则较长的组合前缀长度获胜
3. `calling_prefix` 和 `called_prefix` 可以一起使用以进行精确匹配

示例：

```
config :omniss7,
  gt_nat_enabled: true,
  gt_nat_rules: [
    # 高优先级：同时匹配呼叫来自 "8772" 和被叫到 "555"
    %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

    # 中等优先级：仅匹配呼叫来自 "8772"
    %{calling_prefix: "8772", weight: 10, response_gt:
"99988411553"},

    # 中等优先级：仅匹配被叫到 "555"
    %{called_prefix: "555", weight: 10, response_gt:
"99988411554"},

    # 仅匹配呼叫来自 "8773"
    %{calling_prefix: "8773", weight: 10, response_gt:
"99988411554"},

    # 通配符回退规则（匹配所有，最高权重）
    %{weight: 100, response_gt: "99988411555"}
  ]
```

另见：有关详细用法和示例，请参见 GT NAT 指南。

M3UA 连接参数

MAP 客户端模式的 M3UA 连接配置。有关详细用法和示例，请参见 MAP 客户端指南。

□□	□□	□□	□□	□
`map_client_m3ua.mode`	□□□	-	□	□□□□□ `"ASP"`
`map_client_m3ua.callback`	□□	-	□	□□□□/□□□ `{Ma` `:handle_pay`
`map_client_m3ua.process_name`	□□	-	□	□□□□□□□
`map_client_m3ua.local_ip`	□□□□□□	-	□	□□□□□□ IP □□ `0, 0, 1}` □□ □□□ `[{10, 0,` `{10, 0, 0,`
`map_client_m3ua.local_port`	□□	`2905`	□	□□ SCTP □□
`map_client_m3ua.remote_ip`	□□□□□□	-	□	□□ STP/SGW `{10, 0, 0,` `[{10, 0, 0,` `0, 0, 11}]`
`map_client_m3ua.remote_port`	□□	`2905`	□	□□ SCTP □□
`map_client_m3ua.routing_context`	□□	-	□	M3UA □□□□□
`map_client_m3ua.opc`	□□	`5013`	□	□□□□□ MAP □ □□□□□□□□□□□□□

参数	类型	默认	必	描
`map_client_m3ua.dpc`	整数	`5011`	否	用于发出的 MAP [果未设置，默认为遗留[
`map_client_m3ua.receive_watchdog`	布尔	`true`	否	启用 ASP 接收看门 时，如果在 `receive_wat` 秒内未接收到 SCTP 除。设置为 `false` 接收看门狗 下文。
`map_client_m3ua.receive_watchdog_idle`	整数	`15`	否	空闲阈值（秒）。如果[SCTP 有效负载，[触发重新连接。仅在 `receive_wat` `true` 时适用。

示例（单个 **IP**）：

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1,
    opc: 5013,    # 原始点代码 (2-114-5)
    dpc: 5011     # 目标点代码 (2-114-3)
  }
```

> 点代码格式：`X-Y-Z` 格式的点代码转换为整数为：`(X * 2048) + (Y * 8) + Z`。例如，`2-114-5` = `(2 * 2048) + (114 * 8) + 5` = `5013`。

示例（**SCTP** 多宿主）：

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # 多个本地 IP 用于冗余
    local_ip: [{10, 179, 4, 11}, {10, 179, 4, 12}],
    local_port: 2905,
    # 多个远程 IP 用于 STP 冗余
    remote_ip: [{10, 179, 4, 10}, {10, 179, 4, 20}],
    remote_port: 2905,
    routing_context: 1
  }
```

> 注意：有关 SCTP 多宿主配置和好处的详细信息，请参见 常见指南中的 SCTP 多宿主。

接收看门狗

接收看门狗 监控 SCTP 连接以防止僵尸套接字。僵尸套接字是一个在操作系统级别保持 ESTABLISHED 状态的关联，但远程端已静默停止发送数据。如果没有看门狗，节点可能不会在下一个发送尝试失败之前检测到死连接。

任何接收到的 SCTP 有效负载都会重置空闲计时器，包括 M3UA BEAT 响应、NOTIFY 消息和应用数据。根据 RFC 4666 §3.8, M3UA BEAT 是可选的。SCTP 已经在传输层执行强制心跳。当远程 SG 不发送定期的应用层流量（并且不发送 M3UA BEAT 时），请禁用看门狗以避免不必要的重新连接周期。

参数	类型	默认	描述
`receive_watchdog`	布尔	`true`	启用或禁用接收看门狗。当 `false` 时，空闲连接不会被看门狗拆除；SCTP 传输层心跳仍然正常运行。
`receive_watchdog_idle`	整数	`15`	在看门狗拆除连接[illegible]触发重新连接之前的非活动秒数。当 `receive_watchdog: false` 时没有效果。

示例：禁用 **ASP** 客户端的看门狗：

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1,
    receive_watchdog: false        # □□ - □□ SG □□□□□□□□
  }
```

□□□□□□□□□□□□

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1,
    receive_watchdog: true,
    receive_watchdog_idle: 30      # □ 30 □□□□□□□□□□□□□15 □□
  }
```

SCCP □□□□□

OmniSS7 □□□□□□□□□□□□□□□□□□□□ SCCP □□□□□□□□□□□SST□□ITU-T Q.714 §5.3.4□□ `local_subsystems` □□□□□□□□□□□□□□□□□□□□SSN□□□□□ SSN □ SST □□□□ `Subsystem-Allowed` (SSA)□□□□□ SSN □□□□ `Subsystem-Prohibited` (SSP)□

参数	类型	默认	必需	描述
`local_subsystems`	整数列表或 `:all`	`:all`	???	用于回答 SST 的在服务本地 SCCP SSN。`:all`（默认）将每个子系统视为可用。这适合没有托管任何子系统的纯中继。终止特定 SSN 的操作员必须列出它们，以便 SST 反映现实。

常见 **SSN**：6 = HLR, 7 = VLR, 8 = MSC, 9 = EIR, 146 = gsmSCF/CAMEL。

示例：

```
config :omniss7,
  local_subsystems: [6, 7, 8, 146]
```

网关筛选

网关筛选（SCCP 防火墙）在路由之前允许或拒绝中转流量。规则根据源对等体、OPC、DPC、服务指示符、SCCP 被叫/主叫 GTA 前缀和 SSN 匹配；第一个匹配的启用规则（最低优先级）决定 `allow`/`deny`。当没有规则匹配时，下面的默认操作适用。被拒绝的消息将被丢弃，并计入 `m3ua_stp_routing_failures_total{reason="screened"}`。有关完整模型，请参见 STP 指南。

参数	类型	默认	必需	描述
`screening_default_action`	`:allow` 或 `:deny`	`:allow`	否	当没有筛选规则匹配时应用于中转消息的操作。`:allow`（默认）是默认允许姿态；`:deny` 是默认拒绝（允许列表）。设置为 `:deny` 即使没有配置规则也会使筛选生效。

示例：

```
config :omniss7,
  # 默认拒绝：仅转发匹配显式允许规则的流量。
  screening_default_action: :deny
```

规则本身保存在 `ss7_screen_rule` Mnesia 表中并以编程方式管理；仅通过配置设置默认操作。

USSD 网关参数

USSD 网关将网络发起和移动发起的 USSD 桥接到 HTTP/JSON 后端。通过 `ussd_gateway_enabled: true` 启用它（见 操作模式标志），然后配置 `ussd_gateway` 映射。有关完整的回调协议和会话生命周期，请参见 USSD 网关指南。

参数	类型	默认	必需	描述
`ussd_gateway.routes`	映射列表	`[]`	是（如果启用）	短代码 → 回调 URL 路由规则。每个条目为 `%{pattern: "<prefix>", url: "<callback URL>"}`。最长匹配前缀获胜；使用 `"*"` 风格的短模式作为捕获所有回退。
`ussd_gateway.session_timeout_ms`	整数	`180_000`	否	最大总会话生命周期（毫秒）（3 分钟）。如果超过，则会话终止。
`ussd_gateway.turn_timeout_ms`	整数	`30_000`	否	在多轮对话中等待用户回复的最大时间（毫秒）。
`ussd_gateway.http_timeout_ms`	整数	`5_000`	否	对回调到您的应用程序 `` HTTP 请求/连接超时（毫秒）。
`ussd_gateway.max_text_length`	整数	`182`	否	USSD 文本字符串的最大字符数（GSM 7 位）。较长的文本将被截断并记录警告。

路由：当 USSD 代码到达时，网关按降序 `pattern` 长度对配置的路由进行排序。然后，网关选择第一个 `pattern` 是拨打代码前缀的路由。这提供了确定性的最长前缀匹配。

Parse error on line 3: ...配路由} B -->|模式 \"*100\"| C["POST 到其 U ----------------------^ Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND', 'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND', 'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'STR'

□□

□□□

```
config :omniss7,
  ussd_gateway_enabled: true,
  ussd_gateway: %{
    routes: [
      %{pattern: "*100", url: "http://balance-app:9000/ussd"},
      %{pattern: "*200", url: "http://topup-app:9000/ussd"},
      %{pattern: "*",    url: "http://default-app:9000/ussd"}
    ],
    session_timeout_ms: 180_000,
    turn_timeout_ms: 30_000,
    http_timeout_ms: 5_000,
    max_text_length: 182
  }
```

□□□ / □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ OmniSS7□□□□□□□□□□□□□□□□□□□□ Web □□□□□□□□

参数	类型	默认	必需	描述
`start_web_interfaces`	布尔	`true`	否	启动控制面板 UI、`:api_ex` REST 服务器和 SS7 事件记录器。当 `false` 时，仅启动数据包记录器（`SS7.Events`）。可以通过 `ENABLE_WEB_INTERFACES` 环境变量（`true`/`1` 或 `false`/`0`）在启动时覆盖。
`start_http_server`	布尔	`true`	否	启动 Plug.Cowboy HTTP 服务器，端口为 `api_port`（REST `/api/*`、`/metrics`、`/swagger`）。见 REST API 服务器。
`api_port`	整数	`9090`	否	Plug.Cowboy HTTP 服务器的 TCP 端口（REST `/api/*`、`/metrics`、`/swagger`）。如果更改，请更新 Prometheus 抓取目标。
`headless`	布尔	`false`	否	仅库最小监督。当 `true` 时，OmniSS7 仅启动编解码器/事务监督者（MAP、CAP、USSD 注册和动态监督者）：没有 Web UI，没有许可证客户端，没有 SCTP 处理程序，没有 M3UA 服务器。旨在将 OmniSS7 编解码器作为依赖项嵌入。
`vlr_cache_ttl_seconds`	整数	`3600`	否	在 SMSc 交付路径中缓存 VLR 查找的生存时间（秒）。

示例（无头库模式）：

```
config :omniss7,
  headless: true
```

示例（无 **Web** 层，仅 **REST API**）：

```
config :omniss7,
  start_web_interfaces: false,
  start_http_server: true
```

基础设施参数

系统基础设施组件的配置，包括许可、Web 界面、API 服务器和日志记录。

许可证配置

参数	类型	默认	必需	描述
`license_client.license_server_api_urls`	字符串列表	-	是	许可证服务器 API 端点 URL
`license_client.licensee`	字符串	-	是	许可证持有者组织名称

示例：

```
config :license_client,
  license_server_api_urls: ["https://localhost:10443/api"],
  licensee: "Omnitouch Network Services Pty. Ltd."
```

□□□□ Web □□

□□	□□	□□
`control_panel.parent_application_readable_name`	□□□	`"OmniSS7 Stack"`
`control_panel.use_additional_pages`	□□□□□	`[]`
`control_panel.page_order`	□□□□□□	`[]`
`ControlPanelWeb.Endpoint.url.host`	□□□	`"0.0.0.0"`
`ControlPanelWeb.Endpoint.https.port`	□□	`8087`
`ControlPanelWeb.Endpoint.https.keyfile`	□□□	`"priv/cert/omnit`
`ControlPanelWeb.Endpoint.https.certfile`	□□□	`"priv/cert/omnit`

□□□

```
config :control_panel, ControlPanelWeb.Endpoint,
  url: [host: "0.0.0.0", path: "/"],
  https: [
    port: 8087,
    keyfile: "priv/cert/omnitouch.pem",
    certfile: "priv/cert/omnitouch.crt"
  ],
  parent_application_readable_name: "OmniSS7 Stack STP"

config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 □□"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "□□□"}
  ],
  page_order: ["/events", "/m3ua", "/application",
"/configuration"]
```

REST API □□□

□□	□□	□□	□□	□□
`start_http_server`	□□	`true`	□	□□/□□ HTTP □ □□□□ 9090□
`api_ex.api.port`	□□	`8445`	□	API □□□ □
`api_ex.api.listen_ip`	□□□	`"0.0.0.0"`	□	API □□□ □ IP
`api_ex.api.product_name`	□□□	`"OmniSS7"`	□	API □□□ □
`api_ex.api.title`	□□□	`"API - OmniSS7"`	□	Swagg UI □ AP □□
`api_ex.api.hostname`	□□□	`"localhost"`	□	API □□□
`api_ex.api.enable_tls`	□□	`true`	□	□□ API □□□ TL
`api_ex.api.tls_cert_path`	□□□	`"priv/cert/omnitouch.crt"`	□	API □ TLS □□ □

□□	□□	□□	□□	□□
`api_ex.api.tls_key_path`	□□□	`"priv/cert/omnitouch.pem"`	□	API □ TLS □□□

□□□

```
config :omniss7,
  start_http_server: true

config :api_ex,
  api: %{
    port: 8445,
    listen_ip: "0.0.0.0",
    product_name: "OmniSS7",
    title: "API - OmniSS7",
    hostname: "localhost",
    enable_tls: true,
    tls_cert_path: "priv/cert/omnitouch.crt",
    tls_key_path: "priv/cert/omnitouch.pem"
  }
```

□□□□□□□□

□□	□□	□□□	
9090	HTTP (Plug.Cowboy)	`APIhandler`	MAP/CAP REST `/api/*`□`/metrics`□ □ `start_http_serv`
8087	HTTPS (Bandit/Phoenix)	`ControlPanelWeb.Endpoint`	Web □□□□ UI□Web □ □`start_web_interf`
8445	HTTPS / TLS	`:api_ex` REST □□□	□□□□□ CRUD□□□□□□ Web □□□□□□

API □□□

- MAP/CAP REST API + Swagger: `http://[server-ip]:9090/...`□□ API □□□
- `:api_ex` REST □□□: `https://[server-ip]:8445/api/*`
- Web □□□□: `https://[server-ip]:8087/`
- Prometheus □□: `http://[server-ip]:9090/metrics`

API □□ SSL □□

□□□ HLR/SMSc API □□□□□□□□□□□□□□ `*_verify_ssl` □□□□□□□□□ `:api` □□□□□

□□	□□	□□	□□
`api.base_url`	□□□	□□□ `smsc_api_base_url`	□□ API □□□□□□ URL□
`api.timeout`	□□	`5000`	□□□□□□□□□
`api.verify_ssl`	□□	`false`	□□□□□□ TLS □□□□□□□□ **SSL** □□□□ □□ `hlr_api_verify_ssl` □ `smsc_api_verify_ssl`□

□□□

```
config :omniss7,
  api: %{
    base_url: "https://backend.example.com:8443",
    timeout: 5000,
    verify_ssl: false
  }
```

□□□□

□□	□□	□□	□□	
`logger.level`	□□□	`:debug`	□	□□□□

数据库参数

Mnesia 数据库持久性的配置。

参数	类型	默认	必需	描述
`mnesia_storage_type`	原子	`:disc_copies`	否	Mnesia 存储类型: `:disc_copies` 或 `:ram_copies`

示例:

```
config :omniss7,
  mnesia_storage_type: :disc_copies  # 生产
  # mnesia_storage_type: :ram_copies  # 仅用于测试
```

存储类型:

- `:disc_copies` - 持久磁盘存储（重启后仍然存在） - 推荐用于生产
- `:ram_copies` - 仅在内存中（重启时丢失） - 仅用于测试

Mnesia 表:

- `m3ua_peer` - M3UA 对等连接
- `m3ua_route` - 点代码路由
- `m3ua_gt_route` - 全局标题路由

位置: `Mnesia.{node_name}/` 目录

硬编码值

以下值是 硬编码在源代码中 的，无法通过配置更改。

超时

值	影响	解决方法
MAP 请求超时: **10** 秒	所有 MAP 操作在 10 秒后超时	修改源代码
ISD 超时: **10** 秒	每个 ISD 消息在 10 秒后超时	修改源代码

HTTP 服务器

值	影响	解决方法
HTTP IP: **0.0.0.0**	指标/Swagger 服务器在所有接口上监听	修改源代码
HTTP 端口: **9090**	指标/Swagger 端点在端口 9090 上运行	设置 `config :omniss7, api_port: <port>`（默认 `9090`）

注册间隔

值	影响	解决方法
前端注册: **5** 分钟	SMSc 每 5 分钟与后端注册	修改源代码

Web UI 自动刷新

页面	间隔
路由管理	5 秒
活跃用户	2 秒

□□□□

□□ HLR □□

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  hlr_api_base_url: "https://omnihss.example.com:8443",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112",

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

□□ SMSc □□

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  smsc_api_base_url: "https://omnimessage.example.com:8443",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112",

  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

STP □□□□□□

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,

  enable_gt_routing: true,
  mnesia_storage_type: :disc_copies,

  sctp_handler: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  },

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 10},
    local_port: 2906,
    remote_ip: {10, 179, 4, 11},
    remote_port: 2905,
    routing_context: 1
  }
```

□□

□□□□□□85+

□□□□

- □□□□□6 □□□
- HLR □□□17 □□□
- SMSc □□□8 □□□

- STP 模式: 5+ 个参数（加上 m3ua_peers、m3ua_routes、m3ua_gt_routes 列表）
- CAMEL GW 模式: 14 个参数
- 全局标题 NAT: 2 个参数
- M3UA 连接: 8 个参数
- SCCP 子系统状态: 1 个参数
- USSD 网关: 5 个参数
- 运行时 / 监控: 4 个参数
- 基础设施（许可证、Web、API、日志）：23 个参数
- 数据库: 1 个参数

按模式所需参数：

HLR 模式：

- `hlr_api_base_url`
- `hlr_service_center_gt_address`
- `smsc_service_center_gt_address`
- 所有 `map_client_m3ua.*` 参数（8）

SMSc 模式：

- `smsc_api_base_url`
- `smsc_service_center_gt_address`
- `auto_flush_dest_smsc`（如果启用自动刷新）
- 所有 `map_client_m3ua.*` 参数（8）

STP 模式：

- `sctp_handler.point_code`（如果启用 SCTP 处理程序）
- `sctp_handler.local_ip`
- `sctp_handler.local_port`

CAMEL GW 模式：

- `cgrates_url`（如果启用 CGrateS��
- 所有 `cap_client_m3ua.*` 参数（8）

基础设施：

- `license_client.license_server_api_urls`
- `license_client.licensee`

相关文档

- **HLR □南** - HLR 特定配置
- **SMSc □南** - SMSc 特定配置
- **STP □南** - STP 路由配置
- **API □南** - REST API 参考
- **USSD 网关□南** - USSD 网关配置和 HTTP 回调协议
- **Web UI □南** - Web 界面文档

全球标题 NAT □南

概述

全球标题网络地址转换（GT NAT）是一个功能，允许 OmniSS7 根据呼叫方的 GT 前缀、被叫方的 GT 前缀或两者的组合来响应不同的全球标题地址。这在处理多个全球标题时至关重要，并且需要确保响应使用正确的 GT，具体取决于哪个网络或对等方在呼叫和/或他们呼叫了哪个 GT。

新功能（增强的 GT NAT）

GT NAT 功能已增强，具有强大的新能力：

新特性

1. 被叫方前缀匹配：规则现在可以匹配 `called_prefix`，除了 `calling_prefix` 之外
2. 组合匹配：规则可以同时匹配呼叫和被叫前缀
3. 基于权重的优先级：规则现在使用 `weight` 字段（值越低优先级越高），而不仅仅是前缀长度
4. 灵活匹配：您现在可以创建具有以下内容的规则：
    - 仅呼叫前缀
    - 仅被叫前缀
    - 同时使用呼叫和被叫前缀
    - 两者都不使用（通配符/回退规则）

新规则□□式

必需字段：

- `weight`：整数优先级（值越低优先级越高）
- `response_gt`：要响应的 GT

可选字段（建议至少□定一个以实现特定匹配）：

- `calling_prefix`：匹配呼叫方 GT 前缀
- `called_prefix`：匹配被叫方 GT 前缀

示例：

```
gt_nat_rules: [
  # 同时具有两个前缀的特定规则 - 最高优先级
  %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

  # 特定规则 - 中等优先级
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},
  %{called_prefix: "555", weight: 10, response_gt: "333333"},

  # 通配符回退 - 最低优先级
  %{weight: 100, response_gt: "999999"}
]
```

用例

多网络操作

当您有多个对等网络，每个网络期望从特定 GT 接收响应时：

- 网络 **A** 呼叫您的 GT `111111` 并期望从 `111111` 接收响应
- 网络 **B** 呼叫您的 GT `222222` 并期望从 `222222` 接收响应

如果没有 GT NAT，您将需要单独的实例或复杂的路由。使用 GT NAT，单个 OmniSS7 实例可以智能地处理这一点。

漫游场景

当作为 HLR 或 SMSc 进行漫游协议操作时：

- 本地网络 用户使用 GT `555000`
- 漫游合作伙伴 **1** 使用 GT `555001`
- 漫游合作伙伴 **2** 使用 GT `555002`

GT NAT 确保每个合作伙伴从其配置的正确 GT 接收响应。

测试和迁移

在网络迁移或测试期间：

逐步将流量从旧 GT 迁移到新 GT
在过渡期间维护两个 GT
根据呼叫者使用的 GT 路由响应

工作原理

地址转换流程

1. 传入请求：OmniSS7 接收到一个 SCCP 消息，包含：

被叫方 GT：55512341112（您的 GT）
呼叫方 GT：877234567（他们的 GT）

2. GT NAT 查找：系统检查呼叫 GT 877234567 是否与配置的前缀规则匹配

3. 前缀匹配：找到最长的匹配前缀（例如，8772 匹配 877234567）

4. 响应 GT 选择：使用匹配规则中的 response_gt（例如，55512341112）

5. 发送响应：SCCP 响应使用：

被叫方 GT：877234567（反转 - 他们的 GT）
呼叫方 GT：55512341112（NAT'd GT）

受影响的响应类型

GT NAT 适用于 SS7 栈的多个层：

SCCP 层（所有响应）

所有响应消息中的 SCCP 被叫/呼叫 GT 地址 ISD（InsertSubscriberData）确认 UpdateLocation 响应

- 错误响应

MAP 层（操作特定）

- **SRI-for-SM** 响应：`networkNode-Number`（SMSc GT 地址）
- **UpdateLocation**：响应中的 `hlr-Number`
- **InsertSubscriberData**：ISD 消息中的 HLR GT

配置

基本配置

添加到 `config/runtime.exs`：

```
config :omniss7,
  # 启用 GT NAT
  gt_nat_enabled: true,

  # 定义 GT NAT 规则
  gt_nat_rules: [
    # 规则 1：来自前缀 "8772" 的呼叫从 "55512341112" 获取响应
    %{calling_prefix: "8772", response_gt: "55512341112"},

    # 规则 2：来自前缀 "8773" 的呼叫从 "55512341111" 获取响应
    %{calling_prefix: "8773", response_gt: "55512341111"},

    # 默认规则（空前缀匹配所有内容）
    %{calling_prefix: "", response_gt: "55512311555"}
  ]
```

配置参数

有关完整的配置参考，请参见 配置参考中的全球标题 NAT 参数。

参数	类型	必需	描述
`gt_nat_enabled`	布尔	是	启用/禁用 GT NAT 功能
`gt_nat_rules`	映射列表	是（如果启用）	前缀匹配规则列表

规则格式

每个规则是一个具有以下键的映射：

```
%{
  calling_prefix: "8772",       # （可选）与呼叫 GT 匹配的前缀
  called_prefix: "555",         # （可选）与被叫 GT 匹配的前缀
  weight: 10,                   # （必需）优先级值（值越低优先级越高）
  response_gt: "55512341112"    # （必需）在响应中使用的 GT
}
```

规则字段：

- **`calling_prefix`**（可选）：与传入呼叫 GT 匹配的字符串前缀
  - 匹配是作为字符串前缀匹配进行的
  - 空字符串 `""` 或 `nil` 作为通配符（匹配任何呼叫 GT）
  - 可以省略以匹配任何呼叫 GT
- **`called_prefix`**（可选）：与传入被叫 GT 匹配的字符串前缀
  - 匹配是作为字符串前缀匹配进行的
  - 空字符串 `""` 或 `nil` 作为通配符（匹配任何被叫 GT）
  - 可以省略以匹配任何被叫 GT
- **`weight`**（必需）：整数优先级值
  - 较低的权重 = 较高的优先级（首先处理）
  - 必须 >= 0
  - 用作匹配规则的主要排序标准

- **`response_gt`**（必需）：在响应中使用的全球标题地址
    - 必须是有效的 E.164 数字字符串
    - 应与您配置的 GT 之一匹配

至少应□定 **`calling_prefix`** 或 **`called_prefix`** 之一以实现特定路由。两者都可以省略以实现通配符/回退规则。

规则匹配逻辑

规则的评估顺序为 权重优先（升序），然后是组合前缀特异性：

匹配算法：

1. 过滤所有指定前缀匹配的规则
    - 如果设置了 `calling_prefix`，它必须与呼叫 GT 匹配
    - 如果设置了 `called_prefix`，它必须与被叫 GT 匹配
    - 如果两者都设置，则两者都必须匹配
    - 如果都未设置，则规则作为通配符
2. 按以下顺序对匹配规则进行排序：
    - 主要：权重（升序 - 较低的值优先）
    - 次要：组合前缀长度（降序 - 较长 = 更具体）
3. □回第一个匹配规则

示例：

```
# 示例规则
gt_nat_rules: [
  # 权重 1: 最高优先级 - 匹配两个前缀
  %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

  # 权重 10: 中等优先级 - 特定规则
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  #
仅呼叫
  %{called_prefix: "555", weight: 10, response_gt: "333333"},    #
仅被叫

  # 权重 100: 最低优先级 - 通配符回退
  %{weight: 100, response_gt: "444444"}  # 匹配所有内容
]

# 匹配示例:
# 呼叫: "877234567", 被叫: "555123" -> "111111" (权重 1, 两者匹配)
# 呼叫: "877234567", 被叫: "999999" -> "222222" (权重 10, 仅呼叫)
# 呼叫: "999999999", 被叫: "555123" -> "333333" (权重 10, 仅被叫)
# 呼叫: "999999999", 被叫: "888888" -> "444444" (权重 100, 通配符)
```

示例

示例 1: 两个网络合作伙伴

场景: 您运营一个 SMSc, 拥有两个网络合作伙伴。每个合作伙伴期望从不同的 GT 接收响应。

```
config :omniss7,
  gt_nat_enabled: true,

  # 默认 SMSc GT（在禁用 GT NAT 或没有规则匹配时使用）
  smsc_service_center_gt_address: "5551000",

  # 合作伙伴的 GT NAT 规则
  gt_nat_rules: [
    # 合作伙伴 A（前缀 4412）期望从 GT 5551001 接收响应
    %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

    # 合作伙伴 B（前缀 4413）期望从 GT 5551002 接收响应
    %{calling_prefix: "4413", weight: 10, response_gt: "5551002"},

    # 默认：使用标准 SMSc GT（通配符回退）
    %{weight: 100, response_gt: "5551000"}
  ]
```

流量流程：

```
来自 44121234567 的传入 SRI-for-SM：
  被叫 GT：5551001（您的 GT，合作伙伴 A 使用）
  呼叫 GT：44121234567（合作伙伴 A 的 GT）

GT NAT 查找：
  "44121234567" 匹配前缀 "4412"
  选择的 response_gt："5551001"

响应 SRI-for-SM 到 44121234567：
  被叫 GT：44121234567（反转）
  呼叫 GT：5551001（NAT'd）
  networkNode-Number：5551001（在 MAP 响应中）
```

示例 2：具有区域 GT 的 HLR

场景：国家 HLR 每个区域有不同的 GT。

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "555000",  # □□ HLR GT

  gt_nat_rules: [
    # □□□□ VLR□□□ 5551□
    %{calling_prefix: "5551", weight: 10, response_gt: "555100"},

    # □□□□ VLR□□□ 5552□
    %{calling_prefix: "5552", weight: 10, response_gt: "555200"},

    # □□□□ VLR□□□ 5553□
    %{calling_prefix: "5553", weight: 10, response_gt: "555300"},

    # □□□□□□□□□□□□□□□
    %{weight: 100, response_gt: "555000"}
  ]
```

□□ 3□□□□□

□□□□□□□ GT □□□□ GT□

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "123456789",  # □ GT□□□□

  gt_nat_rules: [
    # □□□□□□□□□□□□□□□□□□
    %{calling_prefix: "555", weight: 10, response_gt:
"987654321"},  # □ GT
    %{calling_prefix: "666", weight: 10, response_gt:
"987654321"},  # □ GT

    # □□□□□□□□□ GT□□□□□
    %{weight: 100, response_gt: "123456789"}  # □ GT
  ]
```

示例 4：被叫方前缀匹配（新）

场景：您有多个 GT 用于不同服务，并希望根据呼叫的 GT 响应正确的 GT。

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # 当他们呼叫您的 SMS GT（5551xxx）时，响应该 GT
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # 当他们呼叫您的语音 GT（5552xxx）时，响应该 GT
    %{called_prefix: "5552", weight: 10, response_gt: "555200"},

    # 当他们呼叫您的数据 GT（5553xxx）时，响应该 GT
    %{called_prefix: "5553", weight: 10, response_gt: "555300"},

    # 默认回退
    %{weight: 100, response_gt: "555000"}
  ]
```

流量流程：

```
传入请求到被叫 GT：555100（您的 SMS GT）
呼叫 GT：441234567（任何呼叫者）

GT NAT 查找：
  被叫 GT "555100" 匹配前缀 "5551"
  选择的 response_gt："555100"

响应使用呼叫 GT：555100（匹配他们呼叫的内容）
```

示例 5：组合呼叫 + 被叫前缀匹配（高级）

场景：不同的合作伙伴呼叫不同的 GT，您希望进行细粒度控制。

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # 合作伙伴 A 呼叫您的 SMS GT - 最高优先级（权重 1）
    %{calling_prefix: "4412", called_prefix: "5551", weight: 1,
response_gt: "555101"},

    # 合作伙伴 B 呼叫您的 SMS GT - 最高优先级（权重 1）
    %{calling_prefix: "4413", called_prefix: "5551", weight: 1,
response_gt: "555102"},

    # 任何人呼[illegible]您的 SMS GT - 中等优先级（权重 10）
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # 合作伙伴 A 呼叫任何 GT - 中等优先级（权重 10）
    %{calling_prefix: "4412", weight: 10, response_gt: "555200"},

    # 默认回退 - 低优先级（权重 100）
    %{weight: 100, response_gt: "555000"}
  ]
```

匹配示例：

```
# 合作伙伴 A 呼叫 SMS GT
呼叫: "441234567", 被叫: "555100"
→ 匹配权重 1 规则（两个前缀）→ "555101"

# 合作伙伴 A 呼叫语音 GT
呼叫: "441234567", 被叫: "555200"
→ 匹配权重 10 规则（仅呼叫）→ "555200"

# 未知呼叫者呼叫 SMS GT
呼叫: "999999999", 被叫: "555100"
→ 匹配权重 10 规则（仅被叫）→ "555100"

# 未知呼叫者呼叫语音 GT
呼叫: "999999999", 被叫: "555200"
→ 匹配权重 100 通配符 → "555000"
```

操作模式

GT NAT 在所有 OmniSS7 操作模式下工作:

HLR 模式

GT NAT 影响:

- UpdateLocation 响应（响应中的 HLR GT）
- InsertSubscriberData 消息（作为呼叫方的 HLR GT）
- SendAuthenticationInfo 响应
- 取消位置响应

有关 HLR 操作的更多信息，请参见 HLR 配置指南。

配置:

```
config :omniss7,
  hlr_mode_enabled: true,
  hlr_service_center_gt_address: "5551234567",  # 默认 HLR GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "331", weight: 10, response_gt:
"5551234568"},  # 法国
    %{calling_prefix: "44", weight: 10, response_gt:
"5551234569"},   # 英国
    %{weight: 100, response_gt: "5551234567"}  # 默认通配符
  ]
```

SMSc 模式

GT NAT 影响:

- SRI-for-SM 响应（`networkNode-Number` 字段） - 参见 SRI-for-SM 详细信息
- MT-ForwardSM 确认

有关 SMSc 操作的更多信息，请参见 SMSc 配置指南。

□□□

```
config :omniss7,
  smsc_mode_enabled: true,
  smsc_service_center_gt_address: "5559999",  # □□ SMSc GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "1", weight: 10, response_gt: "5559991"},
# □□
    %{calling_prefix: "44", weight: 10, response_gt: "5559992"},
# □□
    %{calling_prefix: "86", weight: 10, response_gt: "5559993"},
# □□
    %{weight: 100, response_gt: "5559999"}  # □□□□□
  ]
```

CAMEL □□□□

GT NAT □□□

- □□ SCCP □□□□gsmSCF GT □□□□□□
- CAMEL/CAP □□□□□InitialDP□EventReportBCSM □□
- RequestReportBCSMEvent □□
- ApplyCharging □□
- Continue □□

□□□

```
config :omniss7,
  camelgw_mode_enabled: true,
  camel_gsmscf_gt_address: "55512341112",  # 默认 gsmSCF GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "555", weight: 10, response_gt:
"55512341111"},  # 网络 A
    %{calling_prefix: "666", weight: 10, response_gt:
"55512311555"},  # 网络 B
    %{weight: 100, response_gt: "55512341112"}  # 默认通配符
  ]
```

用例：当作为多个网络的 gsmSCF（服务控制功能）操作时，每个网络的 gsmSSF 可能期望从特定的 gsmSCF GT 接收响应。GT NAT 确保根据哪个 gsmSSF 在呼叫使用正确的 GT。

日志记录和调试

启用 **GT NAT** 日志记录

GT NAT 包括所有转换的自动日志记录：

```
# 在日志中，您将看到：
[info] GT NAT [SRI-for-SM 响应]: 呼叫 GT 877234567 -> 响应 GT
55512341112
[info] GT NAT [UpdateLocation ISD]: 呼叫 GT 331234567 -> 响应 GT
55512341111
[info] GT NAT [MAP BEGIN 响应]: 呼叫 GT 441234567 -> 响应 GT 55512311555
```

上下文字段显示了 NAT 应用的位置：

- `"SRI-for-SM 响应"` - 在 SRI-for-SM 处理程序中
- `"UpdateLocation ISD"` - 在 InsertSubscriberData 消息中
- `"UpdateLocation END"` - 在 UpdateLocation END 响应中
- `"MAP BEGIN 响应"` - 通用 MAP BEGIN 响应
- `"ISD ACK"` - ISD 确认

- `"HLR □□□□"` - HLR □□□□□
- `"CAMEL □□"` - CAMEL/CAP □□□□□gsmSCF□

□□

□□□□□□□□□ GT NAT □□□

```
# □□ GT NAT □□
iex> GtNat.validate_config()
{:ok, [
  %{calling_prefix: "8772", weight: 10, response_gt:
"55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt:
"55512341111"}
]}

# □□□□□□□
iex> GtNat.enabled?()
true

# □□□□□□□
iex> GtNat.get_rules()
[
  %{calling_prefix: "8772", weight: 10, response_gt:
"55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt:
"55512341111"}
]
```

□□ GT NAT

□□□□□□□□ GT NAT □□□

```
# □□□□□□ GT □□□□□□called_gt □ nil□
iex> GtNat.translate_response_gt("877234567", nil, "default_gt")
"55512341112"

# □□□□□□□□□ GT □□□□□
iex> GtNat.translate_response_gt("877234567", "555123",
"default_gt")
"55512341112"

# □□□□□□□□□nil □□ GT□
iex> GtNat.translate_response_gt_with_logging("877234567", nil,
"default_gt", "test")
# □□□GT NAT [test]: □□ GT 877234567 -> □□ GT 55512341112
"55512341112"

# □□□□□□□□□□□ GT□
iex> GtNat.translate_response_gt_with_logging("877234567",
"555123", "default_gt", "test")
# □□□GT NAT [test]: □□ GT 877234567, □□ GT 555123 -> □□ GT
55512341112
"55512341112"

# □□□□□□□□□□□□□□□
iex> GtNat.translate_response_gt("999999999", "888888",
"default_gt")
"default_gt"
```

□□□□

□□□GT NAT □□□

□□ 1□□□□□□

```
iex> Application.get_env(:omniss7, :gt_nat_enabled)
true  # □□□ true
```

□□ 2□□□□□□□□

```
iex> Application.get_env(:omniss7, :gt_nat_rules)
[%{calling_prefix: "8772", response_gt: "55512341112"}, ...]  # 应该返回列表
```

检查 **3**：检查日志 搜索日志中的 "GT NAT" 以查看是否发生了转换。

问题：响应中的 **GT** 错误

症状：响应使用意外的 GT 地址

原因：规则前缀匹配可能过于宽泛或默认规则捕获了流量

解决方案：审查规则权重和前缀：

```
# 不好：低权重的通配符（首先捕获所有内容）
gt_nat_rules: [
  %{weight: 1, response_gt: "111111"},                # 这首先匹配所有内容！
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"}  # 从未到达
]

# 好：具有较低权重的特定规则，较高权重的通配符
gt_nat_rules: [
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  # 特定，低权重
  %{weight: 100, response_gt: "111111"}  # 通配符，高权重（回退）
]
```

问题：**GT NAT** 未应用于特定消息类型

症状：某些响应使用 NAT'd GT，其他则不使用

当前覆盖范围：

- □ SCCP 呼叫 GT（所有响应）
- □ SRI-for-SM 响应（networkNode-Number）
- □ UpdateLocation ISD 消息（HLR GT）
- □ UpdateLocation END 响应

✅ ISD 确认
✅ MAP BEGIN 响应

如果特定消息类型未使用 GT NAT，可能尚未实现。检查源代码或联系支持。

性能考虑

查找性能

GT NAT 使用简单的前缀匹配，复杂度为 O(n)，其中 n 是规则的数量。

性能提示：

将规则数量保持在 100 以下以获得最佳性能
使用特定前缀以减少规则数量
默认规则（空前缀）应放在最后

基准测试（典型系统）：

10 条规则：每次查找 < 1µs
50 条规则：每次查找 < 5µs
100 条规则：每次查找 < 10µs

内存使用

每条规则需要约 100 字节的内存：

10 条规则 ≈ 1 KB
100 条规则 ≈ 10 KB

最佳实践

1. 始终包含一个通配符回退规则

```
gt_nat_rules: [
  %{calling_prefix: "8772", weight: 10, response_gt: "111111"},
  %{calling_prefix: "8773", weight: 10, response_gt: "222222"},
  %{weight: 100, response_gt: "default_gt"}  # 始终有一个高权重的通配符
]
```

2. 使用有意义的前缀和权重

```
# 好：清晰、特定的前缀和适当的权重
%{calling_prefix: "331", weight: 10, response_gt: "..."}  # 法国
%{calling_prefix: "44", weight: 10, response_gt: "..."}   # 英国

# 不好：过于宽泛的前缀或混淆的权重
%{calling_prefix: "3", weight: 5, response_gt: "..."}     # 太多国家
%{calling_prefix: "331", weight: 100, response_gt: "..."}  # 权重应为特定规则降低
```

3. 记录您的规则

```
gt_nat_rules: [
  # 合作伙伴 XYZ - 英国网络（GT 范围：4412xxxxxxx）
  # 权重 10：标准合作伙伴优先级
  %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

  # 合作伙伴 ABC - 法国网络（GT 范围：33123xxxxxx）
  # 权重 10：标准合作伙伴优先级
  %{calling_prefix: "33123", weight: 10, response_gt: "5551002"}
]
```

4. 部署前测试

```
# 在部署前 iex 中测试
iex> GtNat.translate_response_gt("44121234567", nil, "default")
"5551001"  # 预期结果

# 使用被叫 GT 测试
iex> GtNat.translate_response_gt("44121234567", "555123",
"default")
"5551001"  # 预期结果
```

5. 监控日志

启用 INFO 级别日志记录，以查看生产中的所有 GT NAT 转换。

与其他功能的集成

STP 模式

GT NAT 独立于 STP 路由工作。STP 根据点代码和目标 GT 进行路由，而 GT NAT 处理响应地址。

有关 STP 路由的更多信息，请参见 STP 配置指南。

CAMEL 集成

GT NAT 完全集成与 CAMEL/CAP 操作:

SCCP 层:

- 所有 CAMEL 响应中的呼叫方 GT
- 根据传入的 gsmSSF GT 自动应用

配置:

- `camel_gsmscf_gt_address` - 默认 gsmSCF GT (可选)
- 如果未配置，则使用传入请求中的被叫方 GT
- GT NAT 规则根据呼叫方前缀覆盖默认设置

示例：

```
# 当 gsmSSF 555123456 呼叫您的 gsmSCF
# 传入：被叫=55512341112，呼叫=555123456
# GT NAT 查找："555" -> response_gt="55512341111"
# 响应：被叫=555123456，呼叫=55512341111
```

负载均衡

GT NAT 可以与 M3UA 负载均衡结合使用，以实现高级流量管理。

迁移□南

在现有系统上启用 GT NAT

1. 准备配置

```
# 添加到 runtime.exs（最初保持禁用）
config :omniss7,
  gt_nat_enabled: false,  # 开始时禁用
  gt_nat_rules: [
    # 您的规则在这里□□□带有权重
    %{calling_prefix: "877", weight: 10, response_gt:
"111111"},
    %{weight: 100, response_gt: "999999"}  # 通配符回退
  ]
```

2. 测试配置

```
# 验证配置是否编译
mix compile

# 在 iex 中测试
iex -S mix
iex> GtNat.validate_config()
```

3. 在暂存环境中启用

```
gt_nat_enabled: true  # 更改为 true
```

4. 监控日志

```
tail -f log/omniss7.log | grep "GT NAT"
```

5. 部署到生产

    - 在维护窗口期间进行部署
    - 密切监控前 24 小时
    - 准备回滚计划（将 `gt_nat_enabled` 设置为 false）

支持

如有问题或疑问：

- 检查日志中的 "GT NAT" 消息
- 使用 `GtNat.validate_config()` 验证配置
- 查看本指南的故障排除部分
- 联系 OmniSS7 支持并附上日志摘录

另见

- HLR 指南 - HLR 模式配置
- SMSC 指南 - SMSc 模式配置
- STP 指南 - STP 路由配置
- 配置参考 - 完整配置参考

HLR 配置□南

← □回主文档

本指南提供了使用 OmniSS7 作为 家庭位置寄存器 **(HLR/HSS)** 的配置，后端订阅者数据库为 **OmniHSS**。

OmniHSS 集成

OmniSS7 HLR 模式作为 **SS7** 信令前端，与 **OmniHSS**（一个功能齐全的家庭订阅者服务器 HSS 后端）进行接口。该架构分离了关注点：

- **OmniSS7 (HLR** 前端**)**：处理所有 SS7/MAP 协议信令、SCCP 路由和网络通信
- **OmniHSS (HSS** 后端**)**：管理订阅者数据、身份验证、配置和高级功能

为什么选择 OmniHSS？

OmniHSS 提供运营商级的订阅者管理，具有以下功能：

- 多 **IMSI** 支持：每个订阅者可以有多个 IMSI 与单个 MSISDN 关联，以便于国际漫游、网络切换和 eSIM 配置
- 灵活的身份验证：支持 Milenage（3G/4G/5G）和 COMP128（2G）身份验证算法
- 电路与数据会话跟踪：独立跟踪 CS（电路交换）和 PS（分组交换）网络注册
- 高级配置：可定制的服务配置、补充服务和 CAMEL 订阅数据
- **API** 优先设计：RESTful HTTP API 用于与计费、CRM 和配置系统集成
- 实时更新：位置跟踪、会话管理和身份验证向量生成

所有订阅者数据、身份验证凭据和服务配置都存储在 OmniHSS 中。OmniSS7 通过 HTTPS API 调用查询 OmniHSS，以响应 MAP 操作，如 UpdateLocation、SendAuthenticationInfo 和 SendRoutingInfo。

重要：OmniSS7 HLR 模式仅为 信令前端。所有订阅者管理逻辑、身份验证算法、配置规则和数据库操作均由 OmniHSS 处理。本指南涵盖 OmniSS7 中的 SS7/MAP 协议配置。有关订阅者配置、身份验证配置、服务配置和管理操作的信息，请 参考 **OmniHSS** 文档。

多 IMSI 支持

OmniHSS 原生支持多 **IMSI** 配置，允许单个订阅者（通过 MSISDN 识别）拥有多个 IMSI。这使得：

- 国际漫游配置：不同地区的不同 IMSI 以降低漫游费用
- **eSIM** 多配置：单个 eSIM 设备上的多个网络配置
- 网络切换：在不更改 MSISDN 的情况下无缝切换网络
- 双 **SIM** 协调：跨多个物理或虚拟 SIM 的协调
- 测试与开发：多个测试 IMSI 指向同一订阅者

工作原理：

- 每个 IMSI 具有自己的身份验证凭据（Ki、OPc、算法）
- 每个 IMSI 可以独立进行电路和数据会话注册
- 订阅者服务和配置可以按 IMSI 共享或定制
- OmniSS7 ��� IMSI 查询 OmniHSS，OmniHSS □回适当的订阅者数据
- 计费系统可以按 IMSI 跟踪使用情况，同时将所有 IMSI 关联到单个账户

示例多 **IMSI** 场景：

