

通知中心设计：邮件、短信、Push与站内信

Original 极特架构笔记 极特架构笔记 2025年11月18日 08:00 广东

业务量的增长，往往伴随着需求的失控。

订单、营销、风控等各个系统都在向用户发送消息，渠道混乱、体验割裂、成本高昂等问题随之而来。

当一个系统需要面对多渠道、多场景的触达需求时，一个设计良好的统一通知中心就成了刚需。它不仅是技术架构的优化，更是对用户体验和运营效率的战略投资。

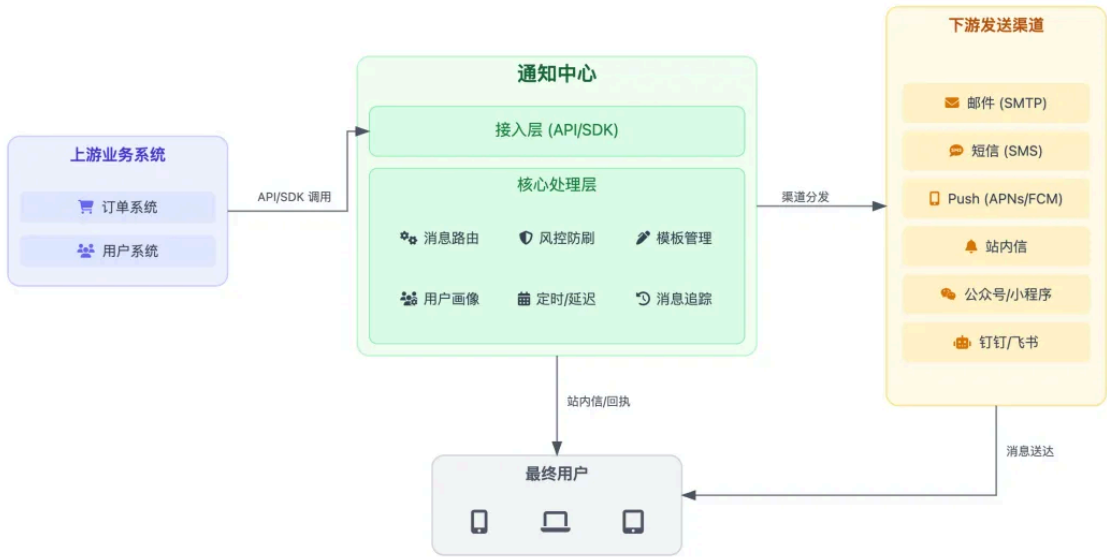
今天，极特就跟大家探讨下统一通知中心的设计、核心架构与实现策略。

什么是统一通知中心

统一通知中心是一种服务于全局的中间件，将所有业务线的消息发送需求进行收口，并统一处理所有下游渠道（邮件、短信、Push、站内信等）的发送逻辑，扮演着交通枢纽的角色。

与微服务架构中的API网关类似，通知中心屏蔽了下游渠道的复杂性。上游业务方不再需要关心每个渠道的SDK、API密钥、发送频率限制和模板规范，只需调用一个统一的接口，告诉通知中心要给谁、发什么内容、在什么场景下发即可。

通知中心总体架构图



通知中心总体架构图

通知中心带来的价值

- **开发减负**：一次接入，所有业务线复用，无须重复开发对接各个渠道的逻辑；
- **体验升级**：全局管控频率与时段，把发不发的控制权统一管控，让用户体验更优；
- **运营赋能**：统一数据看板与A/B测试，让最佳文案和渠道一目了然；
- **成本可控**：所有付费渠道集中管理，每一笔花费都指向明确的效果。

通知中心的核心架构设计

通知中心要设计得足够健壮，就要应对好异步化、可扩展、高可用等处理流程。极特就来给大家分析下通知中心的核心处理流程与关键技术选型。

核心处理流程



通知中心核心流程图

- **消息接入**：支持多协议接入，并为每条消息分配唯一ID，用于全链路追踪；
- **风控预处理**：发送前必经风控，去重、扫黑、控频，拦截无效流量；
- **模板匹配**：按规则匹配消息模板，实现千人千面的个性化起点；
- **内容组装**：将动态变量填入模板，生成最终的发送内容；
- **智能路由与发送**：由路由引擎决策最优渠道并发送，同时获取送达回执；
- **回执与统计**：全量记录发送数据，基于送达率、点击率进行优化。

