美团点评技术团队

大众点评支付渠道网关系统的实践之路

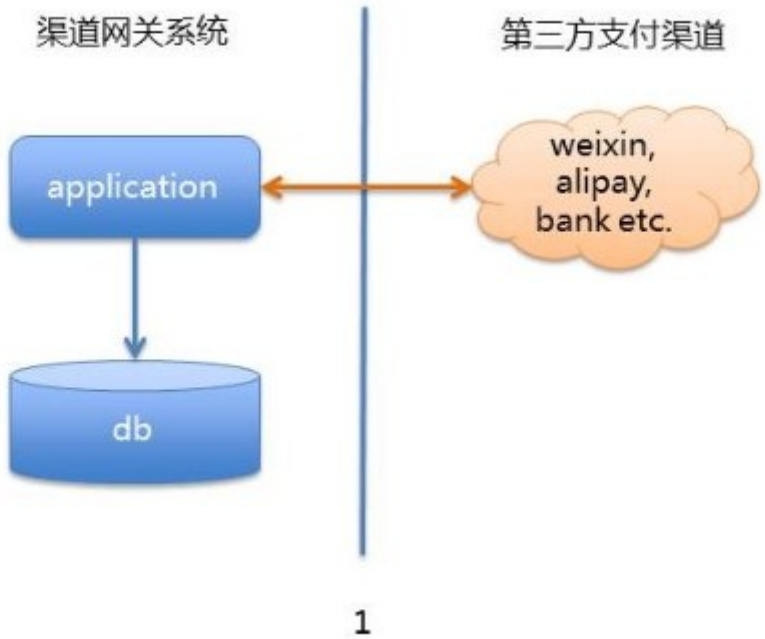
李力 ·2016-05-20 18:00

业务的快速增长，要求系统在快速迭代的同时，保持很好的扩展性和可用性。其中，交易系统除了满足上述要求之外，还必须保持数据的强一致性。对系统开发人员而言，这既是机遇，也是挑战。本文主要梳理大众点评支付渠道网关系统在面对这些成长烦恼时的演进之路，以及过程中的一些思考和实践。

在整个系统的演进过程中，核心思路是：大系统做小，做简单（具体描述可参考《高可用性系统在大众点评的实践与经验 (http://tech.meituan.com/high-availability-systems-dianping.html)》）。在渠道网关系统实践过程中，可以明显区分出几个有代表性的阶段。

一、能用阶段

早期业务流量还不是很大，渠道网关系统业务逻辑也很简单，一句话总结就是：让用户在交易的时候，能顺利把钱给付了。做的事情可简单概括成3件：发起支付请求、接收支付成功通知以及用户要求退款时原路退回给用户的支付账户。这个阶段系统实践比较简单，主要就是“短、平、快”，快速接入新的第三方支付渠道并保证能用。系统架构如图1。



1

二、可用阶段

在系统演进初期的快速迭代过程中，接入的第三方支付渠道不多，系统运行还算比较平稳，一些简单问题也可通过开发人员人工快速解决。但随着接入的第三方支付渠道不断增多，逐渐暴露出一些新的问题：