```
订阅者 MSISDN: +1-555-123-4567
├─ IMSI 1: 310260123456789 (美国本地网络 - Milenage 身份验证)
├─ IMSI 2: 208011234567890 (法国漫游配置 - Milenage 身份验证)
└─ IMSI 3: 440201234567891 (英国漫游配置 - COMP128 身份验证)
```

所有三个 IMSI 可以独立用于网络注册，但它们都属于同一订阅者账户。OmniHSS 管理 IMSI 到订阅者的映射，并确保每个 IMSI 的正确身份验证和配置。

目录

1. OmniHSS 集成
2. 多 IMSI 支持
3. 什么是 HLR 模式？
4. 启用 HLR 模式
5. 订阅者数据库
6. 身份验证向量
7. 位置更新
8. CAMEL 集成
9. 漫游订阅者处理
10. HLR 操作

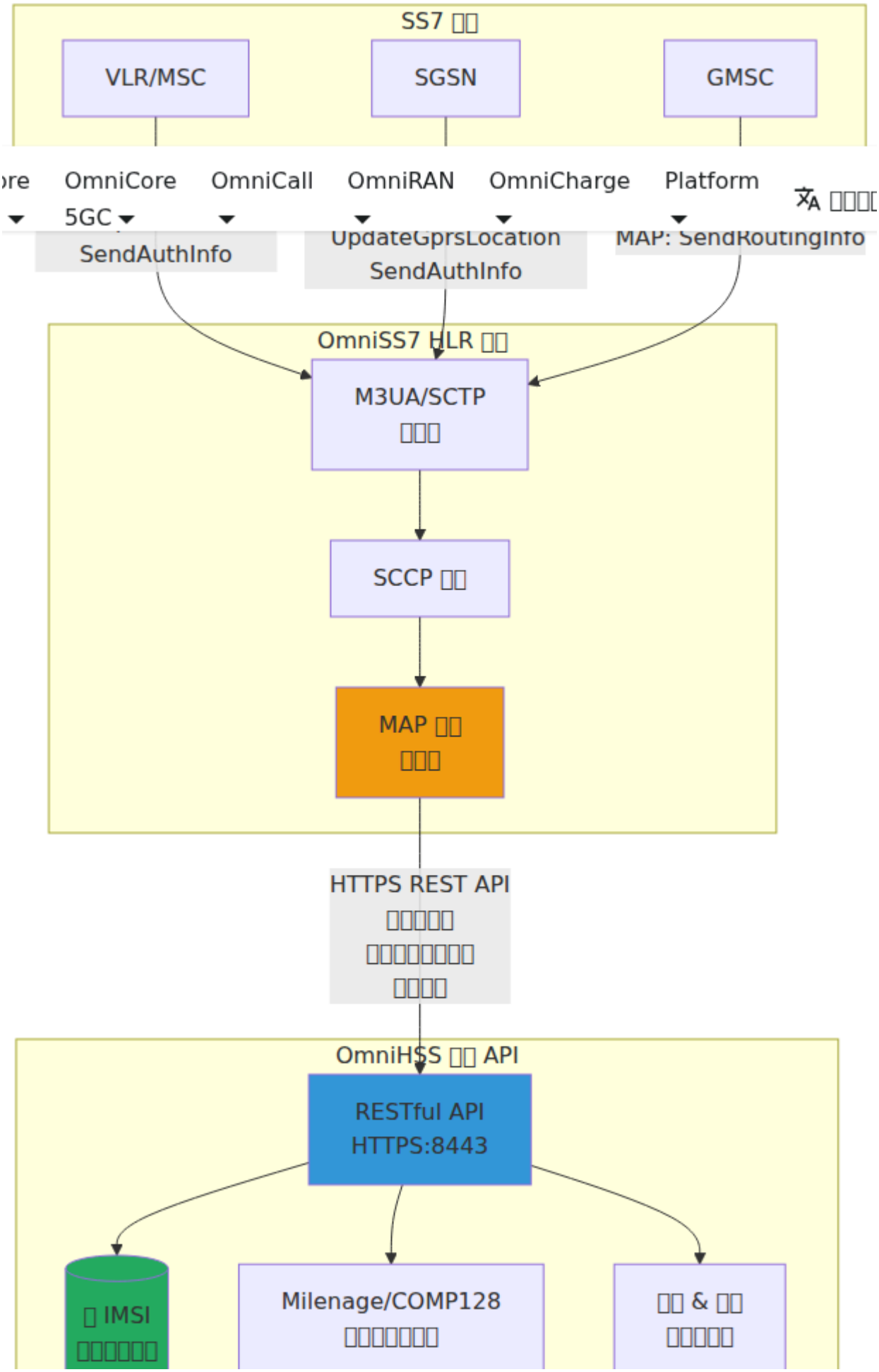
响应字段映射
SendRoutingInfo (SRI)
UpdateLocation / ISD
SendRoutingInfoForSM
字段源摘要

什么是 HLR 模式？

HLR 模式使 OmniSS7 能够作为家庭位置寄存器运行：

订阅者管理：存储和管理订阅者数据
身份验证：为网络访问生成身份验证向量
位置跟踪：处理来自 VLR 的位置更新
路由信息：提供呼叫和 SMS 的路由信息

HLR □□


SS7 □□
VLR/MSC
SGSN
GMSC
ore
OmniCore 5GC
OmniCall
OmniRAN
OmniCharge
Platform
□□□□
SendAuthInfo
UpdateGprsLocation
SendAuthInfo
MAP: SendRoutingInfo
OmniSS7 HLR □□
M3UA/SCTP □□□
SCCP □□
MAP □□ □□□
HTTPS REST API □□□□□ □□□□□□□□□ □□□□
OmniHSS □□ API
RESTful API HTTPS:8443
□ IMSI □□□□□□
Milenage/COMP128 □□□□□□□
□□ & □□ □□□□□



启用 HLR 模式

OmniSS7 可以在不同模式下运行（STP、HLR、SMSc）。通过一组模式标志和 config/config.exs 中的相关配置选择模式，该文件是应用程序的默认配置文件。

切换到 HLR 模式

要将 OmniSS7 作为 HLR 运行，请设置模式标志以启用 HLR 功能并禁用 SMSc 功能，然后添加特定于 HLR 的配置：

1. 打开 config/config.exs
2. 设置 config:omniss7 下的模式标志：
map_client_enabled: true：必需以发送和接收 MAP 操作
hlr_mode_enabled: true：启用 HLR 特定处理
smsc_mode_enabled: false：禁用 SMSc 功能
3. 添加 HLR 配置参数（API 端点、GT 地址、ISD/CAMEL 设置、M3UA 连接），如下所示
4. 自定义参数以适应您的部署
5. 重启应用程序以使更改生效

注意：config/runtime.exs 是一个仅用于自动化测试的最小存根。它不包含操作模式定义。所有模式和 HLR 配置都位于 config/config.exs（环境特定的覆盖在如 config/test.exs 的文件中）。生产部署提供自己的 config/config.exs。

HLR 模式配置

完整的 HLR 配置（在 config/config.exs 中）如下所示：

```
config :omniss7,
  # 模式标志 - 仅启用 HLR 功能
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  # OmniHSS 后端 API 配置
  hlr_api_base_url: "https://omnihss.example.com:8443",

  # HLR 服务中心 GT 地址用于 SMS 操作
  hlr_service_center_gt_address: "1234567890",

  # MSISDN ↔ IMSI 映射配置
  # 参见: MSISDN ↔ IMSI 映射部分以获取详细信息
  hlr_imsi_plmn_prefix: "50557",
  hlr_msisdn_country_code: "61",
  hlr_msisdn_nsn_offset: 0,
  hlr_msisdn_nsn_length: 9,

  # InsertSubscriberData 配置
  # 网络接入模式: :packetAndCircuit、:packetOnly 或 :circuitOnly
  isd_network_access_mode: :packetAndCircuit,

  # 发送 ISD #2（补充服务数据）
  isd_send_ss_data: true,

  # 发送 ISD #3（呼叫限制数据）
  isd_send_call_barring: true,

  # CAMEL 配置（用于 SendRoutingInfo 响应）
  # CAMEL 服务启动的服务密钥
  camel_service_key: 11_110,

  # CAMEL 触发检测点
  # 选项: :termAttemptAuthorized、:tBusy、:tNoAnswer、:tAnswer
  camel_trigger_detection_point: :termAttemptAuthorized,

  # 家庭 VLR 前缀
  # 被视为“家庭”网络的 VLR 地址前缀列表
  # 如果订阅者的 VLR 以这些前缀之一开头，则使用标准 SRI 响应
  # 否则，订阅者正在漫游，我们需要发送 PRN 以获取 MSRN
  # 如果省略，则默认为 []（无前缀 = 所有 VLR 视为漫游）
  home_vlr_prefixes: ["555123"],
```

```
# M3UA □□□□
# □□ ASP □□□□□ MAP □□□UpdateLocation□SendAuthInfo □□
map_client_m3ua: %{
  mode: "ASP",
  callback: {MapClient, :handle_payload, []},
  process_name: :hlr_client_asp,
  # □□□□□HLR □□□
  local_ip: {10, 179, 4, 11},
  local_port: 2905,
  # □□ STP □□
  remote_ip: {10, 179, 4, 10},
  remote_port: 2905,
  routing_context: 1
}
```

□□□□□□□□

□□□□□□□□□□□□□□□□□□□ □□□□□

□□	□□	□□□	
`hlr_api_base_url`	□□□	*□□*	**Om**
`hlr_service_center_gt_address`	□□□	*□□*	HLR Upd
`smsc_service_center_gt_address`	□□□	*□□*	SMS for-S
`hlr_smsc_alert_gts`	□□	`[]`	□ Up □ al □ SM
`hlr_alert_location_expiry_seconds`	□□	`172800`	SMS alert □□□□
`hlr_imsi_plmn_prefix`	□□□	`"50557"`	MSIS PLM ↔ IM
`hlr_msisdn_country_code`	□□□	`"61"`	□□ I □□□□ IMSI
`hlr_msisdn_nsn_offset`	□□	`0`	MSIS □□□□ □□□
`hlr_msisdn_nsn_length`	□□	`9`	□□□□ MSIS

□□	□□	□□□	
`isd_network_access_mode`	□□	`:packetAndCircuit`	Inse□□□
`isd_send_ss_data`	□□	`true`	□□ IS
`isd_send_call_barring`	□□	`true`	□□ IS
`camel_service_key`	□□	`11_110`	Sen□ ◆◆
`camel_trigger_detection_point`	□□	`:termAttemptAuthorized`	CAM
`home_vlr_prefixes`	□□	`[]`	□□□‘ □□□□ □□□
`local_ip`	□□	*□□*	□□ H
`local_port`	□□	`2905`	□□ S
`remote_ip`	□□	*□□*	□□ S
`remote_port`	□□	`2905`	□□ S
`routing_context`	□□	`1`	M3U

启用 **HLR** 模式时会发生什么

当 `hlr_mode_enabled: true` 时，Web UI 将显示：

- □ **SS7** 事件 - 事件日志
- □ **SS7** 客户端 - MAP 操作测试
- □ □□□ - 连接状态（M3UA/SCTP 对等体）
- □ **HLR** 链接 - HLR API 状态 + 订阅者管理 ← *HLR □□*
- □ □□ - 系统监控
- □ □□ - 配置查看器

路由、路由测试和 **SMSc** 链接 选项卡将被隐藏。

重要说明

- 必需配置：`hlr_service_center_gt_address` 参数是 强制 的。如果未配置，应用程序将无法启动。
- **OmniHSS** 后端：OmniHSS API 后端必须在配置的 `hlr_api_base_url` 可访问。
- **API** 请求超时：所有 OmniHSS API 请求都有 硬编码的 **5** 秒超时。
- **MAP** 请求超时：所有 MAP 请求（SRI、UpdateLocation、SendAuthInfo 等）都有 硬编码的 **10** 秒超时。
- **ISD** 超时：每个 UpdateLocation 序列中的 InsertSubscriberData (ISD) 消息都有 硬编码的 **10** 秒超时。
- 需要与 STP 的 M3UA 连接以接收 MAP 操作。
- 更改模式后，必须重启应用程序以使更改生效。
- **Web UI**：有关使用 Web 界面的信息，请参见 Web UI 指南。
- **API** 访问：有关 REST API 文档和 Swagger UI 访问的信息，请参见 API 指南。

订阅者数据库

OmniHSS 管理所有订阅者数据，包括身份、身份验证凭据、服务配置和位置信息。OmniSS7 通过 RESTful API 调用检索这些数据。

OmniHSS 订阅者模型

OmniHSS 存储全面的订阅者信息：

- 每个订阅者多个 **IMSI**：支持多 IMSI 配置（eSIM、漫游配置、网络切换）
- 身份验证凭据：Ki、OPc 和算法选择（Milenage 或 COMP128）
- 服务配置：订阅者类别、允许的服务、QoS 参数
- 位置跟踪：当前 VLR/MSC（电路会话）和 SGSN/GGSN（数据会话）独立跟踪
- **CAMEL** 订阅数据：服务密钥□□触发点和 gsmSCF 地址
- 补充服务：呼叫转移、限制、等待、CLIP/CLIR 配置
- 管理状态：启用/禁用、服务限制、到期日期

身份验证向量

生成身份验证向量

OmniHSS 生成身份验证向量，使用 Milenage 或 COMP128 算法，基于每个订阅者配置的身份验证方法。当 OmniSS7 接收到 **sendAuthenticationInfo** MAP 请求时：

1. OmniSS7 从 MAP 请求中提取 IMSI
2. OmniSS7 调用 OmniHSS API 生成身份验证向量
3. OmniHSS 检索订阅者的 Ki 和 OPc 凭据
4. OmniHSS 生成请求的向量数量（RAND、XRES、CK、IK、AUTN）
5. OmniSS7 将向量编码为 MAP 格式并□回给请求的 VLR/SGSN

OmniHSS API 集成

OmniSS7 通过 HTTPS REST API 与 OmniHSS 通信，以检索订阅者信息、更新位置信息并生成身份验证向量：

```
config :omniss7,
  hlr_api_base_url: "https://omnihss-server:8443"
```

当 OmniSS7 从 SS7 网络接收到 MAP 操作时，它查询 OmniHSS 以：

- 按 **IMSI** 或 **MSISDN** 检索订阅者数据
- 使用存储的 **Ki/OPc** 凭据生成身份验证向量
- 在订阅者执行 **UpdateLocation** 时更新电路会话位置
- 检查订阅者状态和服务权限

位置更新

更新位置处理

在接收到 **updateLocation** MAP 请求时���OmniSS7 与 OmniHSS 协调以在新 VLR 注册订阅者：

1. 从 **UpdateLocation** 请求中提取位置信息（IMSI、新 VLR GT、新 MSC GT）
2. 查询 **OmniHSS** 以验证订阅者是否存在且已启用
3. 在 **OmniHSS** 中更新电路会话，使用新 VLR/MSC 位置
4. 发送 **InsertSubscriberData (ISD)** 消息以在新 VLR 中配置订阅者
5. 返回 **UpdateLocation** 响应 给 VLR（包括来自 `hlr_service_center_gt_address` 的 HLR GT）
6. 发送 **alertServiceCenter** 给配置的 SMSc GT（如果 `hlr_smsc_alert_gts` 被填充）

注意：`hlr_service_center_gt_address` 配置参数指定在 UpdateLocation 响应中□回的 HLR 的全球标题。这允许 VLR/MSC 识别并将消息路由回此 HLR。

警报服务中心集成

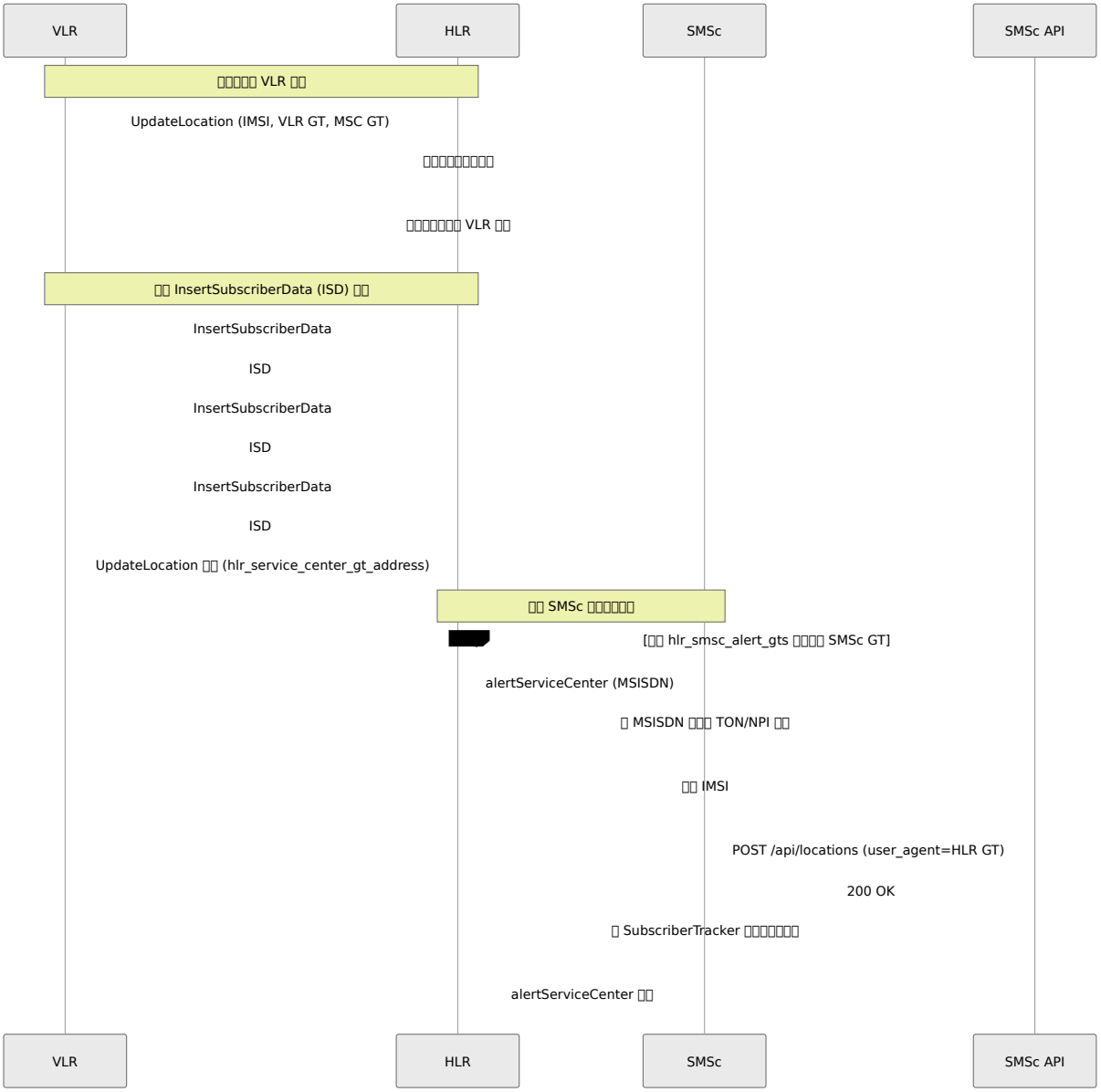
在成功的 UpdateLocation 之后，HLR 可以通过发送 **alertServiceCenter**（MAP 操作码 64）消息自动通知 SMSc 系统，表明订阅者现在可达。有关 SMSc 如何处理这些警报的信息，请参见 SMSc 指南中的警报服务中心处理。

配置

配置要通知的 SMSc 全球标题列表：

```
config :omniss7,
  # □ UpdateLocation □□□ alertServiceCenter □ SMSc GT □□
  hlr_smsc_alert_gts: [
    "15559876543",
    "15559876544"
  ],

  # SMSc □□□ alertServiceCenter □□□□□□□□□□□□□48 □□□
  hlr_alert_location_expiry_seconds: 172800
```

□□□



□□

当订阅者执行 UpdateLocation 时：

1. HLR 向 hlr_smsc_alert_gts 列表中的每个 SMSc GT 发送 alertServiceCenter
2. 消息包括订阅者的 MSISDN
3. HLR 使用 hlr_service_center_gt_address 作为呼叫方 GT
4. SCCP 定址：呼叫 SSN=6（HLR），被叫 SSN=8（SMSc）

SMSc 接收警报并：

从 MSISDN 中剥离 TON/NPI 前缀（例如，“19123123213” → “123123213”）
在其位置数据库中将订阅者标记为可达（通过 POST 到 /api/locations）
在调用 API 时将 user_agent 字段设置为 HLR GT（用于跟踪哪个 HLR 发送了警报）
根据 hlr_alert_location_expiry_seconds 设置位置过期时间
在 SMSc 订阅者跟踪器中跟踪订阅者以进行监控

测试

使用 Web UI 中的活动订阅者页面手动发送 alertServiceCenter 消息进行测试：

1. 导航到“活动订阅者”选项卡
2. 找到“测试警报服务中心”部分
3. 输入 MSISDN、SMSc GT 和 HLR GT（默认值已从配置中预填充）
SMSc GT 默认值为 hlr_smsc_alert_gts 中的第一个条目
HLR GT 默认值为 hlr_service_center_gt_address
4. 点击“发送 alertServiceCenter”

这对于测试 SMSc 警报处理非常有用，而无需完整的 UpdateLocation 流程。该表单使用 phx-blur验证，以避免在输入时显示错误。

InsertSubscriberData (ISD) 配置

在成功的 UpdateLocation 之后，HLR 使用InsertSubscriberData（ISD）消息将订阅者配置数据发送到 VLR。ISD 配置允许您自定义发送的数据及其方式。

有关配置参数参考，请参见配置参考中的 ISD 配置。

ISD 序列

HLR 可以发送最多 3 个顺序 ISD 消息：

1. **ISD #1**（始终发送） - 基本订阅者数据：
    - IMSI
    - MSISDN
    - 订阅者类别
    - 订阅者状态（serviceGranted）
    - 承载服务列表
    - 电信服务列表
    - 网络接入模式
2. **ISD #2**（可选） - 补充服务（SS）数据：
    - 呼叫转移设置（无条件、忙、无回复、不可达）
    - 呼叫等待
    - 呼叫保持
    - 多方服务
    - 补充服务状态和功能
3. **ISD #3**（可选） - 呼叫限制数据：
    - 阻止所有外呼（BAOC）
    - 阻止外部国际呼叫（BOIC）
    - 访问限制数据

配置选项

```
# InsertSubscriberData 配置
# 网络接入模式：:packetAndCircuit、:packetOnly 或 :circuitOnly
isd_network_access_mode: :packetAndCircuit,

# 发送 ISD #2（补充服务数据）
isd_send_ss_data: true,

# 发送 ISD #3（呼叫限制数据）
isd_send_call_barring: true,
```

网络接入模式

`isd_network_access_mode` 参数控制订阅者被允许的网络接入类型：

值	描述	用例
`:packetAndCircuit`	支持分组交换（GPRS/LTE）和电路交换（语音）	默认 - 完整服务订阅者
`:packetOnly`	仅支持分组交换（数据/LTE）	数据卡、物联网设备
`:circuitOnly`	仅支持电路交换（语音/SMS）	传统设备、仅语音计划

控制 **ISD** 消息

您可以根据网络需求控制发送哪些 ISD 消息：

发送所有 **ISD**（默认 - 完整功能集）：

```
isd_send_ss_data: true,
isd_send_call_barring: true,
```

仅发送基本订阅者数据（最小配置）：

```
isd_send_ss_data: false,
isd_send_call_barring: false,
```

发送基本 + 补充服务（无呼叫限制）：

```
isd_send_ss_data: true,
isd_send_call_barring: false,
```

ISD 流程示例

当接收到 UpdateLocation 时：

```
VLR → HLR: UpdateLocation (开始)
HLR → VLR: InsertSubscriberData #1 (继续) - 基本数据
VLR → HLR: ISD #1 确认 (继续)
HLR → VLR: InsertSubscriberData #2 (继续) - SS 数据 [如果启用]
VLR → HLR: ISD #2 确认 (继续)
HLR → VLR: InsertSubscriberData #3 (继续) - 呼叫限制 [如果启用]
VLR → HLR: ISD #3 确认 (继续)
HLR → VLR: UpdateLocation 响应 (结束)
```

如果 `isd_send_ss_data` 或 `isd_send_call_barring` 设置为 `false`，则跳过这些 ISD 消息，并更早发送 UpdateLocation 结束。

最佳实践

- 默认配置：使用 `:packetAndCircuit` 并启用所有 ISD 以获得最大兼容性
- 物联网/**M2M**：使用 `:packetOnly` 并禁用 SS 数据/呼叫限制以适应仅数据设备
- 互操作性：某些旧 VLR 可能不支持所有补充服务 - 如果遇到问题，请禁用 `isd_send_ss_data`
- 性能：禁用未使用的 ISD 可以减少消息开销，加快位置更新速度

CAMEL 集成

CAMEL 配置用于 **SendRoutingInfo**

在响应来自 GMSC（网关 MSC）的 **SendRoutingInfo**（SRI）请求时，HLR 可以指示 GMSC 调用 CAMEL 服务以进行智能呼叫路由和服务控制。

有关配置参数参考，请参见 CAMEL 配置中的配置参考。

什么是 **CAMEL**？

CAMEL（定制应用程序用于移动网络增强逻辑）是一种协议，允许在 GSM/UMTS 网络中实现智能网络服务。它允许网络运营商实施增值服务，例如：

- 预付费计费
- 呼叫筛选和限制
- 虚拟专用网络（VPN）

- 高价服务
- 带有自定义逻辑的呼叫转移
- 基于位置的服务

配置选项

```
# CAMEL 配置（用于 SendRoutingInfo 响应）
# CAMEL 服务启动的服务密钥
camel_service_key: 11_110,

# CAMEL 触发检测点
# 选项: :termAttemptAuthorized，:tBusy，:tNoAnswer，:tAnswer
camel_trigger_detection_point: :termAttemptAuthorized,
```

服务密钥

`camel_service_key` 标识应在 gsmSCF（服务控制功能）处调用的 CAMEL 服务。这是一个在您的网络中配置的数字标识符：

服务密钥	典型用例
`11_110`	预付费终止呼叫控制（默认）
`100`	发起预付费服务
`200`	带有自定义逻辑的呼叫转移
`300`	虚拟专用网络（VPN）
自定义	运营商特定服务

配置示例：

```
# 用于预付费终止呼叫控制
camel_service_key: 11_110,

# 用于 VPN 服务
camel_service_key: 300,
```

触发检测点

`camel_trigger_detection_point` 指定在呼叫设置期间何时触发 CAMEL 服务:

检测点	描述	触发时机
`:termAttemptAuthorized`	呼叫尝试已授权（默认）	在呼叫路由到订阅者之前
`:tBusy`	终止忙碌	当订阅者忙时
`:tNoAnswer`	终止未应答	当订阅者未应答时
`:tAnswer`	终止应答	当订阅者接听电话时

配置示例:

标准预付费控制（在路由之前触发）:

```
camel_trigger_detection_point: :termAttemptAuthorized,
```

自定义忙碌处理（在忙碌时触发）:

```
camel_trigger_detection_point: :tBusy,
```

基于应答计费（在应答时触发）:

```
camel_trigger_detection_point: :tAnswer,
```

带 CAMEL 的 SRI 响应

当配置时，SendRoutingInfo 响应包括 CAMEL 订阅信息：

```
GMSC → HLR: SendRoutingInfo (开始)
HLR → GMSC: SRI 响应 (结束)，包括：
   - IMSI
   - VLR 编号
   - 订阅者状态
   - CAMEL 路由信息：
     * 服务密钥：11_110
     * gsmSCF 地址：<配置地址>
     * 触发检测点：termAttemptAuthorized
     * 默认呼叫处理：continueCall

GMSC 在触发点联系 gsmSCF 以执行 CAMEL 服务
```

最佳实践

- 生产网络：使用与您的 gsmSCF 提供商达成一致的标准化服务密钥
- 测试：使用 `:termAttemptAuthorized` 进行最全面的测试
- 预付费服务：服务密钥 `11_110` 是预付费终止呼叫的常见行业标准
- 回退处理：`defaultCallHandling: :continueCall` 确保在 gsmSCF 无法访问时呼叫继续进行

漫游订阅者处理

家庭 **VLR** 与漫游 **VLR** 检测

当 HLR 接收到 **SendRoutingInfo**（SRI）请求时，它需要确定订阅者是在“家庭” VLR（在您的网络内）还是在漫游 VLR（访问其他网络）。基于此确定，行为有所不同：

有关配置参数参考，请参见 配置参考中的家庭 VLR 前缀。

- 家庭 **VLR**：□回带有 CAMEL 路由信息的标准 SRI 响应
- 漫游 **VLR**：发送提供漫游号码（PRN）请求以获取 MSRN，然后在 SRI 响应中□回

配置

```
# 家庭 VLR 前缀
# 被视为“家庭”网络的 VLR 地址前缀列表
# 如果订阅者的 VLR 地址以这些前缀之一开头，则使用标准 SRI 响应
# 否则，订阅者正在漫游，我们需要发送 PRN 以获取 MSRN
home_vlr_prefixes: ["555123"],
```

配置示例：

```
# 单一家庭网络
home_vlr_prefixes: ["555123"],

# 多个家庭网络（例如，不同地区或子公司）
home_vlr_prefixes: ["555123", "555124", "555125"],
```

工作原理

1. 家庭订阅者流（标准）

当订阅者的 VLR 地址以配置的家庭前缀开头时：

```
GMSC → HLR: SendRoutingInfo (MSISDN: "1234567890")
HLR 查询后端 API 以获取订阅者数据
HLR 检查 VLR 地址："5551234567"
HLR 确定：VLR 以 "555123" 开头 → 家庭网络
HLR → GMSC: SRI 响应，带有 CAMEL 路由信息：
  - IMSI
  - VLR 编号："5551234567"
  - gsmSCF 地址（MSC）："5551234501"
  - CAMEL 服务密钥：11_110
  - 触发检测点：termAttemptAuthorized
```

2. 漫游订阅者流（需要 **PRN**）

当订阅者的 VLR 地址与任何家庭前缀不匹配时：

```
GMSC → HLR: SendRoutingInfo (MSISDN: "1234567890")
HLR □□□□ API □□□□□□□□
HLR □□ VLR □□□"49170123456"
HLR □□□VLR □□ "555123" □□ → □□
HLR → MSC: ProvideRoamingNumber (PRN):
  - MSISDN: "1234567890"
  - IMSI: "999999876543210"
  - MSC □□: "49170123456"
  - GMSC □□: "5551234501"
MSC → HLR: PRN □□□□ MSRN: "49170999888777"
HLR → GMSC: SRI □□□□□□□□□
  - IMSI
  - VLR □□: "49170123456"
  - □□□□□MSRN□: "49170999888777"
```

□□□□□□

□□□□□ **SRI** □□

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :camelRoutingInfo, %{
      gmscCamelSubscriptionInfo: %{
        "t-CSI": %{
          serviceKey: 11_110,
          "gsmSCF-Address": "5551234501",
          defaultCallHandling: :continueCall,
          "t-BcsmTriggerDetectionPoint": :termAttemptAuthorized
        }
      }
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "5551234567"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

□□□□□ **SRI** □□

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :routingInfo, %{
      roamingNumber: "49170999888777"  # 来自 PRN 的 MSRN
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "49170123456"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

提供漫游号码（**PRN**）操作

PRN 请求结构

发送到 MSC/VLR 的 PRN 请求包含:

字段	来源	描述
MSISDN	SRI 请求	订阅者的电话号码
IMSI	HLR API	订阅者的 IMSI
MSC 编号	HLR API	为漫游订阅者服务的 MSC（`serving_msc`）
GMSC 地址	SRI 请求	发起原始 SRI 请求的 GMSC
呼叫参考编号	静态	呼叫参考标识符
支持的 **CAMEL** 阶段	静态	GMSC 支持的 CAMEL 阶段

PRN 响应处理

HLR 期望 PRN 响应包含:

- **MSRN**（移动台漫游号码）：由访问网络分配的临时号码，用于路由呼叫

错误处理：

- 如果 PRN 超时 → 在 SRI 响应中回错误 27（缺失订阅者）
- 如果 PRN 失败 → 在 SRI 响应中回错误 27（缺失订阅者）
- 如果无法提取 MSRN → 在 SRI 响应中回错误 27（缺失订阅者）

配置示例

单一家庭网络运营商

```
# 所有 VLR 地址以 "555123" 开头被视为家庭
home_vlr_prefixes: ["555123"],
```

- VLR `5551234567` → 家庭（CAMEL 响应）
- VLR `5551235001` → 家庭（CAMEL 响应）
- VLR `49170123456` → 漫游（PRN + MSRN 响应）

多区域运营商

```
# 跨不同区域的多个家庭网络
home_vlr_prefixes: ["555123", "555124", "555125"],
```

- VLR `5551234567` → 家庭（区域 1）
- VLR `5552341234` → 家庭（区域 2）
- VLR `5553411111` → 家庭（区域 3）
- VLR `44201234567` → 漫游（国际）

测试配置

为了测试 PRN 功能，将列表设置为空，以将所有 VLR 视为漫游：

```
# 所有 VLR 被视为漫游（用于测试 PRN 流程）
home_vlr_prefixes: [],
```

最佳实践

- 前缀选择：使用唯一的最短前缀来标识您网络的 VLR（例如，国家代码 + 网络代码）
- 多个前缀：包括您网络中的所有 VLR 前缀，包括不同地区和子公司
- 漫游协议：确保漫游合作伙伴网络正确支持 PRN
- 测试：在生产部署之前彻底测试家庭和漫游场景
- 监控：监控 PRN 超时率，以识别���漫游合作伙伴的连接问题

故障排除

症状：所有订阅者被视为漫游

- 原因：未配置 `home_vlr_prefixes` 或前缀与 VLR 地址不匹配
- 解决方案：检查数据库中的 VLR 地址并相应更新前缀

症状：PRN 请求超时

- 原因：与漫游合作伙伴 MSC/VLR 的网络连接问题
- 解决方案：验证 M3UA/SCCP 路由到远程 MSC 地址

症状：SRI 响应中的 MSRN 无效

- 原因：漫游合作伙伴的 PRN 响应格式与预期结构不匹配
- 解决方案：查看 PRN 响应日志，并根据需要调整 `extract_msrn_from_prn/1`

HLR 操作

支持的 MAP 操作

OmniSS7 按操作码（独立于 TCAP 阶段）调度入站 MAP 操作。以下入站操作被处理：

操作码	操作	目的
2	`updateLocation`	注册 VLR 位置（触发 ISD 序列 + 可选的 alertServiceCenter）
23	`updateGprsLocation`	注册 SGSN（分组交换）位置
3	`cancelLocation`	从旧 VLR 注销
7	`insertSubscriberData`	接收订阅者配置（例如，当作为 VLR 端时）
22	`sendRoutingInfo` (SRI)	提供 MSRN/CAMEL 路由以进行呼叫（家庭与漫游）
45	`sendRoutingInfoForSM` (SRI-for-SM)	提供 IMSI + SMSC GT 以进行 SMS 路由
44	`MT-Forward-SM`	移动终止 SMS 交付
46	`MO-Forward-SM`	移动发起 SMS 接收
47	`reportSM-DeliveryStatus`	SMS-GMSC 交付状态报告
64	`alertServiceCenter`	向 SMSc 发送订阅者可达警报
66	`readyForSM`	MSC/SGSN 表示订阅者已准备好接收 SMS
56	`sendAuthenticationInfo`	生成身份验证向量
57	`restoreData`	VLR 订阅者数据恢复
67	`purgeMS`	在 HLR 中清除订阅者记录
59	`processUnstructuredSS-Request`	USSD（在启用时通过 USSD 网关处理）
71	`anyTimeInterrogation` (ATI)	订阅者位置/状态查询

操作码	操作	目的
10	`registerSS`	补充服务注册（例如，无条件呼叫转移）

未处理的操作码将以 `facilityNotSupported`（21）错误进行响应。

响应字段映射

本节详细说明 HLR 响应中每个字段的来源。

SendRoutingInfo (SRI) 响应

目的：提供有关订阅者的来电路由信息。

HLR 根据订阅者是在家庭 VLR 还是漫游提供两种不同的响应类型：

家庭订阅者响应（**CAMEL** 路由）

用于订阅者的 VLR 地址以配置的 `home_vlr_prefixes` 值开头时。

响应结构：

字段	来源	描述
IMSI	OmniHSS API	来自 OmniHSS 数据库的订阅者 IMSI
VLR 编号	OmniHSS API	当前为订阅者服务的 VLR（`circuit_session.assigned_vlr`）
订阅者状态	静态	始终为 `notProvidedFromVLR`
extendedRoutingInfo	-	类型：`camelRoutingInfo`
gsmSCF 地址	OmniHSS API	为订阅者服务的 MSC（`circuit_session.assigned_msc`）
服务密钥	config.exs	CAMEL 服务标识符（`camel_service_key`）
触发检测点	config.exs	何时触发 CAMEL（`camel_trigger_detection_point`）
CAMEL 能力处理	静态	CAMEL 阶段支持级别
默认呼叫处理	静态	如果 gsmSCF 无法访问的回退

漫游订阅者响应（**MSRN** 路由）

用于订阅者的 VLR 地址与任何配置的 `home_vlr_prefixes` 值不匹配时。

响应结构：

字段	来源	描述	
IMSI	OmniHSS API	来自 OmniHSS 数据库的订阅者 IMSI	"
VLR 编号	OmniHSS API	当前为订阅者服务的 VLR（`circuit_session.assigned_vlr`）	"
订阅者状态	静态	始终为 `notProvidedFromVLR`	:
extendedRoutingInfo	-	类型: `routingInfo`	-
漫游号码（**MSRN**）	PRN 响应	从提供漫游号码请求中获得的 MSRN	"

路由决策逻辑:

```
1. OmniSS7 接收到 SendRoutingInfo 请求
2. OmniSS7 从 OmniHSS API 查询订阅者数据
3. OmniSS7 检查 VLR 地址与 home_vlr_prefixes:

   如果 VLR 以家庭前缀开头:
     → 返回 CAMEL 路由信息（家庭订阅者流）

   如果 VLR 不匹配任何家庭前缀:
     → 向 MSC 发送提供漫游号码（PRN）
     → 从 PRN 响应中提取 MSRN
     → 返回带 MSRN 的路由信息（漫游订阅者流）
```

数据流:

- OmniSS7 查询 OmniHSS 以获取订阅者信息
- OmniHSS □回 IMSI、当前 VLR/MSC 位置和订阅者状态
- OmniSS7 使用这些数据构建 MAP 响应

配置要求:

```
# 在 config/config.exs 中
home_vlr_prefixes: ["555123"],  # 家庭 VLR 前缀列表
```

错误响应：

- 如果 `serving_vlr` 和 `serving_msc` 为 `null`：回错误 27（缺失订阅者）
- 如果未找到订阅者：回错误 1（未知订阅者）
- 如果 PRN 请求超时（漫游情况）：回错误 27（缺失订阅者）
- 如果 PRN 响应无效（漫游情况）：回错误 27（缺失订阅者）

UpdateLocation 响应与 InsertSubscriberData

目的：在新 VLR 中注册订阅者并配置订阅者数据。

UpdateLocation 结束响应

字段	来源	描述	示例
HLR 编号	config.exs	本 HLR 的全球标题（`hlr_service_center_gt_address`）	`"5551234568"`
TCAP 消息类型	静态	在所有 ISD 之后的最终响应	`END`

InsertSubscriberData #1（基本订阅者数据）

字段	来源	描述	示例
IMSI	请求	来自 UpdateLocation 请求	`"999999876543210"`
MSISDN	OmniHSS API	来自 OmniHSS 的订阅者电话号码	`"555123456"`
类别	静态	订阅者类别	`"\n"` (0x0A)
订阅者状态	静态	服务状态	`:serviceGranted`
承载服务列表	静态	支持的承载服务	`[<<31>>]`
电信服务列表	静态	支持的电信服务	`[<<17>>, "!", "\""]`
网络接入模式	config.exs	分组/电路接入（`isd_network_access_mode`）	`:packetAndCircuit`

InsertSubscriberData #2（补充服务） **-** 可选

字段	来源	描述	由谁控制
配置的 **SS**	静态	补充服务数据	`isd_send_ss_data: true`
呼叫转移	静态	转移配置（无条件、忙、无回复、不可达）	配置启用
呼叫等待	静态	呼叫等待服务状态	配置启用
多方服务	静态	会议呼叫支持	配置启用

ISD #2 包含:

- 无条件呼叫转移（SS 代码 21）
- 忙时呼叫转移（SS 代码 41）
- 无回复呼叫转移（SS 代码 42）

- 不可达呼叫转移（SS 代码 62）
- 呼叫等待（SS 代码 43）
- 多方服务（SS 代码 51）
- CLIP/CLIR 服务

InsertSubscriberData #3（呼叫限制） **-** 可选

字段	来源	描述	由谁控制
呼叫限制信息	静态	呼叫限制配置	`isd_send_call_barring: true`
BAOC	静态	阻止所有外呼（SS 代码 146）	配置启用
BOIC	静态	阻止外部国际呼叫（SS 代码 147）	配置启用
访问限制数据	静态	网络接入限制	配置启用

ISD 序列控制：

- ISD #1：始终发送 **-** 包含基本订阅者数据
- ISD #2：仅在 `isd_send_ss_data: true` 时发送
- ISD #3：仅在 `isd_send_call_barring: true` 时发送

SendRoutingInfoForSM (SRI-for-SM) 响应

目的：提供用于 SMS 交付的 MSC/SMSC 路由信息。当 SMSc 需要将 SMS 发送到订阅者时，它向 HLR 发送 SRI-for-SM 请求，以确定将消息□□由到哪里。

HLR 以两种模式之一进行响应，由请求的 **SCCP** 呼叫方 **GT** 决定：

1. 家庭路由（默认：每个外部发起者）。此模式□回合成 IMSI 以及 IP-SM-GW 自身的 GT 作为网络节点编号。HLR 从不向外部运营商暴露真实 IMSI 和服务节点。MT-ForwardSM □回到 IP-SM-GW，然后处理交付。这是互联和漫游流量的正确行为。

字段	来源	描述	如何生成	
IMSI	计算	从 MSISDN 派生的合成 IMSI	`PLMN_PREFIX + zero_padded_MSISDN`	`"001001`
网络节点编号	config.exs	SMSC GT 地址用于 SMS 路由	`smsc_service_center_gt_address`	`"555123`

2. 内部直接路由（仅当请求来自我们自己的 **SMSc / IP-SM-GW** 节点时）。当呼叫 GT 匹配配置的 `home_smsc_gts` 列表时，HLR 在其后端查找订阅者并□回 真实 **IMSI** 和 真实服务 **MSC** 作为网络节点编号，以便我们的 SMSc 可以将 MT-ForwardSM 直接发送到 MSC。如果没有可用的服务 MSC，则回退到家庭路由。

字段	来源	描述	示例
IMSI	HLR ��端	真实订阅者 IMSI	`"505570123456789"`
网络节点编号	HLR 后端	来自上次 UpdateLocation 的真实 `serving_msc`	`"5559876543"`

配置参数（来自 `config/config.exs`）：

```
# 服务中心 GT 地址（在家庭路由的 SRI-for-SM 响应中返回）
# 这告诉请求的 SMSc 将 MT-ForwardSM 消息发送到哪里
smsc_service_center_gt_address: "5551234567",  # 必需

# 内部直接路由：我们自己的 SMSc / IP-SM-GW 节点的 SCCP 呼叫 GT。
# 来自这些的 SRI-for-SM 将获得真实服务 MSC，而不是家庭路由。
# 其他所有人都将家庭路由。将其留空（[]）以家庭路由所有内容。
home_smsc_gts: ["5551234567"],                  # 可选（默认 []）

# MSISDN ↔ IMSI 映射配置
# PLMN 前缀：MCC（001 = 测试网络）+ MNC（01 = 测试运营商）
hlr_imsi_plmn_prefix: "001001",                   # 唯一需要的配置参数！
```

> 注意：内部直接路由仅在 HLR 实际持有订阅者的 `serving_msc` 时有效。也就是说，服务 MSC 对此 HLR 执行了 MAP `updateLocation`。如果 MSC 通过其他机制注册（例如，向单独的 HLR 进行 GSUP），则后端没有位置，查询将进行家庭路由。

MSISDN ↔ IMSI 映射

配置参数：

这些参数控制 OmniSS7 如何为 SRI-for-SM 响应从 MSISDN 生成合成 IMSI：

- **`hlr_imsi_plmn_prefix`**：在构建合成 IMSI 时使用的 MCC+MNC 前缀（例如，“50557”表示 MCC=505，MNC=57）
- **`hlr_msisdn_country_code`**：在进行反向 IMSI→MSISDN 映射时要添加的国家代码（例如，“61”表示澳大利亚，“1”表示美国/加拿大）
- **`hlr_msisdn_nsn_offset`**：MSISDN 中国家订阅者号码（NSN）开始的字符位置（通常为 0，如果 MSISDN 不包括国家代码，或者为国家代码的长度）
- **`hlr_msisdn_nsn_length`**：要从 MSISDN 中提取的 NSN 的位数

有关更多配置详细信息，请参见 配置参考中的 MSISDN ↔ IMSI 映射。

为什么需要 **MSISDN** 到 **IMSI** 的映射？

MAP 协议的 **SendRoutingInfoForSM**（SRI-for-SM）要求 HLR 在其响应中□回 **IMSI**（国际移动订阅者身份）。然而，请求的 SMSc 只知道订阅者的 **MSISDN**（电话号码）。



在传统网络中：

- SMSc 发送 SRI-for-SM，目标 MSISDN（例如，“5551234567”）
- HLR 必须在其数据库中查找订阅者以找到他们的 IMSI
- HLR 在 SRI-for-SM 响应中回 IMSI
- SMSc 然后在将 MT-ForwardSM 发送到 MSC/VLR 时使用此 IMSI

OmniSS7 的方法 - 合成 IMSI：

OmniSS7 使用简单的编码方案直接从 MSISDN 计算合成 IMSI，而不是维护完整的订阅者数据库与 MSISDN 到 IMSI 的映射。此方法提供了两个关键好处：

1. 隐私：存储在 HLR 数据库中的真实订阅者 IMSI 从不在通过 SS7 网络发送的 SRI-for-SM 响应中暴露
2. 简化：在 SRI-for-SM 操作期间无需查询 HLR 数据库进行 IMSI 查找 - IMSI 是根据 MSISDN 动态计算的

工作原理：

MSISDN 直接编码到 IMSI 的订阅者部分（MCC+MNC 之后的数字）：

```
IMSI = PLMN_PREFIX + zero_padded_MSISDN
```

其中：

- **PLMN_PREFIX**：MCC + MNC（例如，“001001”表示测试网络）
- **MSISDN**：电话号码中的所有数字字符
- 零填充：左侧填充零以使 IMSI 恰好为 15 位数字

逐步示例：

```
# 配置
plmn_prefix = "001001"  # MCC 001 + MNC 01

# 输入：来自 SRI-for-SM 请求的 MSISDN（TBCD 解码）
msisdn = "555123456"    # 9 位数字

# 步骤 1：计算订阅者号码的可用空间
subscriber_digits = 15 - String.length("001001")  # = 9 位数字

# 步骤 2：左填充 MSISDN 以填充订阅者部分
padded_msisdn = String.pad_leading("555123456", 9, "0")  # =
"555123456"（无需填充）

# 步骤 3：连接 PLMN 前缀 + 填充的 MSISDN
imsi = "001001" <> "555123456"  # = "001001555123456"（恰好 15 位数字）
```

完整示例：

□□ MSISDN	PLMN □□	□□□□□□□□	□□□ MSISDN	□□ IMSI	□□
`"555123456"`	`"001001"` (6)	9	`"555123456"`	`"001001555123456"`	□□□□□□□□
`"99"`	`"001001"` (6)	9	`"000000099"`	`"001001000000099"`	□□□□
`"999999999"`	`"001001"` (6)	9	`"999999999"`	`"001001999999999"`	□□□□
`"91123456789"`	`"001001"` (6)	9	`"555123456"`	`"001001555123456"`	□□□□□□□□□□□9□□□□

□□□□□□□□

- □ **MSISDN**□□□□□□□□□□□“99” → “000000099”□
- □ **MSISDN**□□□□□□□□□□□□□□□□□□□□□□□“91123456789” → “555123456”□
- **IMSI** □□□□□□□□□ 15 □□□

□□□□□IMSI → MSISDN□□

SMSc 可以反向此映射，将 IMSI 转换回 MSISDN:

```
# 输入: 来自 SRI-for-SM 响应的 IMSI
imsi = "001001555123456"

# 步骤 1: 剥离 PLMN 前缀
plmn_prefix = "001001"
subscriber_portion = String.slice(imsi, 6, 9)  # = "555123456"

# 步骤 2: 去除前导零以获取实际 MSISDN
msisdn = String.replace_leading(subscriber_portion, "0", "")  # =
"555123456"
```

反向映射示例:

输入 **IMSI**	**PLMN** 前缀	订阅者部分	去除前导零	最终 **MSISD**
`"001001555123456"`	`"001001"`	`"555123456"`	`"555123456"`	`"5551234`
`"001001000000099"`	`"001001"`	`"000000099"`	`"99"`	`"99"`
`"001001999999999"`	`"001001"`	`"999999999"`	`"999999999"`	`"9999999`

此映射的属性:

- □ □□□□□□□ MSISDN 始终生成相同的 IMSI
- □ □□□□□□□ IMSI 转换回 MSISDN
- □ □□□□□□□□ `hlr_imsi_plmn_prefix`
- □ □□□□□□□ IMSI 从不暴露
- □ □□□□□□□□□□□□□□□ API 调用
- □ □□ **15** 位: IMSI 始终恰好为 15 位数字

MSISDN 输入处理:

当 HLR 接收到 SRI-for-SM 请求时，MSISDN 会经历 TBCD 解码:

1. **TBCD** 解码: 将二进制 TBCD 转换为字符串（可能包含 TON/NPI 前缀，如“91”）
2. 提取数字: 仅保留数字字符，剥离任何非数字字符

3. 标准化：如果超过可用空间，则保留右侧数字；如果较短，则左填充零
4. 编码：连接 PLMN 前缀 + 标准化的 MSISDN

安全考虑：

在 SRI-for-SM 响应中回的合成 IMSI 仅用于路由目的。它们不是存储在 HLR 订阅者数据库中的真实 IMSI。这为隐私保护提供了额外的保护层，因为真实订阅者 IMSI 仅在绝对必要时暴露（例如，在需要真实身份验证向量的 UpdateLocation 或 SendAuthenticationInfo 操作中）。

响应流：

```
1. SMSc → HLR: SRI-for-SM 请求
   - MSISDN（TBCD）："91123456789"（包含 TON/NPI）

2. HLR 处理：
   - TBCD 解码："91123456789"
   - 提取数字："91123456789"（11 位数字）
   - 适应 9 位数字："555123456"（保留右侧 9 位）
   - 添加 PLMN："001001" + "555123456" = "001001555123456"
   - 从配置中获取 SMSC GT："5551234567"

3. HLR → SMSc: SRI-for-SM 响应
   - IMSI："001001555123456"（合成，始终 15 位）
   - 网络节点编号："5551234567"（发送 MT-ForwardSM 的位置）

4. SMSc 将 MT-ForwardSM 发送到 "5551234567"，IMSIN 为 "001001555123456"
```

配置：

以下参数在 `config/config.exs` 中使用：

```
# PLMN 前缀：MCC（001 = 测试网络）+ MNC（01 = 测试运营商）
hlr_imsi_plmn_prefix: "001001",

# NSN 提取（如果 MSISDN 包括国家代码）
hlr_msisdn_country_code: "1",       # 用于反向映射（IMSI→MSISDN）
hlr_msisdn_nsn_offset: 1,           # 跳过 1 位国家代码
hlr_msisdn_nsn_length: 10           # 提取 10 位 NSN
```

NSN 提取配置：

□□□□ MSISDN □□□□□□□□□□“99988000088”□□□□□“88000088”□□□□□□□ NSN □□□

- **`hlr_msisdn_nsn_offset`**□NSN □□□□□□□□□□□□□□□□□□□
- **`hlr_msisdn_nsn_length`**□□□ MSISDN □□□□□□□

□□□

□□	□□□□	**MSISDN □□**	**nsn_offset**	**nsn_length**	□□□ **NSN**
1 □ CC	`"9"`	`"95551234567"`	`1`	`10`	`"5551234567"`
2 □ CC	`"99"`	`"99412345678"`	`2`	`9`	`"412345678"`
3 □ CC	`"999"`	`"99988000088"`	`3`	`8`	`"88000088"`

□□□□□

1. **MSISDN → IMSI**□□ MSISDN □□ NSN□□□□□□□ PLMN □□