技术选型

- 消息队列 (Kafka/RocketMQ)：削峰填谷，系统更稳健；
- 分布式缓存 (Redis)：存规则、偏好、模板，减轻 DB 压力；
- 分布式任务调度 (XXL-Job/PowerJob)：调度定时与重试，可靠省心；

智能路由

智能路由，主要就是决策：什么消息，走什么路，什么时候走。



通知渠道优先级与降级策略图

场景化渠道优先级

根据不同的场景，发送渠道的选择也有些不同。

- 强验证类 (登录/注册)：必达高效。短信 -> 语音 -> 站内信；
- 强提醒类 (订单/支付)：强感知。Push -> 微信/短信；
- 营销类 (促销/新品)：高覆盖。Push -> 公众号 -> 邮件/短信；
- 关怀类 (祝福/升级)：有温度。站内信 -> 邮件 -> Push；
- 通知类 (促销/新品)：必达可追溯。站内信 -> 邮件。

动态降级策略 (Fallback)

- **渠道不可用**：渠道不可用，立即切换；
- **用户偏好**：用户选择，高于一切规则；
- **成本与时效**：成本或延迟过高，则降级非关键消息。

与视觉设计相连

消息发出去了，用户收到了吗？打开率如何？为什么某个渠道的发送成功率突然下降了？没有数据，我们就如同在黑暗中裸奔。

通知中心数据流与可观测性

构建闭环、可度量、可监控的通知体系

三 消息生命周期与数据流转

+

1. 消息创建 & 入库

API接收请求，生成唯一MessageID，初始状态为“待处理”，存入数据库。

🔄

2. 状态变更：处理中

核心服务消费消息，进行风控、模板匹配、渠道选择，状态更新为“处理中”。

📤

3. 状态变更：已发送

消息交由各渠道网关，调用三方API成功后，状态更新为“已发送”。

🏁

4. 最终状态：成功 / 失败

通过异步回调或定时轮询获取渠道回执，更新最终状态。

👥 系统可观测性

📊

Metrics (指标)

- 各渠道发送量 (QPS/QPM)，成功率，延迟
- API接口响应时间 (P95/P99)，消息队列积压数

📄

Logging (日志)

- 记录含TraceID的结构化日志，覆盖请求、决策、调用
- 便于快速检索和问题排查

👤

Tracing (追踪)

- 可视化消息从触发到接收的全链路，定位性能瓶颈

📊 数据分析与运营看板

📬

送达/打开率

👥

用户活跃时段

📊

渠道分布

👉

用户点击转化

通知中心数据流与可观测性设计图

可观测性的三大支柱

指标 (Metrics)

我们需要一套生命体征监测系统，实时看护着通知中心的心跳、血压和体温。可采用 **Prometheus**实现。观察指标：

- **业务指标**：各渠道发送量、成功率、打开率、点击率。
- **系统指标**：API响应时间、MQ积压数、数据库连接池使用率。
- **成本指标**：各渠道的费用消耗。

它会第一时间发出警报：系统，生病了。

日志 (Logging)

日志就是现场快照。采用结构化的 **JSON**格式，抛弃杂乱的字符串。每一条日志都必须包含关键字段，如 `traceId`，`userId`，`templateId`，`channel`。

这些结构化的日志被收集到 **ELK**或 **Loki**中，我们才能快速地从海量信息中，通过 `traceId` 筛选出某个用户收到的所有消息，拼凑出问题的全貌。

追踪 (Tracing)

一条消息从诞生到消亡，经历了哪些服务？在哪一站耽搁了？

这需要一个贯穿全程的唯一标识，也就是 `traceId`。像一根无形的线，将消息从生产者，到消息队列，再到消费者的完整生命周期串联起来。我们通常遵循 **OpenTelemetry**规范来实现。

有了这根线，我们才能画出它完整的行动路线图（GPS），精准定位性能瓶颈和错误根源。

数据分析与运营

眼睛让我们看见当下，大脑则让我们预见未来。

数据分析，就是通知中心的大脑。所有的数据沉淀到**数据仓库**中，通过 **Tableau**或 **Metabase**等BI工具进行可视化分析。

- **渠道质量评估**：哪个短信供应商成本更低？
- **用户行为分析**：用户更愿意在什么时间段接收营销信息？
- **内容效果度量**：通过A/B测试，评估不同文案、不同落地页的转化效果。

写在最后

通知中心，就是做到通知渠道统一。

但这只是基础。能否依靠数据和策略，实现精准触达，才是关键。

这决定了它只是一个成本部门，还是一个价值源泉。核心差异，就在于此。



架构设计