(1) 所有的业务逻辑都在同一个物理部署单元，不同业务之间互相影响（例如退款业务出现问题，但是与此同时把支付业务也拖垮了）；

(2) 随着业务流量的增大，数据库的压力逐渐增大，数据库的偶尔波动造成系统不稳定，对用户的支付体验影响很大；

(3) 支付、退款等状态的同步很大程度上依赖第三方支付渠道的异步通知，一旦第三方支付渠道出现问题，造成大量客诉，用户体验很差，开发、运营都很被动。

针对(1)中的业务之间互相影响问题，我们首先考虑进行服务拆分，将之前一个大的物理部署单元拆成多个物理部署单元。有两种明显的可供选择的拆分策略：

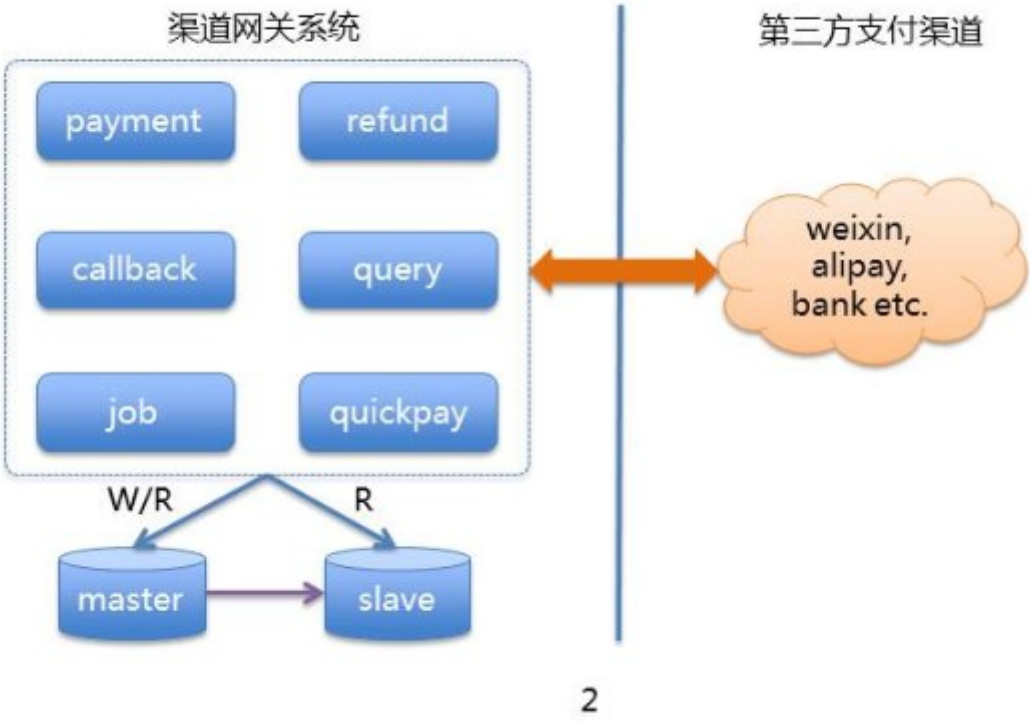
- **按照渠道拆分**，不同的第三方支付渠道独立一个物理部署单元，例如微信一个部署单元，支付宝一个部署单元等；
- **按照业务类型拆分**，不同的业务独立一个物理部署单元，例如支付业务一个部署单元，退款业务一个部署单元等。

考虑到在当时的流量规模下，支付业务优先级最高，退款等业务的优先级要低；而有些渠道的流量占比很小，作为一个独立的部署单元，会造成一定的资源浪费，且增加了系统维护的复杂度。基于此，我们做了一个符合当时系统规模的trade-off：选择了第2种拆分策略 — 按照业务类型拆分。

针对(2)中的DB压力问题，我们和DBA一起分析原因，最终选择了Master-Slave方案。通过增加Slave来缓解查询压力；通过强制走Master来保证业务场景的强一致性；通过公司的DB中间件Zebra来做负载均衡和灾备切换，保证DB的高可用性。

针对(3)中的状态同步问题，我们对不同渠道进行梳理，在已有的第三方支付渠道异步通知的基础上，通过主动查询定时批量同步状态，解决了绝大部分状态同步问题。对于仍未同步的少量Case，系统开放出供内部使用的API，方便后台接入和开发人员手动补单。

在完成上述的实践之后，渠道网关系统已达到基本可用阶段，通过内部监控平台可以看到，核心服务接口可用性都能达到**99.9%**以上。演化之后的系统架构如图2。



2

三、柔性可用阶段

在解决了业务隔离、DB压力、状态同步等问题后，渠道网关系统度过一段稳定可用的时期。但架不住业务飞速增长的压力，之前业务流量规模下的一些小的系统波动、流量冲击等异常，在遭遇流量洪峰时被急剧放大，最终可能成为压垮系统的最后一根稻草。在新的业务流量规模下，我们面临着新的挑战：