```
MSISDN: "99988000088"
NSN: String.slice("99988000088", 3, 8) = "88000088"
□□□ NSN: "088000088"□9 □□□□
IMSI: "547050" + "088000088" = "547050088000088"
```

2. **IMSI → MSISDN**□□□ PLMN □□□□□□□□□□□□□□□□

```
IMSI: "547050088000088"
□□□□□□: "088000088"
□□□: "88000088"
MSISDN: "+999" + "88000088" = "+99988000088"
```

API 要求：无 - SRI-for-SM 使用计算值和配置。无需后端 API 调用。

字段源摘要

源类型	描述	示例
OmniHSS API	来自 OmniHSS 订阅者数据库的动态数据	IMSI、MSISDN、来自 circuit_session 的服务 VLR/MSC
config.exs	OmniSS7 配置参数	`smsc_service_center_gt_address`、`camel_service_key`、`isd_network_access_mode`
静态	响应生成器中的硬编码值	订阅者状态、承载服务、SS 代码
请求	从传入的 MAP 请求中提取的字段	来自 UpdateLocation 的 IMSI，来自 SRI 的 MSISDN
计算	使用逻辑派生的值	SRI-for-SM 中的合成 IMSI（hlr_imsi_prefix + NSN）

配置依赖关系

在 **config/config.exs** 中必需：

- `hlr_service_center_gt_address` - 用于 UpdateLocation 响应
- `smsc_service_center_gt_address` - 用于 SRI-for-SM 响应（MT-ForwardSM 应路由到的位置）

在 **config/config.exs** 中可选（带默认值）：

- `camel_service_key` - 默认值：`11_110`
- `camel_trigger_detection_point` - 默认值：`:termAttemptAuthorized`

- `isd_network_access_mode` - 默认值: `:packetAndCircuit`
- `isd_send_ss_data` - 默认值: `true`
- `isd_send_call_barring` - 默认值`true`
- `hlr_imsi_plmn_prefix` - 默认值: `"001001"`（用于 MSISDN↔IMSI 映射的 PLMN 前缀）

来自 **OmniHSS** 的必需:

OmniHSS 必须提供 REST API 端点以:

- 按 IMSI 和 MSISDN 查询订阅者
- 更新电路会话位置（VLR/MSC 分配）
- 生成身份验证向量
- 查询订阅者状态和服务配置

相关文档

OmniSS7 文档:

- ← 回主文档
- 常见功能指南
- MAP 客户端指南
- 技术参考
- 配置参考

OmniHSS 文档：有关订阅者管理、配置、身份验证配置和管理操作的信息，请参考 **OmniHSS** 产品文档。OmniHSS 包含所有订阅者数据库逻辑、身份验证算法、服务配置规则和多 IMSI 管理能力。

OmniSS7 由 Omnitouch 网络服务提供

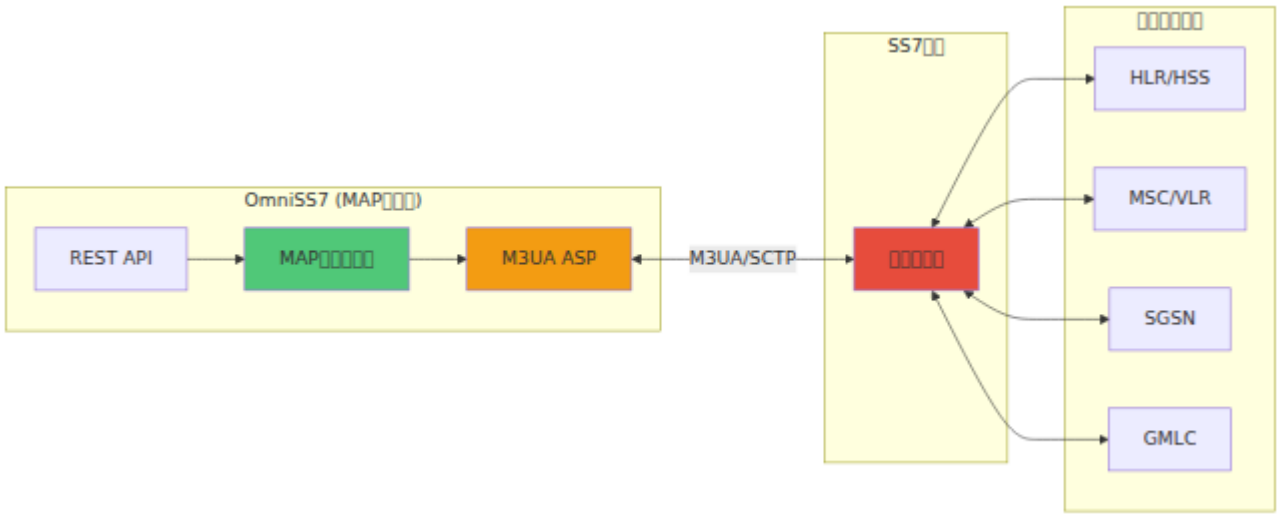
MAP客户端配置□南

← □回主文档

本指南涵盖如何将OmniSS7作为**MAP**客户端进行操作。在此角色中，OmniSS7作为应用服务器进程（ASP）连接到SS7网络。它向HLR、MSC/VLR、SGSN、GMLC和其他核心网络元素发出MAP（移动应用部分）操作。协议行为遵循3GPP TS 29.002。

目录

1. 什么是MAP客户端模式?
2. 启用MAP客户端模式
3. MAP操作参考
4. 常见操作示例
5. 通过API发送请求
6. 指标和监控
7. 故障排除



什么是**MAP**客户端模式?

MAP客户端模式允许OmniSS7作为应用服务器进程（**ASP**）通过**SCTP**连接到**M3UA**对等体（**STP**或**SGP**）并交换**MAP**消息。每个操作都作为HTTP端点暴露。API将请求码为TCAP/MAP对话，并通过SCCP和M3UA传输到SS7网络。然后，OmniSS7解码网络的响应并同步回给调用者。

典型用途包括:

- 移动管理：更新位置、取消位置、清除MS、提供漫游号码
- 呼叫处理：发送路由信息、提供用户信息
- 短信：SRI-for-SM、MO/MT-ForwardSM、交付状态报告
- 身份验证：发送身份验证信息
- 用户数据：插入/删除用户数据、随时查询
- 补充服务：注册/删除/激活/查询SS，密码处理
- 位置服务（**LCS**）和**GPRS**路由

网络架构

启用**MAP**客户端模式

MAP客户端模式在`config/runtime.exs`中配置。有关完整的M3UA连接参考，请参见 M3UA连接参数。

```
config :omniss7,
  # □□MAP□□□□□
  map_client_enabled: true,

  # MAP□□□□M3UA□□□□□ASP□□□□□STP/SGP□
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},
    remote_port: 2905,
    routing_context: 1
  }
```

`map_client_m3ua`参数

参数	类型	必需	默认值	描述
`mode`	字符串	是	-	M3UA角色。`"ASP"`用于客户端模式（连接到STP/SGP）。
`callback`	元组	是	-	`{Module, Function, Args}`，每个传入的M3UA有效负载调用。对于MAP客户端，这是`{MapClient, :handle_payload, []}`。
`process_name`	原子	是	-	M3UA ASP进程的注册名称。
`local_ip`	元组	是	-	本地SCTP绑定地址，作为IP元组，例如`{10, 0, 0, 100}`。
`local_port`	整数	否	2905	本地SCTP端口。2905是IANA注册的M3UA端口。
`remote_ip`	元组	是	-	远程STP/SGP的IP地址，作为元组。
`remote_port`	整数	否	2905	远程STP/SGP的SCTP端口。
`routing_context`	整数	否	-	M3UA路由上下文值，当对等体需要时。

> 顶级键：`map_client_enabled`（布尔值，默认`false`）启用MAP客户端；`map_client_m3ua`（映射）包含上述连接。

您需要单独配置出站操作的SCCP寻址（呼叫/被叫方全球标题、SSN、点码）。请参见配置参考。

MAP操作参考

下面的每个操作都可以通过`POST /api/<endpoint>`访问。操作码是来自3GPP TS 29.002的MAP操作码。请求是同步的：API会阻塞，直到SS7响应到达或请求超时（请参见响应代码）。

移动管理

操作	端点	操作码	描述
更新位置	`/api/updateLocation`	2	注册用户的服务VLR与HLR。
取消位置	`/api/cancelLocation`	3	指示VLR删除用户记录。
提供漫游号码	`/api/prn`	4	从服务MSC获取MSRN。
清除MS	`/api/purgeMS`	67	标记用户在HLR处为已清除。
发送身份识别	`/api/sendIdentification`	55	从先前的VLR检索IMSI/身份验证数据。
重置	`/api/reset`	37	HLR发起的重置，针对VLR。
恢复数据	`/api/restoreData`	57	VLR请求���HLR恢复用户数据。

呼叫处理与用户数据

操作	端点	操作码	描述
发送路由信息	`/api/sri`	22	查询HLR以获取语音呼叫路由信息。
提供用户信息	`/api/provideSubscriberInfo`	70	从VLR请求用户状态/位置。
删除用户数据	`/api/deleteSubscriberData`	8	删除VLR处的用户数据。
随时查询	`/api/anyTimeInterrogation`	71	gsmSCF在HLR查询用户信息/位置。
随时订阅查询	`/api/anyTimeSubscriptionInterrogation`	62	查询HLR处的订阅数据。
随时修改	`/api/anyTimeModification`	65	修改HLR处的订阅数据。
注意用户数据已修改	`/api/noteSubscriberDataModified`	5	HLR通知gsmSCF数据已更改。
注意MM事件	`/api/noteMM-Event`	89	向gsmSCF报告移动管理事件。

短信

操作	端点	操作码	描述
发送短信路由信息	`/api/sri-for-sm`	45	查询HLR以获取用于短信交付的服务节点。
MT-转发短信	`/api/MT-forwardSM`	44	将移动终止的短信交付给服务MSC/SGSN。
MO-转发短信	`/api/forwardSM`	46	向SMSC提交移动发起的短信。
发送短信（PDU）	`/api/sendSM`, `/api/deliverPDU`	44	构建并交付SMS-DELIVER PDU。
报告短信交付状态	`/api/reportSM-DeliveryStatus`	47	向HLR报告短信交付结果。
准备接收短信	`/api/readyForSM`	66	通知HLR某个MS可以接收短信。
警报服务中心	`/api/alertServiceCentre`	64	警报SMSC，通知某个用户可用。

身份验证与设备

操作	端点	操作码	描述
发送身份验证信息	`/api/send-auth-info`	56	从HLR检索身份验证向量。
发送IMSI	`/api/sendIMSI`	58	将MSISDN解析为HLR处的IMSI。
检查IMEI	`/api/checkIMEI`	43	查询设备状态。

补充服务

操作	端点	操作码	描述
注册SS	`/api/registerSS`	10	注册补充服务（例如呼叫转移/CFU）。
删除SS	`/api/eraseSS`	11	删除补充服务。
激活SS	`/api/activateSS`	12	激活补充服务。
停用SS	`/api/deactivateSS`	13	停用补充服务。
查询SS	`/api/interrogateSS`	14	查询补充服务的状态。
注册密码	`/api/registerPassword`	17	注册SS密码。
获取密码	`/api/getPassword`	18	检索/验证SS密码�?
转发检查SS指示	`/api/forwardCheckSS-Indication`	38	向VLR进行未确认检查。

位置服务（**LCS**）与**GPRS**

操作	端点	操作码	描述
发送LCS路由信息	`/api/sendRoutingInfoForLCS`	85	GMLC查询HLR以获取LCS路由。
提供用户位置	`/api/provideSubscriberLocation`	83	从服务节点请求用户的位置。
用户位置报告	`/api/subscriberLocationReport`	86	向GMLC报告用户的位置。
发送GPRS路由信息	`/api/sendRoutingInfoForGprs`	24	查询HLR以获取GPRS路由（GGSN选择）。
更新GPRS位置	`/api/updateGprsLocation`	23	注册用户的服务SGSN与HLR。

> **LCS**范围：位置服务操作仅为出站（请求者）。MAP客户端将它们触发到HLR、服务节点或GMLC。该节点不充当LCS □□□（它不会响应传入的ProvideSubscriberLocation或处理传入的SubscriberLocationReport）。编码器携带强制信息元素以及上述核心选项。更广泛的LCS参数集（QoS、优先级、GAD形状、代码字、参考编号）尚未公开。

CAP / CAMEL与原始注入

操作	端点	描述
CAP InitialDP	`/api/cap/initialDP`	向gsmSCF触发CAMEL对话。
CAP Connect / Continue / ReleaseCall / ApplyCharging	`/api/cap/connect`, `/api/cap/continue`, `/api/cap/releaseCall`, `/api/cap/applyCharging`	CAMEL呼叫控制操作。请参见CAMEL网关指南。
原始SCCP	`/api/raw/sccp`	注入手工构建的SCCP有效负载（测试/诊断）。
原始M3UA	`/api/raw/m3ua`	注入手工构建的M3UA有效负载（测试/诊断）。

常见操作示例

发送短信路由信息（**SRI-for-SM**）

查询HLR以获取用于交付短信的服务节点。有关HLR端如何处理此请求，请参见HLR指南中的SRI-for-SM。

SCCP寻址（**SMSc**短信交付路径）。默认情况下，SRI-for-SM是通过**GT**路由到目标**MSISDN**发送的（称为SSN 6/HLR），这要求STP为该号码范围持有GTT条目。对于本网用户，SMSc则直接在配置的**HLR GT**上寻址SRI。然后，它在HLR的自身（已可路由）对等地址上路由，因此不需要每个范围的GTT，并且HLR□回真实的服务MSC。这是根据目标选择的：

□□□`:omniss7`□	□□	□□
`hlr_gt_address`	□□□□SRI□□□□HLR□GT	`"61455501111"`
`home_msisdn_prefixes`	□□MSISDN□□□□□□□□HLR GT	`["6777"]`

□`home_msisdn_prefixes`□□□□□□□□□□□`hlr_gt_address`□□□□□□HLR GT□□□SSN 6□□□□□□□□□□□MSISDN□□□GT□□□SRI-for-SM□□□□`sccp_opts`□`:called_party`, `:called_ssn`□□□□□□□□□□SCCP□□□□

□□□ `POST /api/sri-for-sm`

□□□

```
{
  "msisdn": "447712345678",
  "serviceCenter": "447999123456"
}
```

□□□

```
{
  "result": {
    "imsi": "234509876543210",
    "locationInfoWithLMSI": {
      "networkNode-Number": "447999555111"
    }
  }
}
```

cURL□

```
curl -X POST http://localhost/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{"msisdn": "447712345678", "serviceCenter": "447999123456"}'
```

□□□□□□□SRI□

□□□ `POST /api/sri`

□□□

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456"
}
```

□□□□□□□PRN□

□□□ `POST /api/prn`

□□□

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456",
  "msc_number": "447999555111",
  "imsi": "234509876543210"
}
```

□□□□□□□□

□□□ `POST /api/send-auth-info`

□□□

```
{
  "imsi": "234509876543210",
  "vectors": 5
}
```

□□□

```
{
  "result": {
    "authenticationSetList": [
      {
        "rand": "0123456789ABCDEF0123456789ABCDEF",
        "xres": "ABCDEF0123456789",
        "ck": "0123456789ABCDEF0123456789ABCDEF",
        "ik": "FEDCBA9876543210FEDCBA9876543210",
        "autn": "0123456789ABCDEF0123456789ABCDEF"
      }
    ]
  }
}
```


□□□□

□□□□□□VLR□□□HLR□HLR□□□□□□□□□□□□□□□□□□□□HLR□□□□□□□□□□

□□□ `POST /api/updateLocation`

□□□


```
{
  "imsi": "234509876543210",
  "vlr": "447999555111"
}
```


□□□□□ATI□

□□□ `POST /api/anyTimeInterrogation`

□□□


```
{
  "hlr_gt": "447999123456",
  "msisdn": "447712345678"
}
```



操作类别

通过API发送请求

使用Swagger UI

Swagger UI提供了每个端点的交互式界面。

1. 导航到`http://your-server/swagger`。
2. 选择一个端点（例如`/api/sri-for-sm`）并点击试用。
3. 填写请求体并点击执行。
4. 解码的SS7响应显示在下面。

原始OpenAPI文档可在`http://your-server/swagger.json`获取。

API响应代码

代码	含义
200	成功：解码结果在响应体中。
400	错误请求：参数无效或缺失。
504	网关超时：在请求超时内没有SS7响应（默认10秒）。

某些操作是未确认的（立即发送），并在消息发送后立即□回。

MAP客户端□标

所有指标都在Prometheus端点（`/metrics`）上导出。

□标：`map_requests_total` 类型：计数器 描述：发送的总MAP请求。标签：`operation`：例如`sri`、`sri_for_sm`、`prn`、`authentication_info`、`update_location`。

□标：`map_request_errors_total` 类型：计数器 描述：结��为错误或超时的总MAP请求。标签：`operation`。

□标：`map_request_duration_milliseconds` 类型：直方图 描述：MAP请求往□时间的分布。标签：`operation`。

□标：`map_pending_requests` 类型：计数器 描述：当前等待响应的MAP请求。

示例Prometheus查询

```
# 最近一小时的SRI-for-SM请求
increase(map_requests_total{operation="sri_for_sm"}[1h])

# 平均SRI响应时间
rate(map_request_duration_milliseconds_sum{operation="sri"}[5m])
  / rate(map_request_duration_milliseconds_count{operation="sri"}
[5m])

# 每个操作的错误率
sum(rate(map_request_errors_total[5m])) by (operation)

# 当前待处理请求
map_pending_requests
```

故障排除MAP客户端

请求超时（504）

症状： API□回504；没有来自HLR/MSC的响应。

可能的原因：

- M3UA与STP/SGP的关联未处于活动状态。
- STP没有目标全球标题的路由。
- SCCP寻址不正确（被叫/呼叫方GT、SSN）或路由上下文。

解决方案：

1. 在仪表板上确认M3UA关联状态或通过STP指南状态视图确认。
2. 验证与STP的网络连接（SCTP）。
3. 确认目标全球标题是可路由的，并且目标子系统（例如HLR SSN 6）���于服务状态。
4. 查看日志以获取SCCP□回消息（见下文）。

SCCP返回/错误消息

症状：日志显示SCCP单元数据服务（UDTS）□回消息。

常见返回原因：

- 地址没有翻译：全球标题不在STP路由表中。
- 子系统故障：目标子系统（例如HLR SSN 6）不可用。
- 网络故障/拥塞：瞬时网络问题。

解决方案：

- 确认目标GT的STP路由/翻译配置。
- 验证目标子系统是否正常运行。
- 在拥塞消除后重试。

相关文档

- ← 回主文档
- 配置参考：所有配置参数
- 常见功能指南：Web UI、API、监控
- STP指南：路由和全球标题翻译
- HLR指南：HLR端操作处理
- 短信中心指南：短信交付
- CAMEL网关指南：CAP/CAMEL操作

短信中心 (SMSc) 配置□南

← □回主文档

本指南提供了使用 OmniSS7 作为 短信中心 (SMSc) 前端与 OmniMessage 作为后端消息存储和传递平台的详细配置。

OmniMessage 集成

OmniSS7 SMSc 模式作为 SS7 信令前端，与 OmniMessage（一个运营商级 SMS 平台）进行接口。该架构分离了关注点：

- OmniSS7 (SMSc 前端)：处理所有 SS7/MAP 协议信令、SCCP 路由和网络通信
- OmniMessage (SMS 后端)：管理消息存储、排队、重试逻辑、交付跟踪和路由决策

为什么选择 OmniMessage？

OmniMessage 提供运营商级 SMS 消息传递能力，具有以下功能：

- 消息队列管理：持久存储，具有可配置的重试逻辑和优先级排队
- 交付跟踪：实时交付状态、交付报告 (DLR) 和失败原因跟踪
- 多 SMSc 支持：多个前端实例可以连接到单个 OmniMessage 后端以实现负载均衡和冗余
- 路由智能：基于目的地、发送者、消息内容和时间的高级路由规则
- 速率限制：每条路由的 TPS（每秒事务）控制，以防���网络拥堵
- API 优先设计：用于与计费系统、客户门户和第三方应用集成的 RESTful HTTP API
- 分析与报告：消息量统计、交付成功率和性能指标

所有消息数据、交付状态和路由配置都存储并由 OmniMessage 管理。OmniSS7 通过 HTTPS API 调用查询 OmniMessage，以检索待处理消息、更新交付状态并注册为活动前端。

重要：OmniSS7 SMSc 模式仅为 信令前端。所有消息路由逻辑、队列管理、重试算法、交付跟踪和业务规则均由 OmniMessage 处理。本指南涵盖 OmniSS7 中的 SS7/MAP 协议配置。有关消息路由、队列配置、交付报告、速率限制和分析的信息，请 参考 OmniMessage 文档。

目录

1. OmniMessage 集成
2. 什么是短信中心模式？
3. 启用 SMSc 模式
4. HTTP API 配置
5. 短信消息流
6. 警报服务中心处理
7. 循环预防
8. SMSc 订阅者跟踪
9. VLR 地址缓存
10. 自动刷新配置
11. 指标和监控
12. 故障排除

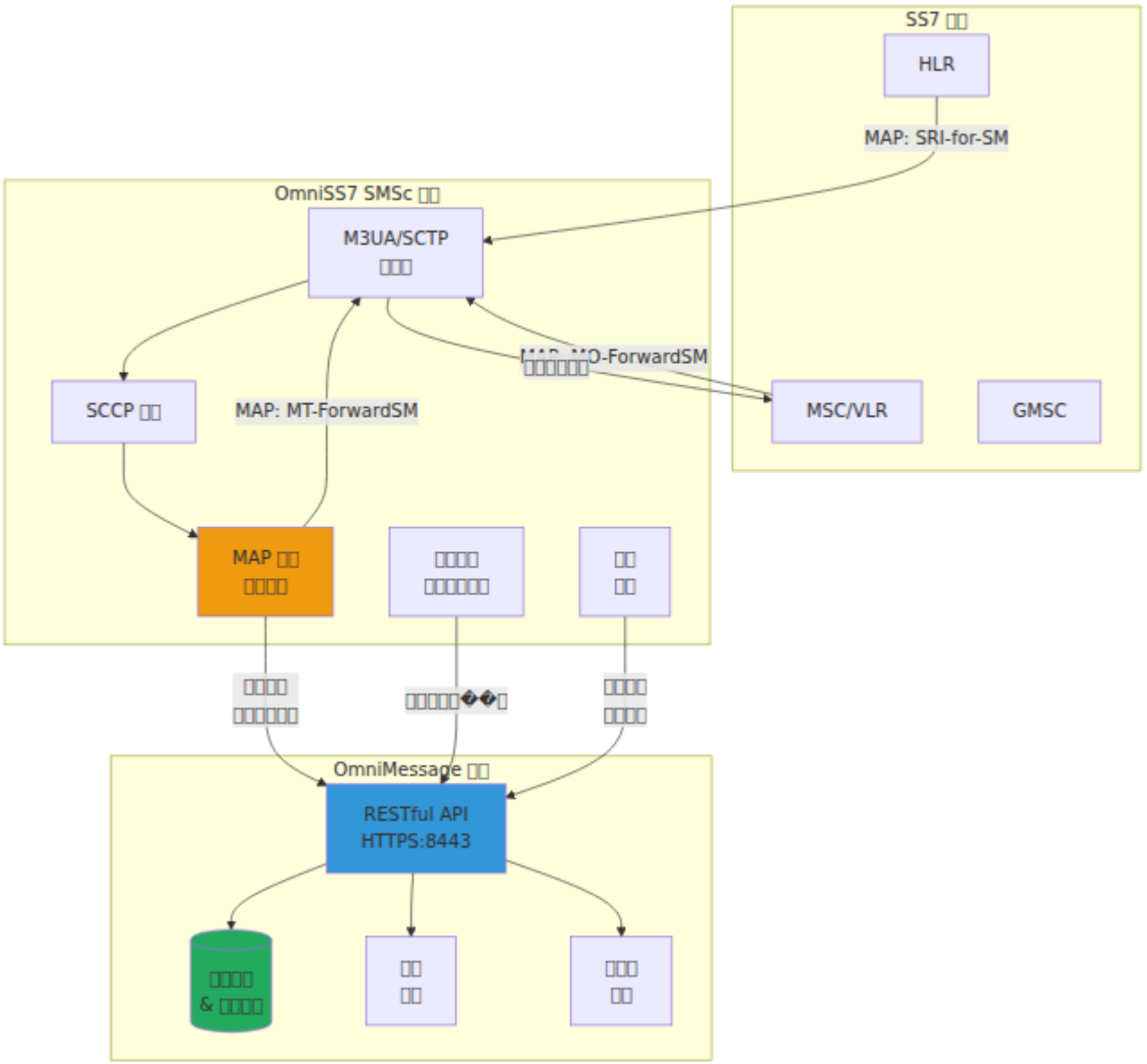
什么是短信中心模式？

注意：本节仅涵盖 OmniSS7 的 SS7 信令配置。有关消息路由规则、队列管理、交付跟踪和业务逻辑配置，请参见 OmniMessage 产品文档。

短信中心模式使 OmniSS7 能够作为 SMSc 运行，用于：

MT-SMS 交付：将移动终止的 SMS 交付给订阅者
MO-SMS 处理：接收和路由移动发起的 SMS
消息排队：基于数据库的消息队列，具有重试逻辑
自动刷新：从队列自动交付 SMS
交付报告：跟踪消息交付状态

短信中心架构



启用 SMSc 模式

OmniSS7 可以以不同的模式运行（STP、HLR、SMSc）。通过一组模式标志和 config/config.exs 中的相关配置选择模式，这是应用程序的默认配置文件。

切换到 SMSc 模式

要将 OmniSS7 作为 SMSc 运行，请设置模式标志以启用 SMSc 功能，然后添加 SMSc 特定的配置：

1. 打开 `config/config.exs`
2. 设置 `config :omniss7` 下的模式标志：
   - `map_client_enabled: true`：路由/MAP 功能所需
   - `smsc_mode_enabled: true`：启用 SMSc 特定处理
   - `hlr_mode_enabled: false`：禁用 HLR 功能
3. 添加 SMSc 配置参数（API 端点、SMSc 名称、GT 地址、自动刷新设置、M3UA 连接），如下所示
4. 自定义 您的部署参数
5. 重启 应用程序以使更改生效

> 注意：`config/runtime.exs` 是一个仅用于自动测试的最小存根。它不包含操作模���定义。所有模式和 SMSc 配置都位于 `config/config.exs` 中（在 `config/test.exs` 等文件中具有特定环境的覆盖）。生产部署提供自己的 `config/config.exs`。

SMSc 模式配置

完整的 SMSc 配置（在 `config/config.exs` 中）如下所示：

```
config :omniss7,
  # □□□□ - □□ STP + SMSc □□
  # □□□map_client_enabled □ true□□□ SMSc □□□□□□□
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  # OmniMessage □□ API □□
  smsc_api_base_url: "https://omnimessage.example.com:8443",
  # □□□□□□□□□ SMSc □□
  smsc_name: "ipsmgw",
  # □□□□□□□□□□ GT □□
  smsc_service_center_gt_address: "5551234567",

  # □□□□□□□□□□ SMS □□□□□□
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  # M3UA □□□□
  # □□ ASP □□□□□/□□ MAP SMS □□
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    # □□□□□SMSc □□□
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    # □□ STP □□
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

Web UI □□□ `config :control_panel` □□□□□□□□ `config/config.exs` □□□□□ `use_additional_pages` □□□□□□ OmniSS7 □□□□ 16 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□peer-status □□□□□ **"Peers"**□

```
config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 □□"},
    {SS7.Web.TestClientLive, "/client", "SS7 □□□"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "Peers"},
    {SS7.Web.SCTPConnectionsLive, "/sctp", "SCTP □□"},
    {SS7.Web.RoutingLive, "/routing", "□□"},
    {SS7.Web.RoutingTestLive, "/routing_test", "□□□□"},
    {SS7.Web.SmscLinksLive, "/smsc_links", "SMSc □□"},
    {SS7.Web.SmscSubscribersLive, "/smsc_subscribers", "SMSc □□
□"},
    {SS7.Web.HlrLinksLive, "/hlr_links", "HLR □□"},
    # ... □□□□□□□□□□□□CAMEL□□□□MSISDN/IMSI □□□□
  ],
  page_order: [
    "/application", "/configuration", "/events", "/logs",
"/client", "/m3ua",
    "/routing", "/routing_test", "/smsc_links",
"/smsc_subscribers", "/hlr_links",
    # ... □□□□□□
  ]
```

□□□□□□□□

□□□□□□□□□□□□□□□□□□ ���□□□□

□□	□□	□□□	□□
`smsc_api_base_url`	□□□□	*□□*	**OmniMessag** □□ API □□
`smsc_name`	□□□□	`"{hostname}_SMSc"`	□□ SMSc □□□□□□□□□
`smsc_service_center_gt_address`	□□□□	*□□*	□□□□□□□□□
`auto_flush_enabled`	□□□□	`true`	□□□□□□□□
`auto_flush_interval`	□□	`10_000`	□□□□□□□□□□
`auto_flush_dest_smsc`	□□□□	`nil`	□□□□□□□ SMSc □□□□□□ `nil` = □□□□□□□□□□
`auto_flush_tps`	□□	`10`	□□□□□□□□□/□□
`local_ip`	□□	*□□*	□□ SMSc □□□ IP □□
`local_port`	□□	`2905`	□□ SCTP □□
`remote_ip`	□□	*□□*	SS7 □□□ STP IP □□

参数	类型	默认值	描述
`remote_port`	整数	`2905`	远程 SCTP 端口
`routing_context`	整数	`1`	M3UA 路由上下文 ID

启用 **SMSc** 模式后会发生什么

当 `smsc_mode_enabled: true` `map_client_enabled: true` 时，Web UI 将显示：

- **SS7** 事件 - 事件日志
- **SS7** 客户端 - MAP 操作测试
- **Peers** - 连接状态（M3UA/SCTP 对等体）
- - 路由表管理（启用 STP）
- - 路由测试（启用 STP）
- **SMSc** 链接 - SMSc API 状态 + SMS 队列管理 ← *SMSc*
- - 系统监控
- - 配置查看器

HLR 链接 标签将被隐藏。

重要说明

- SMSc 模式需要 `map_client_enabled: true` 以获得路由功能
- **OmniMessage** 后端：OmniMessage API 后端必须在配置的 `smsc_api_base_url` 可访问
- 前端注册：系统每 **~85** 秒 通过 `SMS.FrontendRegistry` 模块自动向 OmniMessage 注册
- **API** 请求超时：所有 OmniMessage API 请求都有一个 硬编码的 **5** 秒超时
- **MAP** 请求超时：SRI-for-SM 有 10 秒的对话超时。MT-ForwardSM 有 **30** 秒的对话超时，以适应 2G/CS 网络上较慢的 VLR 响应
- 自动刷新会在后台自动处理 SMS 队列
- 需要与 STP 建立 M3UA 连接以发送/接收 MAP SMS 操作
- 更改模式后，您必须重启应用程序以使更改生效

Web UI：有关使用 Web 界面的信息，请参见Web UI 指南

API 访问：有关 REST API 文档和 Swagger UI 访问的信息，请参见API 指南

HTTP API 配置

OmniMessage 后端设置

OmniSS7 通过 HTTPS REST API 与 OmniMessage 通信，以管理消息交付、跟踪订阅者状态并注册为活动前端：

config:omniss7,
OmniMessage API 基础 URL
smsc_api_base_url: "https://omnimessage.example.com:8443",
短信中心名称标识符用于注册（如果为空，则默认为 hostname_SMSc）
smsc_name: "omni-smsc01",
短信操作的服务中心 GT 地址
smsc_service_center_gt_address: "5551234567"

配置参数：

参数	类型	必需	默认值	
`smsc_api_base_url`	字符串	是	`"https://localhost:8443"`	C A
`smsc_name`	字符串	否	`""`（使用 `"{hostname}_SMSc"`）	[S
`smsc_service_center_gt_address`	字符串	否	`"5551234567"`	[[C [F [[S

前端注册

系统在启动时自动向 OmniMessage 注册，并通过 `SMS.FrontendRegistry` 模块每 **~85** 秒重新注册。这使 OmniMessage 能够：

- 跟踪活动前端以实现负载均衡
- 监控正常运行时间和健康状态
- 收集配置信息
- 管理多个前端之间的分布式 SMS 路由

实现细节：

- 注册间隔：~85 秒（`@registration_interval 85_000`，硬编码）
- 进程：当 `smsc_mode_enabled: true` 时自动启动

注册有效负载：

```
{
  "frontend_name": "omni-smsc01",
  "configuration": "{...}",
  "frontend_type": "SS7",
  "hostname": "smsc-server01",
  "uptime_seconds": 12345
}
```

注意：前端名称取自 `smsc_name` 配置参数。如果未设置，则默认为 `"{hostname}_SMSc"`。

OmniMessage API 通信

当 OmniSS7 从 SS7 网络接收 MAP 操作或处理消息队列时，它与 OmniMessage 通信以：

- 注册为活动前端 并报告健康状态
- 提交从订阅者接收的移动发起 **(MO)** 消息
- 从队列检索移动终止 **(MT)** 消息 以进行交付
- 更新交付状态 以提供成功/失败报告
- 查询路由信息 以进行消息转发

□□	□□	□□	□□□
`/api/frontends`	POST	□□□□□□	`{"frontend_name"` `"configuration":` `"frontend_type":` `"hostname": "...` `"uptime_seconds"`
`/api/messages_raw`	POST	□□□ SMS □□□□□ TPDU□	`{"rp_originator_` `"...",` `"rp_destination_` `"...", "directio` `"message_body":` `"send_time": ".."` `"smsc_node_name"` `"source_smsc": "` `"source_type": "`
`/api/messages`	GET	□□□□□□	□□□`smsc: <smsc_r`
`/api/messages/{id}`	PATCH	□□□□□□□□□	`{"deliver_time":` `"dest_smsc": "..`
`/api/messages/{id}`	PUT	□□□□□□	`{"dest_smsc": nu`
`/api/locations`	POST	□□/□□□□□□□	`{"msisdn": "..."` `"...", "location` `"ims_capable": t` `"csfb": false, "` `"...", "user_age` `"...", "ran_loca` `"...", "imei": "` `"registered": ".`
`/api/events`	POST	□□□□□□	`{"message_id": .` `"name": "...",` `"description": "`

端点	方法	目的	请求体
`/api/status`	GET	健康检查（路径可以通过 `smsc_api_status_path` 配置，默认 `/api/status`；由 SMSc 链接页面轮询，而不是由 `SMSc.APIClient`）	-

API 响应格式

所有 API 响应使用 JSON 格式，遵循以下约定：

- 成功响应：HTTP 200-201，JSON 主体包含结果数据
- 错误响应：HTTP 4xx/5xx，响应主体中包含错误详细信息
- 时间戳：ISO 8601 格式（例如，`"2025-10-21T12:34:56Z"`）
- 消息 **ID**：整数或字符串标识符

API 客户端模块

SMS 系统由三个主要模块组成：

1. SMSc.APIClient

主要 API 客户端模块，提供与 OmniMessage 的所有 HTTP API 通信：

- `frontend_register/4` - 向 OmniMessage 注册前端
- `insert_message/1` - 插入原始 SMS 消息（接受单个消息映射；发布到 `/api/messages_raw`）
- `insert_location/9` - 插入/更新订阅者位置数据
- `get_message_queue/2` - 从队列中检索待处理消息
- `mark_dest_smsc/3` - 将消息标记为已交付或失败
- `add_event/3` - 为消息添加事件跟踪
- `flush_queue/2` - 处理待处理消息（SRI-for-SM + MT-forwardSM）
- `auto_flush/2` - 持续队列处理循环

2. SMS.FrontendRegistry

处理与后端的定期前端注册：

启动时自动注册
每 ~85 秒重新注册
使用配置中的 smsc_name（回退到主机名）
收集系统配置和正常运行时间信息

3. SMS.Utils

SMS 操作的实用函数：

generate_tp_scts/0 - 生成 TPDU 格式的 SMS 时间戳

□□□□□

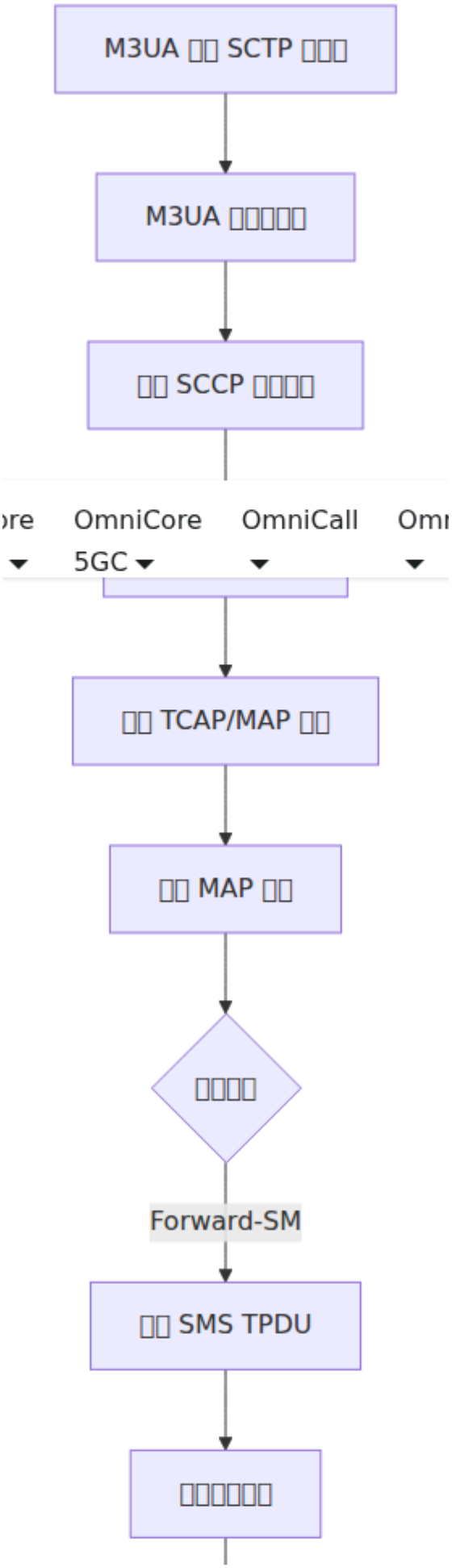
□□ SMS □□□□□□□

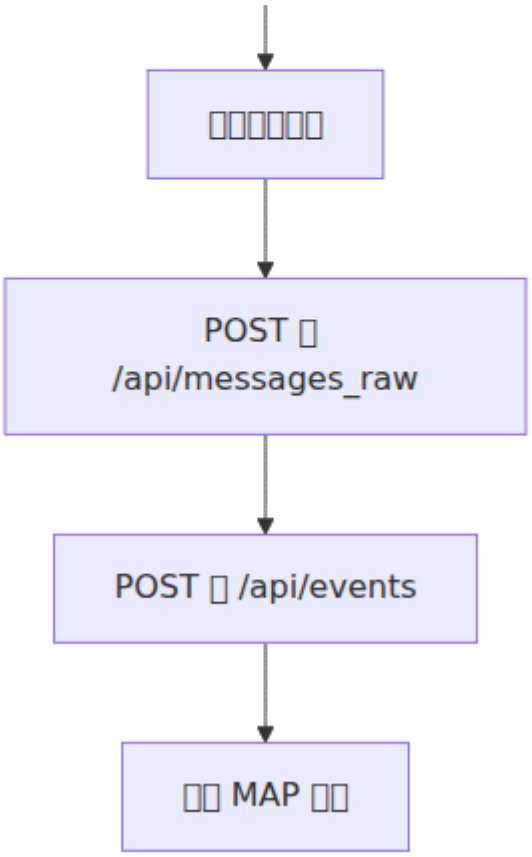

M3UA □□ SCTP □□□
M3UA □□□□□
□□ SCCP □□□□
□□ SCCP □□
□□ TCAP/MAP □□
□□ MAP □□
□□□□
Forward-SM
□□ SMS TPDU
□□□□□□□



出站 **SMS** 流（移动终止）

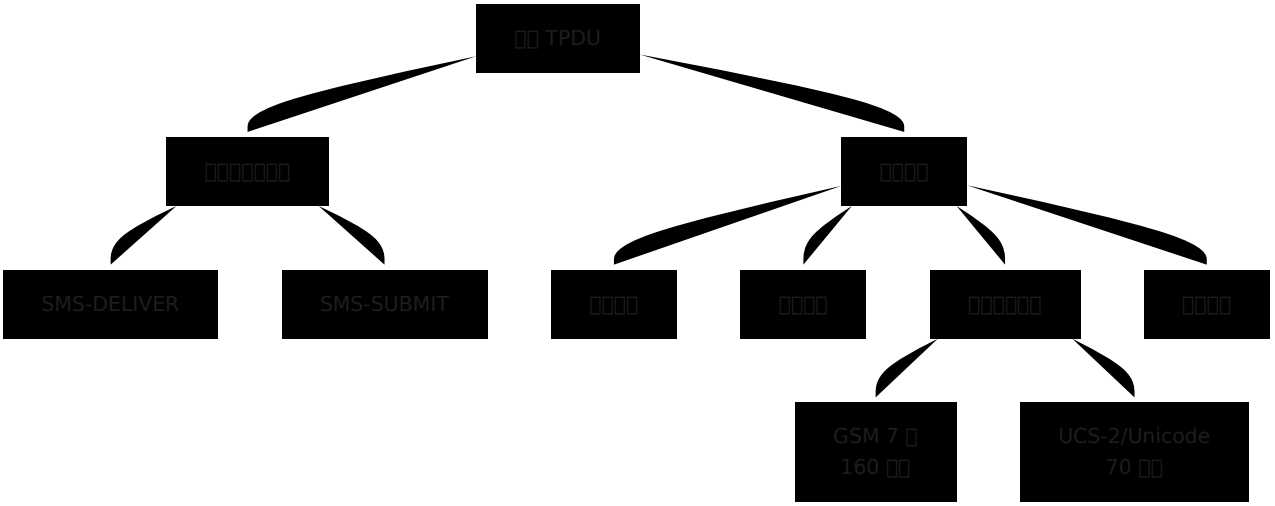
当配置了 HSS/HLR API（`hlr_api_base_url`）时，IP-SM-GW 可以直接查询 HSS API 以获取订阅者的真实 IMSI 和服务 VLR，从而将 MT-SMS 交付给 在网订阅者，而无需进行 SRI-for-SM 往 **VLR** 缓存 在订阅者跟踪器中避免了对同一订阅者的重复 HSS 查找。


M3UA □□ SCTP □□□
M3UA □□□□□
□□ SCCP □□□□
ore
OmniCore
5GC
OmniCall
Omı
□□ TCAP/MAP □□
□□ MAP □□
□□□□
Forward-SM
□□ SMS TPDU
□□□□□□



关键步骤解释：

- **VLR** 缓存检查：在查询 HSS API 之前，系统检查订阅者跟踪器的 VLR 缓存是否有有效条目（IMSI + VLR 地址在 TTL 内）。缓存命中将完全跳过 HSS API。有关详细信息，请参见 VLR 地址缓存。
- **HSS API** 直接交付（在网）：当配置了 `hlr_api_base_url` 时，IP-SM-GW 使用目标 MSISDN 查询 HSS API 以获取订阅者的 真实 **IMSI** 和 服��� **VLR** 地址。如果两者都可用，IP-SM-GW 将 MT-ForwardSM 直接发送到 VLR。这完全绕过了 SRI-for-SM。
- 缺失订阅者（在网）：如果 HSS API 找到订阅者但没有分配服务 VLR，则消息将标记为失败，并带有过期的位置条目。当 HLR 在订阅者重新注册后发送 alertServiceCenter 时，将进行交付重试。
- **SRI-for-SM** 回退（非在网）：当未配置 HSS API 或未在 HSS 中找到订阅者时，使用标准的 SRI-for-SM 流。HLR 将响应一个合成 IMSI 和网络节点号。有关详细信息，请参见 HLR 指南中的 SRI-for-SM。
- 交付失败时清除 **VLR**：当 MT-ForwardSM 交付失败（超时、SCCP 错误等）时，缓存的 VLR 地址将被清除，以便下次尝试重新查询 HSS API 以获取新的 VLR。
- **MT-ForwardSM** 超时：MT-ForwardSM 对话有 **30** 秒超时，以适应较慢的 2G/CS VLR 响应。



SMS TPDU 结构

警报服务中心处理

SMSc 可以接收来自 HLR 的 **alertServiceCenter** 消息，以跟踪订阅者的可达状态。

有关 HLR 如何发送 alertServiceCenter 消息的信息，请参见 HLR 指南中的警报服务中心集成。

什么是 **alertServiceCenter**?

当订阅者在 HLR 执行 UpdateLocation（即，注册到新的 VLR/MSC）时，HLR 可以通过发送 **alertServiceCenter**（MAP 操作码 64）消息来通知 SMSc 系统订阅者现在可达。

配置

位置过期时间在 HLR 中配置：

```
config :omniss7,
  # SMSc 接收到 alertServiceCenter 时的位置过期时间（默认：48 小时）
  hlr_alert_location_expiry_seconds: 172800
```

行为

当 SMSc 接收到 alertServiceCenter 消息时：

1. 解码 **MSISDN**：从消息中提取订阅者的 MSISDN（TBCD 格式）
2. 去除 **TON/NPI** 前缀：删除常见前缀，如 "19"、"11"、"91"（例如，"19123123213" → "123123213"）
3. 解析 **IMSI**：如果配置了 `hlr_api_base_url`，则查询 HSS API 以解析订阅者的真实 **IMSI**。如果 HSS 查找失败或未配置 API，则回退到使用与 SRI-for-SM 相同的映射生成合成 IMSI。真实 IMSI 对于在网 MT 交付给 2G/CS 订阅者至关重要。MSC/VLR 需要在 MT-ForwardSM 中使用正确的 IMSI。
4. **POST** 到 **/api/locations**：使用以下信息更新位置数据库：
   - `msisdn`：订阅者的电话号码（已清理）
   - `imsi`：合成 IMSI
   - `location`：SMSc 名称（例如，"ipsmgw"）
   - `expires`：当前时间 + `hlr_alert_location_expiry_seconds`
   - `csfb`：true（订阅者通过电路交换回退可达）
   - `ims_capable`：false（这是 2G/3G CS 注册，而不是 IMS/VoLTE）
   - `user_agent`：发送警报的 HLR GT（用于跟踪）
   - `ran_location`："SS7"
5. 在 **SMSc** 订阅者跟踪器中跟踪：记录订阅者与 HLR GT，状态=活动，消息计数器为 0
6. 发送 **ACK**：向 HLR 回复 alertServiceCenter 确认

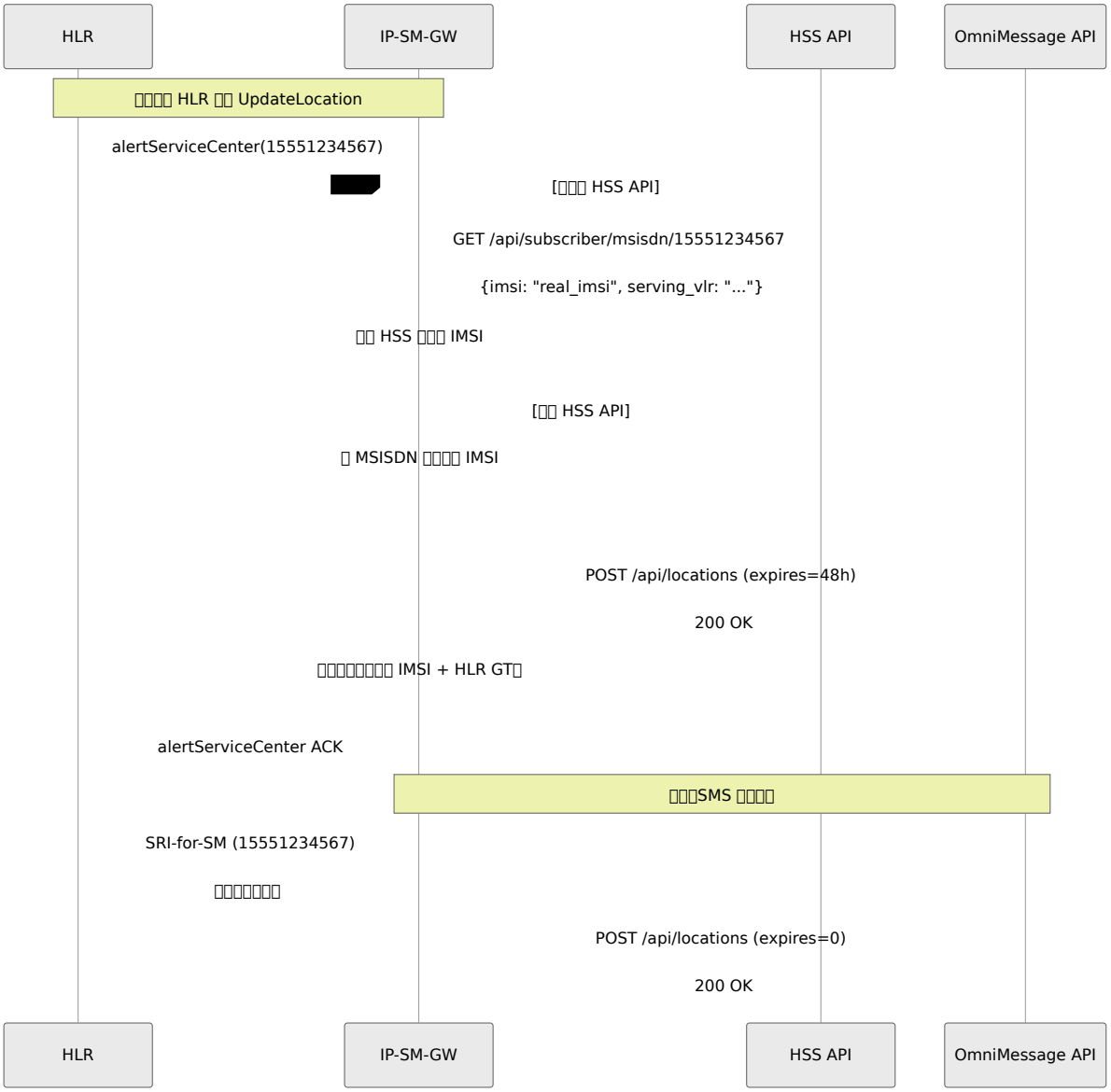
缺失订阅者处理

当 SMSc 尝试交付消息并在 SRI-for-SM 期间收到 "缺失订阅者" 错误时（有关 SRI-for-SM 的更多信息，请参见 HLR 指南中的 SRI-for-SM）：

1. 检测缺失：SRI-for-SM 回 `absentSubscriberDiagnosticSM` 错误
2. 过期位置：POST 到 /api/locations，`expires=0` 以将订阅者标记为不可达
3. 用户代理：设置为 "SS7_AbsentSubscriber" 以标识来源
4. 更新跟踪器：在 SMSc 订阅者跟踪器中将订阅者标记为 `failed`

这确保了位置数据库和跟踪器准确反映订阅者的可达状态。

□□□



API □□

POST /api/locations

```
{
  "msisdn": "15551234567",
  "imsi": "001010123456789",
  "location": "ipsmgw",
  "ims_capable": false,
  "csfb": true,
  "expires": "2025-11-01T12:00:00Z",
  "user_agent": "15551111111",
  "ran_location": "SS7",
  "imei": "",
  "registered": "2025-10-30T12:00:00Z"
}
```

注意：`user_agent` 字段包含发送 alertServiceCenter 的 HLR GT，使 SMSc 能够跟踪提供位置更新的 HLR。

对于缺失的订阅者，`expires` 设置为当前时间（立即过期）。

循环预防

SMSc 实现了 自动循环预防，以避免当消息来自 SS7 网络时的无限消息路由循环，同时仍允许在网订阅者之间的合法 MO→MT 交付。

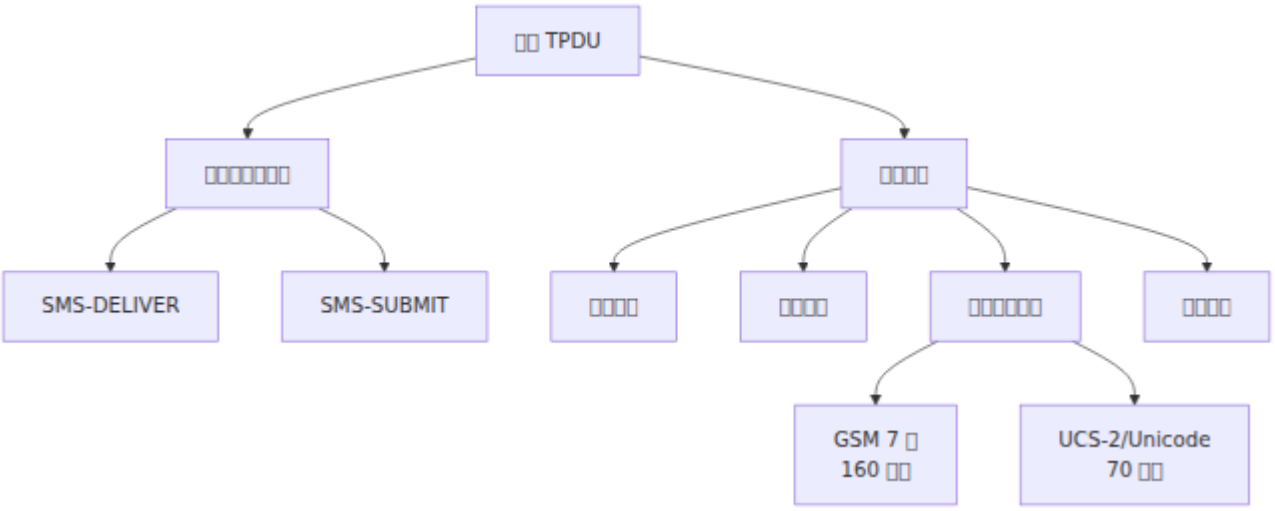
为什么循环预防很重要

当 SMSc 从 SS7 网络接收移动发起（MO）SMS 消息时，它将这些消息插入消息队列，并带有一个 `source_smsc` 字段来标识其来源（例如，`"SS7_MO_15551234567"`）。如果没有循环预防，这些消息可能会：

1. 从 SS7 网络接收 → 使用 `source_smsc` 插入队列，包含 "SS7"
2. 从队列中检索 → 处理以进行交付
3. 发送回 SS7 网络 → 创建循环

它是如何工作的

循环预防逻辑区分 在网 和 非在网 目的地。SMSc 仅在目标订阅者 不在网 时阻止来自 SS7 的消息。这防止了过境路由循环。它仍然允许在同一网络内的合法 MO→MT 交付（例如，MSC 到 MSC 通过 IP-SM-GW）。



在网检测

当消息来自 SS7 时，系统在应用循环预防之前检查目标订阅者是否在网：

1. **VLR** 缓存（最快）：检查订阅者在订阅者跟踪器中是否具有有效的 IMSI 的缓存 VLR 地址
2. **HSS API**（权威）：如果配置了 `hlr_api_base_url` 且缓存未命中，则通过 MSISDN 查询 HSS API。如果订阅者存在于 HSS 中，则他们在网。

如果两个检查都未找到订阅者，则目标被视为 非在网，消息将被阻止。

实现

在处理来自队列的消息时，SMSc 检查 `source_smsc` 字段：

- 如果 **`source_smsc`** 包含 **"SS7"** 且目标为非在网：
  - 跳过消息
  - 添加事件：`"循环预防"`，描述说明目标为非在网
  - 通过 PUT 请求将消息标记为失败
  - 以警告级别记录
- 如果 **`source_smsc`** 包含 **"SS7"** 且目标为在网：
  - 正常处理消息。这是合法的 MO→MT 流。
  - 通过 HSS API / VLR 缓存路径进行交付
- 如果 **`source_smsc`** 不包含 **"SS7"���**

- 正常处理消息（例如，来自 Web API、SMPP 或其他非 SS7 源的消息）

源 SMSC 值

消息可以具有各种 `source_smsc` 值：

来源	示例值	操作
SS7 网络（MO-FSM），非在网目标	`"SS7_MO_15551234567"`	跳过 - 循环预防
SS7 网络（MO-FSM），在网目标	`"SS7_MO_15551234567"`	允许 - 在网 MO→MT 交付
外部 API/SMPP	`"ipsmgw"` 或 `"api_gateway"`	正常处理
其他 SMSc	`"smsc-node-01"`	正常处理

事件跟踪

当由于循环预防而跳过消息时，记录一个事件：

```
{
  "message_id": 12345,
  "name": "循环预防",
  "description": "消息跳过 - source_smsc 'SS7_MO_15551234567' 包含 'SS7' 且目标为非在网，防止消息循环"
}
```

该事件在以下位置可见：

- **Web UI**：SS7 事件页面（`/events`）
- 数据库：通过 API 的 `events` 表
- 日志：警告级别日志条目

配置

循环预防始终启用，无法禁用。这是防止网络因消息循环而中断的关键安全功能。在网旁路需要同时配置 `smsc_mode_enabled: true` 和 VLR 缓存或 `hlr_api_base_url`。

示例场景

场景 **1**：来自 SS7 的 MO-SMS 到非在网订阅者（被阻止）

1. 移动电话 → MSC/VLR → IP-SM-GW（通过 MO-ForwardSM）
2. IP-SM-GW 接收 MO-FSM，插入队列：`source_smsc = "SS7_MO_9991900"`
3. 自动刷新从队列中检索消息
4. 目标未在 VLR 缓存或 HSS 中找到 → 非在网
5. IP-SM-GW 检测到 `"SS7"` 在 `source_smsc` 中 + 非在网目标 → 跳过
6. 记录事件：`"循环预防"`
7. 未发送 SRI-for-SM 或 MT-ForwardSM（循环被阻止）

场景 **2**：来自 SS7 的 MO-SMS 到在网订阅者（允许）

1. 移动电话 → MSC/VLR → IP-SM-GW（通过 MO-ForwardSM）
2. IP-SM-GW 接收 MO-FSM，插入队列：`source_smsc = "SS7_MO_9991900"`
3. 自动刷新从队列中检索消息
4. 目标在 HSS API 中找到 → 在网订阅者
5. IP-SM-GW 允许交付，尽管来源为 SS7
6. HSS 返回真实 IMSI + VLR → MT-ForwardSM 直接发送到 VLR

SMSc 订阅者跟踪

SMSc 包含一个订阅者跟踪器 GenServer，基于 alertServiceCenter 消息和消息交付尝试维护订阅者的实时状态。

目的

跟踪器提供：

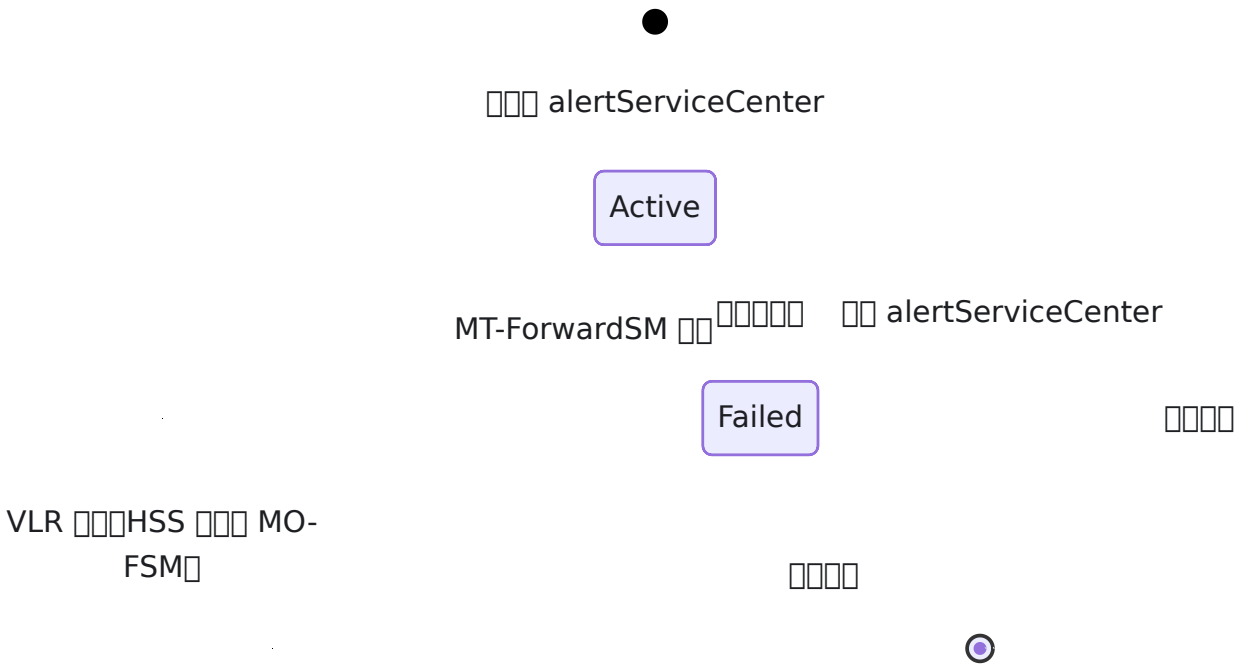
- 可达性监控：当前可达的订阅者

- **HLR** 跟踪：为每个订阅者发送 alertServiceCenter 的 HLR
- 消息计数器：每个订阅者发送/接收的消息数量
- 失败跟踪：当交付尝试失败时将订阅者标记为失败
- **Web UI** 可见性：实时仪表板显示所有跟踪的订阅者

跟踪的信息

对于每个订阅者，跟踪器存储：

字段	描述	示例
`msisdn`	订阅者的电话号码（键）	"15551234567"
`imsi`	订阅者的 IMSI（如果可用，则来自 HSS 的真实 IMSI）	"001010123456789"
`hlr_gt`	发送 alertServiceCenter 的 HLR GT	"15551111111"
`messages_sent`	发送的 MT-FSM 消息计数	5
`messages_received`	接收的 MO-FSM 消息计数	2
`status`	`:active` 或 `:failed`	`:active`
`updated_at`	最后更新的 Unix 时间戳	1730246400
`vlr_address`	缓存的服务 VLR GT 地址	"14155550100"
`vlr_cached_at`	VLR 上次缓存的 Unix 时间戳	1730246400



状态转换

行为

当接收到 **alertServiceCenter** 时：

- 创建或更新订阅者条目
- 设置 `status = :active`
- 记录 HLR GT
- 从 HSS API ���析真实 IMSI（如果配置），否则使用合成 IMSI
- 保留现有 VLR 缓存和消息计数器

当 **HSS API** 返回带 **VLR** 的订阅者时：

- 在跟踪器中缓存 VLR 地址和真实 IMSI
- 后续交付使用缓存的 VLR，而无需重新查询 HSS

当接收到 **MO-ForwardSM** 时：

- 将 SCCP 呼叫 GT 缓存为订阅者的 VLR 地址（MO-ForwardSM 的呼叫 GT 是转发消息的 VLR）
- 如果不存在，则创建跟踪器条目

当 **MT-ForwardSM** 交付失败时：

清除缓存的 VLR 地址，以便下次交付尝试重新查询 HSS API

当 SRI-for-SM 成功时：

增加 messages_sent 计数器
更新 updated_at 时间戳

当 SRI-for-SM 失败时：

设置 status =:failed
保持在跟踪器中以进行监控

当订阅者被移除时：

从 ETS 表中删除
不再出现在 Web UI 中

Web UI - SMSc 订阅者页面

路径：/smsc_subscribers
自动刷新：每 2 秒

注意：此页面仅在 SMSc 模式下运行时可用。在 config/config.exs 中启用 SMSc 配置后，您必须重启应用程序以使路由可用。

SMSc 订阅者页面提供实时监控所有跟踪的订阅者：

功能

1. 订阅者表

MSISDN、IMSI、HLR GT、VLR GT（缓存的服务 VLR 地址）
发送/接收计数器
状态徽章（活动/失败）及颜色编码
最后更新时间戳和持续时间
单个订阅者的移除按钮

2. 摘要统计

- □□□□□□□
- □□□□□□□□
- □□□□□□□□
- □□ HLR □□□

3. □□

- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□

□□□□

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ SMSc □□□□□□□                                    □□: 3                        │
├──────────────────────────────────────────────────────────────────────────────┤
│ MSISDN          IMSI              HLR GT         VLR GT         Msg          │
│                                                                  S/          │
├──────────────────────────────────────────────────────────────────────────────┤
│ 15551234567     001010123456789   15551111111    14155550100    5/2          │
│ 15559876543     001010987654321   15551111111    -              0/0          │
│ 15551112222     001010111222233   15552222222    14155550200    3/1          │
└──────────────────────────────────────────────────────────────────────────────┘

□□□  □□: 3  |  □□: 2  |  □□: 1  |  □□ HLR: 2
```

API □□

□□□□□□□□□□□□□□□□□□□□

```
# 当接收到 alertServiceCenter 时调用
SMSc.SubscriberTracker.alert_received(msisdn, imsi, hlr_gt)

# 增加消息计数器
SMSc.SubscriberTracker.message_sent(msisdn)
SMSc.SubscriberTracker.message_received(msisdn)

# 标记为失败（SRI-for-SM 失败）
SMSc.SubscriberTracker.mark_failed(msisdn)

# 从跟踪中移除
SMSc.SubscriberTracker.remove_subscriber(msisdn)

# 查询函数
SMSc.SubscriberTracker.get_active_subscribers()
SMSc.SubscriberTracker.get_subscriber(msisdn)
SMSc.SubscriberTracker.count_subscribers()
SMSc.SubscriberTracker.clear_all()
```

集成

跟踪器自动集成：

- **alertServiceCenter** 处理程序：在成功位置更新时调用 `alert_received/3`
- **SRI-for-SM** 处理程序：在成功路由时增加 `messages_sent`
- 缺失订阅者处理程序：当订阅者缺失时调用 `mark_failed/1`
- 未知订阅者错误：当 SRI-for-SM 失败时调用 `mark_failed/1`

VLR 地址缓存

当作为 IP-SM-GW 运行并配置 HSS API 时，订阅者跟踪器缓存 VLR 地址，以避免在每次 MT 交付尝试时查询 HSS。

它是如何工作的

VLR 缓存由两个来源填充：

1. **HSS API** 查找：当交付管道查询 HSS API 并获得带有服务 VLR 的订阅者时，IMSI 和 VLR 地址将被缓存到订阅者跟踪器中。
2. **MO-ForwardSM**：当来自订阅者的 MO-ForwardSM 到达时，SCCP 呼叫方 GT（即转发消息的 VLR/MSC）被缓存为订阅者的 VLR 地址。

在后续的 MT 交付尝试中，首先检查缓存。如果存在有效条目（IMSI 和 VLR 均存在，且在 TTL 内），则完全跳过 HSS API。

缓存失效

当发生以下情况时，VLR 缓存将被清除：

- **MT-ForwardSM** 交付失败（超时、SCCP 错误、VLR 拒绝）：缓存的 VLR 可能已过期
- **TTL** 过期：TTL 可以通过 `vlr_cache_ttl_seconds` 配置。缓存将视为未命中的条目超过 TTL。

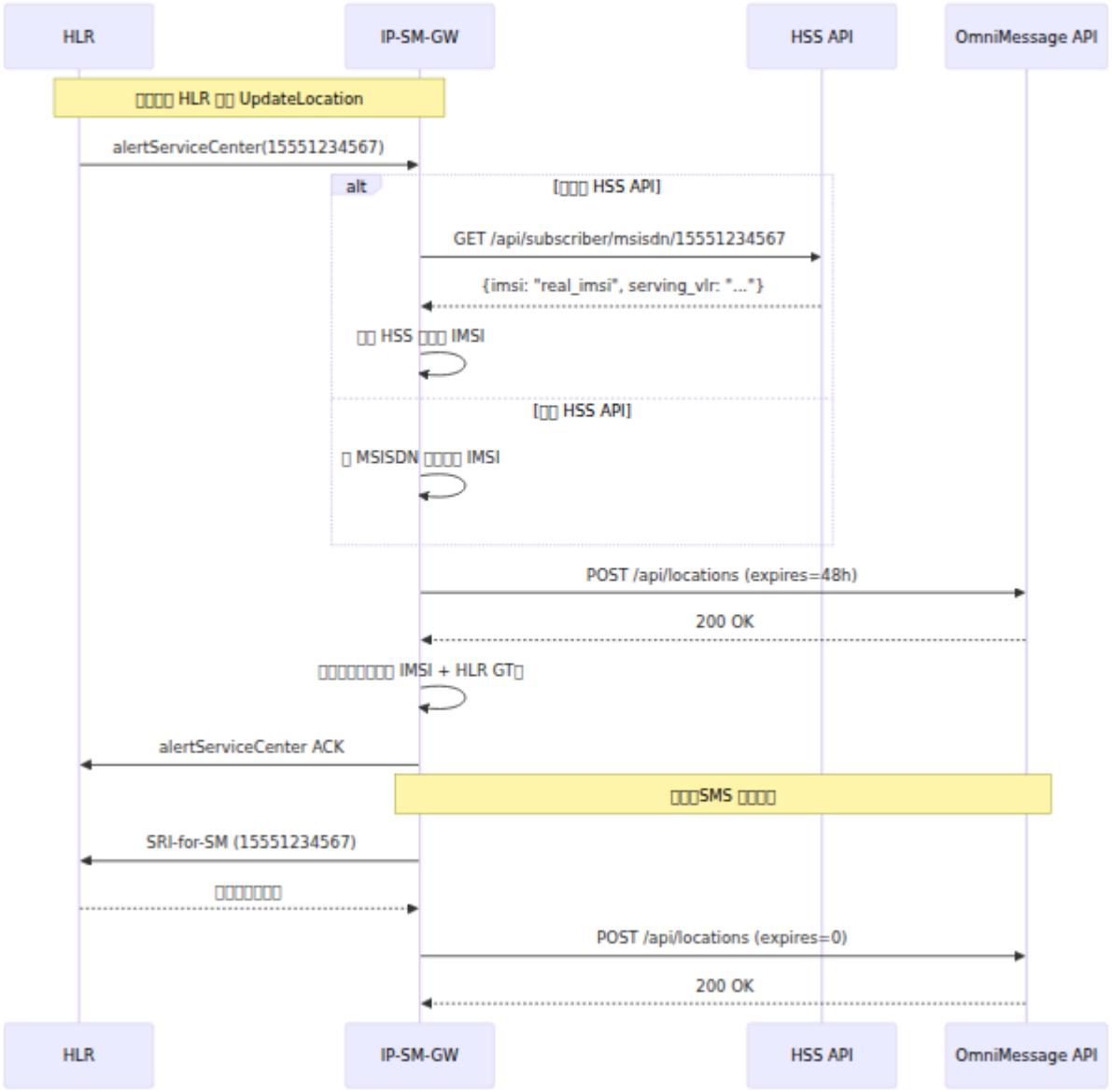
缓存不会在 alertServiceCenter 上清除。现有的 VLR 条目将保留，因为订阅者可能仍然在同一 VLR 可达。

配置

```
config :omniss7,
  # HSS/HLR API 端点（用于 VLR 缓存和在网交付所需）
  hlr_api_base_url: "https://omnihss.example.com:8443",

  # VLR 缓存 TTL（默认：3600 = 1 小时）
  vlr_cache_ttl_seconds: 3600
```

参数	类型	必需	默认值	描述
`hlr_api_base_url`	字符串	否	`nil`	HSS/HLR API 基础 URL。当设置时，启用在网订阅者查找、VLR 缓存和直接 MT 交付。当为 `nil` 时，所有交付使用 SRI-for-SM 流。
`vlr_cache_ttl_seconds`	整数	否	`3600`	缓存的 VLR 条目被视为有效的最大年龄（秒）。在此期间后，下一次交付尝试将重新查询 HSS API。较低的值会增加 HSS API 流量，但提高移动订阅者的准确性。

序列图



自动刷新 SMS 队列

自动刷新服务自动处理待处理的 SMS 消息。
有关配置参数参考，请参见配置参考中的自动刷新配置。

配置

```
config :omniss7,
  auto_flush_enabled: true,              # 启用/禁用自动刷新
  auto_flush_interval: 10_000,           # 轮询间隔（毫秒）
  auto_flush_dest_smsc: nil,             # 过滤器：nil = 所有
  auto_flush_tps: 10                     # 每秒最大事务数
```

它是如何工作的

1. 轮询：每 `auto_flush_interval` 毫秒查询 API 以获取待处理消息
2. 过滤：可选择按 `auto_flush_dest_smsc` 进行过滤
3. 速率限制：每个周期处理最多 `auto_flush_tps` 消息
4. 交付：对于每条消息：
   - 发送 **SRI-for-SM**（发送短消息路由信息）到 HLR 以获取路由信息
     - HLR 返回从 MSISDN 计算的合成 IMSI
     - HLR 返回 MT-ForwardSM 应发送的 SMSC GT 地址
     - 有关完整文档，请参见 HLR 指南中的 SRI-for-SM 详细信息。
   - 成功时，发送 **MT-forwardSM** 到 MSC/VLR
   - 通过 API 更新消息状态（已交付/失败）
   - 通过 API 添加事件跟踪

> 技术深入：有关 SRI-for-SM 工作原理的完整解释，包括 MSISDN 到 IMSI 映射、服务中心 GT 地址配置和隐私保护的合成 IMSI 生成，请参见 HLR 配置指南中的 SRI-for-SM 部分。

SMSc 指标

OmniSS7 暴露 Prometheus 指标。共享的 MAP 指标跟踪 SMS 活动。没有 SMSc 特定的队列/交付计数器。与 SMS 操作相关的指标包括：

□□	□□	□□	□ **SMS** □□□□
`map_requests_total`	□□□	`operation`	□□ MAP □□□□ SMS □□□□□□□ `sri_for_sm`□ `forward_sm`□ `MTforward_sm` □ `prn`□
`map_request_duration_milliseconds`	□□□	`operation`	MAP □□/□□□□□□□ □□□□ SRI-for-SM □ MT-ForwardSM□

□□□□□□□□□□□□□□□ SMS □□□□□`map_requests_total`□`cap_requests_total`□ `ussd_requests_total`□`ussd_active_sessions`□`map_pending_requests`□ `omniss7_license_status` □ `map_request_duration_milliseconds`□

□□□□□

```
# □□ 5 □□ MT-ForwardSM □□□□□
rate(map_requests_total{operation="MTforward_sm"}[5m])

# □□ 5 □□ SRI-for-SM □□□□□
rate(map_requests_total{operation="sri_for_sm"}[5m])

# MT-ForwardSM □□□□□□□□□□□ 5 □□□
rate(map_request_duration_milliseconds_sum{operation="MTforward_sm"}
[5m]) /
rate(map_request_duration_milliseconds_count{operation="MTforward_sm"
[5m])
```

故障排除 SMSc

问题：消息未交付

检查：

1. 验证自动刷新是否启用
2. 检查数据库连接
3. 监控日志以查找错误
4. 验证 M3UA 连接是否为活动状态
5. 检查 TPS 限制

问题：队列深度过高

可能原因：

TPS 限制过低
HLR 超时问题
网络连接问题
无效的目的地号码

解决方案：

增加 auto_flush_tps
检查 HLR 可用性
查看失败的消息日志

MT-forwardSM API

通过 API 发送 SMS

API 端点：POST /api/MT-forwardSM

所有四个字段都是必需的。字段名称为 snake_case（注意 `destination_service_centre` 使用英国拼写）。`smsPDU` 是十六进制编码的 SMS TPDU。

请求：

```
{
  "imsi": "234509876543210",
  "destination_service_centre": "447999555111",
  "originating_service_center": "447999123456",
  "smsPDU":
"040B917477218345F60000157030185714 0C0BD4F29C0E9281C4E1F11A"
}
```

响应：

成功时（HTTP 200），响应包含一个单独的 `result` 键，保存 MT-ForwardSM 对话的格式化 TCAP 结果（没有 `message_id`，值不是字面字符串 `"success"`）：

```
{
  "result": "<formatted TCAP result>"
}
```

如果 MAP 对话超时，则端点回 HTTP 504 和 `{"error": "timeout"}`。请求格式错误/缺失字段回 HTTP 400 和 `{"error": "invalid request"}`。

相关文档

OmniSS7 文档：

- ← 回主文档
- HLR 配置指南 - HLR 模式设置和操作
  - **SRI-for-SM** 技术细节 - 有关 MSISDN 到 IMSI 映射和服务中心配置的完整文档
- 常见功能指南 - Web UI、API、监控
- MAP 客户端指南 - MAP 操作
- 技术参考 - 协议规范

OmniMessage 文档：有关消息路由配置、队列管理、交付跟踪、速率限制和分析的信息，请参考OmniMessage 产品文档。OmniMessage 包含所有消息路由逻辑、队列重试算法、交付报告处理和业务规则引擎。

OmniSS7由 Omnitouch Network Services 提供

M3UA & M2PA STP 配置□南

← □回主文档

本指南提供了使用 OmniSS7 作为 信令传输点 **(STP)** 的详细配置。

目录

1. 什么是 STP?
2. STP 网络角色
3. 通过信令网关与 TDM 网络接口
4. 启用 STP 模式
5. 配置对等体
6. M2PA 协议支持
    - M3UA 与 M2PA 的关键区别
    - 配置 M2PA 对等体
    - SLTM 配置
    - 通过 Web UI 管理 M2PA
    - M2PA 指标
7. 点码路由
8. 全局标题路由
9. SCCP 过境处理
    - GTT 后的 SSN 路由
    - 跳数计数器强制执行
    - 失败时□□□息□回
10. 目的地状态（信令路由管理）
11. 网关筛选（SCCP 防火墙）
12. 路由管理功能
    - 禁用路由
    - DROP 路由 - 防止路由循环
13. 高级路由
14. 测试配置

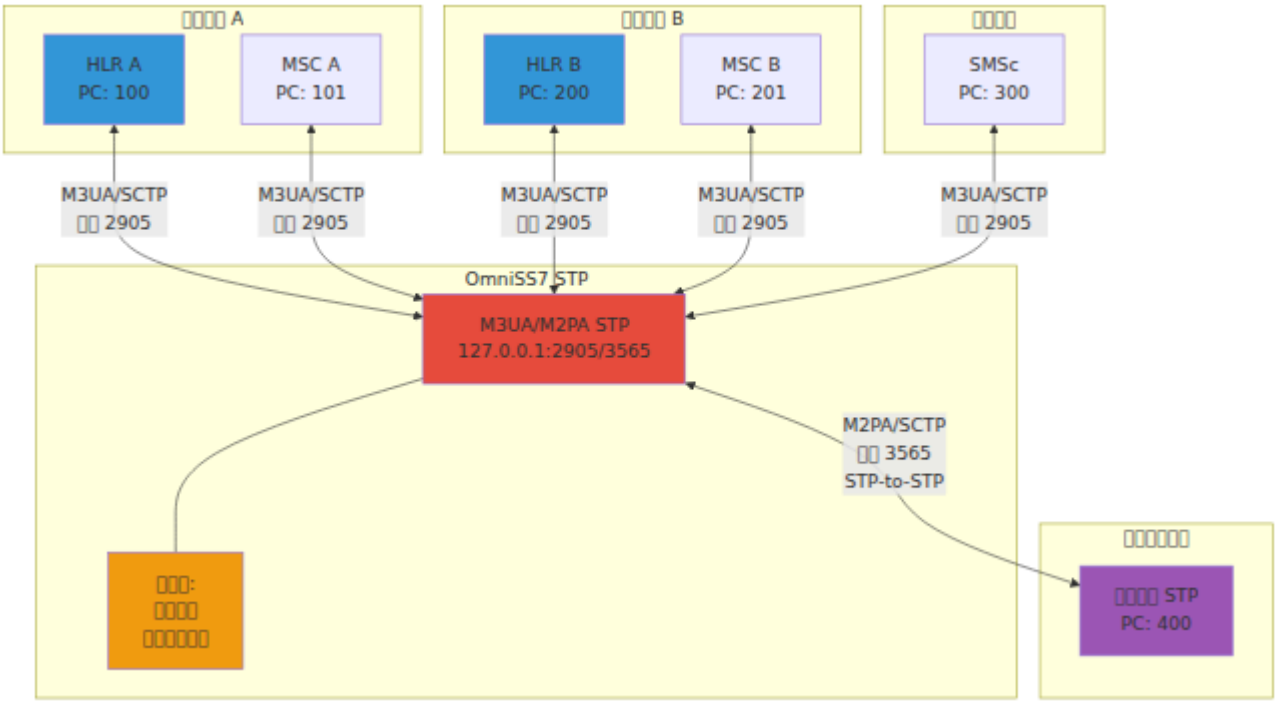
15. 指标和监控
16. M3UA 对等体监控
    - M3UA 路由上下文处理
    - 同 IP 多对等体消歧义

什么是信令传输点 **(STP)**?

信令传输点 **(STP)** 是 SS7 和基于 IP 的信令网络中的关键网络元素，负责在网络节点之间路由信令消息。

STP 功能

- 消息路由：根据目的地点码 (PC) 或全局标题 (GT) 路由 SS7 信令流量
- 协议转换：将传统 SS7 网络与基于 IP 的 M3UA/SCTP 网络连接起来
- 负载分配：使用基于优先级的路由在多个目的地之间分配流量
- 网络网关：连接不同的信令网络和服务提供商
- 拓扑隐藏：可以重写地址以隐藏内部网络拓扑



STP 网络图

STP 网络角色解释

ASP（应用服务器进程）

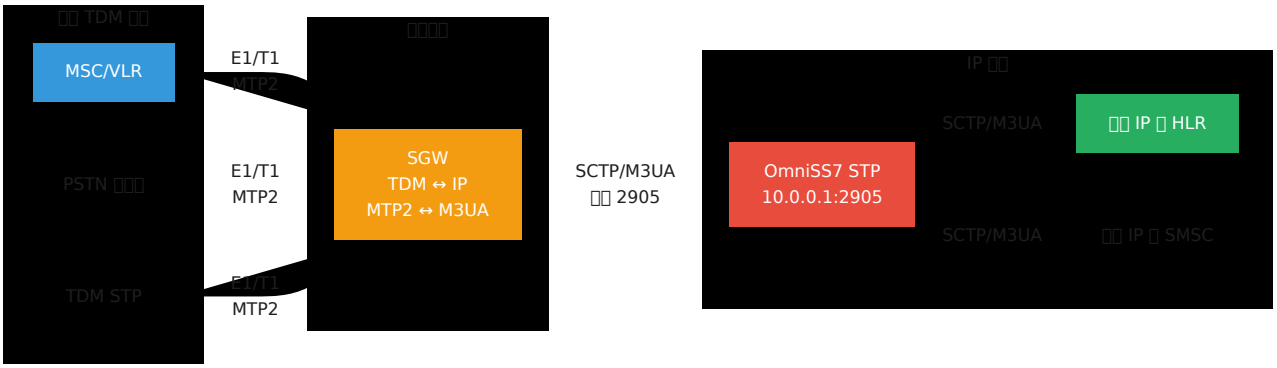
- 角色：连接到远程 SGP/STP 的客户端
- 方向：出站连接
- 用例：您的 STP 连接到合作伙伴网络的 STP

SGP（信令网关进程）

- 角色：接受来自 ASP 的连接的服务器
- 方向：入站连接
- 用例：合作伙伴网络连接到您的 STP

AS（应用服务器）

- 定义：一个或多个 ASP 的逻辑分组



- 目的：提供冗余和负载共享
- 用例：多个 ASP 为同一目的地服务

通过信令网关与 **TDM** 网络接口

OmniSS7 是一个基于 IP 的信令平台，使用 SIGTRAN 协议（M3UA/M2PA 通过 SCTP）。要与传统的基于 **TDM** 的 **SS7** 网络交换信令，您需要一个信令网关 **(SGW)** 来桥接这两个世界。

什么是信令网关？

信令网关 **(SGW)** 是一个网络元素，它在以下两者之间转换 SS7 信令：

- **TDM** 侧：使用 E1/T1 链路的传统 SS7，采用 MTP1/MTP2/MTP3
- **IP** 侧：使用 SCTP/IP 的 SIGTRAN，采用 M3UA 或 M2PA

SGW 充当协议转换器。它使基于 IP 的应用程序（如 OmniSS7）能够与不支持 IP 信令的传统 TDM 设备（交换机、STP、HLR）进行通信。

TDM 到 **IP** 架构

协议栈比较

层级	TDM SS7	信令网关	IP (SIGTRAN)
用户部分	SCCP/TCAP/MAP	← 透明 →	SCCP/TCAP/MAP
网络	MTP3	转换	M3UA/M2PA
链路	MTP2	终止	SCTP
物理	MTP1 (E1/T1)	终止	IP/Ethernet

SGW 在 TDM 侧终止 MTP1/MTP2，并在 IP 侧通过 M3UA 或 M2PA 提供 MTP3 用户数据。上层（SCCP、TCAP、MAP、CAP）透明通过。

将 **OmniSS7** 连接到信令网关

OmniSS7 作为 **ASP（**应用服务器进程**）** 连接到信令网关，而 SGW 充当 **SGP（**信令网关进程**）**。

配置示例

```
config :omniss7,
  # □□ ASP □□□□□□□□
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :sgw_connection,
    # □□□□□ (OmniSS7)
    local_ip: {10, 0, 0, 1},
    local_port: 0,                    # □□□□□□□
    # □□□□□□□
    remote_ip: {10, 0, 0, 100},       # SGW IP □□
    remote_port: 2905,                # SGW M3UA □□
    routing_context: 1                # SGW □□□□□□□
  },

  # □□□□□ STP □□□□□□□
  peers: [
    %{
      peer_id: 1,
      name: "TDM_Gateway",
      role: :client,                  # OmniSS7 □□□□□
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 100},     # SGW IP □□
      remote_port: 2905,
      routing_context: 1,
      point_code: 100,                # SGW □□□□□□□□
      network_indicator: :international
    }
  ],

  # □□□□□□□ TDM □□□□□
  m3ua_routes: [
    %{
      dest_pc: 100,                   # TDM MSC □□
      peer_id: 1,                     # □□ SGW □□□□□□
      priority: 1,
      network_indicator: :international
    },
    %{
      dest_pc: 200,                   # TDM HLR □□
      peer_id: 1,
      priority: 1,
```

```
      network_indicator: :international
    }
  ]
```

SGW 配置注意事项

配置信令网关以与 OmniSS7 一起使用时:

参数	描述	典型值
远程 **IP**	OmniSS7 的 IP 地址	您的服务器 IP
远程端口	OmniSS7 的 SCTP 端口	2905
路由上下文	SGW 的 AS 标识符	SGW 管理员分配
点码	通过 SGW 可访问的 TDM 网络点码	网络特定
流量模式	ASP 流量处理模式	覆盖或负载共享

部署场景

场景 **1**: **IP** 应用访问 **TDM** 网络

OmniSS7 通过 SGW 向基于 TDM 的 HLR 发送 MAP 查询 (SRI-SM, PRN):

```
OmniSS7 (ASP) → SGW (SGP) → TDM 网络 → HLR
     M3UA           MTP2           MTP3
```

场景 **2**: **OmniSS7** 作为 **IP** 和 **TDM** 之间的 **STP**

OmniSS7 在基于 IP 的网络元素和 TDM 网络之间路由流量:

```
IP SMSC → OmniSS7 STP → SGW → TDM HLR
                ↓
          基于 IP 的 HLR
```

□□ **3□□ SGW □□□□□**

□□□□□□□□□□□□□□□□□□□□

```
peers: [
  %{
    peer_id: 1,
    name: "SGW_Primary",
    role: :client,
    remote_ip: {10, 0, 0, 100},
    remote_port: 2905,
    point_code: 100,
    # ... □□□□
  },
  %{
    peer_id: 2,
    name: "SGW_Backup",
    role: :client,
    remote_ip: {10, 0, 0, 101},
    remote_port: 2905,
    point_code: 100,
    # ... □□□□
  }
],

m3ua_routes: [
  # □□ SGW_Primary □□□□□
  %{dest_pc: 100, peer_id: 1, priority: 1, network_indicator:
:international},
  # □□ SGW_Backup □□□□□
  %{dest_pc: 100, peer_id: 2, priority: 2, network_indicator:
:international}
]
```

□□ M3UA STP □□

OmniSS7 □□□□□□□□□□□□□□□□□□ STP□□□□□ `:omniss7` □□□□□□□□ STP □□□□□

配置存放位置

文件	目的
`config/config.exs`	默认的、始终加载的配置（Web UI、端口 8445 的 API 监听、日志记录）
`config/stp_example_runtime.exs`	完整的、可编辑的 **STP** 示例（`sctp_handler`、对等体、PC 路由、GT 路由）
`config/runtime.exs`	每个部署的运行时覆盖。在此存储库中，它是一个最小的存根；生产服务器部署自己的

> 注意：早期版本在 `config/runtime.exs` 中包含三个注释掉的模式块。该布局不再存在。STP 示例现在位于其自己的文件 `config/stp_example_runtime.exs` 中。

切换到 **STP** 模式

要启动 STP，请将示例配置复制到您的部署在运行时加载的配置中：

1. 打开 `config/stp_example_runtime.exs` 并查看 STP 示例（SCTP 处理程序、对等体、PC 路由、GT 路由）。
2. 复制 `config :omniss7, ...` 块到您部署的 `config/runtime.exs`（或合并相关键）。
3. 自定义 您网络的 IP 地址、端口、点码、对等体和路由。
4. 重启 应用程序：`iex -S mix`

STP 模式配置

config/stp_example_runtime.exs 中的 STP 示例配置了一个入站 STP 监听器（sctp_handler），启用了 GT 路由，并定义了对等体以及 PC 和 GT 路由：

```
config :omniss7,
  # SCTP 监听器（入站 STP 服务器） - 请参见下面的独立 STP 模式
  sctp_handler: %{
    enabled: true,
    local_ip: {127, 0, 0, 1},
    local_port: 2905,
    point_code: 101             # 此 STP 的点码
  },

  # 启用全局标题路由（除了 PC 路由外）
  enable_gt_routing: true,

  # M3UA 对等体、PC 路由和 GT 路由如下
  m3ua_peers: [ %{peer_id: 1, name: "STP_West", role: :client,
...} ],
  m3ua_routes: [ %{dest_pc: 100, peer_id: 1, priority: 1,
network_indicator: :international} ],
  m3ua_gt_routes: [ %{gt_prefix: "44", peer_id: 2, priority: 1,
description: "英国号码"} ]
```

请参见 配置 M3UA 对等体、点码路由 和 全局标题路由 以获取完整的参数集。

需要自定义的配置参数

有关所有配置参数的完整参考，请参见 配置参考。

以下是 `config :omniss7` 下的顶级 STP 键:

参数	类型	默认值	描述	示例
`sctp_handler`	Map	□□	入站 STP 监听器配置（请参见 独立 STP 模式）	`%{enabled: true, local_port: 2905, point_code: 101}`
`enable_gt_routing`	Boolean	`false`	启用全局标题路由（除了 PC 路由外）	`true`
`m3ua_peers`	List	`[]`	M3UA / M2PA 对等体定义（请参见 配置 M3UA 对等体）	无
`m3ua_routes`	List	`[]`	点码路由（请参见 点码路由）	无
`m3ua_gt_routes`	List	`[]`	全局标题路由（请参见 全局标题路由）	无

> 提示：通过为 `local_ip` 和/或 `remote_ip` 提供 IP 地址列表来使用 SCTP 多宿主，以启用自动故障转移。请参见 SCTP 多宿主指南。

启用 **STP** 模式后发生的事情

一旦对等体和路由配置完成，Web UI 显示：

- □ **SS7** 事件 - 事件日志
- □ **M3UA** - 连接状态
- □ □□ - 路由表管理 ← *STP □□*
- □ □□□□ - 路由测试 ← *STP □□*
- □ □□ - 系统监控
- □ □□ - 配置查看器

重要注意事项

- 必须允许 SCTP 协议 (IP 协议 132) 通过防火墙
- 默认 M3UA 端口为 2905（行业标准）
- 确保有足够的系统资源来处理路由流量
- 路由持久性：通过 Web UI 或 API 配置的所有路由都存储在 **Mnesia** 数据库中，并且在重启后仍然有效
- 配置合并：`runtime.exs` 中的路由在启动时加载并与 Mnesia 路由合并
- 更改配置后，必须重启应用程序以使更改生效
- **Web UI**：请参见 Web UI 指南以通过 Web 界面管理路由
- **API** 访问：请参见 API 指南以获取 REST API 文档和 Swagger UI 访问

独立 **STP** 模式

您可以运行一个独立的 **M3UA STP** 服务器，以监听传入连接。这是 `config/stp_example_runtime.exs` 中示例使用的模式。

启用独立 **STP**

在运行时配置的 `config :omniss7` 下添加以下配置：

```
config :omniss7,
  sctp_handler: %{
    enabled: true,
    local_ip: {127, 0, 0, 1},     # 监听的 IP 地址
    local_port: 2905,              # 监听的端口
    point_code: 100                # 此 STP 的点码
  }
```

STP 配置参数

参数	类型	默认值	描述	示例
`enabled`	Boolean	`false`	启用独立 STP 服务器	`true`
`local_ip`	Tuple	`{127, 0, 0, 1}`	监听连接的 IP 地址	`{0, 0, 0, 0}`
`local_port`	Integer	`2905`	监听的端口	`2905`
`point_code`	Integer	`□□`	此 STP 的 SS7 点码	`100`

何时使用独立 STP

- 纯路由：当您只需要 M3UA 路由而不需要 MAP 客户端功能时
- 中央 **STP**：创建一个中央信令路由器，以连接多个网络元素
- 中心架构：通过中央 STP 连接多个 HLR、MSC 和 SMSC

注意：如果您同时需要出站连接和入站 STP 功能，可以同时启用 `map_client_m3ua` 和 `sctp_handler`。

路由表持久性 (Mnesia)

所有路由表（对等体、点码路由和全局标题路由）都存储在 **Mnesia** 数据库中以实现持久性。

路由工作原理

1. **Runtime.exs** 路由：在应用程序启动时加载在 `config/runtime.exs` 中定义的路由，包括 `peers`（或遗留的 `m3ua_peers`）、`m3ua_routes` 和 `m3ua_gt_routes`
2. **Web UI** 路由：通过 Web UI 路由页面添加的路由存储在 Mnesia 中
3. 路由合并：在重启时，runtime.exs 路由与现有的 Mnesia 路由合并（不重复）
4. 持久性：通过 Web UI 配置的所有路由在应用程序重启后仍然有效

Mnesia 存储类型

控制路由表的存储方式。有关数据库配置的更多详细信息，请参见配置参考中的数据库参数。

config:omniss7,
mnesia_storage_type::disc_copies # 或:ram_copies 用于测试

存储类型 描述 持久性 用例

:disc_copies 磁盘支持的存储（默认） 在重启后仍然有效 生产环境

:ram_copies 仅在内存中 重启时丢失 测试、开发

默认：:disc_copies

Mnesia 数据库位置

Mnesia 将路由表存储在应用程序的 Mnesia 目录中：

位置：Mnesia.{node_name}/（例如，Mnesia.nonode@nohost/）
表（所有在 M3UARouting.init_database中创建）：

表 目的

m3ua_peer M3UA / M2PA 连接端点

m3ua_route 点码路由 (DPC → 对等体，支持掩码)

m3ua_gt_route 全局标题路由 (前缀 → 对等体，支持 SSN/TT/NPI/NAI 匹配和重写)

由 MTP3 信令路由管理驱动的每个目的地可用性状态 (TFP/TFA/TFR, Q.704
m3ua_dest_state
§13)

ss7_screen_rule 网关筛选 (SCCP 防火墙) 允许/拒绝规则

管理路由

您有三种管理路由的选项：
1. Runtime.exs - 启动时加载的静态配置
2. Web UI - 交互式路由管理（请参见Web UI 指南）
3. REST API - 程序化路由管理（请参见API 指南）
最佳实践：使用 runtime.exs 进行基本配置，并使用 Web UI 在运行过程中进行动态路由更改。

配置 M3UA 对等体

对等体表示 M3UA 连接端点（其他 STP、HLR、MSC、SMSC）。将对等体添加到 config/runtime.exs。

对等体配置示例

注意：配置键 peers 是当前标准。遗留键 m3ua_peers 仍然支持以保持向后兼容性。

```
config :omniss7,
  peers: [
    # □□□□□□□□ STP □□□□□□□□□□□:client□
    %{
      peer_id: 1,                              # □□□□□□
      name: "Partner_STP_West",                # □□□□□□
      role: :client,                           # :client □□□□□□:server
□□□□□
      local_ip: {10, 0, 0, 1},                # □□ IP □□
      local_port: 0,                           # 0 = □□□□□□□
      remote_ip: {10, 0, 0, 10},              # □□□□□□ IP
      remote_port: 2905,                       # □□□□□□□□
      routing_context: 1,                      # M3UA □□□□□□
      point_code: 100,                         # □□□□□□□□
      network_indicator: :international     # :international □
:national
    },

    # □□□□□□ HLR□□□□□:client□
    %{
      peer_id: 2,
      name: "Local_HLR",
      role: :client,
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 20},
      remote_port: 2905,
      routing_context: 2,
      point_code: 200,
      network_indicator: :international
    },