(1) 随着团队的壮大，新加入的同学在接入新的渠道或者增加新的逻辑时，往往都会优先选用自己熟悉的方式完成任务。但熟悉的不一定是合理的，有可能会引入新的风险。特别是在与第三方渠道对接时，系统目前在使用的HTTP交互框架就有`JDK HttpURLConnection/HttpsURLConnection`、`Httpclient3.x`、`Httpclient4.x`（4.x版本内部还分别有使用不同的小版本）。仅在这个上面就踩过好几次惨痛的坑。

(2) 在按业务类型进行服务拆分后，不同业务不再互相影响。但同一业务内部，之前流量规模小的时候，偶尔波动一次影响不大，现在流量增大后，不同渠道之间就开始互相影响。例如支付业务，对外统一提供分布式的支付API，所有渠道共享同一个服务RPC连接池，一旦某一个渠道的支付接口性能恶化，导致大量占用服务RPC连接，其他正常渠道的请求都无法进来；而故障渠道性能恶化直接导致用户无法通过该渠道支付成功，连锁反应导致用户多次重试，从而进一步导致恶化加剧，最终引起系统雪崩，拒绝服务，且重启后的服务还有可能被大量的故障渠道重试请求

恶化加剧，最终引起系统雪崩，拒绝服务，且重启后的服务还有可能被大量的故障渠道重试请求给再次击垮。

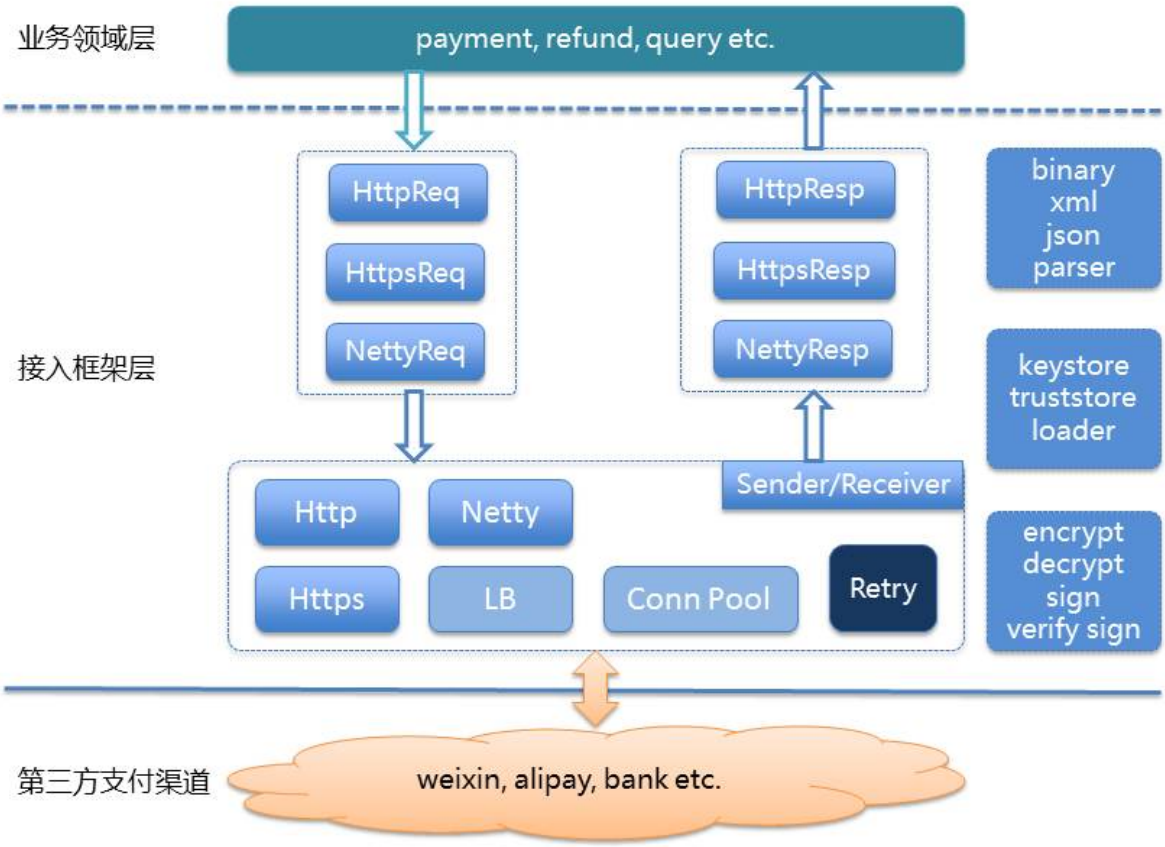
(3) 目前接入的第三方支付渠道，无论是第三方支付公司、银行或是其他外部支付机构，基本都是通过重定向或SDK的方式引导用户完成最终支付动作。在这条支付链路中，渠道网关系统只是在后端与第三方支付渠道进行交互（生成支付重定向URL或预支付凭证），且只能通过第三方支付渠道的异步通知或自己主动进行支付查询才能得知最终用户支付结果。一旦某个第三方支付渠道内部发生故障，渠道网关系统完全无法得知该支付链路已损坏，这对用户支付体验造成损害。