    # □□□□□ MSC □□□□□□□□□□□:server□
    # □□ :server □□□STP □□□□□□□
    %{
      peer_id: 3,
      name: "Remote_MSC",
      role: :server,                           # □□□□□□□
      remote_ip: {10, 0, 0, 30},              # □□□ IP
      remote_port: 2905,                       # □□□□□□□0 = □□□□□□□□□□
      routing_context: 3,
      point_code: 300,
      network_indicator: :international
```

```
  },

  # 具有动态源端口的入站连接（无端口过滤）
  %{
    peer_id: 4,
    name: "Dynamic_Client",
    role: :server,
    remote_ip: {10, 0, 0, 40},            # 预期源 IP
    remote_port: 0,                        # 0 = 接受来自任何源端口的连接
    routing_context: 4,
    point_code: 400,
    network_indicator: :international
  }
]
```

对等配置参数

参数	类型	必需	描述
`peer_id`	整数	是	对等的唯一数字标识符
`name`	字符串	是	用于日志和监控的人类可读名称
`role`	原子	是	`:client`（出站）或 `:server`（入站）
`local_ip`	元组或列表	是（客户端）	本地 IP 地址绑定。单个：`{10, 0, 0, 1}` 或多个用于 SCTP 多宿主：`[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`local_port`	整数	是（客户端）	本地端口（0 表示动态）
`remote_ip`	元组或列表	是	客户端角色：单个元组或多宿主远程对等的列表。服务器角色：仅单个元组 - 远程对等连接的 IP（见下文注释）
`remote_port`	整数	是	远程对等端口（0 表示入站 = 接受任何源端口）
`routing_context`	整数	是	M3UA 路由上下文标识符
`point_code`	整数	是	此对等的 SS7 点码
`network_indicator`	原子	否	`:international` 或 `:national`

> 服务器角色的 **SCTP** 多宿主（入站连接）：当接受来自多宿主远程对等的入站连接时，您只需指定远程对等用于启动 SCTP 连接的单个 **IP** 地址（SCTP INIT）。这应该是一个 单个元组，而不是列表。SCTP 将在关联握手期间自动发现对等的其他多宿主 IP 地址。`remote_ip` 的列表格式仅在连接到多宿主远程对等时用于 客户端角色。
>
> 客户端角色的 **SCTP** 多宿主（出站连接）：在连接到远程对等时，为了网络冗余，您可以为 `local_ip` 和 `remote_ip` 配置多个 IP 地址，使用列表。这使得在一个网络路径失败时可以自动故障转移。有关详细的配置示例和最佳实践，请参见 SCTP 多宿主指南。

入站连接的源端口过滤

对于 入站连接（角色：`:server`），`remote_port` 参数控制源端口过滤：

- 特定端口（例如，`remote_port: 2905`）：仅接受来自该确切源端口的连接
  - 通过验证源端口提供额外的安全性
  - 当远程对等使用固定源端口时使用
- 任何端口（`remote_port: 0`）：接受来自任何源端口的连接
  - 当远程对等使用动态/临时源端口时很有用
  - 仅验证源 IP 地址
  - 更灵活但安全性稍低

示例：

```
# 仅接受来自 192.0.2.10:2905（特定端口）
%{
  peer_id: 1,
  name: "Strict_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 2905,
  # ... 其他配置
}

# 接受来自 192.0.2.10 的任何源端口
%{
  peer_id: 2,
  name: "Flexible_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 0,  # 接受来自任何源端口
  # ... 其他配置
}
```

M2PA 协议支持

OmniSS7 支持 **M3UA** 和 **M2PA** 协议用于 SS7 信令传输。

什么是 M2PA?

M2PA（MTP2 用户对等适配层）是一个 IETF 标准化协议（RFC 4165），用于通过 IP 网络使用 SCTP 传输 SS7 MTP3 消息。

M3UA 与 M2PA：关键区别

特性	M3UA	M2PA
架构	客户端/服务器（ASP/SGW）	对等
用例	SS7 与 IP 之间的网关	直接点对点链接
链路状态管理	应用级（ASPUP/ASPAC）	MTP2 风格（对齐、验证、准备）
序列号	无固有排序	24 位 BSN/FSN 用于有序交付
典型部署	SS7 到 IP 网关，STP	节点之间的直接信令链接
RFC	RFC 4666	RFC 4165

协议选择□导

建议：默认使用 **M3UA**。仅在特定需要时使用 **M2PA**。

何时使用 M3UA（推荐���

M3UA 是大多数部署的推荐协议：

- **STP** 部署：标准信令传输点实现
- 网关功能：将 SS7 网络与基于 IP 的信令连接
- 网络元素连接：将 HLR、MSC、SMSC 和其他网络元素连接到您的 STP

- 信令网关（**SGW**）：接受来自多个应用服务器的连接的中央网关
- 灵活拓扑：具有集中控制的客户端/服务器架构
- 多供应商网络：广泛支持的行业标准（RFC 4666）

使用 **M3UA** 将网络元素（**HLR**、**MSC**、**SMSC**、**VLR** 等）连接到您的 **STP**。

何时使用 **M2PA**（仅限特殊情况）

M2PA 仅应在特定场景中使用：

- **STP** 到 **STP** 链接：在多 STP 网络中信号传输点之间的直接点对点连接
- 传统 **TDM** 替代：在远程系统特别需要 M2PA 时替代传统 SS7 TDM 链接
- 需要 **MTP2** 兼容性：当连接到强制要求 MTP2 风格链路状态管理的遗留系统时
- 合作伙伴要求：当合作伙伴或互联特别要求 M2PA 协议时

重要：不要使用 M2PA 将网络元素（HLR、MSC、SMSC）连接到您的 STP - 请改用 M3UA。M2PA 旨在用于 STP 到 STP 的互连，其中双方都作为路由节点操作。

配置 **M2PA** 对等

M2PA 对等的配置与 M3UA 对等相同，增加了一个 `protocol` 参数。

M2PA 对等配置

将 M2PA 对等添加到 `config/runtime.exs` 中的 `peers` 配置（M3UA 和 M2PA 对等共享相同的配置部分，通过 `protocol` 参数区分）：

M2PA 的关键参数：

参数	值	描述
`protocol`	`:m2pa`	指定 M2PA 协议（如果省略，默认为 `:m3ua`）
`role`	`:client` 或 `:server`	连接方向
`local_port`	整数	本地 SCTP 端口（标准 M2PA 端口为 **3565**）
`remote_port`	整数	远程 SCTP 端口（标准 M2PA 端口为 **3565**）
`point_code`	整数	您的点码
`adjacent_point_code`	整数	远程对等的点码（M2PA 特定）
`send_sltm`	布尔	控制 SLTM 行为（见下文 SLTM 配置）
`network_indicator`	原子	`:international` 或 `:national` - 必须与远程对等匹配

注意：M2PA 使用 **3565** 端口作为行业标准（与 M3UA 的 2905 端口不同）。

SLTM 配置

SLTM（信令链路测试消息）和 **SLTA**（信令链路测试确认）是 MTP3 维护消息，用于在 M2PA 链路达到 READY 状态后验证端到端连接。根据 ITU-T Q.707，一方发送 SLTM，另一方响应 SLTA 以确认链路处于操作状态。

SLTM 行为

`send_sltm` 参数控制哪一方发起 SLTM 测试：

`send_sltm` 值	行为
`true`	链路变为 READY 时发送 SLTM，等待 SLTA
`false`	等待对等发送 SLTM，响应 SLTA
未设置（默认）	遵循 Q.707：SCTP 响应方发送 SLTM，SCTP 发起方等待

默认行为（符合 **Q.707**）：

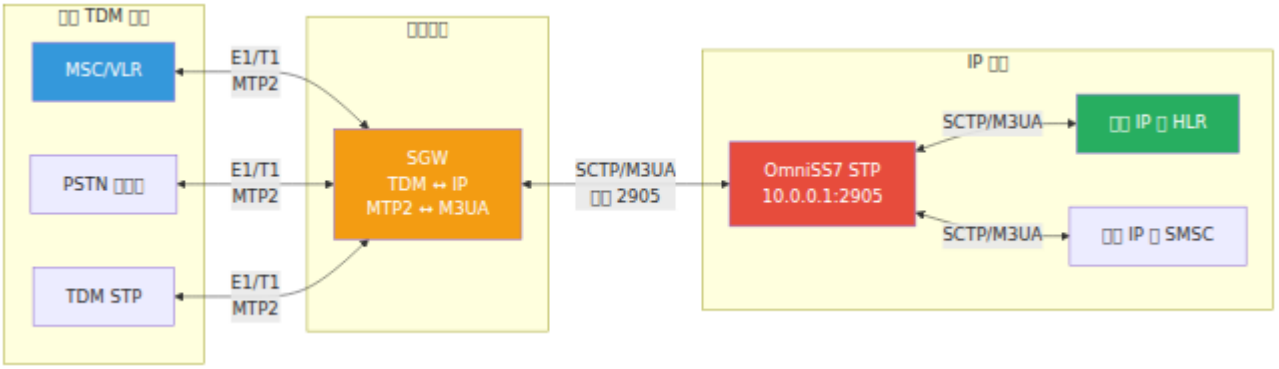
- 如果 `initiate_connection: true`（SCTP 发起方/客户端）→ 我们等待对等发送 SLTM
- 如果 `initiate_connection: false`（SCTP 响应方/服务器）→ 我们发送 SLTM

何时覆盖 **SLTM** 默认值

在连接到不遵循 Q.707 规范的设备时覆盖默认行为：

```
# 示例：强制我们的一方发送 SLTM，无论 SCTP 角色如何
%{
  peer_id: 100,
  name: "Partner_STP",
  protocol: :m2pa,
  role: :client,
  local_ip: {10, 0, 0, 1},
  local_port: 3565,
  remote_ip: {10, 0, 0, 2},
  remote_port: 3565,
  point_code: 7415,
  adjacent_point_code: 15528,
  network_indicator: :international,
  send_sltm: true  # 覆盖：即使我们是 SCTP 发起方，我们也发送 SLTM
}
```

SLTM 消息流



排查 **SLTM** 问题

症状：链路达到 **READY** 但没有流量流动

双方可能在等待对方发送 SLTM。检查日志以查找：

```
"waiting for peer to send SLTM"
```

解决方案：在一方配置 `send_sltm: true` 以打破僵局。

症状：发送 **SLTM** 但未收到 **SLTA**

可能的原因：

1. `adjacent_point_code` 不正确（SLTM 发送时使用了错误的 DPC）
2. 网络指示符不匹配（`: international` 与 `: national`）
3. 远程对等未配置为响应我们的点码

解决方案：验证 `adjacent_point_code` 与 STP 自身的点码匹配（在 `sctp_handler.point_code` 中配置）。

M2PA 链路状态

M2PA 链路在初始化期间经历几个状态：

1. **Down** - 未建立连接
2. **Alignment** - 初始同步阶段（约 1 秒）
3. **Proving** - 链路质量验证（约 2 秒）
4. **Ready** - 链路活动并准备好传输流量

链路状态进展确保在交换流量之前进行可靠的信令。

通过 Web UI 管理 M2PA 对等

Web UI 中的路由页面提供对 M2PA 对等的完整管理支持：

1. 导航到路由页面
2. 选择 “对等” 标签
3. 点击 “添加新对等”
4. 从协议下拉菜单中选择 “M2PA (RFC 4165)”
5. 填写对等配置：
对等名称（描述性标识符）
协议：M2PA
角色：客户端或服务器
点码（您的 PC）
本地/远程 IP 地址
本地/远程端口（通常 M2PA 为 3565）
网络指示符（国际或国家）
6. 点击 “保存对等”

对等表显示协议类型并进行颜色编码：

蓝色 - M3UA 对等
绿色 - M2PA 对等

M2PA 路由行为

M2PA 对等与 OmniSS7 的路由系统无缝集成：

点码路由：M2PA 和 M3UA 的工作方式相同
全局标题路由：在 M2PA 链路上完全支持
路由优先级：M2PA 和 M3UA 对等可以混合在同一路由表中
消息中继：消息可以在 M2PA 上到达并路由到 M3UA，反之亦然

M2PA □标

M2PA 提供全面的 Prometheus 指标用于监控链路健康和流量:

流量□标:

- `m2pa_messages_sent_total` - 每个链路发送的 MTP3 消息总数
- `m2pa_messages_received_total` - 每个链路接收的 MTP3 消息总数
- `m2pa_bytes_sent_total` - 通过 M2PA 发送的字节总数
- `m2pa_bytes_received_total` - 通过 M2PA 接收的字节总数

所有流量指标按: `link_name`、`point_code`、`adjacent_pc` 标记

链路状态□标:

- `m2pa_link_state_changes_total` - 链路状态转换 (DOWN → ALIGNMENT → PROVING → READY)
  - 标签: `link_name`、`from_state`、`to_state`

错误□标:

- `m2pa_errors_total` - 按类型统计的错误总数
  - `decode_error` - M2PA 消息解码失败
  - `encode_error` - M2PA 消息编码失败
  - `sctp_send_error` - SCTP 传输失败
  - 标签: `link_name`、`error_type`

访问□标:

- Prometheus 端点: `http://your-server:9090/metrics`
- 指标在应用启动时自动注册

M2PA 最佳实践

1. 端口选择: 使用 3565 端口用于 M2PA (行业标准)
2. 链路监控: 通过指标监控链路状态变化
3. 防火墙规则: 确保允许 SCTP (IP 协议 132)
4. 点码: 确保双方的相邻点码正确配置

5. 网络□示符：必须在对等之间匹配（国际或国家）
6. 测试：使用路由测试页面在配置后验证连接

M2PA 套接字要求

M2PA 使用 **SCTP.SocketHandler** 共享套接字。所有 M2PA 对等自动使用 SCTP.SocketHandler 进行套接字管理，这允许多个对等有效地共享同一 SCTP 端口。

要求：

- SCTP.SocketHandler 必须启用并运行
- 共享套接字必须在 M2PA 对等启动之前配置

示例：

```
# □□ SCTP.SocketHandler
sctp_handler: %{
  enabled: true,
  local_ip: {10, 179, 4, 10},
  local_port: 3565,
  point_code: 100
}

# M2PA □□□□□□□□□□□□
peers: [
  %{
    peer_id: 1,
    name: "M2PA_Link_1",
    protocol: :m2pa,
    role: :client,
    local_ip: {10, 179, 4, 10},
    local_port: 3565,
    remote_ip: {10, 179, 4, 20},
    remote_port: 3565,
    point_code: 100,
    adjacent_point_code: 200
  },
  %{
    peer_id: 2,
    name: "M2PA_Link_2",
    protocol: :m2pa,
    role: :client,
    local_ip: {10, 179, 4, 10},
    local_port: 3565,           # □□□□□□□□□□□□□
    remote_ip: {10, 179, 4, 30},
    remote_port: 3565,
    point_code: 100,
    adjacent_point_code: 300
  }
]
```

SCTP □□□

- □□□□□□ `unordered` □□□
- PPID 5□□□ RFC 4165 □ M2PA □□□□

□□□□□□

□□□□□□□ MTP3 □□□ □□□□ **(DPC)** □□□□□

□□ SS7 □□□□□□□

□□□□□ SS7 □□□□□□□□□□□□□□□□□□□□□□

□□□□□

```
┌─────────────────────────────────────────┐
│  □□□ (SCCP/TCAP/MAP)                    │
├─────────────────────────────────────────┤
│  MTP3 □                                 │ ← □□□□□□□
│  - □□□□: DPC, OPC, SLS                  │ ← □□ STP □□
│  - □□□□□□ (SIO)                         │
├─────────────────────────────────────────┤
│  M3UA □ M2PA (□□□)                      │ ← □□□□
│  - □□□□ (□□ MTP3)                       │
│  - □□□□ (DUNA/DAVA)                     │ ← □□□□
├─────────────────────────────────────────┤
│  SCTP (□□)                              │
└─────────────────────────────────────────┘
```

□□□□□□□□

1. **MTP3** □□□□□□□□□□
   - □□ MTP3 □□□□□□□DPC□OPC□
   - □□□ M3UA □□□□□□□□□ 528□□
   - □□□ M2PA □□□□□□□
   - **STP** □□□□ **DPC** □□□□□□□
   - □□□□□□□□□□□□□□
2. **M3UA** □□□□□□□□□□□□
   - □□□ M3UA □□□□□□□DUNA□DAVA□SCON□DUPU□
   - □□□□□□□□□□□□□□□□

- 告诉对等哪些目的地是可用/不可���
- 不用于路由用户数据

STP 路由工作原理：

- 对于 **M3UA DATA** 消息：STP 从协议数据参数（标签 528）中提取 MTP3 消息，其中包含 MTP3 路由标签（DPC, OPC, SLS）。使用 MTP3 层的 DPC 查找路由。
- 对于 **M2PA** 用户数据消息：STP 从 M2PA 用户数据字段中提取 MTP3 消息，然后读取 MTP3 路由标签中的 DPC。
- **M3UA** 管理消息：网络管理消息（DUNA, DAVA, SCON）在 M3UA 层包含受影响的点码，以便在对等之间信令网络状态。

基本点码路由

将路由添加到 `config/runtime.exs`：

```
config :omniss7,
  m3ua_routes: [
    # 将所有流量路由到 PC 100 到对等 1（合作 STP）
    %{
      dest_pc: 100,                    # 目标点码
      peer_id: 1,                      # 通过的对等
      priority: 1,                     # 优先级（数字越低优先级越高）
      network_indicator: :international
      # mask: 14                       # 可选：默认为 14（精确匹配）
    },

    # 将所有流量路由到 PC 200 到对等 2（本地 HLR）
    %{
      dest_pc: 200,
      peer_id: 2,
      priority: 1,
      network_indicator: :international
    },

    # 负载均衡示例：PC 300 具有主路由和备份路由
    %{
      dest_pc: 300,
      peer_id: 3,                      # 主路由
      priority: 1,
      network_indicator: :international
    },
    %{
      dest_pc: 300,
      peer_id: 4,                      # 备份路由（优先级数字较高）
      priority: 2,
      network_indicator: :international
    }
  ]
```

注意：`mask` 字段是可选的，默认为 `14`（精确点码匹配）。仅在需要基于范围的路由时指定 `mask`（请参见点码掩码部分）。

路由逻辑

1. STP 接收 M3UA DATA 或 M2PA 用户数据消息
2. STP 从协议数据（M3UA）或用户数据（M2PA）字段提取 **MTP3** 消息
3. STP 从 MTP3 路由标签读取目标点码 **(DPC)**

4. 查找路由表以匹配 DPC（考虑掩码）
5. 如果存在多个路由，选择具有最特定掩码（最高掩码值），然后最低优先级数字的路由
6. 将 MTP3 消息包装在 M3UA DATA 或 M2PA 用户数据中以发送到目标对等
7. 将消息路由到相应的对等
8. 如果所选对等处于关闭状态，则尝试下一个优先级最高的路由

点码掩码

点码是 14 位值（范围 0-16383）。默认情况下，路由精确匹配单个点码（掩码 /14）。但是，您可以使用点码掩码创建匹配点码范围的路由。

理解掩码

掩码指定路由的目标 PC 与传入消息的 DPC 之间必须匹配的最重要的位数。其余位可以是任何值，从而创建匹配点码的范围。

掩码参考表：

掩码	匹配的点码	用例
`/14`	1 PC（精确匹配）	单一目标（默认）
`/13`	2 PCs	小范围
`/12`	4 PCs	小范围
`/11`	8 PCs	小范围
`/10`	16 PCs	中等范围
`/9`	32 PCs	中等范围
`/8`	64 PCs	中等范围
`/7`	128 PCs	中大范围
`/6`	256 PCs	大范围
`/5`	512 PCs	大范围
`/4`	1,024 PCs	非常大范围
`/3`	2,048 PCs	非常大范围
`/2`	4,096 PCs	极其大范围
`/1`	8,192 PCs	一半的所有 PCs
`/0`	16,384 PCs	所有 PCs（默认/后备路由）

点码掩码示例

注意：在所有示例中，`mask` 字段是 可选的。如果省略，默认为 `14`（精确匹配）。

示例 **1**：单个点码（默认行为）

```
# 没有掩码字段（推荐用于单个 PC）
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  network_indicator: :international
}
# 掩码默认为 14 - 匹配: 仅 PC 1000

# 显式掩码（相同结果）
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  mask: 14,                                    # 显式精确匹配
  network_indicator: :international
}
# 匹配: 仅 PC 1000
```

示例 **2**: 小范围

```
%{
  dest_pc: 1000,
  peer_id: 2,
  priority: 1,
  mask: 12,                                    # 匹配 4 PCs
  network_indicator: :international
}
# 匹配: PC 1000，1001，1002，1003
```

示例 **3**: 中等范围

```
%{
  dest_pc: 1000,
  peer_id: 3,
  priority: 1,
  mask: 8,                                    # 匹配 64 PCs
  network_indicator: :international
}
# 匹配: PC 1000-1063 (64 个连续点码)
```

示例 **4**: 默认/后备路由

```
%{
  dest_pc: 0,
  peer_id: 4,
  priority: 10,                               # 低优先级（高数字）
  mask: 0,                                    # 匹配所有 PCs
  network_indicator: :international
}
# 匹配: 所有点码 (0-16383)
# 用作低优先级的捕获/默认路由
```

结合特定和掩码路由

您可以结合特定路由和掩码路由以实现灵活路由:

```
config :omniss7,
  m3ua_routes: [
    # 特定路由用于 PC 1000（优先级最高）
    %{
      dest_pc: 1000,
      peer_id: 1,
      priority: 1,
      network_indicator: :international
      # mask 默认为 14（精确匹配）
    },

    # 范围路由用于 PCs 1000-1063
    %{
      dest_pc: 1000,
      peer_id: 2,
      priority: 1,
      mask: 8,                              # 匹配 64 个 PC
      network_indicator: :international
    },

    # 默认/后备路由用于所有其他 PC
    %{
      dest_pc: 0,
      peer_id: 3,
      priority: 10,                         # 低优先级
      mask: 0,                              # 匹配所有 PC
      network_indicator: :international
    }
  ]
```

DPC 1000 的路由决策:

1. 匹配 mask `/14` 路由（PC 1000 精确匹配） - 选择（最具体）
2. 也匹配 mask `/8` 路由（PC 1000-1063 范围） - 被忽略（不够具体）
3. 也匹配 mask `/0` 路由（所有 PC） - 被忽略（最不具体）

DPC 1015 的路由决策:

1. 不匹配 mask `/14` 路由（仅 PC 1000）
2. 匹配 mask `/8` 路由（PC 1000-1063 范围） - 选择（最具体匹配）
3. 也匹配 mask `/0` 路由（所有 PC） - 被忽略（不够具体）

DPC 5000 的路由决策：

1. 不匹配 mask /14 路由
2. 不匹配 mask /8 路由
3. 匹配 mask /0 路由（所有 PC） - 选择（唯一匹配，后备路由）

最佳实践

1. 单一目的地省略 mask：对于精确的点代码匹配，完全省略 mask字段（默认为 /14）
2. 仅在必要时显式使用 /14：仅在需要在文档中明确说明或与范围路由混合时指定 mask: 14
3. 对网络块使用范围掩码：将整个网络段路由到特定对等体，使用掩码 /0 到 /13
4. 使用 /0 作为后备：创建一个低优先级的默认路由以捕获未匹配的流量
5. 最具体的胜出：路由引擎始终首先选择最具体（最高掩码值）匹配的路由
6. 优先级作为平局打破者：如果多个路由具有相同的掩码，则优先级数字最低的胜出

配置全球标题（GT）路由

全球标题路由允许使用基于内容的路由，使用电话号码或 IMSI 值而不是点代码。有关基于呼叫/被叫方的高级全球标题地址转换，请参见全球标题 NAT 指南。

□□□□

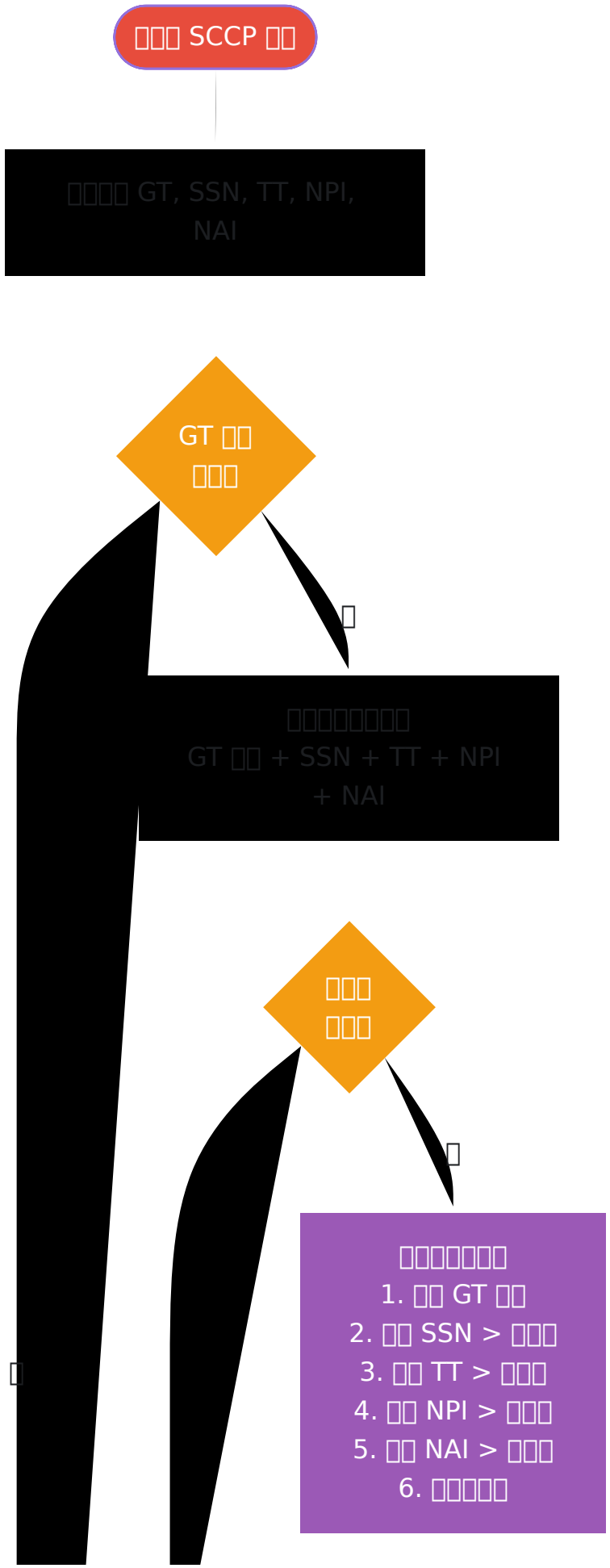
- □□ GT □□□□ `config/runtime.exs` □□□ `enable_gt_routing: true`

GT 路由配置

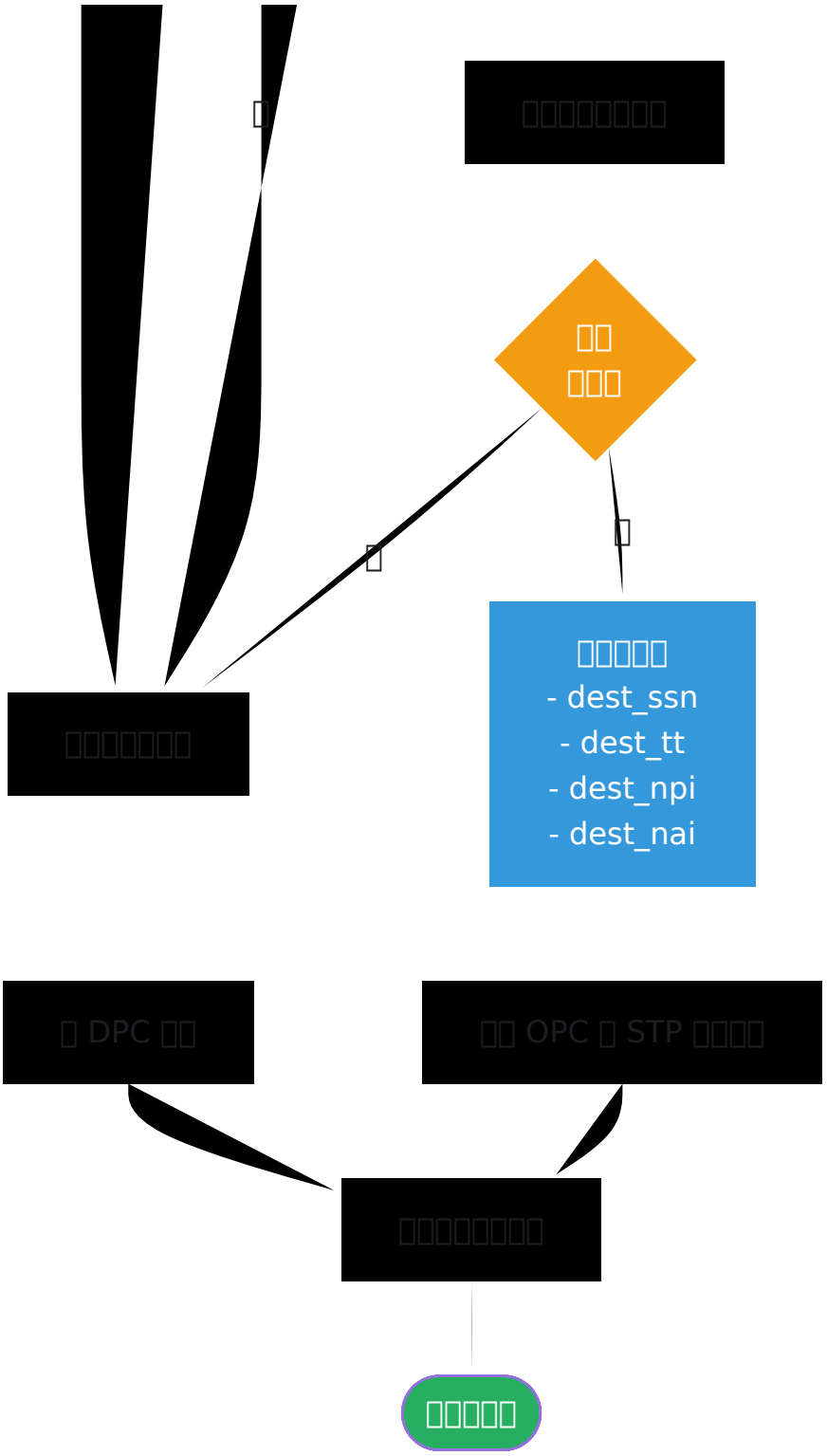
```
config :omniss7,
  # 启用 GT 路由
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # 将所有英国号码（前缀 44）路由到对等体 1
    %{
      gt_prefix: "44",                    # 匹配的全球标题前缀
      peer_id: 1,                         # 目标对等体
      priority: 1,                        # 优先级（数字越低优先级越高）
      description: "英国号码"             # 日志记录的描述
    },

    # 将美国号码（前缀 1）路由到对等体 2
    %{
      gt_prefix: "1",
      peer_id: 2,
      priority: 1,
      description: "美国号码"
    },

    # 更具体的路由：以 447 开头的英国手机号码
    %{
      gt_prefix: "447",                   # 最长前缀匹配胜出
      peer_id: 3,
      priority: 1,
      description: "英国手机号码"
    },

    # SSN 特定路由（可选）
    %{
      gt_prefix: "555",
      source_ssn: 8,                      # 仅在源 SSN = 8（SMSC）时匹配
      peer_id: 4,
      dest_ssn: 6,                        # 将目标 SSN 重写为 6（HLR）
      priority: 1,
      description: "61 前缀的 SMS 流量"
    }
  ]
```

GT □□□□

GT □□□□□□□□□□□□□□



传入的 SCCP 消息
□□□□ GT, SSN, TT, NPI, NAI
GT 路由启用?
是
□□□□□□□□□
GT □□ + SSN + TT + NPI + NAI
是否有匹配?
是
按具体性排序:
1. 最长 GT 前缀
2. 特定 SSN > 通配符
3. 特定 TT > 通配符
4. 特定 NPI > 通配符
5. 特定 NAI > 通配符
6. 最低优先级



□□□□□□

1. □□□□□□□□□STP □□□□□ GT □□□□□□□□□□□□□□□□□□□□□□□□□□□□
    - □□□GT "447712345678" □□ "44" □ "447"□□ "447" □□□□□□□□□□
2. **SSN** □□□□□□□□□

- 如果指定了 `source_ssn`，则仅在 SCCP 被叫方 SSN 等于该值时路由匹配
- 如果 `source_ssn` 为 `nil`，则路由匹配任何 SSN（通配符）

3. **TT/NPI/NAI** 匹配（可选）：

- 如果指定了 `source_tt`、`source_npi` 或 `source_nai`，则路由必须匹配这些指示符
- `nil` 值作为通配符（匹配任何值）

4. 基于具体性的选择：

- 具有更具体匹配标准的路由胜过通配符
- 优先级顺序：GT 前缀长度 → SSN → TT → NPI → NAI → 优先级数字

5. □示符重写（可选）：

- 如果指定了 `dest_ssn`、`dest_tt`、`dest_npi` 或 `dest_nai`，STP 将重写这些指示符

- 有助于协议规范化和网络互连

6. 回退到点代码：
    - 如果没有 GT 路由匹配，STP 将回退到使用 DPC 的点代码路由

高级 **GT** 路由：翻译类型、**NPI** 和 **NAI**

除了 GT 前缀和 SSN 匹配外，STP 还支持基于 SCCP 全球标题指示符的路由和转换：

- 翻译类型（**TT**）：标识编号计划和地址类型
- 编号计划□示符（**NPI**）：定义编号计划（例如，ISDN、数据、传真）
- 地址性质□示符（**NAI**）：指定地址格式（例如，国际、国家、用户）

匹配（源□示符）

路由可以根据传入消息指示符进行匹配：

- `source_tt`：匹配具有特定翻译类型的消息
- `source_npi`：匹配具有特定编号计划指示符的消息
- `source_nai`：匹配具有特定地址性质指示符的消息
- `nil` 值 = 通配符（匹配任何值）

转换（目标□示符）

路由在转发时可以重写指示符：

dest_tt：将翻译类型转换为新值
dest_npi：将编号计划指示符转换为新值 dest_nai：将地址性质指示符转换为新值 nil 值 = 保留原始值（不进行转换）

基于具体性的选择

当多个路由匹配时，选择最具体的路由，优先级顺序如下：

1. 最长 GT 前缀匹配
2. 特定源 SSN 胜过通配符 SSN
3. 特定源 TT 胜过通配符 TT
4. 特定源 NPI 胜过通配符 NPI
5. 特定源 NAI 胜过通配符 NAI
6. 最低优先级数字

配置示例

```
config :omniss7,
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # 示例 1: 匹配并转换翻译类型
    %{
      gt_prefix: "44",
      peer_id: 1,
      source_tt: 0,       # 匹配 TT=0 (未知)
      dest_tt: 3,         # 转换为 TT=3 (国家)
      priority: 1,
      description: "英国号码: TT 0→3 转换"
    },

    # 示例 2: 匹配特定 NPI 并转换 NAI
    %{
      gt_prefix: "1",
      peer_id: 2,
      source_npi: 1,      # 匹配 NPI=1 (ISDN/电信)
      source_nai: 4,      # 匹配 NAI=4 (国际)
      dest_nai: 3,        # 转换为 NAI=3 (国家)
      priority: 1,
      description: "美国号码: 国际→国家 NAI"
    },

    # 示例 3: 组合 SSN 和指示符路由
    %{
      gt_prefix: "33",
      source_ssn: 8,      # 匹配 SMSC 流量
      source_tt: 0,       # 匹配 TT=0
      dest_ssn: 6,        # 将 SSN 重写为 HLR
      dest_tt: 2,         # 转换为 TT=2
      dest_npi: 1,        # 置 NPI=1 (ISDN)
      dest_nai: 4,        # 置 NAI=4 (国际)
      peer_id: 3,
      priority: 1,
      description: "法国 SMS: 完全规范化"
    },

    # 示例 4: 通配符 TT, 特定 NPI
    %{
      gt_prefix: "49",
      source_tt: nil,     # 匹配任何 TT (通配符)
```

```
        source_npi: 6,      # 匹配 NPI=6（数据）
        dest_npi: 1,        # 转换为 NPI=1（ISDN）
        peer_id: 4,
        priority: 1,
        description: "德国数据网络规范化"
      }
    ]
```

常见 **TT/NPI/NAI** 值

翻译类型（**TT**）：

- `0` = 未知
- `1` = 国际
- `2` = 国家
- `3` = 网络特定

编号计划□示符（**NPI**）：

- `0` = 未知
- `1` = ISDN/电信（E.164）
- `3` = 数据（X.121）
- `4` = 传真（F.69）
- `6` = 移动通信（E.212）

地址性质□示符（**NAI**）：

- `0` = 未知
- `1` = 用户号码
- `2` = 保留用于国家使用
- `3` = 国家显著号码
- `4` = 国际号码

路由决策示例

对于传入消息：

- GT: "447712345678"

- SSN: 8
- TT: 0
- NPI: 1
- NAI: 4

配置了以下路由：

```
# 路由 A：通配符 TT
%{gt_prefix: "447", peer_id: 1, priority: 1}

# 路由 B：特定 TT
%{gt_prefix: "447", source_tt: 0, peer_id: 2, priority: 1}

# 路由 C：特定 TT + NPI
%{gt_prefix: "447", source_tt: 0, source_npi: 1, peer_id: 3,
priority: 1}
```

结果：选择路由 C（最具体：匹配 GT + TT + NPI）

消息根据路由 C 的 `dest_tt`、`dest_npi`、`dest_nai` 值进行转发，指示符被转换。

GT 路由示例

被叫 **GT**	源 **SSN**	**TT**	**NPI**	**NAI**	匹配路由	原因
447712345678	6	-	-	-	"447" → 对等体 3	最长前缀匹配
441234567890	6	-	-	-	"44" → 对等体 1	前缀匹配，没有更具体的路由
12125551234	6	-	-	-	"1" → 对等体 2	美国号码的前缀匹配
555881234567	8	-	-	-	"555"（SSN 8）→ 对等体 4	GT + SSN 匹配，将 SSN 重写为 6
555881234567	6	-	-	-	"555"（SSN 通配符）→ 对等体 X	GT 匹配，没有 SSN 重写
441234567890	6	0	1	4	"44"（TT=0）→ 对等体 1	GT + TT 匹配，将 TT 转换为 3
12125551234	8	0	1	4	"1"（TT=0，NPI=1，NAI=4）	最具体：GT+TT+NPI+NAI 匹配

TT/NPI/NAI 路由的实际用例

1. 网络互连规范化
    - 不同网络可能使用不同的指示符约定
    - 在互连点转换指示符以确保兼容性
    - 示例：合作网络使用 TT=0 表示国际，你的网络使用 TT=1
2. 协议转换
    - 在不同网络类型之间路由时在编号计划之间转换

- 示例：从移动网络（NPI=6）路由到 PSTN（NPI=1）

3. 地址格式标准化

   - 规范化所有传入流量以使用一致的 NAI 值
   - 示例：将所有国际格式（NAI=4）转换为国家格式（NAI=3）以进行国内路由

4. 运营商特定路由

   - 根据翻译类型路由到不同的服务提供商
   - 示例：TT=0 路由到运营商 A，TT=2 路由到运营商 B

5. 遗留系统集成

   - 现代系统可能使用与遗留系统不同的指示符值
   - 在 STP 中进行转换以保持向后兼容性

SCCP 传输处理

除了选择目标对等体，STP 还执行 SCCP 层传输 功能，这些功能是一个真正的信令连接控制点所需的。这些适用于 STP 转发的无连接消息类：**UDT**（0x09）、**XUDT**（0x11）和 **LUDT**（0x13）。

全球标题转换后的 **SSN** 路由

默认情况下，STP 保留被叫方路由指示符作为路由到 **GT** 当它转发由 GT 路由匹配的消息时。这是故意的：通过 GT 前缀选择下一个跳点是 *GT 模式*（中间），而不是 *最终 GT 模式*。根据 ITU-T Q.714 §2.4，中间节点必须将路由指示符保留为 路由到 GT，以便下游节点执行转换。切换到 路由到 SSN 会错误地将中间结果标记为最终。消息携带一个 GT，路由到 SSN 并且没有点代码，这会导致下游节点（例如 期望运行其自身 GTT 的 MSS）拒绝并返回 `providerAbort(abnormalDialogue)`。

仅当 *a* STP 是最终 翻译器，并且路由交付到无法执行 进一步 GTT 的 SSN 仅端点时，将路由指示符设置为路由到 SSN。这是通过 GT 路由上的 `route_on_ssn: true` 标志进行的显式每路由选择：

```
m3ua_gt_routes: [
  # 传输路由 - 保留路由到 GT（默认）；下游执行 GTT
  %{gt_prefix: "44", peer_id: 2, priority: 1, description: "英国号
码"},

  # 最终交付到 SSN 仅端点 - 切换到路由到 SSN
  %{
    gt_prefix: "1234",
    peer_id: 1,
    priority: 1,
    dest_ssn: 8,
    route_on_ssn: true,
    description: "最终交付到 SSN 仅 MSC"
  }
]
```

该标志也可以通过路由 API 设置（`route_on_ssn` 在 `POST /api/routing/gt-routes` 上），默认为 `false`。启用时，切换仅适用于当前以路由到 GT 地址的被叫方，且携带非零 SSN（ITU-T Q.714 §2.4）。

跳数计数器强制执行（XUDT / LUDT）

XUDT 和 LUDT 消息携带一个 **SCCP** 跳数计数器，以防止在 SCCP 中继节点之间发生路由循环。在每个传输跳中，STP：

1. 递减转发消息中的跳数计数器（Q.714 §2.4）。
2. 如果递减会使计数器达到 零，则消息将 丢弃 而不是转发（Q.714 §2.8.6）。
3. 丢弃会增加 `m3ua_stp_routing_failures_total{reason="hcounter_violation"}`。
4. 如果原始消息请求错误□回，则会生成一个 `XUDTS`/`LUDTS` 消息□回到发起者（见下文）。

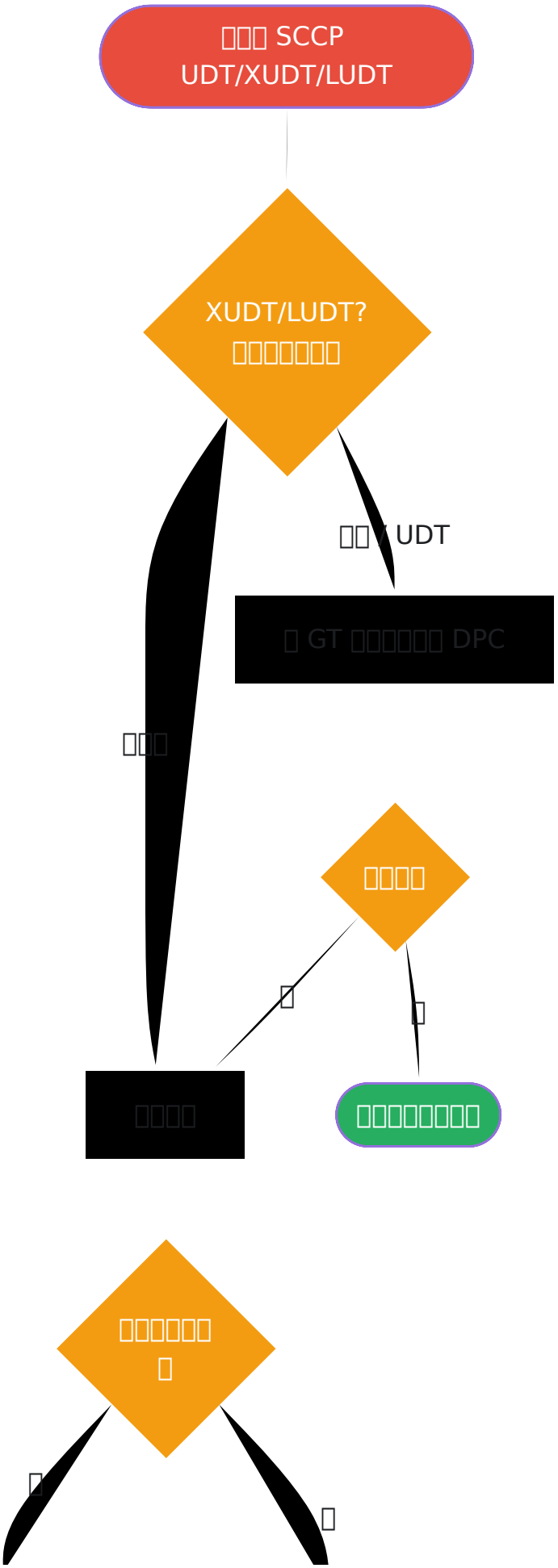
UDT 没有跳数计数器，并在此步骤中保持不变地转发。

路由失败时的消息返回（UDTS / XUDTS / LUDTS）

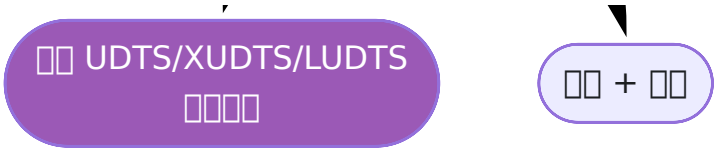
当无连接消息无法送达（没有路由、禁止的 目的地、跳数计数器耗尽等）且发起者设置了 返回错误选项，STP 构建一个 SCCP 消息□回并将其 发送回发起者（ITU-T Q.714 §4.2）：

原始消息	返回为
UDT (0x09)	UDTS (0x0A)
XUDT (0x11)	XUDTS (0x12)
LUDT (0x13)	LUDTS (0x14)

□回原因在消息中反映失败原因（例如，禁止目的地的 MTP 失败，或耗尽计数器的跳数计数器违反）。如果发起者未请求□回错误，则消息将简单丢弃并计数。



传入的 SCCP
UDT/XUDT/LUDT
XUDT/LUDT?
递减跳数计数器
正常 UDT
GT DPC
达
可交付?
转发到目标对等体
请求回错误
?
否



目的地状态（信令路由管理）

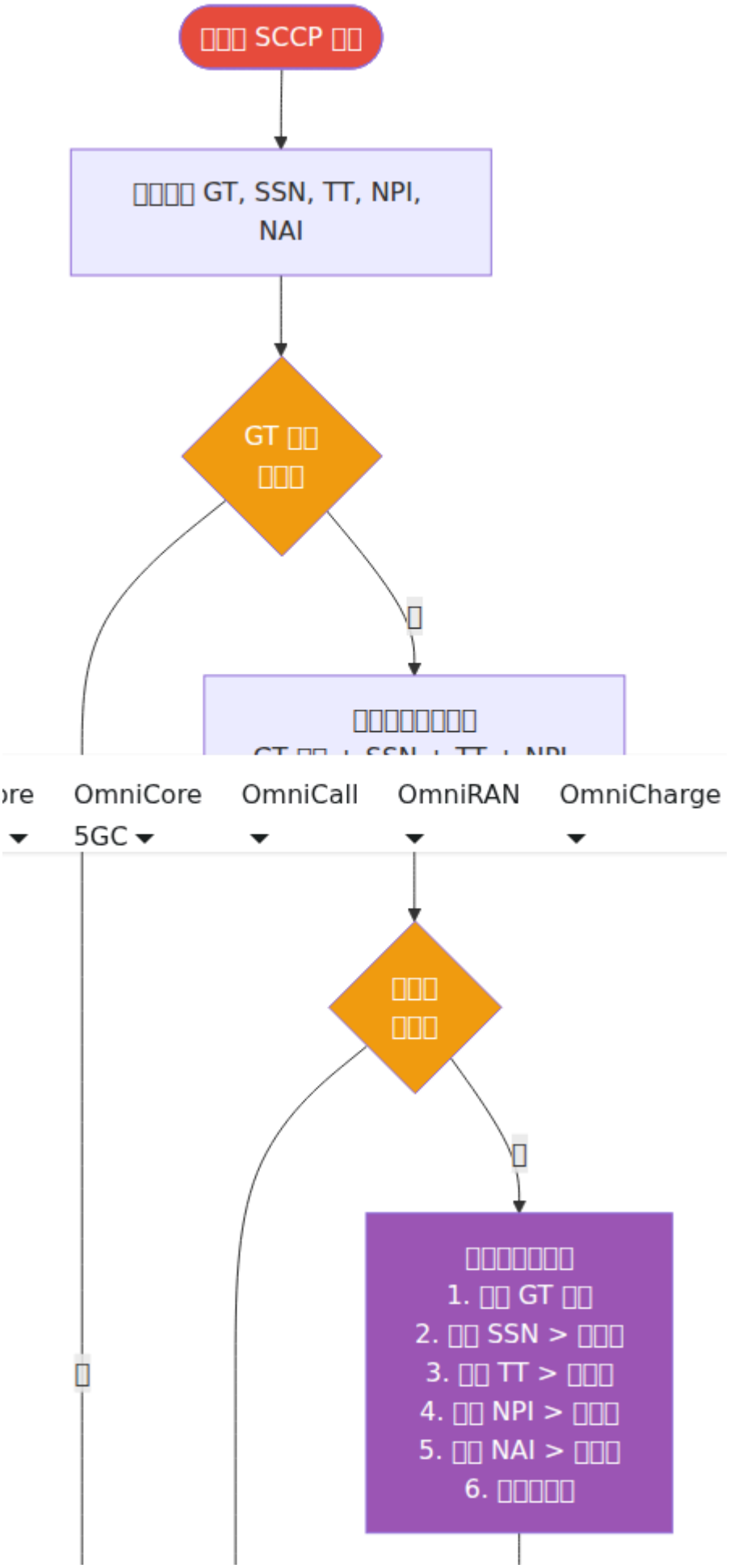
STP 使用 MTP3 信令路由管理消息跟踪每个目的地点代码的可用性，按照 ITU-T Q.704 §13。状态保存在 `m3ua_dest_state` Mnesia 表中；缺少行意味着目的地是 允许的（默认）。

TFP / TFA / TFR 处理

消息	意义	对 **STP** 状态的影响
TFP（禁止转移）	通过此路由不再可达的目的地	目的地标记为 `:prohibited`
TFA（允许转移）	目的地再次可达	清除目的地状态（恢复为允许）
TFR（限制转移）	目的地仅在没有替代方案时使用	目的地标记为 `:restricted`

对路由的影响

- 在点代码查找期间，STP 首先检查目的地状态。标记为 禁止转移 的目的地将被跳过。查找□回 `{:error, :prohibited}`，消息不会路由到那里（Q.704 §13.2.3）。这会增加 `m3ua_stp_routing_failures_total{reason="prohibited"}`。
- **DAUD**（目的地审计）请求根据此实际状态进行响应，而不是以“可用”作为 blanket: 可达的目的地通过 **DAVA** 报告，而不可达或被禁止的目的地通过 **DUNA**（RFC 4666 §3.4.3）。


□□□ SCCP □□
□□□□ GT, SSN, TT, NPI,
NAI
GT □□
□□□
□
□□□□□□□□□
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
□□□
□□□
□
□□□□□□□□
1. □□ GT □□
2. □□ SSN > □□□
3. □□ TT > □□□
4. □□ NPI > □□□
5. □□ NAI > □□□
6. □□□□□
□

网关筛选（**SCCP** 防火墙）

网关筛选允许 STP 在路由之前允许或拒绝传输流量。如果没有它，STP 将无条件转发每个消息。通过它，运营商可以在网关处阻止不必要的信令。这是核心 SS7 安全控制。

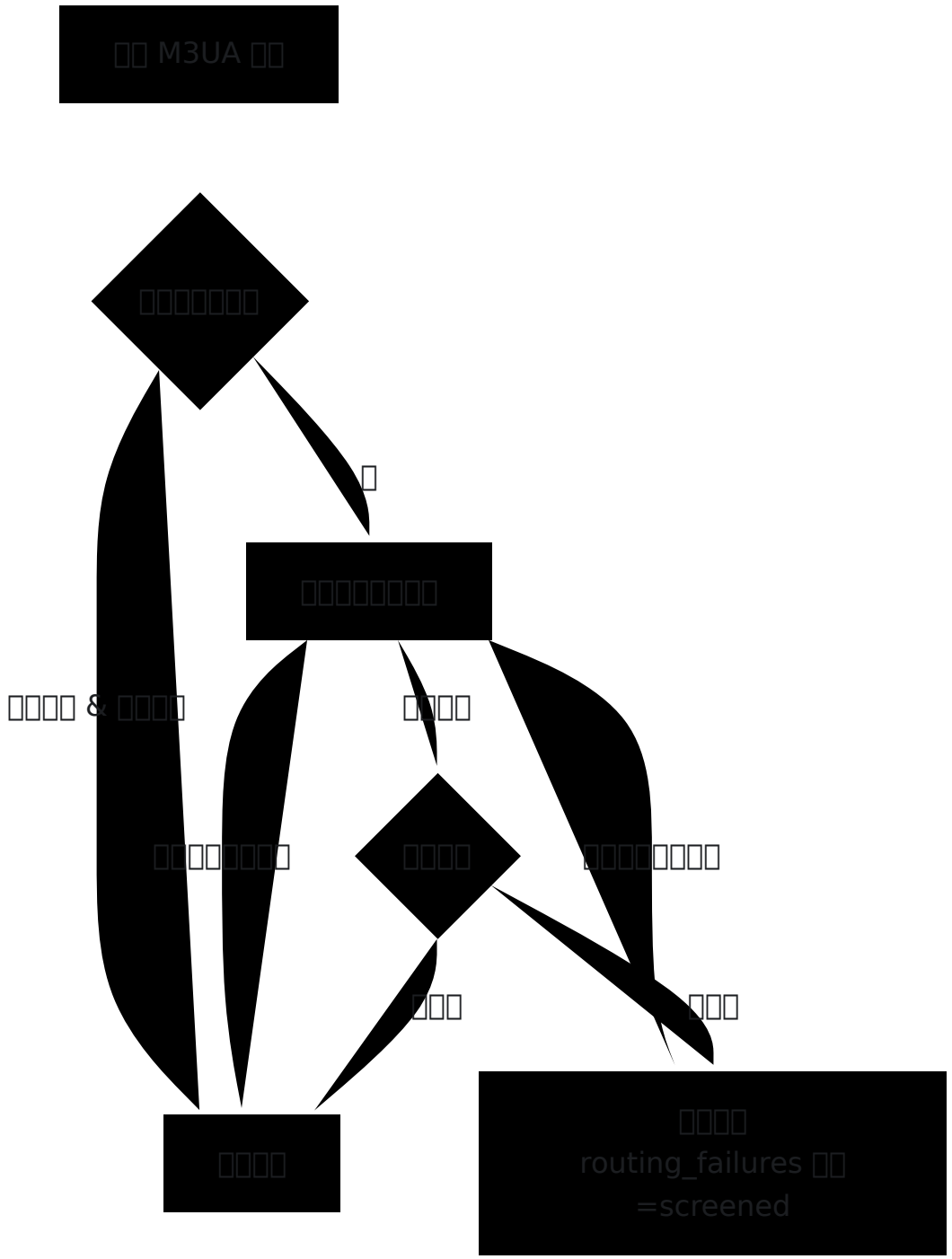
每个传输 M3UA 数据消息都根据一组有序的规则进行评估。规则可以匹配以下属性的任意组合（未设置的属性为通配符）：

属性	匹配
源对等体	消息到达的对等体
OPC	源点代码
DPC	目标点代码
服务指示符	MTP3 SI
被叫方 GTA 前缀	SCCP 被叫方全球标题（前缀匹配）
主叫方 GTA 前缀	SCCP 主叫方全球标题（前缀匹配）
SSN	SCCP 子系统编号

第一个匹配的启用规则（优先级值最低的优先）决定结果：`allow` 或 `deny`。如果没有规则匹配，则应用可配置的 默认操作（`config :omniss7, :screening_default_action`，默认值为 `:allow`）。

被拒绝的消息将 静默丢弃，并计为 `m3ua_stp_routing_failures_total{reason="screened"}`。不会生成 UDTS/□回，因为防火墙不能将阻止信息泄露回发送者。

□□



□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `:allow` □□□□□□□□□□□□□□□□□□□ `:deny` □□□□□□□□□□□□□□

默认操作

参数	类型	默认值	描述
`screening_default_action`	`:allow` | `:deny`	`:allow`	当没有规则匹配时应用的操作。设置为 `:deny` 以实现默认拒绝（允许列表）姿态。

```
config :omniss7,
  # 阻止所有未被规则明确允许的内容。
  screening_default_action: :deny
```

规则存储在 `ss7_screen_rule` Mnesia 表中，并通过编程方式进行管理（每个规则都有一个优先级、上述匹配属性、一个 `allow`/`deny` 操作和一个 `enabled` 标志）。有关默认操作参数，请参见 配置参考。

路由管理功能

禁用路由

可以暂时禁用路由而不删除它们。这对于测试、维护或流量管理非常有用。

启用标志

点代码和全局标题路由都支持可选的 `enabled` 标志：

```
config :omniss7,
  m3ua_routes: [
    # 活动路由
    %{
      dest_pc: 100,
      peer_id: 1,
      priority: 1,
      network_indicator: :international,
      enabled: true  # 路由处于活动状态（如果省略则为默认值）
    },

    # 禁用路由（在路由过程中不进行评估）
    %{
      dest_pc: 200,
      peer_id: 2,
      priority: 1,
      network_indicator: :international,
      enabled: false  # 路由已禁用
    }
  ],

  m3ua_gt_routes: [
    # 禁用 GT 路由
    %{
      gt_prefix: "44",
      peer_id: 1,
      priority: 1,
      description: "英国号码 - 暂时禁用",
      enabled: false
    }
  ]
```

默认行为：

- 如果未指定 `enabled`，则路由默认设置为 `enabled: true`
- 禁用路由在路由查找过程中完全跳过
- 使用 Web UI 切换路由的启用/禁用，而无需编辑配置

用例：

- 测试流量流动变化

临时维护窗口

A/B 测试不同的路由路径

新路由的逐步推出

DROP 路由：防止路由循环

DROP 路由（peer_id: 0）静默丢弃流量，而不是转发它。这可以防止路由循环并启用高级流量过滤。

配置 DROP 路由

config:omniss7,
m3ua_routes: [
# 针对特定点代码的 DROP 路由
%{
dest_pc: 999,
peer_id: 0, # peer_id=0 意味着 DROP priority: 1,
network_indicator::international
}
],

m3ua_gt_routes: [
# 针对 GT 前缀的 DROP 路由
%{
gt_prefix: "999",
peer_id: 0, # peer_id=0 意味着 DROP priority: 99,
description: "阻止测试范围"
}
]

DROP 路由的工作原理

当消息匹配 DROP 路由时：

1. 路由引擎识别 peer_id: 0

2. 消息被静默丢弃（不转发）

3. 生成 **INFO** 日志：`"DROP route matched for DPC 999"` 或 `"DROP route matched for GT 999"`
4. 路由查找□回 `{:error, :dropped}`

重要提示：丢弃的流量在 INFO 级别记录，以便于监控和故障排除。

常见用例：前缀白名单

DROP 路由最强大的用途之一是 前缀白名单 - 仅允许特定号码在大范围内，同时阻止所有其他号码。

模式：

1. 为整个前缀创建一个优先级 高的 **DROP** 路由（例如，99）
2. 为单个号码创建特定的允许路由，优先级 低（例如，1）
3. 由于低优先级的路由会优先评估，因此允许的路由会在 DROP 路由之前匹配
4. 任何未明确允许的号码都会被 DROP 路由捕获

示例场景：

您有一个 GT 前缀 `1234`，表示 10,000 个号码的范围（1234000000 - 1234999999），但您只想路由 3 个特定号码：`1234567890`、`1234555000` 和 `1234111222`。

```
config :omniss7,
  m3ua_gt_routes: [
    # 优先级高的 DROP 路由（最后评估）
    %{
      gt_prefix: "1234",
      peer_id: 0,               # DROP
      priority: 99,             # 高数字 = 低优先级 = 最后评估
      description: "阻止所有 1234* 除非是白名单号码"
    },

    # 优先级低的特定允许路由（优先评估）
    %{
      gt_prefix: "1234567890",
      peer_id: 1,               # 路由到对等方 1
      priority: 1,              # 低数字 = 高优先级 = 优先评估
      description: "允许的号码 1"
    },
    %{
      gt_prefix: "1234555000",
      peer_id: 1,
      priority: 1,
      description: "允许的号码 2"
    },
    %{
      gt_prefix: "1234111222",
      peer_id: 1,
      priority: 1,
      description: "允许的号码 3"
    }
  ]
```

路由行为：

来自 **GT**	匹配路由	选择的路由	操作
1234567890	□ "1234567890"（优先级 1） □ "1234" DROP（优先级 99）	"1234567890"（最具体，最高优先级）	路由到对等方 **1**
1234555000	□ "1234555000"（优先级 1） □ "1234" DROP（优先级 99）	"1234555000"（最具体，最高优先级）	路由到对等方 **1**
1234111222	□ "1234111222"（优先级 1） □ "1234" DROP（优先级 99）	"1234111222"（最具体，最高优先级）	路由到对等方 **1**
1234999999	□ "1234" DROP（优先级 99）	"1234" DROP（唯一匹配）	丢弃 + 记录
1234000000	□ "1234" DROP（优先级 99）	"1234" DROP（唯一匹配）	丢弃 + 记录

结果：

- □ □ 3 个特定号码被路由到对等方 1
- □ □□□□ `1234*` 号码被静默丢弃
- □ 所有丢弃的流量被记录以便监控

生成的日志：

```
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for GT 1234000000
```

针对点代码的 **DROP** 路由

相同的白名单模式适用于点代码路由：

```
config :omniss7,
  m3ua_routes: [
    # DROP □□□□ /8□64 □□□□□1000-1063□
    %{
      dest_pc: 1000,
      peer_id: 0,
      priority: 99,
      mask: 8,
      network_indicator: :international
    },

    # □□□□□ PC
    %{dest_pc: 1010, peer_id: 1, priority: 1, network_indicator:
:international},
    %{dest_pc: 1020, peer_id: 1, priority: 1, network_indicator:
:international},
    %{dest_pc: 1030, peer_id: 1, priority: 1, network_indicator:
:international}
  ]
```

□□□ □ PC 1010□1020 □ 1030 □□□□1000-1063 □□□□□□□□□ PC □□□□□

□□ DROP □□

□□□□□

```
# □□□□□□□□
tail -f logs/app.log | grep "DROP route matched"

# □□□□□
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for DPC 1050
```

□□ Web UI□

- □□□ □□□□ □□□
- □ `INFO` □□□□
- □□ "DROP route matched"

□□□□□



1. ⚠ 定期监控日志，以确保 DROP 路由不会阻止合法流量
2. □ 使用描述性 `description` 字段记录路由被丢弃的原因
3. □ 使用高优先级数字（90-99）为 DROP 路由，以确保它们是捕获所有路由
4. □ □□□□□□□□□□ DROP 路由行为
5. □ 设置警报以监控丢弃流量的意外增加

高级路由：基于 **SSN** 的路由和重写

子系统号码（**SSN**）

子系统号码识别应用层：

- **SSN 6**: HLR（家庭位置寄存器）
- **SSN 7**: VLR（访客位置寄存器）
- **SSN 8**: MSC（移动交换中心）/ SMSC（短信中心）
- **SSN 9**: GMLC（网关移动位置中心）

基于 **SSN** 的路由示例

根据号码前缀将 SMS 流量���由到不同的 HLR：

```
m3ua_gt_routes: [
  # 将英国号码的 SMS 路由到英国 HLR，将 SSN 从 8（SMSC）重写为 6（HLR）
  %{
    gt_prefix: "44",
    source_ssn: 8,                    # 匹配传入 SSN 8（SMSC）
    peer_id: 1,
    dest_ssn: 6,                      # 重写为 SSN 6（HLR）
    priority: 1,
    description: "英国 SMS 到 HLR"
  },

  # 将英国号码的语音流量（SSN 6）路由而不重写
  %{
    gt_prefix: "44",
    source_ssn: 6,                    # 匹配传入 SSN 6（HLR）
    peer_id: 1,
    dest_ssn: nil,                    # 无 SSN 重写
    priority: 1,
    description: "英国语音流量"
  }
]
```

测试 **STP** 路由配置

配置好对等方和路由后，验证您的配置：

1. 检查对等方状态

通过 **Web UI**：

- 导航到 http://localhost
- 检查 M3UA 状态页面
- 验证对等方显示 状态：**ACTIVE**

通过 **IEx** 控制台：

```
# □□□□□□□□□□□
SCTP.SocketHandler.get_peers_status()

# □□□□□□
# [
#   %{peer_id: 1, name: "Partner_STP_West", status: :active,
point_code: 100, ...},
#   %{peer_id: 2, name: "Local_HLR", status: :active, point_code:
200, ...}
# ]
```

2. □□□□□□□□

```
# □ DPC 100 □□□□□ M3UA □□
test_payload = <<1, 2, 3, 4>>  # □□□□
SCTP.SocketHandler.route_by_pc(100, test_payload, 0)

# □□□□□□□□□□□□□□
# □□□□□□"Routing message: OPC=... -> DPC=100 via peer 1"
```

3. □□□□□□□□□

```
# □□□□□ GT □□
M3UARouting.lookup_peer_by_gt("447712345678")

# □□ {:ok, peer, rewrites}□`peer` □ 13 □□□ m3ua_peer □□□
# `rewrites` □□□□□□□ MAP□nil = □□□□□□□
# {:ok, {:m3ua_peer, 3, "UK_Mobile_Peer", ...},
#       %{dest_ssn: nil, dest_tt: nil, dest_npi: nil, dest_nai:
nil}}

# □□□ SSN □ GT □□□□□□ SSN 8 → 6 □□□□
M3UARouting.lookup_peer_by_gt("555881234567", 8)

# SSN □□□□□□□□□□□□□□□□□□□□□□□□□□□□
# {:ok, {:m3ua_peer, 4, "SMS_HLR_Peer", ...},
#       %{dest_ssn: 6, dest_tt: nil, dest_npi: nil, dest_nai:
nil}}
```

4. □□□□□□

□□ `/metrics` □ Prometheus □□

□□□□□

```
# □□□□□□□□□□□□
m3ua_stp_messages_received_total{peer_name="Partner_STP_West",point_c
1523

# □□□□□□□□□□□□
m3ua_stp_messages_sent_total{peer_name="Local_HLR",point_code="200"}

# □□□□□□□□□□
m3ua_stp_routing_failures_total{reason="no_route"} 5
m3ua_stp_routing_failures_total{reason="no_gt_route"} 2
m3ua_stp_routing_failures_total{reason="no_active_peer"} 1
m3ua_stp_routing_failures_total{reason="prohibited"} 3
m3ua_stp_routing_failures_total{reason="hcounter_violation"} 4
m3ua_stp_routing_failures_total{reason="screened"} 7
```

STP □标和监控

可用□标

每个对等方流量□标:

- `m3ua_stp_messages_received_total` - 从每个对等方接收的总消息
  - 标签: `peer_name`, `point_code`
- `m3ua_stp_messages_sent_total` - 转发到每个对等方的总消息
  - 标签: `peer_name`, `point_code`

路由失败□标:

- `m3ua_stp_routing_failures_total` - 按原因统计的路由失败次数
  - 标签: `reason`

`reason`	含义
`no_route`	找不到目标的点代码路由
`no_gt_route`	找不到全局标题路由，并且 PC 路由也失败
`no_active_peer`	匹配了路由，但其对等方未知/未激活
`prohibited`	目标被标记为禁止转移（收到了 TFP, Q.704 §13）
`hcounter_violation`	SCCP 跳数计数器在 XUDT/LUDT 上达到零（Q.714 §2.8.6）

□标解释

- 高消息计数：表示活跃的流量流动
- 路由失败：表示缺少路由、不可达的目标或循环消息
  - `no_route`：找不到目标的点代码路由
  - `no_gt_route`：找不到全局标题路由，并且 PC 路由也失败
  - `no_active_peer`：路由存在，但目标对等方已关闭/未知
  - `prohibited`：目标被信令路由管理标记为禁止转移

- `hcounter_violation`：跳数计数器耗尽；消息被丢弃（如果设置了错误□回则□回）

使用□标进行故障排除

场景：没有流量到达目标

1. 检查是否接收到消息：

```
m3ua_stp_messages_received_total{peer_name="Source_Peer"} > 0
```

2. 检查是否发送了消息：

```
m3ua_stp_messages_sent_total{peer_name="Dest_Peer"} > 0
```

3. 检查路由失败：

```
m3ua_stp_routing_failures_total{reason="no_route"} > 0
```

解决方案：如果路由失败较高，请在配置中添加缺失的路由。

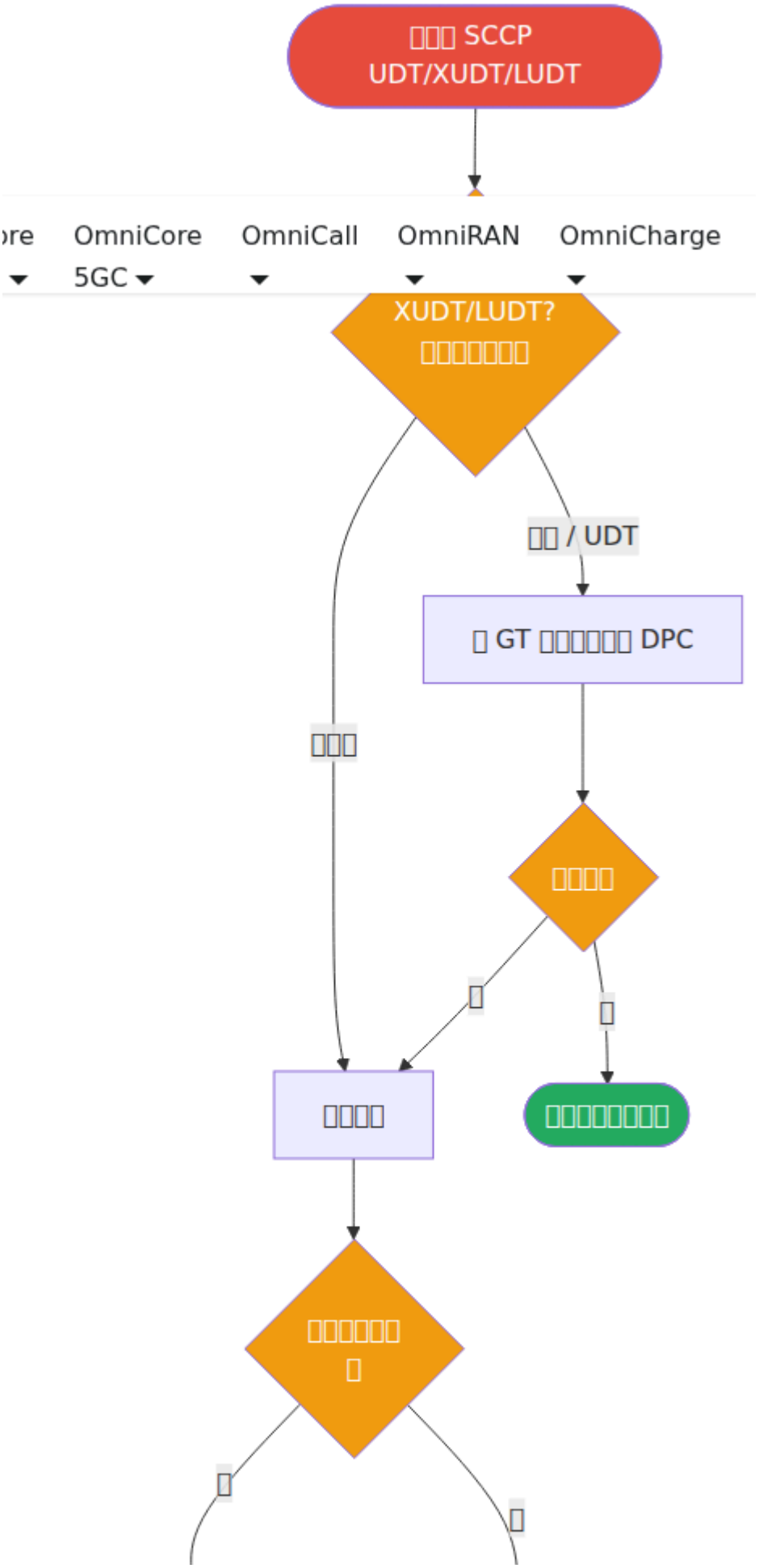
M3UA 对等方状态监控

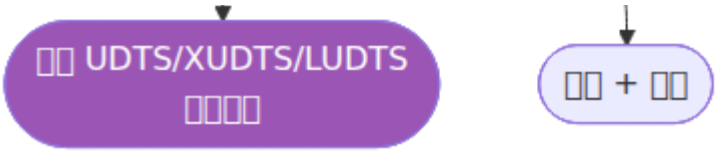
理解 **M3UA**

M3UA（MTP3 用户适配层）是一种协议，允许通过 SCTP 在 IP 网络上传输 SS7 信令。

M3UA 连接状态

M3UA 连接经历多个状态：


□□□ SCCP
UDT/XUDT/LUDT
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
XUDT/LUDT?
□□□□□□□□
□□ / UDT
□ GT □□□□□□ DPC
□□□
□□□□
□
□
□□□□
□□□□□□□□□
□□□□□□□
□
□
□



状态描述：

- **DOWN** - 没有 SCTP 连接
- **CONNECTING** - SCTP 连接正在进行
- **ASPUP_SENT** - 等待 ASPUP 确认
- **INACTIVE** - ASP 已启动但未主动承载流量（可以通过 ASPAC 转换为 ACTIVE）
- **ASPAC_SENT** - 等待 ASPAC 确认
- **ACTIVE** - 准备接收流量，完全操作
- **ASPDOWN_SENT** - 正在进行优雅关闭

通过 **Web UI** 监控 **M3UA** 对等方

Web UI 提供 M3UA 对等方连接的实时监控。

访问 **M3UA** 状态页面：

1. 导航到 Web UI 首页
2. 点击导航菜单中的 "M3UA 状态"
3. 页面每秒自动刷新

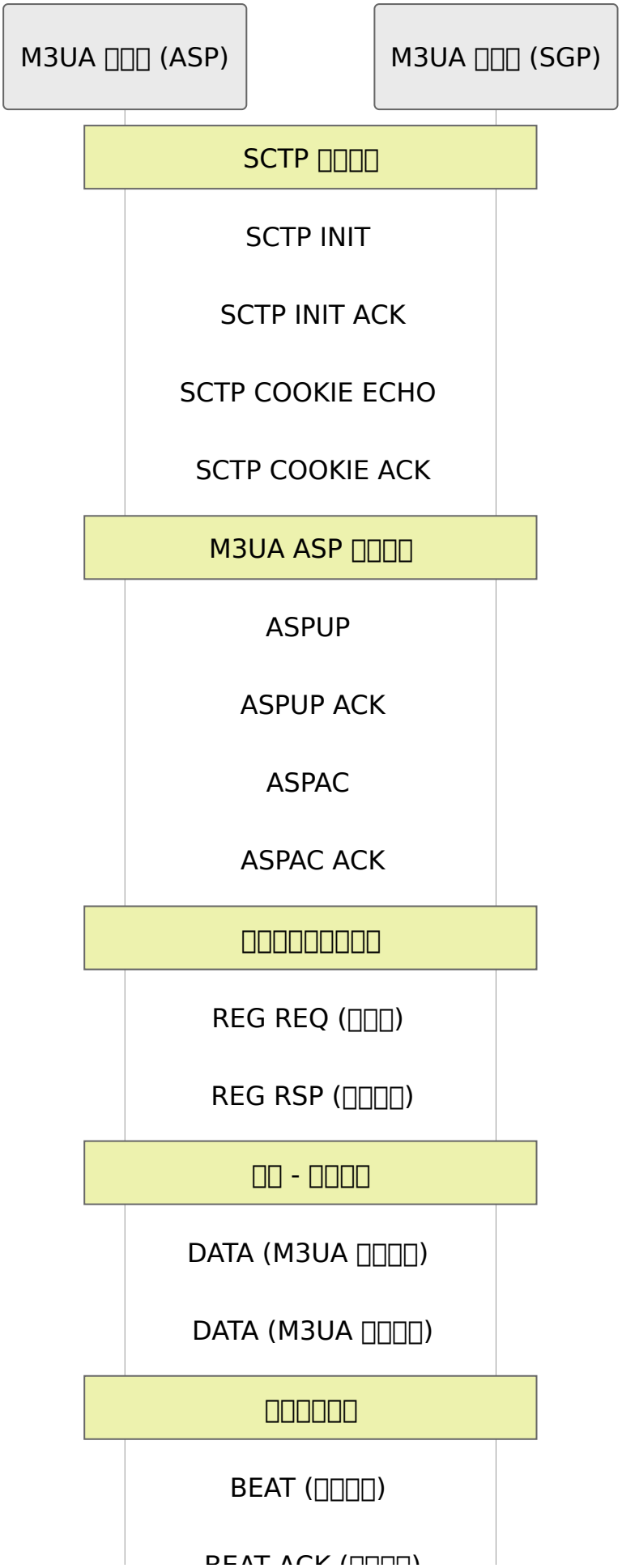
M3UA 状态表：

列	描述
名称	连接名称（例如，`testASP`）
PID	进程标识符
状态	UP（绿色）或 DOWN（红色）
ASP 状态	当前 M3UA 状态（例如，ACTIVE, INACTIVE）
Assoc/SCTP	SCTP 关联状态
本地	本地 IP:端口
远程	远程 IP:端口
RC	路由上下文 ID

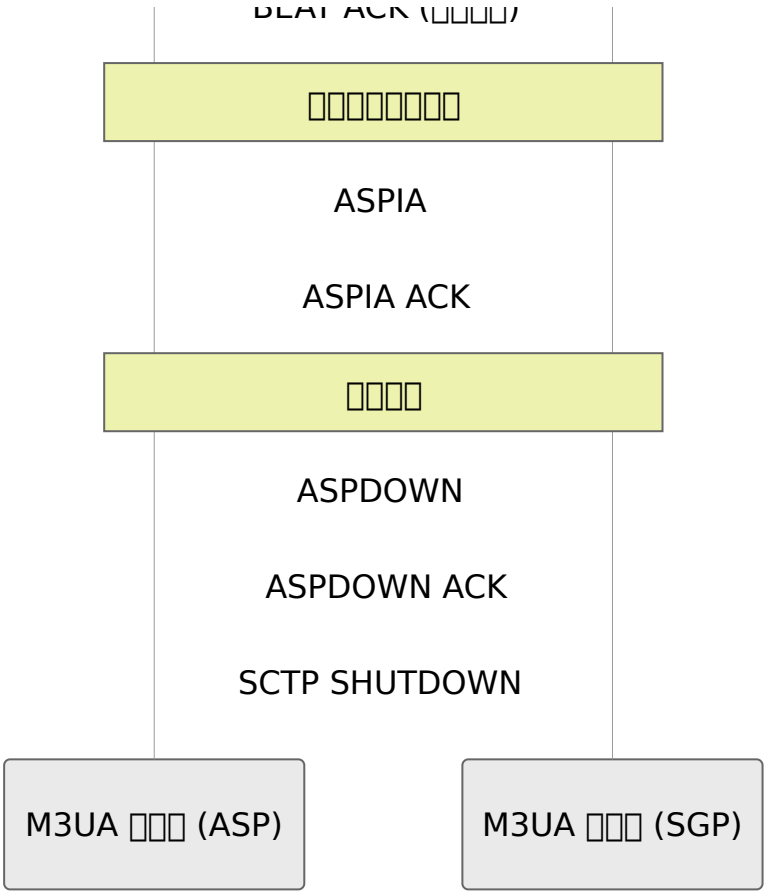
状态□示器:

- 绿色（**UP**） - 连接处于活动状态且健康
- 红色（**DOWN**） - 连接已关闭或不可用
- **ASP** 状态 - 显示当前 M3UA 连接状态
- **Assoc/SCTP** - 显示 SCTP 关联状态

M3UA □□□



M3UA 客户端 (ASP)
M3UA 服务器 (SGP)
SCTP 关联设置
SCTP INIT
SCTP INIT ACK
SCTP COOKIE ECHO
SCTP COOKIE ACK
M3UA ASP 状态管理
ASPUP
ASPUP ACK
ASPAC
ASPAC ACK
路由键管理（可选）
REG REQ (路由键)
REG RSP (注册结果)
活动 - 数据传输
DATA (M3UA 协议数据)
DATA (M3UA 协议数据)
心跳（定期）
BEAT (心跳数据)



RFC 4666 M3UA □□□□

OmniSS7 □□□ RFC 4666 □□□□□□ M3UA □□  □□□

类别	类型	消息	方向	行为
0 (MGMT)	0	ERR	双向	记录错误代码和诊断信息
0 (MGMT)	1	NTFY	双向	记录通知
1 (XFER)	1	DATA	双向	通过 PC 或 GT 路由负载
2 (SSNM)	1	DUNA	接收	记录目标不可用
2 (SSNM)	2	DAVA	接收	记录目标可用
2 (SSNM)	3	DAUD	接收	根据目标状态响应：可达 PC □回 DAVA，无法达/禁止的 PC □回 DUNA
2 (SSNM)	4	SCON	接收	记录拥塞
2 (SSNM)	5	DUPU	接收	记录用户部分不可用
2 (SSNM)	6	DRST	接收	记录目标受限
3 (ASPSM)	1	ASPUP	双向	发送/接收 ASPUP ACK
3 (ASPSM)	2	ASPDN	双向	发送 ASPDN ACK，拆除对等方路由
3 (ASPSM)	3	BEAT	双向	在 BEAT ACK 中回显心跳数据
3 (ASPSM)	4	ASPUP ACK	接收	将对等方转换为 UP
3 (ASPSM)	5	ASPDN ACK	接收	将对等方转换为 DOWN

类别	类型	消息	方向	行为
3 (ASPSM)	6	BEAT ACK	接收	确认心跳
4 (ASPTM)	1	ASPAC	双向	发送/接收 ASPAC ACK，激活路由
4 (ASPTM)	2	ASPIA	双向	发送 ASPIA ACK，将对等方转换为 INACTIVE
4 (ASPTM)	3	ASPAC ACK	接收	将对等方转换为 ACTIVE
4 (ASPTM)	4	ASPIA ACK	接收	将对等方转换为 INACTIVE
9 (RKM)	1	REG REQ	SGP 接收	解析路由键，响应 REG RSP
9 (RKM)	2	REG RSP	ASP 接收	记录注册结果
9 (RKM)	3	DEREG REQ	SGP 接收	响应 DEREG RSP
9 (RKM)	4	DEREG RSP	ASP 接收	记录注销结果

心跳（**BEAT/BEAT ACK**）：当 STP 接收到 BEAT 消息时，它会在 BEAT ACK 中回显心跳数据。这允许对等方在不依赖 SCTP 级别超时的情况下检测链路故障。

路由键管理（**RKM**）：当 ASP 发送 REG REQ 时，SGP 解析路由键（本地 RK 标识符、路由上下文、流量模式、DPC）并响应 REG RSP，包含注册状态。这使得动态路由配置成为可能，而无需静态配置。

ASP 下线（**ASPDN**）：当 SGP 接收到 ASPDN 时，它会响应 ASPDN ACK，将对等方标记为 DOWN，并从路由表中删除相关路由。

ASP 非活动（**ASPIA**）：当 SGP 接收到 ASPIA 时，它会响应 ASPIA ACK，并将对等方转换为 INACTIVE，暂停流量而不拆除关联。

M3UA 路由上下文处理

路由上下文（**RC**）是 M3UA 中的一个 32 位标识符，标识每个端点的应用服务器（AS）上下文。理解路由上下文在信令路径中不同点的处理方式对于正确的网络配置至关重要。

什么是路由上下文?

根据 RFC 4666 第 3.3.1 节，路由上下文：

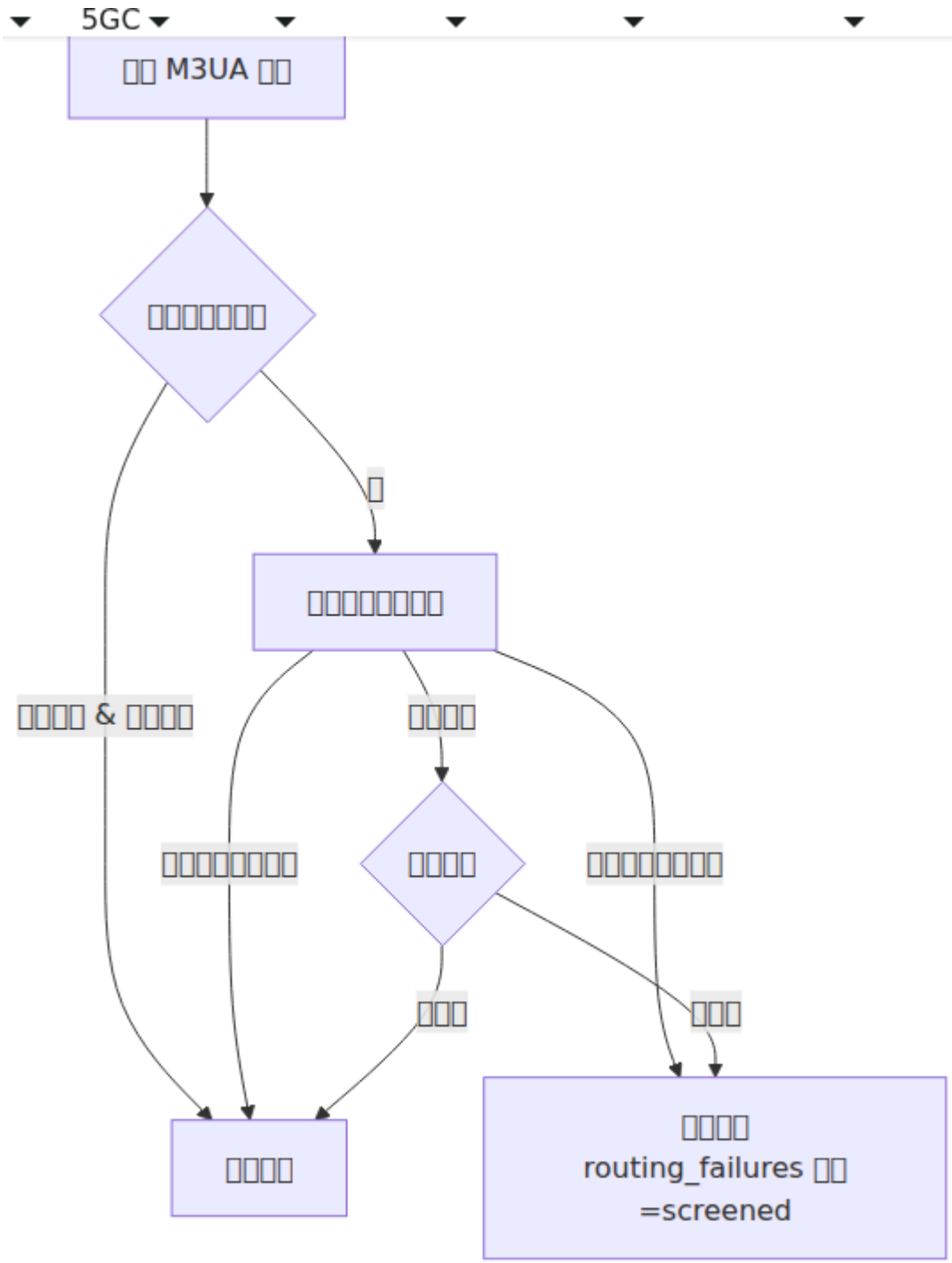
- 唯一标识端点的应用服务器（AS）
- 在 M3UA DATA 消息中携带（参数标签 6）
- 允许信令网关确定哪个 AS 应该接收消息
- 在每跳中有意义，而不是端到端

M3UA 协议中的路由上下文

路由上下文参数（标签 6）出现在 M3UA DATA 消息中，与协议数据参数（标签 528）一起，后者包含实际的 MTP3/SCCP 负载。

OmniSS7 如何处理路由上下文

OmniSS7 根据组件和消息方向以不同方式处理路由上下文：



MAP 客户端和 **CAP** 客户端

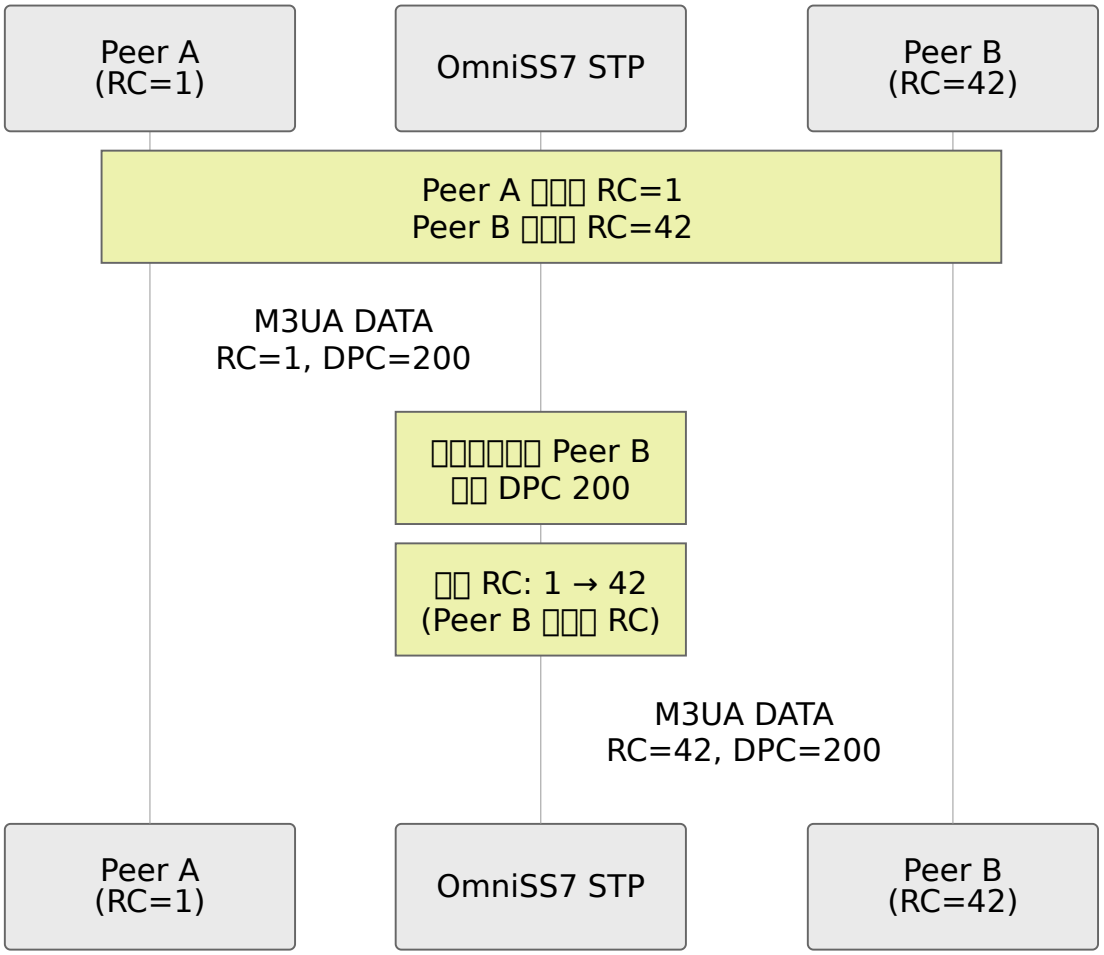
对于应用层客户端（MAP 和 CAP），路由上下文的处理如下：

场景	**RC** 处理	配置
响应传入请求	在响应中镜像传入的 RC	自动 - 无需配置
客户端发起请求 (SRI, PRN, IDP, 等)	使用应用配置中的配置 RC	在 `map_client_m3ua` 或 `cap_client_m3ua` 中设置 `routing_context`
传入请求中没有 **RC**	从响应中省略 RC	保持向后兼容性
客户端请求没有配置 **RC**	从发出消息中省略 RC	保持向后兼容性

配置示例:

```
config :omniss7,
  # MAP 客户端 RC 配置
  map_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 0, 0, 1},
    local_port: 2905,
    remote_ip: {10, 0, 0, 10},
    remote_port: 2905,
    routing_context: 1    # 用于客户端发起的 MAP 请求 (SRI, PRN, 等)
  },

  # CAP 客户端 RC 配置
  cap_client_m3ua: %{
    mode: "ASP",
    local_ip: {10, 0, 0, 1},
    local_port: 2906,
    remote_ip: {10, 0, 0, 20},
    remote_port: 2906,
    routing_context: 2    # 用于客户端发起的 CAP 请求 (InitialDP, 等)
  }
```

STP 路由上下文重写



当 STP 在对等体之间转发消息时，它会 重写 路由上下文为目标对等体配置的 RC。这是根据 RFC 4666 的正确行为，因为：

1. RC 在每个端点识别 AS 上下文
2. 不同的对等体可能与不同的 AS 上下文相关联
3. STP 在不同 AS 之间路由时会转换 RC 值

STP RC 重写流程：

配置：

每个对等体的路由上下文在对等体定义中配置：

```
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "Network_A_STP",
      role: :server,
      remote_ip: {10, 0, 0, 100},
      remote_port: 2905,
      routing_context: 1,        # □□□□□□□□ RC
      point_code: 100,
      network_indicator: :international
    },
    %{
      peer_id: 2,
      name: "Network_B_HLR",
      role: :client,
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 200},
      remote_port: 2905,
      routing_context: 42,       # □□□□□□□□ RC
      point_code: 200,
      network_indicator: :international
    }
  ]
```

□□□□□□□ 1□RC=1□□□□□□□□□□□□□□□ 2 □□STP□

1. □□ DPC □□□□□□200 → □□□ 2□
2. □ RC □ 1 □□□ 42□□□□ 2 □□□ RC□
3. □□□□□□□□□□ 2

RC □□□□

组件	传入请求	发出响应	客户端发起
MAP 客户端	提取 RC	镜像传入的 RC	使用配置的 RC
CAP 客户端	提取 RC	镜像传入的 RC	使用配置的 RC
STP	提取 RC	N/A	重写为目标对等体的 RC

当 **RC** 被省略时

如果某个对等体没有配置 `routing_context`（或设置为 `nil`）：

- **MAP/CAP** 客户端响应：响应中省略 RC 参数
- **MAP/CAP** 客户端请求：请求中省略 RC 参数
- **STP** 转发：RC 不被重写（原始 RC 保留或省略）

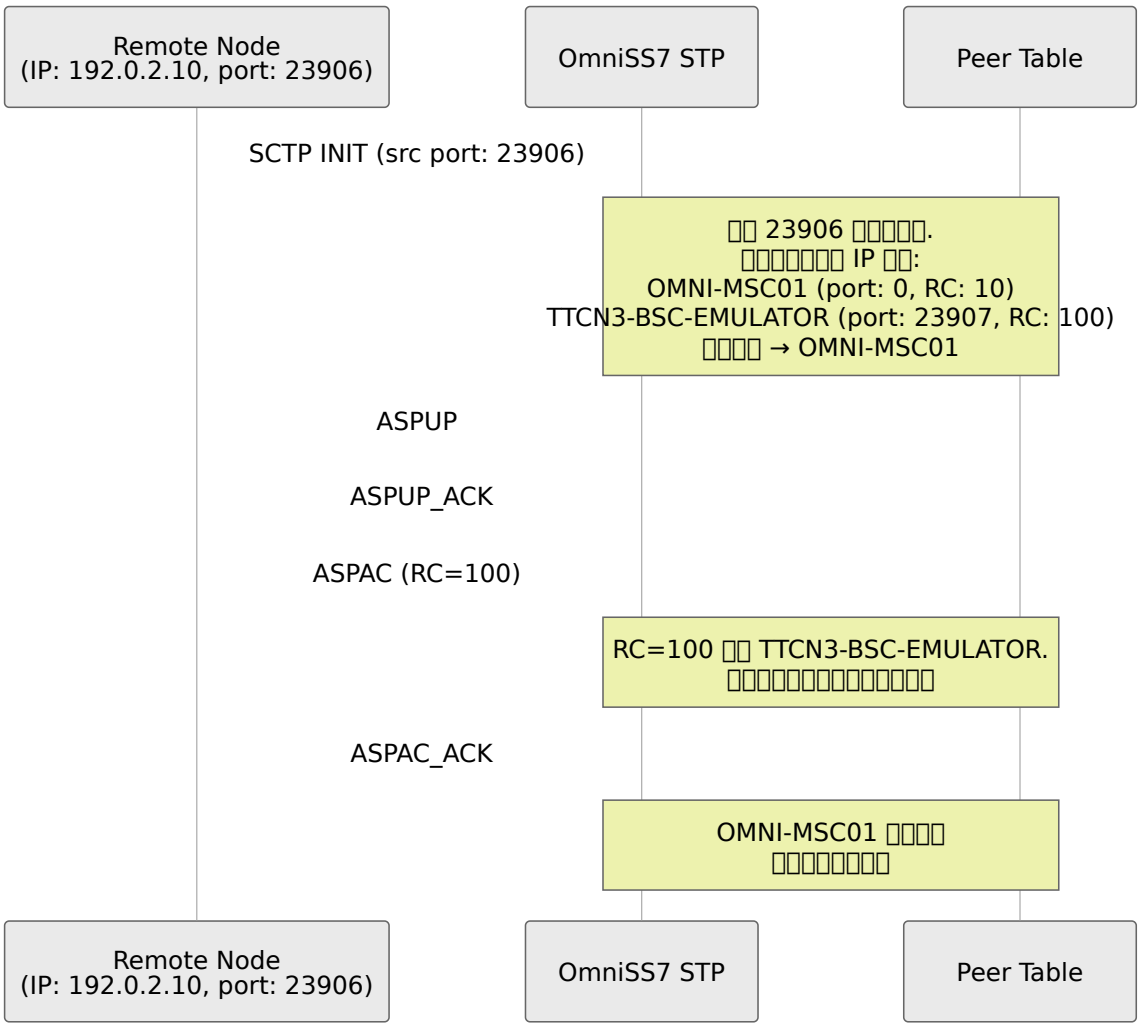
这保持了与不使用或不期望路由上下文的对等体的向后兼容性。

同 **IP** 多对等体歧义消解

两个配置的对等体可以共享相同的远程 IP 地址：例如，生产 MSC 和在同一主机上运行的测试仿真器。当传入的 SCTP 连接到时，OmniSS7 无法仅从源 IP 确定正确的对等体身份。配置为 `remote_port: 0`（通配符）的对等体将匹配来自该 IP 的任何连接，这可能导致来自另一个对等体的新连接暂时分配给错误的条目。

OmniSS7 使用 **M3UA ASPAC** 路由上下文 自动解决此问题。连接被接受并暂时分配给最佳匹配的对等体（特定端口匹配优先于通配符）。当远程端发送携带路由上下文的 `ASPAC` 消息时，系统将该 RC 与每个对等体配置的 `routing_context` 进行比较。如果 RC 匹配不同的对等体，连接将被静默重新映射到正确的对等体。

工作原理


Remote Node
(IP: 192.0.2.10, port: 23906)
OmniSS7 STP
Peer Table
SCTP INIT (src port: 23906)
□□ 23906 □□□□□.
□□□□□□□□ IP □□:
OMNI-MSC01 (port: 0, RC: 10)
TTCN3-BSC-EMULATOR (port: 23907, RC: 100)
□□□□ → OMNI-MSC01
ASPUP
ASPUP_ACK
ASPAC (RC=100)
RC=100 □□ TTCN3-BSC-EMULATOR.
□□□□□□□□□□□□□□□□
ASPAC_ACK
OMNI-MSC01 □□□□
□□□□□□□□□
Remote Node
(IP: 192.0.2.10, port: 23906)
OmniSS7 STP
Peer Table

□□□□

□□□□□□□□□□□□□□□□□OmniSS7 □□□□□□□□□□□□□□□□□□

为了使同 IP 对等体歧义消解正常工作，每个对等体必须配置唯一的 **`routing_context`**。如果同一 IP 上的两个对等体共享相同的 RC（或两者均为 `nil`），则无法确定哪个对等体是正确的。