(4) 现有的渠道网关的DB，某些非渠道网关服务仍可直接访问，这对渠道网关系统的DB稳定性、DB容量规划等带来风险，进而影响渠道网关系统的可用性，内部戏称被戴了“绿帽子”。

(5) 对于退款链路，系统目前未针对退款异常case进行统一收集、整理并分类，且缺乏一个清晰的退款链路监控。这导致用户申请退款后，少量用户的退款请求最终未处理成功，用户发起客诉。同时由于缺乏监控，导致这种异常退款缺乏一个后续推进措施，极端情形下，引起用户二次客诉，极大损害用户体验和公司信誉度。

为最大程度解决问题(1)中描述的风险，在吸取踩坑的惨痛教训后，我们针对第三方渠道对接，收集并整理不同的应用场景，抽象出一套接入框架。接入框架定义了请求组装、请求执行、响应解析和错误重试这一整套网关交互流程，屏蔽了底层的HTTP或Socket交互细节，并提供相应的扩展点。针对银行渠道接入存在前置机这种特殊的应用场景，还基于Netty抽象出连接池（Conn Pool）和简单的负载均衡机制（LB, 提供Round Robin路由策略）。不同渠道在接入时可插入自定义的组装策略（扩展已有的HttpReq、HttpsReq或NettyReq），执行策略[扩展已有（Http、Https或Netty）Sender/Receiver]，解析策略（扩展已有的HttpResp、HttpsResp或NettyResp），并复用框架已提供的内容解析（`binary/xml/json parser`）、证书加载（`keystore/truststore loader`）和加解密签名（`encrypt/decrypt/sign/verify sign`）组