```
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 8,
      name: "OMNI-MSC01",
      role: :server,
      remote_ip: {10, 5, 198, 200},
      remote_port: 0,               # 通配符 - 接受任何源端口
      routing_context: 10,          # 在共享此 IP 的对等体中必须唯一
      point_code: 500,
      network_indicator: :national
    },
    %{
      peer_id: 9,
      name: "TTCN3-BSC-EMULATOR",
      role: :server,
      remote_ip: {10, 5, 198, 200},
      remote_port: 23907,           # 预期的源端口（可能与临时端口不同）
      routing_context: 100,         # 不同的 RC - 用于在 ASPAC 后识别此对等体
      point_code: 193,
      network_indicator: :national
    }
  ]
```

> 关于临时源端口的说明：端口匹配是精确的。如果一个对等体配置为 `remote_port: 23907` 但从端口 23906 连接，则该对等体将被排除在初始匹配之外，而通配符对等体将获胜。这是预期的。ASPAC 路由上下文是权威身份信号。对等体上配置的 `remote_port` 仅用于初始连接分配启发式，而不是最终身份。给每个对等体一个不同的 `routing_context`，无论远程使用哪个源端口，重新映射都将正确解决。

故障排除 **M3UA** 连接

问题：连接无法建立

症状：

- 状态显示为 DOWN

- 没有 SCTP 关联

检查:

1. 验证网络连接: `ping remote_ip`
2. 检查防火墙是否允许 SCTP（协议 132）
3. 验证远程 STP/SGP 是否在正确端口上监听
4. 检查配置中的 `remote_ip` 和 `remote_port`
5. 查看应用日志以获取 SCTP 错误

问题: 连接已建立但 **ASP** 未激活

症状:

- SCTP 关联存在
- ASP 状态停留在 INACTIVE 或 ASPUP_SENT

检查:

1. 验证路由上下文与远程配置匹配
2. 检查远程 STP 是否接受您的点代码
3. 查看日志以获取 ASPUP/ASPAC 拒绝信息
4. 验证没有身份验证/安全要求

问题: ��据未流动

症状:

- ASP 状态显示为 ACTIVE
- 没有消息被路由

检查:

1. 验证消息中的路由上下文
2. 检查 SCCP 地址（GT 格式，SSN 值）
3. 验证路由表配置正确
4. 查看 `/events` 页面以获取 SCCP 错误
5. 检查 STP 级别的点代码路由

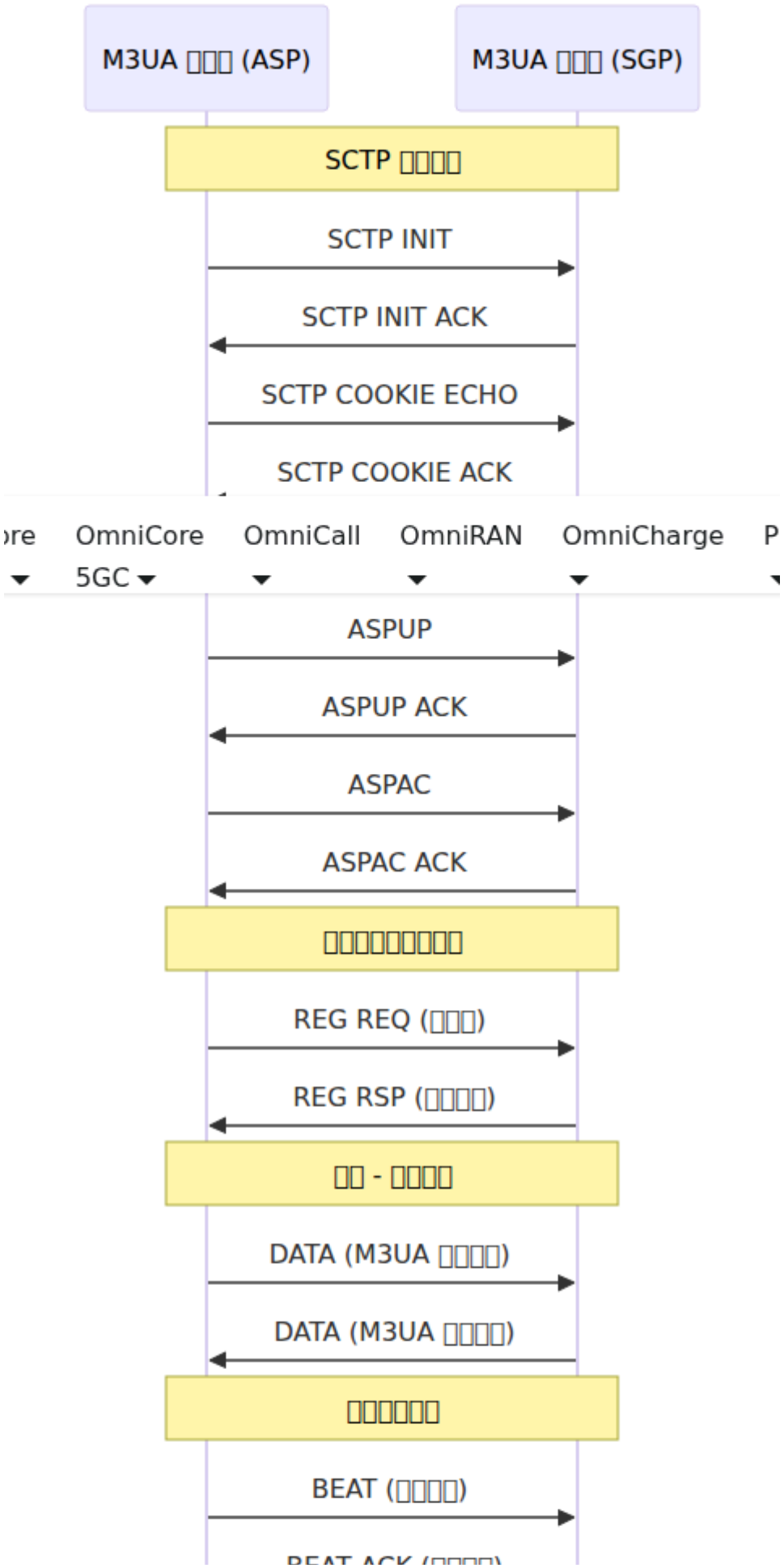
M2PA 对等体状态监控

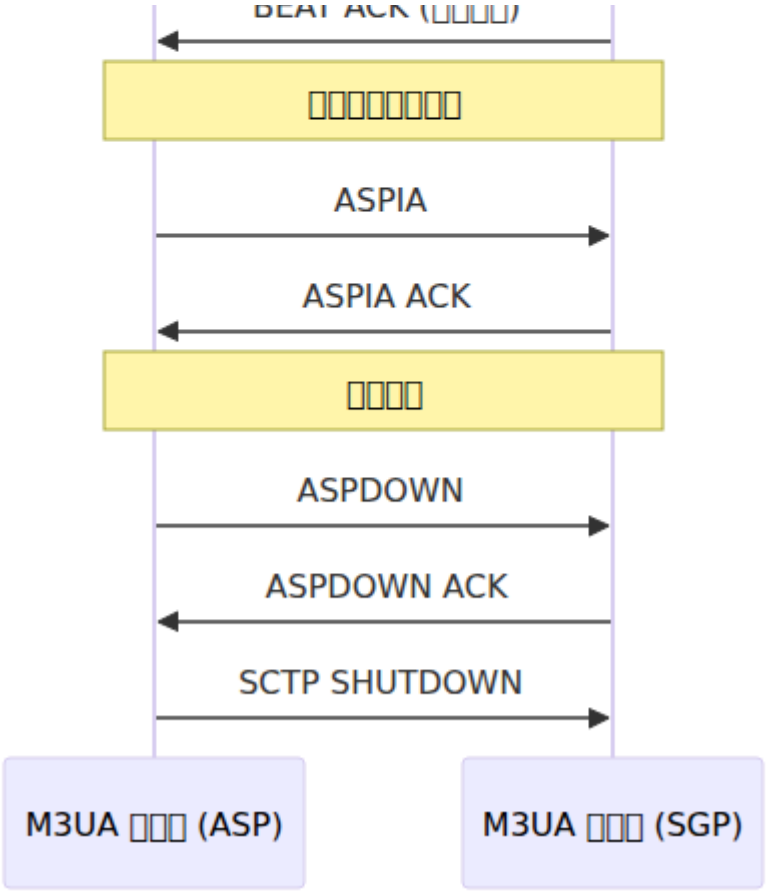
理解 M2PA

M2PA（MTP2 用户对等体到对等体适配层）是 RFC 4165 中定义的协议，提供通过 SCTP 的点对点 MTP3 消息传输。与使用 ASP/SGP 架构的 M3UA 不同，M2PA 提供类似于传统 TDM SS7 链路的对等体链接。

M2PA 链路状态

M2PA 链路在建立过程中经历几个状态：


M3UA □□□ (ASP)
M3UA □□□ (SGP)
SCTP □□□□
SCTP INIT
SCTP INIT ACK
SCTP COOKIE ECHO
SCTP COOKIE ACK
ore
OmniCore
5GC
OmniCall
OmniRAN
OmniCharge
P
ASPUP
ASPUP ACK
ASPAC
ASPAC ACK
□□□□□□□□□□
REG REQ (□□□)
REG RSP (□□□□)
□□ - □□□□
DATA (M3UA □□□□)
DATA (M3UA □□□□)
□□□□□□
BEAT (□□□□)



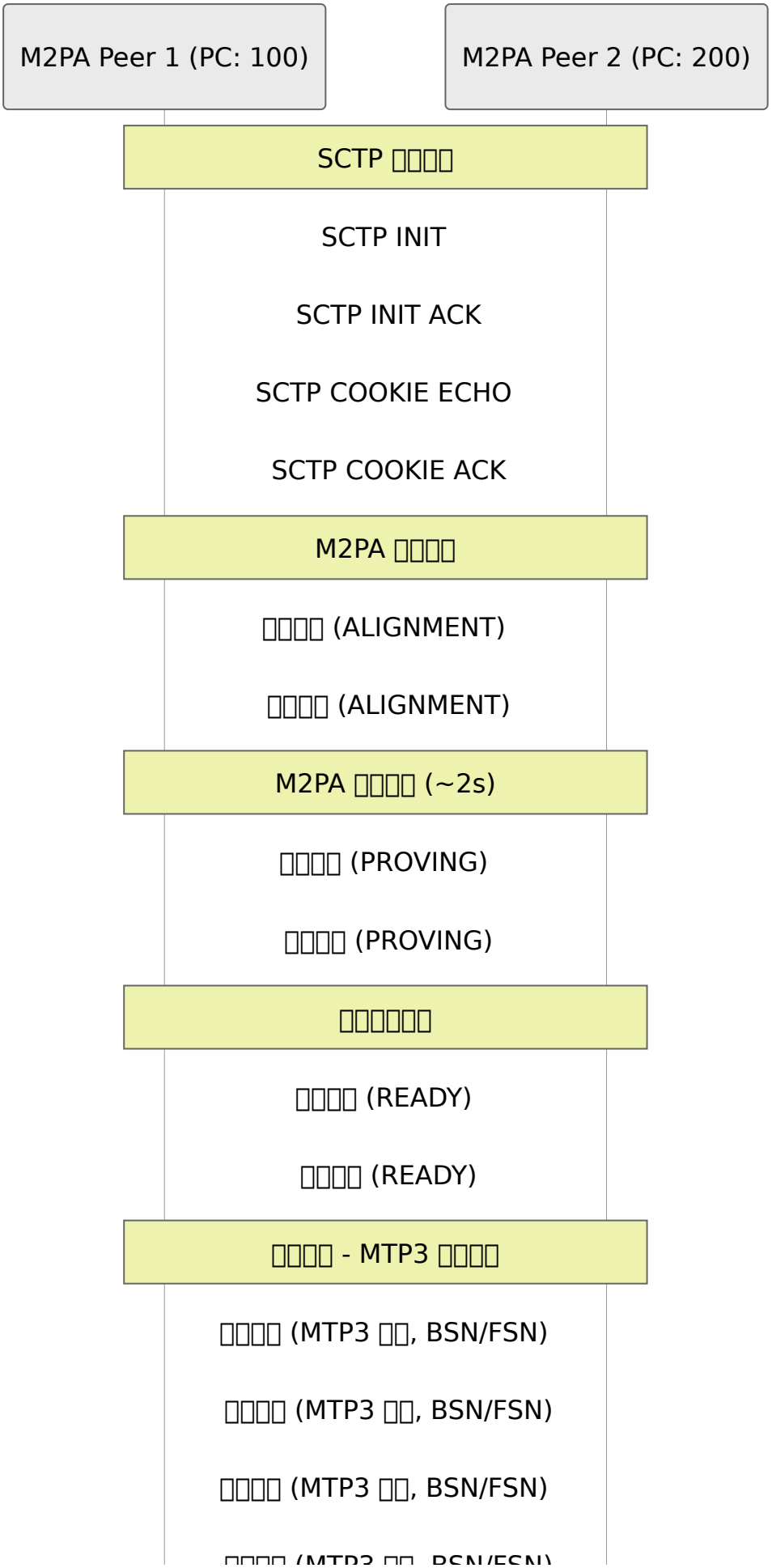
状态描述：

- **DOWN** - 没有 SCTP 连接，链路不活动
- **CONNECTING** - SCTP 关联正在进行
- **ALIGNMENT** - 链路状态消息交换（~1 秒）
- **PROVING** - 链路证明期，测试链路完整性（~2 秒）
- **READY** - 链路操作正常，准备进行 MTP3 用户数据传输
- **ALIGNMENT**（重新进入） - 链路状态变化需要重新对齐

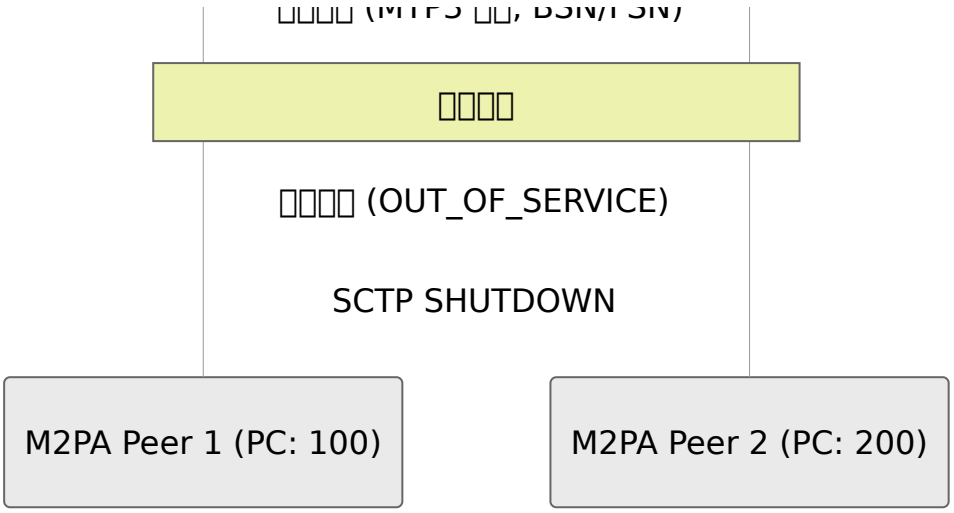
链路状态进展：

1. **SCTP** 连接：建立 SCTP 关联（DOWN → CONNECTING）
2. 对齐：交换链路状态消息以同步（CONNECTING → ALIGNMENT）
3. 证明：测试链路可靠性和序列号同步（ALIGNMENT → PROVING）
4. 准备就绪：链路变为操作正常以进行数据传输（PROVING → READY）

M2PA □□□



M2PA Peer 1 (PC: 100)
M2PA Peer 2 (PC: 200)
SCTP 关联设置
SCTP INIT
SCTP INIT ACK
SCTP COOKIE ECHO
SCTP COOKIE ACK
M2PA 链路对齐
链路状态 (ALIGNMENT)
链路状态 (ALIGNMENT)
M2PA 链路证明 (~2s)
链路状态 (PROVING)
链路状态 (PROVING)
链路准备就绪
链路状态 (READY)
链路状态 (READY)
准备就绪 - MTP3 数据传输
用户数据 (MTP3 消息, BSN/FSN)
用户数据 (MTP3 消息, BSN/FSN)
用户数据 (MTP3 消息, BSN/FSN)



通过 **Web UI** 监控 **M2PA** 对等体

Web UI 提供 M2PA 对等体连接的实时监控。

访问路由管理页面：

1. 导航到 Web UI 首页
2. 点击导航菜单中的 "Routing Management"
3. 查看 "M3UA/M2PA Peers" 表

M2PA 对等体表：

列	描述
对等体 **ID**	唯一对等体标识符
名称	对等体名称（例如，`M2PA_Link_STP_A`）
协议	显示 "M2PA"（绿色）
点代码	本地点代码
邻接 **PC**	邻接对等体点代码
本地	本地 IP:端口（通常为端口 3565）
远程	远程 IP:端口
状态	链路状态（例如，READY, ALIGNMENT, DOWN）

状态□示器：

- **READY**（绿色） - 链路正常运行并传输流量
- **ALIGNMENT**（黄色） - 链路正在对齐，尚未准备好
- **PROVING**（黄色） - 链路处于证明状态
- **DOWN**（红色） - 链路已关闭或不可用

故障排除 **M2PA** 连接

问题：链路停留在 **ALIGNMENT**

症状：

- 链路状态显示为 ALIGNMENT 的时间过���
- 没有进展到 PROVING 或 READY

检查：

1. 验证双方是否配置了正确的点代码

2. 检查 SCTP 防火墙是否允许协议 132
3. 验证 point_code 和 adjacent_point_code 是否正确设置
4. 查看应用日志以获取链路状态消息错误
5. 确保远程对等体也处于 ALIGNMENT 状态

问题：链路停留在 PROVING

症状：

链路达到 PROVING 但未过渡到 READY
证明期超过 2-3 秒

检查：

1. 验证网络稳定性（无丢包）
2. 检查 SCTP 关联错误
3. 查看日志以获取序列号不匹配的信息
4. 确保远程对等体也处于 PROVING 状态
5. 验证 SCTP 多宿主是否导致路由问题

问题：链路抖动（DOWN ↔ READY）

症状：

链路在 READY 和 DOWN 之间反复循环
频繁重新对齐

检查：

1. 检查网络连接稳定性
2. 验证 SCTP 心跳设置
3. 查看防火墙会话超时设置
4. 检查 MTU/分段问题
5. 验证没有重复的 IP 地址

问题：数据未流动

症状：

- 链路状态显示为 READY
- 没有 MTP3 消息被传输

检查:

1. 验证路由表中包含到此对等体的路由
2. 检查 MTP3 点代码路由是否配置
3. 查看消息中的 DPC 值是否匹配预期路由
4. 检查 `/events` 页面以获取路由错误
5. 验证序列号（BSN/FSN）是否在递增

相关文档

- ← 回主文档
- 常见功能指南 - Web UI, API, 监控
- MAP 客户端指南 - 发送 MAP 请求
- 短信中心指南 - 短信投递
- 技术参考 - 协议规范

OmniSS7 由 Omnitouch 网络服务提供

USSD 网关□南

← □回主文档

本指南涵盖 OmniSS7 **USSD** 网关，该网关将 SS7/MAP USSD 对话桥接到 HTTP/JSON 回调，使第三方开发者能够通过简单的 HTTP 端点构建 USSD 应用程序。

目录

1. 概述
2. 架构
3. 启用 USSD 网关
4. 配置
5. HTTP 回调协议
6. 网络发起的 USSD（推送 API）
7. 会话生命周期
8. 错误处理
9. 指标和监控
10. 示例回调服务器
11. 故障排除

概述

USSD 网关处理两个方向的 USSD 流量：

- 移动发起（入站）：订阅者拨打短码（例如 `*100#`）。网关接收 MAP `processUnstructuredSS-Request`（操作码 59）。它将请求转发到您的 HTTP 回调。然后，它通过 SS7 中继您的响应。
- 网络发起（出站）：您的应用程序通过 REST API 向订阅者推送 USSD 消息。网关通过 SS7 发送 MAP `unstructuredSS-Request`（操作码 60）。然后，它将订阅者的回复路由到您的回调。

两个方向都支持 多轮对话：交互式菜单，订阅者回复并接收后续提示。

主要特征

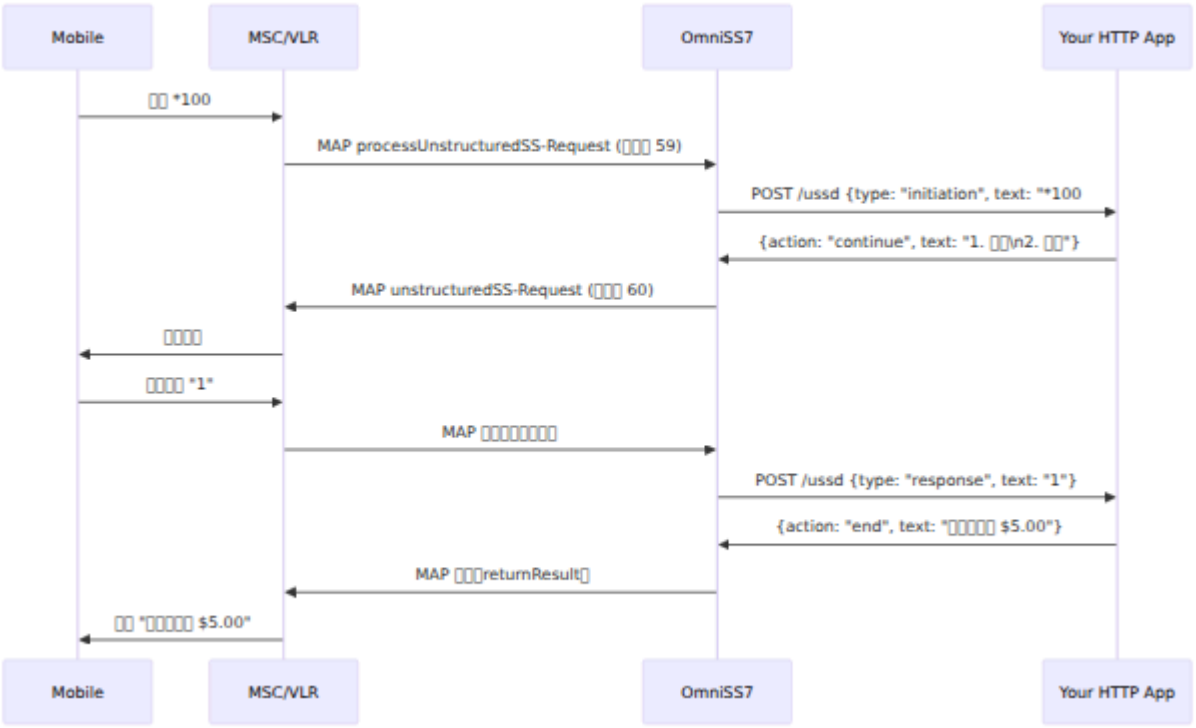
属性	值
传输	每轮同步 HTTP POST
编码	GSM 7 位默认字母表（DCS 0x0F），符合 3GPP TS 23.038
最大文本长度	182 个字符（可配置）
会话跟踪	每个对话由网关生成的 UUID
身份验证	无（信任 SS7 网络）
路由	短码前缀匹配回调 URL

3GPP 参考

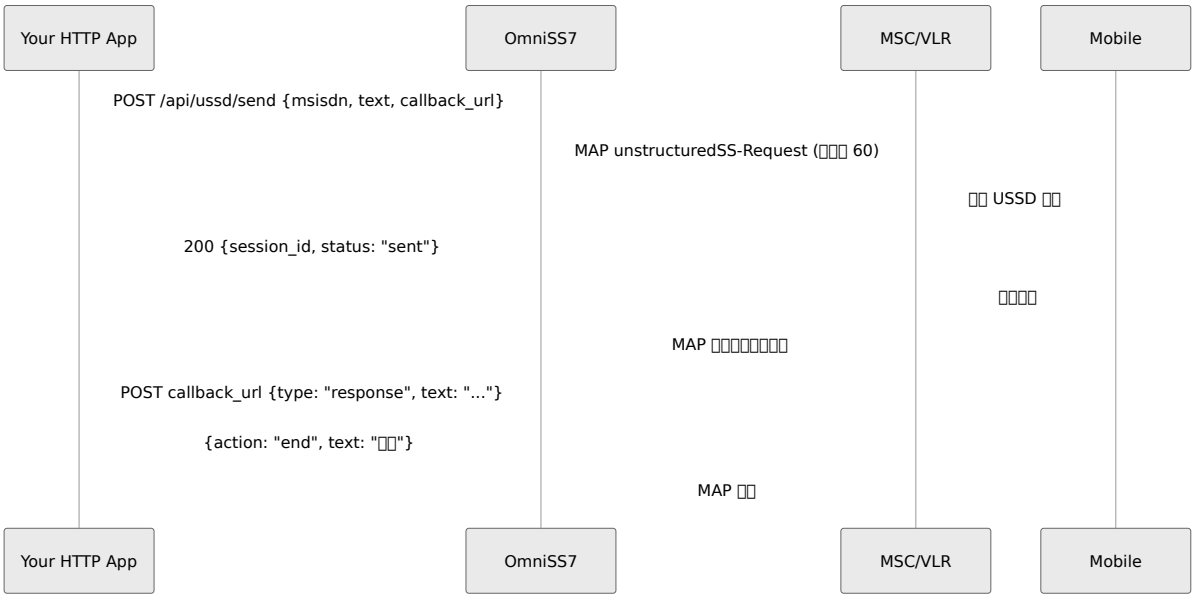
规范	相关性
3GPP TS 23.090	USSD 第二阶段：架构和程序
3GPP TS 24.090	USSD 第三阶段：协议细节
3GPP TS 29.002	MAP 协议：USSD-Arg、USSD-Res、操作码 59/60/61
3GPP TS 23.038	GSM 7 位默认字母表和数据编码方案

□□

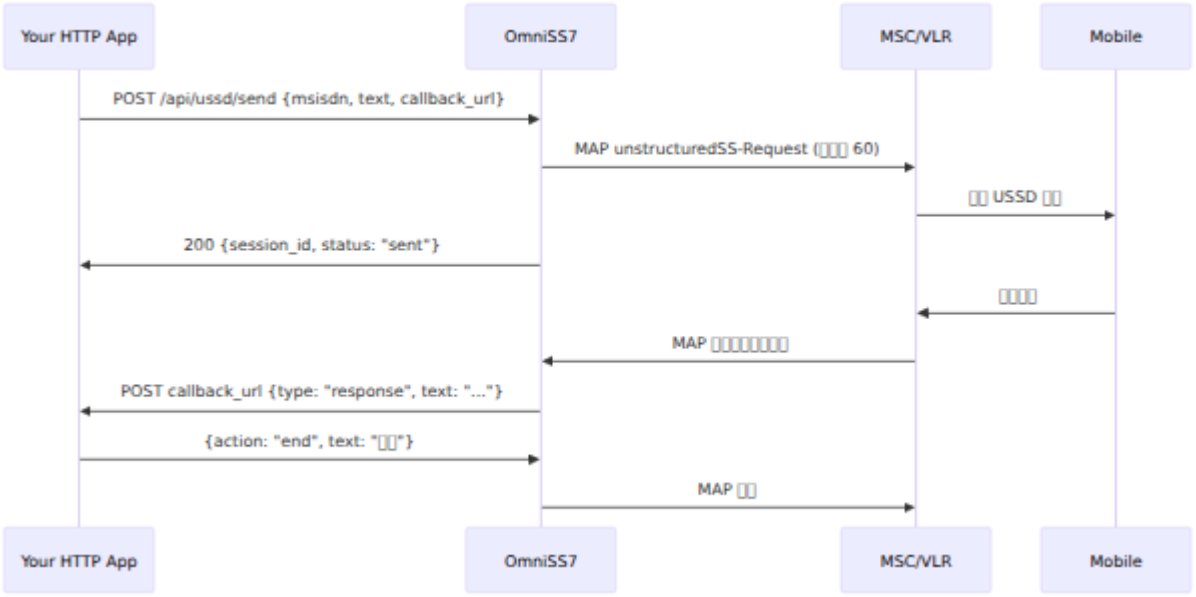
□□□□□□□□□


Mobile
MSC/VLR
OmniSS7
Your HTTP App
□□ *100
MAP processUnstructuredSS-Request (□□□ 59)
POST /ussd {type: "initiation", text: "*100
{action: "continue", text: "1. □□\n2. □□"}
MAP unstructuredSS-Request (□□□ 60)
□□□□
□□□□ "1"
MAP □□□□□□□□□
POST /ussd {type: "response", text: "1"}
{action: "end", text: "□□□□□ $5.00"}
MAP □□□returnResult□
□□ "□□□□□ $5.00"
Mobile
MSC/VLR
OmniSS7
Your HTTP App

□□□□□□□□□□□


Your HTTP App
OmniSS7
MSC/VLR
Mobile
POST /api/ussd/send {msisdn, text, callback_url}
MAP unstructuredSS-Request (□□□ 60)
□□ USSD □□
200 {session_id, status: "sent"}
□□□□
MAP □□□□□□□□□
POST callback_url {type: "response", text: "..."}
{action: "end", text: "□□"}
MAP □□
Your HTTP App
OmniSS7
MSC/VLR
Mobile

□□□□



□□ USSD □□

USSD □□□□□□□ **MAP** □□□□□□□□□□□□□□□□□□□

```
config :omniss7,
  map_client_enabled: true,
  ussd_gateway_enabled: true
```

□□□□□□□□□□□ M3UA □□□□□□ MAP □□□□□ □□□ M3UA □□□□

□□

USSD □□□□

```
config :omniss7,
  ussd_gateway_enabled: true,
  ussd_gateway: %{
    # □□□□ - □□□□□□
    routes: [
      %{pattern: "*100", url: "http://balance-app:9000/ussd"},
      %{pattern: "*200", url: "http://topup-app:9000/ussd"},
      %{pattern: "*",    url: "http://default-app:9000/ussd"}
    ],

    # □□□□
    session_timeout_ms: 180_000,    # □□□□□□□□3 □□□
    turn_timeout_ms: 30_000,        # □□□□□□□□□□□□□□□30 □□

    # HTTP □□□□
    http_timeout_ms: 5_000,         # □□□□□□□ HTTP POST □□□□5 □□

    # □□□□
    max_text_length: 182            # GSM 7 □□□□□□□□□□□□□□□□□
  }
```

参数参考

参数	类型	必需	默认	描述
`ussd_gateway_enabled`	布尔值	是	`false`	USSD 网关功能的主开关
`ussd_gateway.routes`	映射列表	是	`[]`	短码前缀路由规则。每个条目都有 `pattern`（前缀字符串）和 `url`（回调 URL）。最长前缀匹配优先。
`ussd_gateway.session_timeout_ms`	整数	否	`180_000`	最大总会话持续时间（以毫秒为单位）。如果超过，则会话将以错误结束。
`ussd_gateway.turn_timeout_ms`	整数	否	`30_000`	在多轮对话中等待订阅者回复的最大时间（以毫秒为单位）。
`ussd_gateway.http_timeout_ms`	整数	否	`5_000`	对您的应用程序进行回调的 HTTP 请求超时（以毫秒为单位）。包括连接和响应时间。
`ussd_gateway.max_text_length`	整数	否	`182`	USSD 文本字符串中的最大字符数。超过此限制的文本将被截断并记录警告。

路由参数

`routes` 列表中的每个条目都是一个映射：

参数	类型	必需	描述
`pattern`	字符串	是	要匹配的短码前缀。使用 `"*"` 作为捕获所有的后备。较长的前缀优先。
`url`	字符串	是	用于接收匹配短码的回调 POST 的 HTTP 端点 URL。

路由匹配

路由通过 最长前缀优先 进行匹配。对于拨号字符串 `*100#`：

1. `"*100"` 匹配（长度 4）：被选中
2. `"*10"` 匹配（长度 3）：被跳过，较短
3. `"*"` 匹配（长度 1）：后备

如果没有路由匹配，网关将□回 MAP 错误给移动设备并记录警告。

HTTP 回调协议

您的应用程序从网关接收 HTTP POST 请求，并以 JSON 指令做出响应。

从网关到您的应用程序的请求

Content-Type: `application/json`

第一轮（会话初始化）：

```
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "msisdn": "+254712345678",
  "type": "initiation",
  "text": "*100#",
  "turn": 1
}
```

后续轮次（订阅者回复）：

```
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "msisdn": "+254712345678",
  "type": "response",
  "text": "1",
  "turn": 2
}
```

请求字段

字段	类型	描述
`session_id`	字符串	网关生成的 UUID。每个 USSD 对话唯一。使用此字段关联轮次。
`msisdn`	字符串	订阅者的 MSISDN（如果从 MAP 消息中可用）。某些网络可能为空。
`type`	字符串	第一次轮次为 `"initiation"`，后续订阅者回复为 `"response"`。
`text`	字符串	初始化时的拨号字符串（例如 `*100#`），或响应时的订阅者输入（例如 `1`）。
`turn`	整数	从 1 开始的轮次计数器。每次订阅者交互时递增。

从您的应用程序的响应

您的应用程序必须以 JSON 响应，包含 `action` 和 `text`：

继续（显示菜单，等待订阅者输入）：

```
{
  "action": "continue",
  "text": "1. 余额\n2. 充值\n3. 转账"
}
```

结束（显示最终消息，关闭会话）：

```
{
  "action": "end",
  "text": "您的余额为 $5.00"
}
```

响应字段

字段	类型	必需	描述
`action`	字符串	是	`"continue"` 以保持会话打开并等待订阅者输入，或 `"end"` 以显示最终消息并关闭会话。
`text`	字符串	是	显示在订阅者手机上的文本。最大长度由 `max_text_length`（默认 182）控制。使用 `\n` 进行换行。

网络发起的 USSD（推送 API）

从您的应用程序向订阅者推送 USSD 消息。

端点

`POST /api/ussd/send`

请求

```
{
  "msisdn": "+254712345678",
  "text": "您有一笔待付款项。回复 1 以支付。",
  "callback_url": "http://billing-app:9000/ussd"
}
```

请求字段

字段	类型	必需	描述
`msisdn`	字符串	是	目标订阅者的 MSISDN，采用国际格式。
`text`	字符串	是	要显示的初始 USSD 文本。编码为 GSM 7 位。
`callback_url`	字符串	是	用于通过标准回调协议接收订阅者回复的 URL。

响应

成功（**200 OK**）：

```
{
  "session_id": "xyz-789-abc-123",
  "status": "sent"
}
```

错误响应：



HTTP 状态	主体	原因
400	`{"error": "invalid request", "required": ["msisdn", "text", "callback_url"]}`	缺少一个或多个必需字段，或主体不是有效的 JSON
500	`{"error": "{:gsm7_encode_failed, ...}"}`	文本包含不在 GSM 7 位字母表中的字符。编码失败通过通用错误分支显示，因此状态为 **500**，而不是 400。
500	`{"error": "..."}`	任何其他发送失败（例如 M3UA 连接）。`error` 字段包含检查的内部原因。
503	`{"error": "USSD gateway not enabled"}`	`ussd_gateway_enabled` 为 `false`

> 关于验证与发送错误的说明：字段验证问题（缺少键、格式错误的 JSON）□回 **400** `invalid request`。一旦验证通过，任何来自发送路径的失败（包括 GSM 7 位编码失败）将报告为 **500**，并在 `error` 字段中提供检查原因。

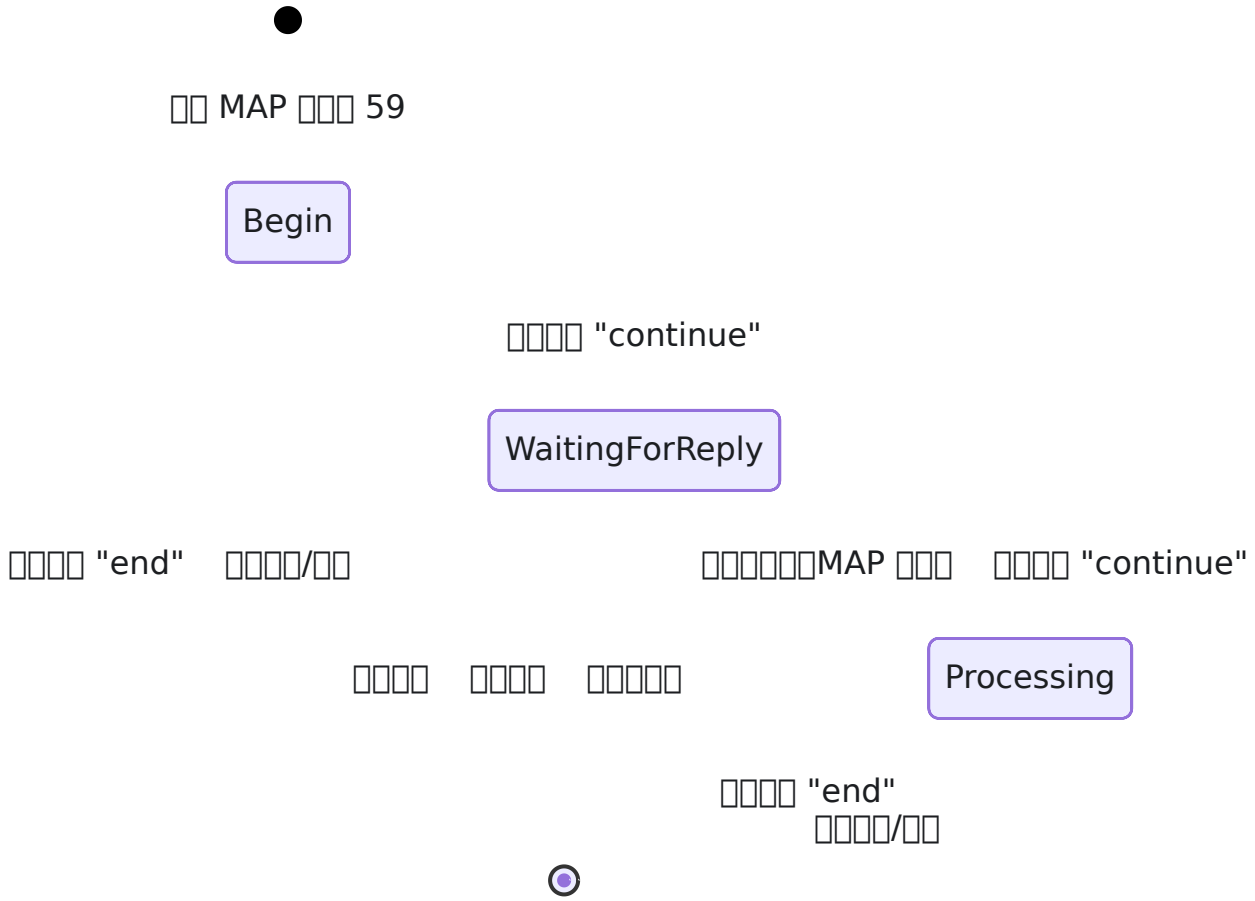
cURL 示例

```
curl -X POST http://localhost:9090/api/ussd/send \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "+254712345678",
    "text": "您有一笔待付款项。回复 1 以支付。",
    "callback_url": "http://billing-app:9000/ussd"
  }'
```

会话生命周期

每个 USSD 对���被跟踪为一个具有唯一 `session_id` 的会话。

□□□□



□□□□

□□□□□□□□□□□□□ `"end"`□□□□□□□□□□□□□□□□□□□□□□□□ MAP □□□□□□□□□

□□□□

□□□□□□□□□ `"continue"`□□□□

1. □□□□□ `UssdGateway.Registry` □□□□ □□ **GenServer**
2. □□□□□□□□ 60□unstructuredSS-Request□□ MAP □□□□□□□□
3. □□□□□□□□□□□□ `turn_timeout_ms`□
4. □□□□□□□□□□□
5. □□□□□□□□□□□ `"end"` □□□□□

超时行为

超时	默认	效果
轮次超时	30 秒	如果订阅者在此窗口内��回复，则会话将以 MAP 错误终止。
会话超时	3 分钟	总会话生命周期。无论活动如何，都会终止会话。
HTTP 回调超时	5 秒	如果您的应用程序未及时响应，网关将向移动设备发送 MAP 错误并终止会话。

错误处理

网关优雅地处理故障，并始终尝试向移动设备发送 MAP 错误响应，以便订阅者看到有意义的消息，而不是网络超时。

场景	网关操作	**MAP** 错误代码
HTTP 回调超时或 5xx	终止会话，发送带错误的 MAP 结束	34 (systemFailure)
回调中无效的 JSON	终止会话，发送带错误的 MAP 结束	34 (systemFailure)
USSD 文本超过 `max_text_length`	截断文本，记录警告，正常继续	N/A（截断，不是错误）
订阅者超时（未回复）	终止会话，发送带错误的 MAP 结束	34 (systemFailure)
没有路由匹配短码	发送带错误的 MAP 结束，记录警告	34 (systemFailure)
会话 GenServer 崩溃	会话终止，订阅者看到网络超时	N/A（进程退出）
USSD 网关未启用	□回不支持的功能	21 (facilityNotSupported)

□标和监控

USSD 网关在标准的 `/metrics` 端点（端口 9090）上公开 Prometheus 指标。

USSD □标

□标：`ussd_requests_total` 类型：计数器 描述：处理的总 USSD 请求 标签：

- `direction`：`"inbound"`（移动发起）或 `"outbound"`（网络发起推送）

□标：`ussd_active_sessions` 类型：计量器 描述：声明为活动 USSD 会话的计量器。

> 注意：此计量器在当前构建中 声明但从未更新：它始终显示为 `0`。请勿依赖它进行实时会话计数。请通过 `ussd_requests_total` 跟踪会话活动。

□标：`map_request_duration_milliseconds` 类型：直方图 描述：USSD 发送操作的持续时间（以毫秒为单位）标签：

- `operation`：`"ussd_send"` 用于出站推送请求

示例 **Prometheus** 查询

```
# 按方向（入站与出站）统计 USSD 请求速率
rate(ussd_requests_total[5m])

# 出站 USSD 延迟（p95）
histogram_quantile(0.95,
rate(map_request_duration_milliseconds_bucket{operation="ussd_send"}
[5m]))
```

□□□□□□□

Python (Flask)

```python
from flask import Flask, request, jsonify

app = Flask(__name__)
sessions = {}

@app.route('/ussd', methods=['POST'])
def ussd():
    data = request.json
    session_id = data['session_id']
    text = data['text']
    turn = data['turn']

    if data['type'] == 'initiation':
        sessions[session_id] = {'state': 'main_menu'}
        return jsonify({
            'action': 'continue',
            'text': '□□□\n1. □□□□\n2. □□□□\n3. □□'
        })

    state = sessions.get(session_id, {}).get('state')

    if state == 'main_menu':
        if text == '1':
            del sessions[session_id]
            return jsonify({
                'action': 'end',
                'text': '□□□□□ $5.00'
            })
        elif text == '2':
            sessions[session_id]['state'] = 'buy_airtime'
            return jsonify({
                'action': 'continue',
                'text': '□□□□□'
            })
        else:
            del sessions[session_id]
            return jsonify({
                'action': 'end',
```

```
                'text': '□□□□□□□□□□'
            })

    elif state == 'buy_airtime':
        del sessions[session_id]
        return jsonify({
            'action': 'end',
            'text': f'□□□□□ ${text} □□□□□□□□'
        })

    return jsonify({'action': 'end', 'text': '□□□□□□□'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9000)
```

Node.js (Express)