件，从而在达到提高渠道接入效率的同时，尽可能减少新渠道接入带来的风险。接入框架的流程结构如图3。



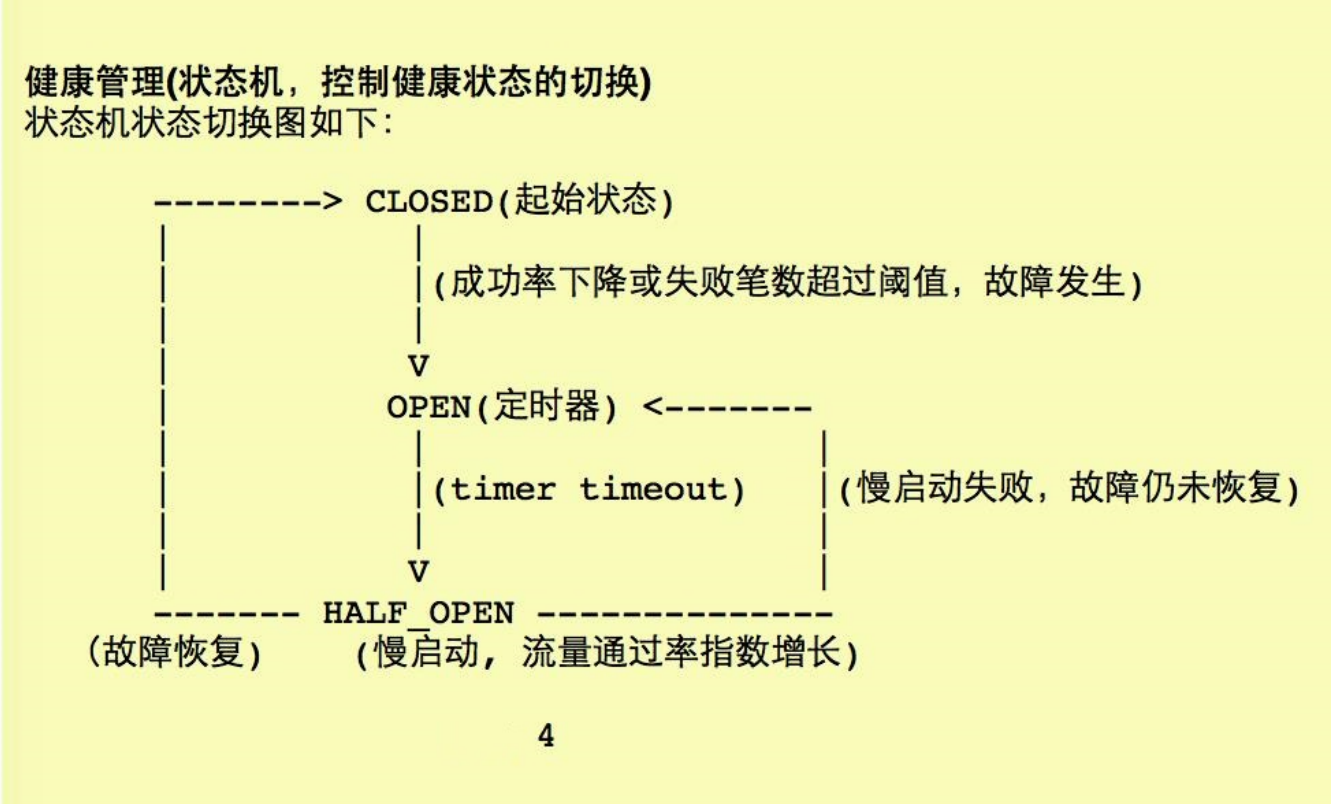
3

为解决问题(2)中渠道之间相互影响，一个简单直观的思路就是渠道隔离。如何隔离，隔离到什么程度？这是2个主要的问题点：

- **如何隔离** 考虑过将支付服务进一步按照渠道拆分，将系统继续做小，但是拆分后，支付API的调用端需要区分不同渠道调用不同的支付API接口，这相当于将渠道隔离问题抛给了调用端；同时拆分后服务增多，调用端需要维护同一渠道支付业务的多个不同RPC-API，复杂度提高，增加了开发人员的维护负担，这在当前的业务流量规模下不太可取。所以我们选择了在同一个支付服务API内部进行渠道隔离。由于共用同一个支付服务服务API连接池，渠道隔离的首要目标就是避免故障渠道大量占用AP连接池，对其他正常渠道造成株连影响。如果能够自动检测出故障渠道，并在其发生故障的初期阶段就快速失败该故障渠道的请求，则从业务逻辑上就自动完成了故障渠道的隔离。
- **隔离到什么程度** 一个支付渠道下存在不同的支付方式(信用卡支付、借记卡支付、余额支付等)，而有些支付方式(例如信用卡支付)还存在多个银行。所以我们直接将渠道隔离的最小粒度定义到支付渠道 -> 支付方式 -> 银行。