```
const express = require('express');
const app = express();
app.use(express.json());

const sessions = new Map();

app.post('/ussd', (req, res) => {
  const { session_id, text, type } = req.body;

  if (type === 'initiation') {
    sessions.set(session_id, { state: 'main_menu' });
    return res.json({
      action: 'continue',
      text: '□□□\n1. □□□□\n2. □□□□'
    });
  }

  const session = sessions.get(session_id);
  if (!session) {
    return res.json({ action: 'end', text: '□□□□□□□' });
  }

  if (session.state === 'main_menu' && text === '1') {
    sessions.delete(session_id);
    return res.json({ action: 'end', text: '□□□□□ $5.00' });
  }

  sessions.delete(session_id);
  return res.json({ action: 'end', text: '□□□' });
});

app.listen(9000, () => console.log('USSD □□□□□ 9000 □□□'));
```

故障排除

USSD 拨号返回 "服务不可用"

症状：订阅者拨打短码，立即收到网络错误。

可能的原因：

- `ussd_gateway_enabled` 为 `false`
- 没有路由匹配拨打的短码
- M3UA 连接中断

解决方案：

1. 验证配置中的 `ussd_gateway_enabled: true`
2. 检查路由模式是否匹配短码（记得包含 `*` 捕获所有）
3. 在 Web UI（对等方页面）中检查 M3UA 对等方状态

回调未接收请求

症状：网关日志显示 USSD 开始，但您的应用程序从未接收到 HTTP POST。

可能的原因：

- 回调 URL 从 OmniSS7 主机无法访问
- 防火墙阻止 OmniSS7 的出站 HTTP
- 回调应用程序未运行

解决方案：

1. 测试连接性：从 OmniSS7 主机执行 `curl -v http://your-app:9000/ussd`
2. 检查出站 HTTP 的防火墙规则
3. 验证您的回调应用程序是否在配置的端口上监听

会话过早超时

症状：多轮会话以 "systemFailure" 结束，订阅者无法回复。

可能的原因：

- `turn_timeout_ms` 对您的订阅者群体来说太短
- `http_timeout_ms` 对您的应用程序的处理时间来说太短
- OmniSS7 与您的回调服务器之间的网络延迟

解决方案：

1. 增加 `turn_timeout_ms`（默认 30 秒对大多数情况应该足够）
2. 如果您的应用程序需要更多处理时间，请增加 `http_timeout_ms`
3. 将回调服务器部署在靠近 OmniSS7 的位置以减少延迟

GSM 7 位编码错误

症状：日志中出现 `gsm7_encode_failed` 错误，或从 `/api/ussd/send` 回 **500** 响应，其 `error` 字段包含 `{:gsm7_encode_failed, ...}`。

可能的原因：

- 文本包含不在 GSM 7 位默认字母表中的字符（例如表情符号、CJK 字符）

解决方案：

- 将 USSD 文本限制为 GSM 基本字符集：ASCII 字母、数字、常见标点符号以及一些希腊/北欧字符
- 请参见 3GPP TS 23.038 第 6.2.1 节 获取完整字符表

相关文档

- **API** 南：完整的 REST API 参考（所有端点，包括 `/api/ussd/send`）
- **MAP** 客户端南：USSD 所需的 M3UA 连接设置
- 配置参考：所有配置参数
- 常见功能南：Web UI、监控和 Prometheus 设置

Web UI 指南

← 返回主文档

本指南提供了使用 OmniSS7 **Web UI**（Phoenix LiveView 界面）的全面文档。

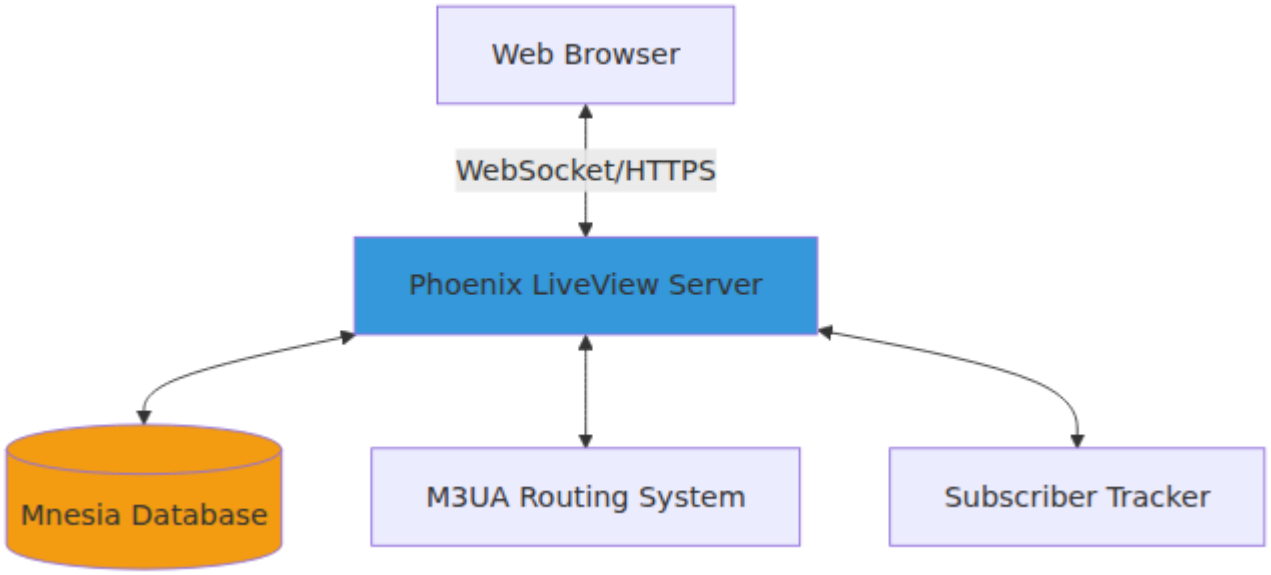
目录

1. 概述
2. 访问 Web UI
3. 路由管理页面
4. 活跃订阅者页面
5. 常见操作
6. 自动刷新行为

概述

OmniSS7 Web UI 是一个 **Phoenix LiveView** 应用程序，提供实时监控和管理功能。可用页面取决于哪个操作模式处于活动状态（STP、HLR 或 SMSc）。

Web UI □□



□□□□□

□□	□
□□	HTTPS (`force_ssl` □□)
□□	`8087`□□ `config :control_panel, ControlPanelWeb.Endpoint` □□□ → `https: [port: 8087]`□
□□ **IP**	`0.0.0.0`□□□□□□□□
□□	`priv/cert/omnitouch.crt`
□□	`priv/cert/omnitouch.pem`
□□□	Bandit (Phoenix)

□□ **URL**: `https://[server-ip]:8087`

访问 Web UI

先决条件
1. SSL 证书: 确保 priv/cert/ 中存在有效的 SSL 证书：
omnitouch.crt - 证书文件
omnitouch.pem - 私钥文件
2. 应用程序运行: 使用 iex -S mix 启动应用程序
3. 防火墙: 确保端口 8087 对 HTTPS 流量开放

按模式可用的页面
此处显示的页面标签与 config/config.exs 中注册的标签完全相同（use_builtin_pages / use_additional_pages）。模式列指示页面通常相关的操作模式（STP/HLR/SMSc）；运行时配置根据部署过滤页面列表。

页面（路由）	标签	**STP** 模式	**HLR** 模式	**SMSc** 模式	描述
`/application`	资源	✅	✅	✅	系统资源和监控
`/configuration`	配置	✅	✅	✅	配置查看器
`/license`	许可证	✅	✅	✅	许可证状态
`/events`	**SS7** 事件	✅	✅	✅	事件日志和 SCCP 消息捕获
`/logs`	系统日志	✅	✅	✅	实时应用程序日志查看器
`/client`	**SS7** 客户端	✅	✅	✅	手动 MAP 操作测试
`/m3ua`	对等体	✅	✅	✅	M3UA 对等体 / ASP 连接状态
`/sctp`	**SCTP** 连接	✅	✅	✅	实时 SCTP 关联状态
`/routing`	路由	✅	✅	✅	M3UA 路由表管理
`/routing_test`	路由测试	✅	✅	✅	路由测试和验证
`/hlr_links`	**HLR** 链接	✅	✅	✅	HLR API 状态和订阅者管理
`/subscribers`	活跃订阅者	✅	✅	✅	实时订阅者位置跟踪 (HLR)
`/smsc_links`	**SMSc** 链接	✅	✅	✅	SMSc API 状态和队列管理

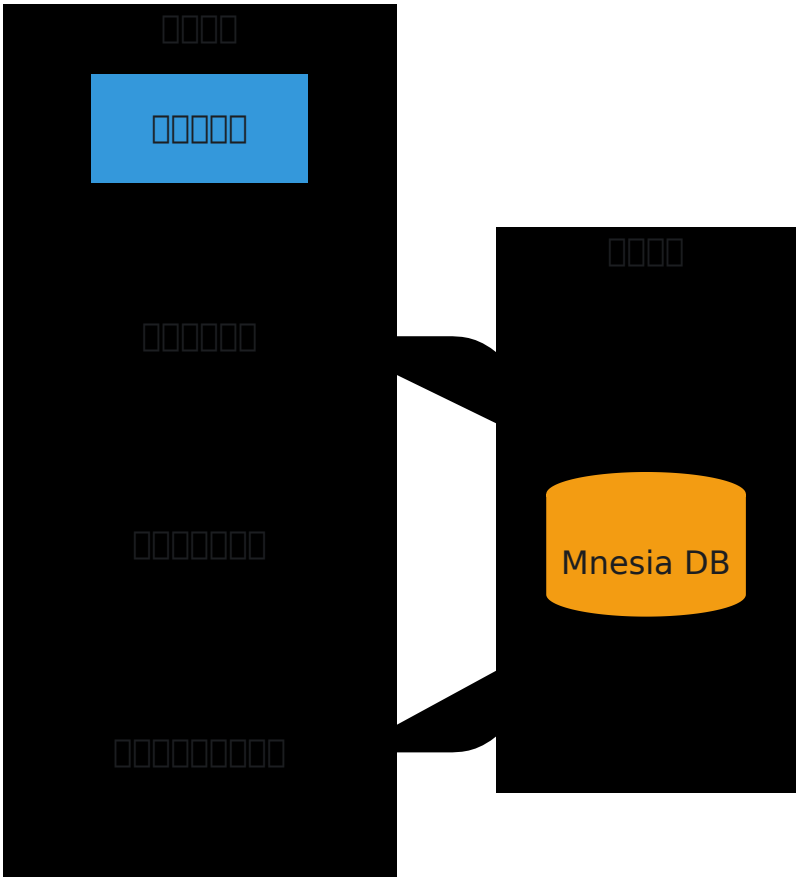
页面（路由）	标签	**STP** 模式	**HLR** 模式	**SMSc** 模式	描述
`/smsc_subscribers`	**SMSc** 订阅者	☐	☐	☐	实时订阅者跟踪 (SMSc)
`/camel_sessions`	**CAMEL** 会话	☐	☐	☐	实时 CAMEL/CAP 对话监控
`/camel_request`	**CAMEL** 请求构建器	☐	☐	☐	交互式 CAP 请求构建器
`/msisdn_imsi_test`	**MSISDN/IMSI** 测试	☐	☐	☐	MSISDN ↔ IMSI 查找 / 转换测试

> M3UA 页面在导航中标记为 "对等体"（路由 `/m3ua`）。

路由管理

页面: `/routing`
模式: STP, SMSc
自动刷新: 每 5 秒

路由管理页面提供了一个选项卡界面，用于管理 M3UA 路由表。



页面布局

对等体选项卡

管理 M3UA 对等体连接（其他 STP、HLR、MSC、SMSC）。

对等体表列

列	描述	示例
ID	唯一对等体标识符	`1`
名称	可读的对等体名称	`"STP_West"`
角色	连接角色	`client`, `server`, `stp`
点码	对等体的 SS7 点码	`100`
远程	远程 IP:端口	`10.0.0.10:2905`
状态	连接状态	`active`, `aspup`, `down`
操作	编辑/删除按钮	-

添加对等体

1. 点击 对等体选项卡
2. 填写 表单字段:
   - 对等体 **ID**: 如果留空则自动生成
   - 对等体名称: 描述性名称（必填）
   - 角色: 选择 `client`、`server` 或 `stp`
   - 点码: SS7 点码（必填）
   - 本地 **IP**: 您系统的 IP 地址
   - 本地端口: 0 表示动态端口分配
   - 远程 **IP**: 对等体的 IP 地址
   - 远程端口: 对等体的端口（通常为 2905）
   - 路由上下文: M3UA 路由上下文 ID
   - 网络□示符: `international` 或 `national`
3. 点击 "添加对等体"

持久性: 对等体立即保存到 Mnesia，并在重启后保留。

编辑对等体

1. 点击 对等体行上的 "编辑" 按钮
2. 根据需要修改 表单字段
3. 点击 "更新对等体"

注意：如果您更改对等体 ID，旧的对等体将被删除并创建一个新的。

删除对等体

1. 点击 对等体行上的 "删除" 按钮
2. 确认 删除（所有使用此对等体的路由也将被删除）

对等体状态□示器

状态	颜色	描述
`active`	□ 绿色	对等体已连接并正在路由消息
`aspup`	□ 黄色	ASP 已启动但尚未激活
`down`	□ 红色	对等体已断开连接

点码路由选项卡

根据目标点码配置路由规则。

路由表列

列	描述	示例
目标 **PC**	目标点码（zone.area.id 格式���	`1.2.3` (100)
掩码	用于 PC 匹配的子网掩码	`/14` (精确)，`/8` (范围)
对等体 **ID**	此路由的目标对等体	`1`
对等体名称	目标对等体的名称	`"STP_West"`
优先级	路由优先级（1 = 最高）	`1`
网络	网络指示符	`international`
操作	编辑/删除按钮	-

添加点码路由

1. 点击 "点码路由" 选项卡
2. 填写 表单字段：
   - 目标点码：输入为 `zone.area.id`（例如，`1.2.3`）或整数（0-16383）
   - 掩码：选择掩码 `/14` 进行精确匹配，较小值用于范围
   - 对等体 **ID**：从下拉列表中选择目标对等体
   - 优先级：输入优先级（1 = 最高，默认）
   - 网络□示符：选择 `international` 或 `national`
3. 点击 "添加路由"

点码格式：您可以以两种格式输入点码：

- **3-8-3** 格式：`zone.area.id`（例如，`1.2.3`）
- 整数格式：`0-16383`（例如，`1100`）

系统会自动在格式之间转换。

理解掩码

点码是 14 位值（0-16383）。掩码指定必须匹配的最高有效位数：

掩码	匹配的 **PC**	用例
`/14`	1 (精确匹配)	���由到特定目标
`/13`	2 PCs	小范围
`/8`	64 PCs	中等范围
`/0`	所有 16,384 PCs	默认/回退路由

示例：

- `PC 1000 /14` → 仅匹配 PC 1000
- `PC 1000 /8` → 匹配 PC 1000-1063（64 个连续 PCs）
- `PC 0 /0` → 匹配所有点码（默认路由）

点码掩码参考卡

网页包括一个交互式参考，显示所有掩码值及其范围。

全局标题路由选项卡

根据 SCCP 全局标题地址配置路由规则。

要求：必须在配置中启用全局标题路由：

```
config :omniss7,
  enable_gt_routing: true
```

路由表列

列	描述	示例
GT 前缀	被叫方 GT 前缀（空 = 回退）	`"1234"`, `""`
源 **SSN**	匹配被叫方 SSN（可选）	`6` (HLR), `any`
对等体 **ID**	目标对等体	`1`
对等体	对等体名称	`"HLR_West (1)"`
目标 **SSN**	转发时重写 SSN（可选）	`6`, `preserve`
优先级	路由优先级	`1`
描述	路由描述	`"US numbers"`
操作	编辑/删除按钮	-

添加全局标题路由

1. 点击 "全局标题路由" 选项卡
2. 填写 表单字段：
   - **GT** 前缀：留空以用于回退路由，或输入数字（例如，`"1234"`）
   - 源 **SSN**：可选 - 按被叫方 SSN 过滤
   - 对等体 **ID**：选择目标对等体
   - 目标 **SSN**：可选 - 转发时重写 SSN
   - 优先级：路由优先级（1 = 最高）
   - 描述：可读描述
3. 点击 "添加路由"

回退路由：如果 GT 前缀为空，则该路由充当不匹配任何其他路由的 GT 的捕获。

常见 **SSN** 值

页面包括一个参考卡，显示常见的 SSN 值：

SSN	网络元素
6	HLR (家庭位置寄存器)
7	VLR (访客位置寄存器)
8	MSC (移动交换中心)
9	EIR (设备身份寄存器)
10	AUC (认证中心)
142	RANAP
145	gsmSCF (服务控制功能)
146	SGSN

SSN 重写

- 源 **SSN**: 匹配传入消息中的被叫方 SSN
- 目标 **SSN**: 如果设置，在转发时重写被叫方 SSN
  - 空 = 保留原始 SSN
  - 值 = 替换为此 SSN

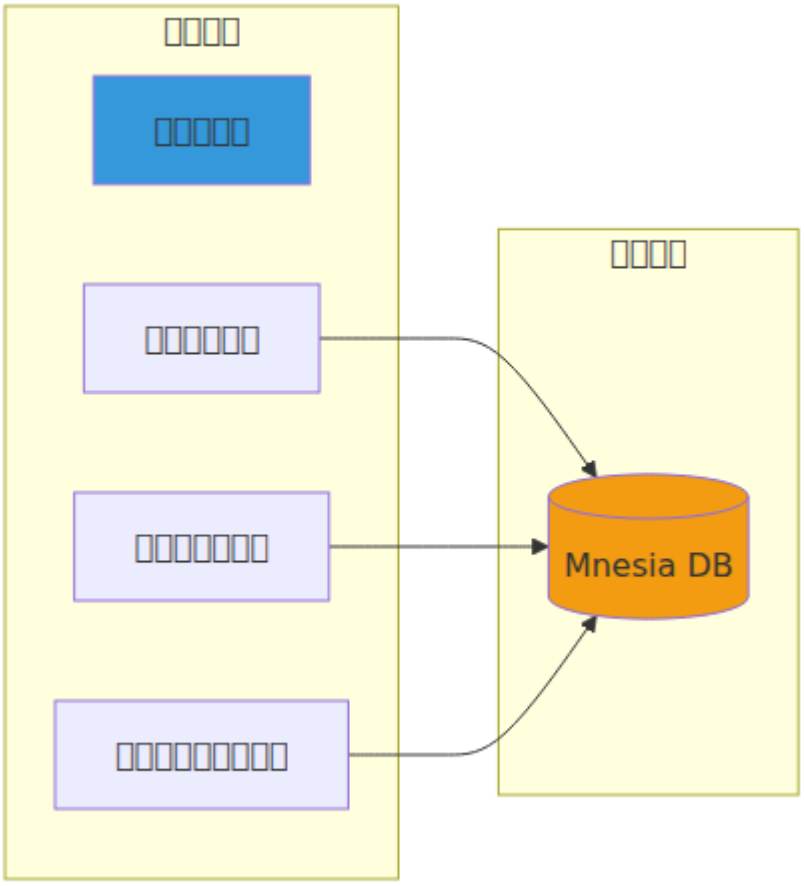
用例: 将 SSN=6 (HLR) 的消息路由到对等体，并在出站时重写为 SSN=7 (VLR)。

路由表持久性

所有路由都存储在 **Mnesia** 中，并在应用程序重启后保留。

路由如何持久化

1. **Web UI** 更改: 所有添加/编辑/删除操作立即保存到 Mnesia
2. 应用程序重启: 路由在启动时从 Mnesia 加载
3. **Runtime.exs** 合并: 来自 `config/runtime.exs` 的静态路由与 Mnesia 路由合并（无重复）



路由优先级

当多个路由匹配一个目标时：

1. 更具体的优先：更高的掩码值（更具体）优先
2. 优先级字段：较低的优先级数字先路由（1 = 最高优先级）
3. 对等体状态：仅使用连接状态为 `active` 的对等体的路由

活跃订阅者

页面：`/subscribers`
模式：仅 HLR
自动刷新：每 2 秒

显示发送 UpdateLocation 请求的订阅者的实时跟踪。

页面功能

订阅者表列

列	描述	示例
IMSI	订阅者 IMSI	`"50557123456789"`
VLR 编号	当前 VLR GT 地址	`"555123155"`
MSC 编号	当前 MSC GT 地址	`"555123155"`
更新时间	最后 UpdateLocation 时间戳	`"2025-10-25 14:23:45 UTC"`
持续时间	自注册以来的时间	`"2h 15m 34s"`

统计摘要

当存在订阅者时，摘要卡显示：

- 总活跃：注册的订阅者总数
- 独特 **VLRs**：不同 VLR 地址的数量
- 独特 **MSCs**：不同 MSC 地址的数量

清除订阅者

清除所有按钮：从跟踪器中移除所有活跃订阅者。

确认：清除之前需要确认（无法撤销）。

用例：在网络维护或测试后清除过时的订阅者记录。

自动刷新

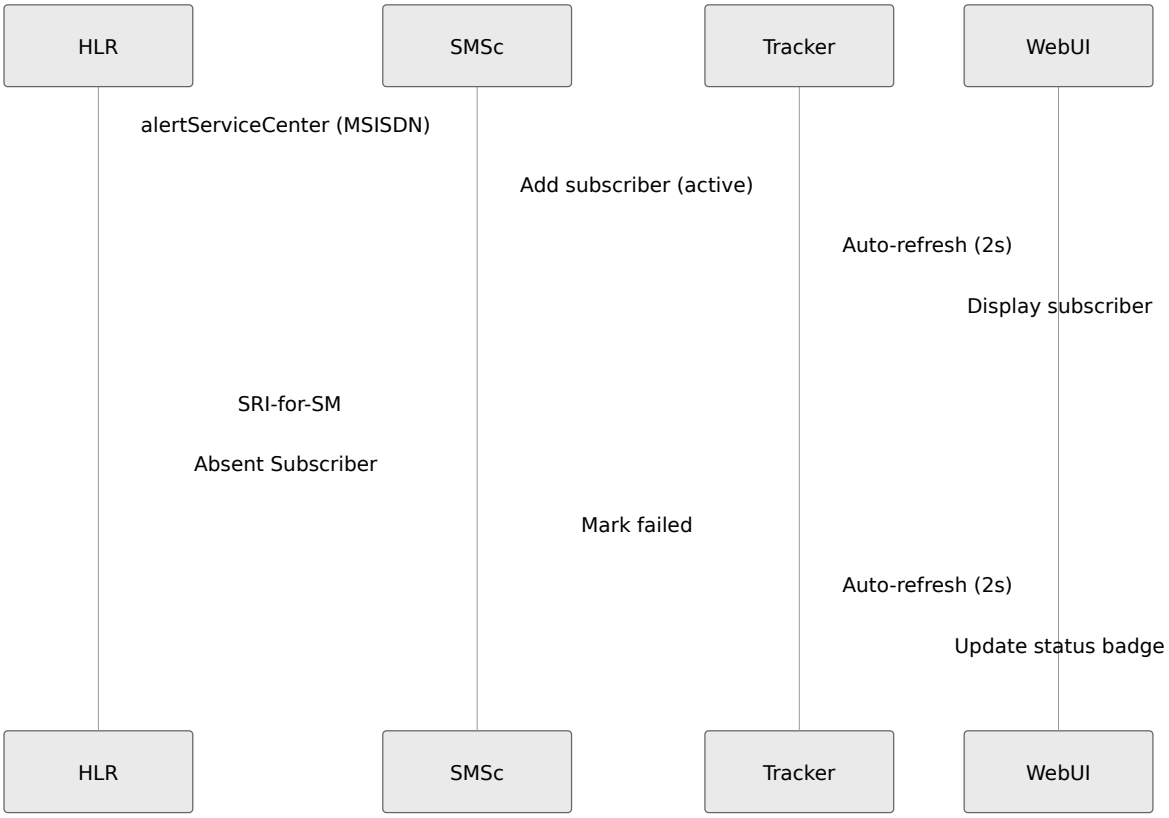
页面每 **2** 秒自动刷新，以显示实时的订阅者更新。

SMSc □□□

□□: `/smsc_subscribers`
□□: □ SMSc
□□□□: □ 2 □

□□□□□ HLR □□□□ alertServiceCenter □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□



订阅者表列

列	描述	示例
MSISDN	订阅者的电话号码	`"15551234567"`
IMSI	订阅者 IMSI	`"001010123456789"`
HLR GT	发送 alertServiceCenter 的 HLR GT	`"15551111111"`
发送的消息	发送的 MT-FSM 消息计数	`5`
接收的消息	从订阅者接收的 MO-FSM 消息计数	`2`
状态	`活跃` 或 `失败`（颜色编码）	`● 活跃`
最后更新	最后更新时间戳	`"2025-10-30 14:23:45 UTC"`
持续时间	自最后更新以来的时间	`"15m 34s"`

状态□示器

- ● 活跃（绿色）：订阅者可达，最后一次成功接收 alertServiceCenter
- ○ 失败（红色）：最后一次交付尝试失败（SRI-for-SM 或缺失订阅者错误）

统计摘要

当存在订阅者时，摘要卡显示：

- 总跟踪：跟踪的订阅者总数
- 活跃：处于活跃状态的订阅者数量
- 失败：处于失败状态的订阅者数量
- 独特 **HLRs**：发送警报的不同 HLR 数量

管理订阅者

移除按钮：从跟踪中移除单个订阅者。

清除所有按钮：移除所有跟踪的订阅者。

确认：清除所有需要确认才能清除（无法撤销）。

用例：

- 在网络问题后移除过时的条���
- 在开发后清除测试数据
- 监控哪些 HLR 正在发送警报

消息计数器

跟踪器会自动增加计数器：

- 发送的消息：当 SRI-for-SM 成功并发送 MT-FSM 时增加
- 接收的消息：当从订阅者接收到 MO-FSM 时增加

自动刷新

页面每 **2** 秒自动刷新，以显示实时的订阅者和状态更新。

常见操作

搜索和过滤

目前，Web UI 不包括内置的搜索/过滤功能。要查找特定路由：

1. 使用浏览器的查找功能 (Ctrl+F / Cmd+F)
2. 搜索对等体名称、点码或 GT 前缀

批量操作

要执行批量路由更改：

1. 选项 **1**：使用 REST API 进行编程访问
2. 选项 **2**：编辑 `config/runtime.exs` 并重启应用程序
3. 选项 **3**：使用 Web UI 进行单个路由更改

导出/导入

注意：Web UI 当前不支持导出或导入路由表。路由是：

- 存储在 Mnesia 数据库文件中
- 在 `config/runtime.exs` 中配置

要备份路由：

1. **Mnesia**：备份 `Mnesia.{node_name}/` 目录
2. 配置：版本控制 `config/runtime.exs`

自动刷新行为

不同页面具有不同的刷新间隔：

页面	刷新间隔	原因
路由管理	5 秒	路由更改不频繁
活跃订阅者	2 秒	订阅者状态频繁变化
M3UA 状态	按页面变化	连接状态监控

WebSocket 连接：所有页面使用 Phoenix LiveView WebSocket 连接进行实时更新。

网络中断：如果 WebSocket 连接丢失，页面将尝试自动重新连接。

故障排除

页面未加载

1. 检查 **HTTPS** 证书：确保 `priv/cert/omnitouch.crt` 和 `priv/cert/omnitouch.pem` 存在
2. 验证端口 **8087**：检查防火墙规则允许端口 `8087` 的 HTTPS 流量
3. 应用程序运行：确认应用程序正在运行，使用 `iex -S mix`
4. 浏览器控制台：检查 SSL 证书错误（自签名证书警告）

路由未持久化

1. 检查 **Mnesia** 存储：验证配置中的 `mnesia_storage_type: :disc_copies`
2. **Mnesia** 目录：确保 Mnesia 目录可写
3. 检查日志：查看应用程序日志中的 Mnesia 错误

自动刷新不起作用

1. **WebSocket** 连接：检查浏览器控制台中的 WebSocket 错误
2. 网络：验证网络连接稳定
3. 页面重新加载：尝试刷新页面 (F5)

相关文档

- **STP □南** - 详细的路由配置
- **HLR □南** - 订阅者管理
- **API □南** - 用于编程访问的 REST API
- 配置参考 - 所有配置参数

总结

OmniSS7 Web UI 提供直观的实时路由表管理和订阅者跟踪：

✅实时更新 - 自动刷新保持数据最新
✅持久存储 - Mnesia 确保路由在重启后保留
✅基于角色的 UI - 页面根据操作模式 (STP/HLR/SMSc) 进行调整 ✅交互式管理 - 添加、编辑、删除路由无需重启
✅状态监控 - 实时连接和对等体状态

有关高级操作或自动化，请参见API 指南。

高级全球标题路由示例

← 回主文档

本文档提供了使用翻译类型（TT）、编号计划指示符（NPI）和地址性质指示符（NAI）进行高级GT路由场景的实用示例。

目录

1. 基本概念
2. 翻译类型（TT）示例
3. 编号计划指示符（NPI）示例
4. 地址性质（NAI）示例
5. 组合示例
6. 真实世界用例

基本概念

示符值参考

翻译类型（**TT**）：

- `0` = 未知
- `1` = 国际
- `2` = 国家
- `3` = 网络特定

编号计划示符（**NPI**）：

- `0` = 未知
- `1` = ISDN/电话（E.164）
- `3` = 数据（X.121）
- `4` = 电传（F.69）

- `6` = 移动通信（E.212）

地址性质□示符（**NAI**）：

- `0` = 未知
- `1` = 用户号码
- `2` = 保留用于国家使用
- `3` = 国家显著号码
- `4` = 国际号码

匹配与转换

- 源□示符（source_tt, source_npi, source_nai）：用于匹配传入消息
- 目标□示符（dest_tt, dest_npi, dest_nai）：用于转换转发时的值
- 留空字段（`nil`）用于通配符匹配或保留原始值

翻译类型（TT）示例

示例 1：未知到国际TT转换

场景：合作伙伴网络发送所有GT，TT=0（未知），但下游HLR期望TT=1（国际）。

```
%{
  gt_prefix: "44",
  peer_id: 1,
  priority: 1,
  description: "英国：将未知TT转换为国际",
  source_tt: 0,       # 匹配传入TT=0
  dest_tt: 1          # 转换为TT=1
}
```

流量流动：

- 传入：GT=441234567890，TT=0
- 路由：匹配gt_prefix="44"，source_tt=0

- 传出: GT=441234567890, TT=1 (已转换)

示例 2: 按翻译类型分开路由

场景: 将不同TT值路由到不同网络。

```
# 将TT=0路由到网络A
%{
  gt_prefix: "33",
  peer_id: 1,
  priority: 1,
  description: "法国号码TT=0路由到网络A",
  source_tt: 0
},

# 将TT=2路由到网络B
%{
  gt_prefix: "33",
  peer_id: 2,
  priority: 1,
  description: "法国号码TT=2路由到网络B",
  source_tt: 2
}
```

流量流动:

- GT="33123456789", TT=0 → 路由到对等体1
- GT="33123456789", TT=2 → 路由到对等体2

编号计划□示符（NPI）示例

示例 1: 移动到ISDN NPI转换

场景: 移动网络使用NPI=6 (E.212), PSTN期望NPI=1 (E.164)。

```
%{
  gt_prefix: "49",
  peer_id: 1,
  priority: 1,
  description: "德国移动→ISDN转换",
  source_npi: 6,      # 匹配NPI=6（移动/E.212）
  dest_npi: 1         # 转换为NPI=1（ISDN/E.164）
}
```

用例：连接具有不同编号计划的移动和固定网络。

示例 2：基于编号计划的路由

场景：将数据网络流量与语音流量分开。

```
# 路由数据网络流量（NPI=3）
%{
  gt_prefix: "1",
  peer_id: 1,
  priority: 1,
  description: "数据网络流量",
  source_npi: 3       # 匹配NPI=3（数据/X.121）
},

# 路由语音网络流量（NPI=1）
%{
  gt_prefix: "1",
  peer_id: 2,
  priority: 1,
  description: "语音网络流量",
  source_npi: 1       # 匹配NPI=1（ISDN）
}
```

地址性质（NAI）示例

示例 1：国际到国家NAI转换

场景：传入国际格式（NAI=4），转换为国家格式（NAI=3）以进行国内路由。

```
%{
  gt_prefix: "1",
  peer_id: 1,
  priority: 1,
  description: "美国: 国际→国家格式",
  source_nai: 4,      # 匹配NAI=4（国���）
  dest_nai: 3         # 转换为NAI=3（国家）
}
```

流量流动：

- 传入：GT=12125551234，NAI=4（国际格式）
- 传出：GT=12125551234，NAI=3（国家格式）

示例 2：用户与网络NAI路由

场景：根据是用户号码还是网络号码进行路由。

```
# 将用户号码路由到HLR
%{
  gt_prefix: "44",
  peer_id: 1,
  priority: 1,
  description: "英国用户号码路由到HLR",
  source_nai: 1      # 匹配NAI=1（用户号码）
},

# 将网络号码路由到MSC
%{
  gt_prefix: "44",
  peer_id: 2,
  priority: 1,
  description: "英国网络地址路由到MSC",
  source_nai: 3      # 匹配NAI=3（国家显著号码）
}
```

组合示例

示例 1：SMS流量标准化

场景：在路由到HLR之前，标准化来自各种来源的所有SMS流量。

```
%{
  gt_prefix: "86",
  source_ssn: 8,            # 匹配SSN=8（SMSC）
  peer_id: 1,
  dest_ssn: 6,              # 重写为SSN=6（HLR）
  priority: 1,
  description: "中国SMS标准化",
  source_tt: 0,             # 匹配TT=0（未知）
  dest_tt: 2,               # 转换为TT=2（国家）
  source_npi: 6,            # 匹配NPI=6（移动）
  dest_npi: 1,              # 转换为NPI=1（ISDN）
  source_nai: 4,            # 匹配NAI=4（国际）
  dest_nai: 3               # 转换为NAI=3（国家）
}
```

此操作的作用：

- 匹配SMS流量（SSN=8）用于中国号码（GT前缀"86"）
- 将所有指示符标准化为标准值
- 将目标SSN从SMSC（8）重写为HLR（6）
- 非常适合指示符使用不一致的互联场景

示例 2：多运营商路由与格式转换

场景：根据TT将流量路由到不同的运营商，并根据需要转换格式。

```
# 运营商A：期望国家TT和ISDN NPI
%{
  gt_prefix: "44",
  peer_id: 1,
  priority: 1,
  description: "英国运营商A：标准化为国家/ISDN",
  source_tt: 0,              # 匹配任何TT=0
  dest_tt: 2,                # 转换为TT=2（国家）
  source_npi: 6,             # 匹配移动NPI
  dest_npi: 1                # 转换为ISDN NPI
},

# 运营商B：期望国际TT，保留NPI
%{
  gt_prefix: "44",
  peer_id: 2,
  priority: 2,               # 较低优先级（数字较高）
  description: "英国运营商B：转换为国际TT",
  source_tt: 2,              # 匹配国家TT
  dest_tt: 1                 # 转换为国际TT
  # NPI留空 = 保留原始值
}
```

真实世界用例

用例 1：遗留系统集成

问题：将现代网络与使用不同指示符约定的遗留系统集成。

解决方案：将所有指示符转换为��留格式。

```
%{
  gt_prefix: "555",
  peer_id: 1,
  priority: 1,
  description: "遗留HLR标准化",
  # 匹配任何现代格式
  source_tt: 1,              # 现代：国际
  dest_tt: 0,                # 遗留：未知
  source_npi: 1,             # 现代：ISDN
  dest_npi: 0,               # 遗留：未知
  source_nai: 4,             # 现代：国际
  dest_nai: 0                # 遗留：未知
}
```

用例 2：国际互联标准化

问题：多个国际合作伙伴发送流量，指示符不一致。

解决方案：将所有传入流量标准化为一致格式。

```
# 合作伙伴A发送TT=0，转换为标准TT=1
%{
  gt_prefix: "1",
  peer_id: 1,
  priority: 1,
  description: "美国合作伙伴A标准化",
  source_tt: 0,
  dest_tt: 1,
  dest_npi: 1,              # 确保ISDN NPI
  dest_nai: 4               # 确保国际NAI
},

# 合作伙伴B发送TT=3（网络特定），转换为TT=1
%{
  gt_prefix: "44",
  peer_id: 2,
  priority: 1,
  description: "英国合作伙伴B标准化",
  source_tt: 3,
  dest_tt: 1,
  dest_npi: 1,
  dest_nai: 4
}
```

用例 3：漫游流量管理

问题：本网和漫游合作伙伴使用不同的地址格式。

解决方案：将漫游流量转换为本网格式。

```
# ###############
%{
  gt_prefix: "12025",      # ########
  peer_id: 1,
  priority: 1,
  description: "########",
  source_nai: 4,           # ##########
  dest_nai: 3,             # #####
  source_npi: 6,           # ####
  dest_npi: 1              # #HLR#ISDN
}
```

4######

##################

####################

```
# 旧格式流量：转换为新格式
%{
  gt_prefix: "999",
  peer_id: 1,
  priority: 1,
  description: "遗留到新格式迁移",
  source_tt: 0,              # 旧：未知
  dest_tt: 2,                # 新：国家
  source_npi: 0,             # 旧：未知
  dest_npi: 1,               # 新：ISDN
  source_nai: 0,             # 旧：未知
  dest_nai: 3                # 新：国家
},

# 新格式流量：保持不变
%{
  gt_prefix: "999",
  peer_id: 1,
  priority: 2,
  description: "新格式直通",
  source_tt: 2,              # 已经是新格式
  # 无dest_*字段 = 保留所有值
}
```

测试您的路由

通过**Web UI**

1. 导航到 路由 → 全球标题路由
2. 点击 **“**显示高级路由（**TT/NPI/NAI**）**”**
3. 填写您的源和目标指示符
4. 点击 添加路由
5. 转到 路由测试 页面测试路由

通过API

GT路由端点挂载在API监听器的`/routing/gt-routes`上（端口 **8445**，启用TLS）。JSON主体使用字段`prefix`（而不是`gt_prefix`），`prefix`和`peer_id`是必需的。TT/NPI/NAI匹配器和重写是可选的。

```
curl -k -X POST https://localhost:8445/routing/gt-routes \
  -H "Content-Type: application/json" \
  -d '{
    "prefix": "44",
    "peer_id": 1,
    "priority": 1,
    "description": "英国TT转换",
    "source_tt": 0,
    "dest_tt": 1
  }'
```

字段	类型	必需	描述
`prefix`	字符串	是	要匹配的GT前缀
`peer_id`	整数	是	目标对等体ID（`0` = 丢弃）
`priority`	整数	否	路由优先级（默认`1`，数字越小优先级越高）
`description`	字符串	否	人类可读的描述
`source_ssn` / `dest_ssn`	整数	否	匹配/重写被叫方SSN
`source_tt` / `dest_tt`	整数	否	匹配/重写翻译类型
`source_npi` / `dest_npi`	整数	否	匹配/重写编号计划指示符
`source_nai` / `dest_nai`	整数	否	匹配/重写地址性质指示符

要删除路由，请对`/routing/gt-routes/{prefix}:{peer_id}`发出`DELETE`请求（例如`/routing/gt-routes/44:1`）。

测试路由匹配

使用路由测试页面模拟具有特定指示符的传入流量：

1. 输入GT前缀（例如，“441234567890”）
2. 设置TT、NPI、NAI值
3. 点击“测试路由”
4. 查看哪个路由匹配以及应用了哪些转换

故障排除

路由未匹配

检查特异性顺序：

1. 最长GT前缀优先
2. 然后是特定的source_ssn优先于通配符
3. 然后是特定的source_tt优先于通配符
4. 然后是特定的source_npi优先于通配符
5. 然后是特定的source_nai优先于通配符
6. 最后，较低的优先级数字获胜

转换未应用

验证dest_字段是否已设置：

空的dest_tt/dest_npi/dest_nai意味着“保留原始值” 必须明确设置dest_*以应用转换

重复路由问题

:bag表允许重复：

GT路由表的类型为:bag（允许具有相同键的多个记录）

- 键为`{gt_prefix, source_ssn}`
- 使用TT/NPI/NAI区分具有相同前缀+SSN的路由
- 删除不需要的重复路由

最佳实践

1. 记录您的转换：使用描述性路由描述
2. 彻底测试：在生产之前使用路由测试页面
3. 监控日志：检查INFO日志以获取已应用的转换
4. 从简单开始：首先添加基本前缀路由，然后添加指示符规则
5. 明智地使用通配符：空的source_*字段匹配任何值
6. 尽可能保留：仅转换需要更改的内容
7. 考虑特异性：更特定的路由覆盖较不特定的路由

有关更多信息，请参见：

- ← 回主文档
- STP指南 - 完整路由文档
- 配置参考 - 所有配置选项
- API指南 - REST API文档

OmniSS7 由Omnitouch网络服务提供

M3UA STP 全局标题路由

← 回主文档

概述

M3UA STP 除了点码 (PC) 路由外，还支持 全局标题 **(GT)** 路由。GT 路由根据 SCCP 层中的被叫方全局标题路由 SS7 消息，并自动回退到 PC 路由。

工作原理

路由优先级

1. **GT** 路由（如果启用）：从 SCCP 中提取被叫 GT → 在 GT 路由表中查找
2. **PC** 路由（回退）：从 MTP3 中提取 DPC → 在 PC 路由表中查找

GT 前缀匹配

- 使用 最长前缀匹配 算法
- 示例: GT "447123456789"
  - 匹配 "447"（长度 3）而不是 "44"（长度 2）
  - 路由到配置为 "447" 前缀的对等方

配置

启用 **GT** 路由

在 `config/runtime.exs` 中:

```
config :omniss7,
  enable_gt_routing: true,  # 置为 false 仅使用 PC 路由
```

定义 GT 路由

```
config :omniss7,
  m3ua_gt_routes: [
    # 将美国号码（以 1234 开头）路由到对等方 1
    %{gt_prefix: "1234", peer_id: 1, priority: 1, description: "美国号码"},

    # 将英国号码（以 44 开头）路由到对等方 2
    %{gt_prefix: "44", peer_id: 2, priority: 1, description: "英国号码"},

    # 更具体的英国移动号码路由（447xxx）
    %{gt_prefix: "447", peer_id: 2, priority: 1, description: "英国移动"},

    # 将澳大利亚号码（以 61 开头）路由到对等方 3
    %{gt_prefix: "61", peer_id: 3, priority: 1, description: "澳大利亚号码"}
  ]
```

> 完整的、可编辑的 STP 配置包含在 `config/stp_example_runtime.exs` 中。有关如何启用它，请参见 STP 指南。

使用示例

示例 1：基本 GT 路由

消息：SCCP UDT，Called GT = "447700123456"

路由决策：

- 提取的 GT："447700123456"
- 最佳匹配："447"（英国移动）→ 对等方 2
- 消息路由到对等方 2

示例 2: 未找到 GT, 回退到 PC

消息: SCCP UDT, Called GT = "331234567890", DPC = 100

路由决策:

- 提取的 GT: "331234567890"
- 未找到 "33" 的 GT 路由
- 回退到 PC 路由
- DPC 100 → 对等方 1 (来自 PC 路由表)
- 消息路由到对等方 1

示例 3: 没有 GT

消息: 没有 SCCP 的 M3UA DATA, 仅 MTP3, DPC = 200

路由决��:

- 没有 SCCP 层, 没有 GT
- 仅使用 PC 路由
- DPC 200 → 对等方 2
- 消息路由到对等方 2

运行时管理

这些函数由 `M3UARouting` 提供, 可以从 `iex -S mix` 控制台调用。

□□ GT □□

```
# □□□□□ GT □□
{:ok, routes} = M3UARouting.get_all_gt_routes()

# □□□□□□□□ 14 □□□□□□□□
# {:m3ua_gt_route, gt_prefix, source_ssn, peer_id, dest_ssn,
priority,
#  description, enabled, source_tt, dest_tt, source_npi, dest_npi,
#  source_nai, dest_nai}
#
# □□□□□□source_ssn □□□□/□□□□□□□□ nil□□
[
  {:m3ua_gt_route, "1234", nil, 1, nil, 1, "□□□□", true, nil, nil,
nil, nil, nil, nil},
  {:m3ua_gt_route, "44", nil, 2, nil, 1, "□□□□", true, nil, nil,
nil, nil, nil, nil},
  {:m3ua_gt_route, "447", nil, 2, nil, 1, "□□□□", true, nil, nil,
nil, nil, nil, nil},
  {:m3ua_gt_route, "61", nil, 3, nil, 1, "□□□□□□", true, nil, nil,
nil, nil, nil, nil}
]
```

□□□□ GT □□

`add_gt_route/3` □□□□□□□□□□□□□□□□□ ID□□□□□□□□□`priority`□`description`□`source_ssn`□`dest_ssn`□TT/NPI/NAI □□□□□□□□□□ `enabled`□□□□□□□□□

```
# □□□add_gt_route(gt_prefix, peer_id, opts \ [])

# □□□□□□□□□□
M3UARouting.add_gt_route("49", 4, priority: 1, description: "□□□
□")

# □□□□ SMS □□□□□□ SSN 8□SMSC□□□ SSN □□□ 6□HLR□
M3UARouting.add_gt_route("55", 4,
  source_ssn: 8,
  dest_ssn: 6,
  priority: 1,
  description: "□□□ HLR □ SMS □□"
)
```

□□ GT □□

```
# □□□□□gt_prefix, peer_id, □□□ source_ssn□
M3UARouting.remove_gt_route("49", 4)
```

□□ GT □□

```
# □□□□□□□□□□□□ GT
# □□□lookup_peer_by_gt(global_title, source_ssn \ nil, opts \ [])
{:ok, peer, rewrites} =
M3UARouting.lookup_peer_by_gt("447700123456")

# `peer` □ 13 □□□ m3ua_peer □□□
# {:m3ua_peer, peer_id, peer_name, protocol, role, local_ip,
local_port,
#  remote_ip, remote_port, routing_context, point_code,
network_indicator, status}
#
# `rewrites` □□□□□□□□□□□□□□ MAP□
# nil □□□“□□□□”□
#   %{dest_ssn: nil, dest_tt: nil, dest_npi: nil, dest_nai: nil}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `"55"` □□□ `dest_ssn: 6`□

```
{:ok, peer, rewrites} =
M3UARouting.lookup_peer_by_gt("5512345678", 8)
# rewrites => %{dest_ssn: 6, dest_tt: nil, dest_npi: nil,
dest_nai: nil}
```

`peer_id: 0`□DROP□□□□□ `{:error, :dropped}`□□□□□ GT □□ `{:error, :no_gt_route}`□

□□□□

STP □□ GT □□□□□

```
[info] □□ M3UA DATA: OPC=1000, DPC=200, Called GT="447700123456"□□
□□□□ 1□
[info] □ GT □□□447700123456
[info] □□□□□□□ 1 □□□□□□□ 2
```

□□ GT □□□□□

```
[info] □□ M3UA DATA: OPC=1000, DPC=200, Called GT="331234567"□□□□□
□ 1□
[info] □□□ GT □□ 331234567
[info] □□□□□□□ 1 □□□□□□□ 2□□□ PC □□□
```

SCCP 支持

支持的 SCCP 消息

消息	代码	支持
UDT (单元数据)	0x09	完全支持：GT 提取、指示重写、路由
UDTS (单元数据服务)	0x0A	完全支持：在路由失败时生成的消息□回
XUDT (扩展单元数据)	0x11	完全支持：GT 提取、指示重写、跳数递减、消息□回
XUDTS (扩展单元数据服务)	0x12	完全支持：在路由失败时生成的消息□回
LUDT (长单元数据)	0x13	完全支持：GT 提取、指示重写、跳数递减、消息□回
LUDTS (长单元数据服务)	0x14	完全支持：在路由失败时生成的消息□回

XUDT 和 LUDT 携带 SCCP 跳数计数器。STP 在每次传输跳数时递减它，如果它达到零，则丢弃消息并将其□回给发起者（当请求□回错误时）。有关详细信息，请参见 STP 指南。

支持的 GT 格式

- **GTI 0-4**：所有 GT 指示类型
- **BCD** 编码：标准电话数字编码
- 地址性质：所有类型（国际、国内等）

性能

- **GT** 查找：O(n)，其中 n = GT 路由数量
- 前缀匹配：通过最长匹配优先排序进行优化
- 建议：保持 GT 路由数量 < 1000 以获得最佳性能

故障排除

GT 未被提取

问题：日志显示 Called GT=nil

解决方案：

1. 检查 SCCP 消息类型是否为 UDT/XUDT/LUDT
2. 验证被叫方地址是否携带全局标题
3. 启用调试日志以查看 SCCP 解析详细信息

选择了错误的对等方

问题：消息路由到错误的对等方

解决方案：

1. 检查 GT 前缀配置（最长匹配优先）
2. 验证 peer_id 是否与配置的对等方匹配
3. 检查优先级值（较低 = 较高优先级）

回退到 PC 不工作

问题：未找到 GT 时消息被丢弃

解决方案：

1. 验证 enable_gt_routing 是否为true
2. 检查 PC 路由表中是否有 DPC 的路由
3. 确保 PC 的对等方处于活动状态

最佳实践

1. 使用特定前缀：更具体的路由覆盖不太具体的路由

- 好：`"447"`、`"448"`、`"449"` 用于不同的英国运营商
- 坏：单一的 `"44"` 用于所有英国

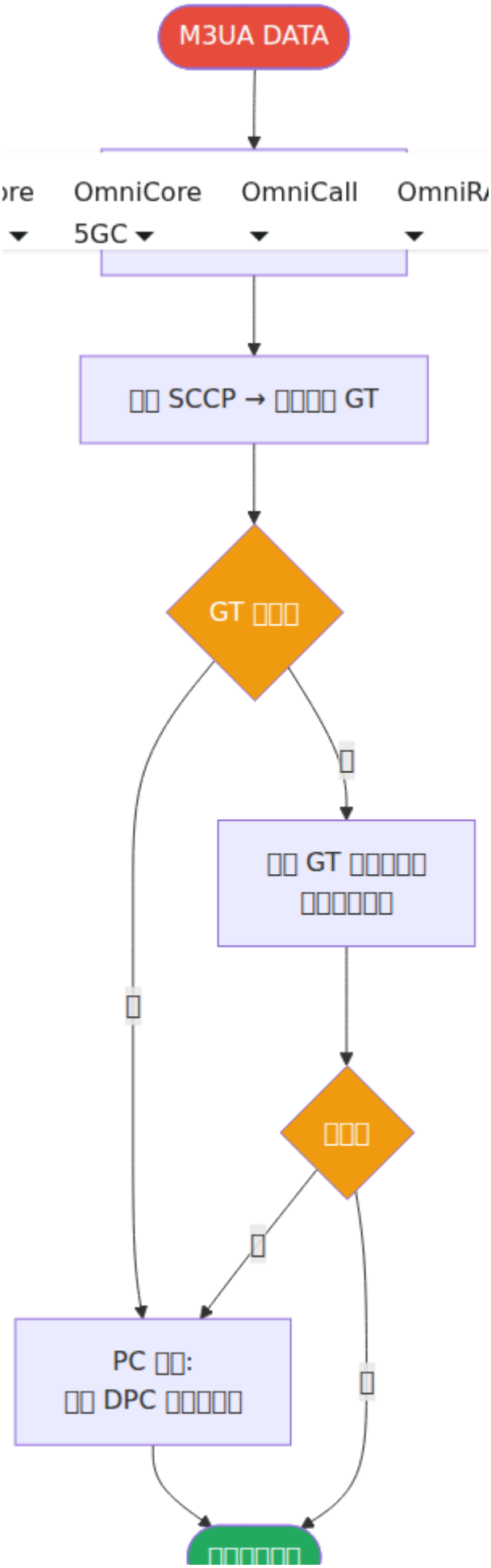
2. 记录路由：使用描述字段

```
M3UARouting.add_gt_route("1234", 1, priority: 1,
  description: "ACME Corp - 美国东海岸")
```

3. 测试路由：使用 `lookup_peer_by_gt/1` 进行验证

4. 监控：检查日志以获取路由失败信息

5. 回退：始终配置 PC 路由作为备份


M3UA DATA
ore
OmniCore
5GC
OmniCall
OmniR
□□ SCCP → □□□□ GT
GT □□□
□
□□ GT □□□□□
□□□□□□
□
□□□
□
PC □□:
□□ DPC □□□□□
□
□□□□□□□

OmniSS7 □ Omnitouch □□□□□□