基于上述的思考，我们设计并实现了一个针对故障渠道的快速失败（fail-fast）机制：

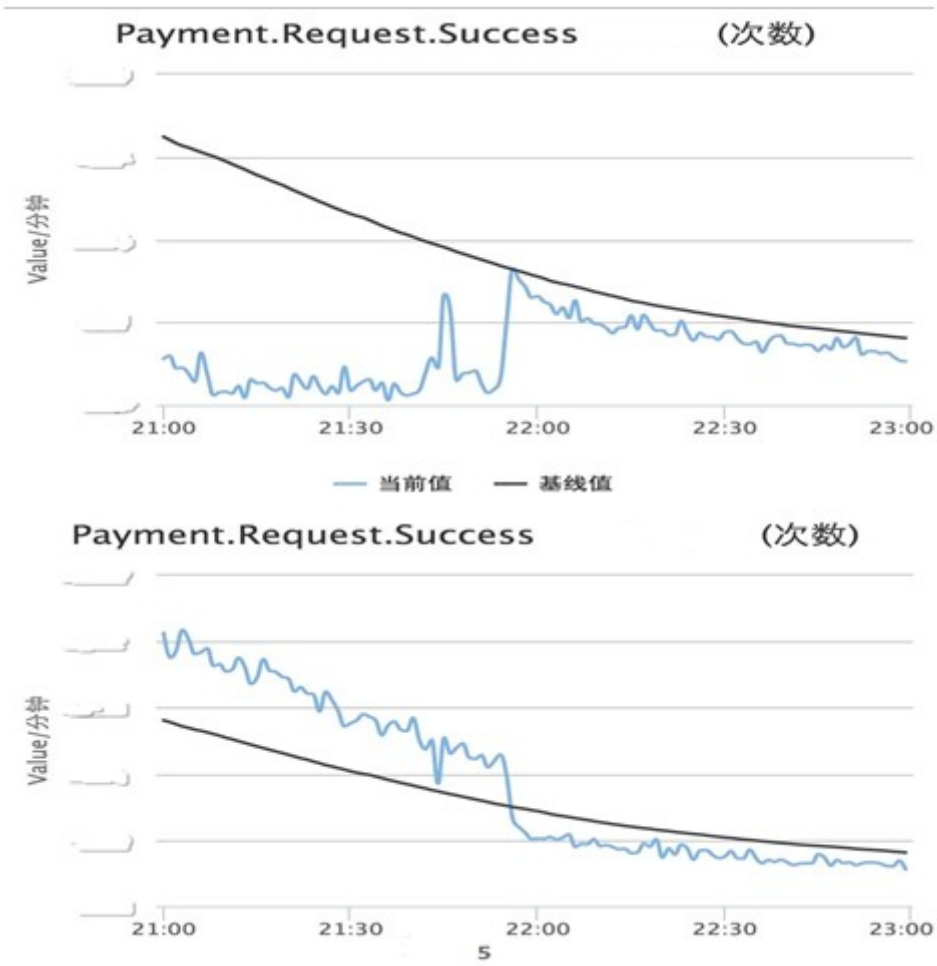
- 将每一笔支付请求所附带的支付信息抽象为一个特定的fail-fast路径，请求抽象成一个fail-fast事务，请求成功即认为事务成功，反之，事务失败。
- 在fail-fast事务执行过程中，级联有2个fail-fast断路开关：
  - 静态开关，根据人工配置（***on/off***），断定某个支付请求是否需快速失败。
  - 动态开关，根据历史统计信息，确定当前健康状态，进而断定是否快速失败当前支付请求。
- 动态断路开关抽象了3种健康状态（closed-放行所有请求；half_open-部分比例的请求放行；open-快速失败所有请求），并依据历史统计信息（总请求量/请求失败量/请求异常量/请求超时量），在其内部维护了一个健康状态变迁的状态机。状态变迁如图4。



- 状态机的每一次状态变迁都会产生一个健康状态事件，收银台服务可以监听这个健康状态事件，实现支付渠道的联动上下线切换。
- 每一笔支付请求结束后都会动态更新历史统计信息。

经过线上流量模拟压测观察，fail-fast机制给系统支付请求增加了**1~5ms**的额外耗时，相比第三方渠道的支付接口耗时，占比**1%~2%**，属于可控范围。渠道故障fail-fast机制上线之后，结合压测配置，经过几次微调，稳定了线上环境的fail-fast配置参数。

在前不久的某渠道支付故障时，通过公司内部的监控平台，明显观察到fail-fast机制起到很好的故障隔离效果，如下图5。



5

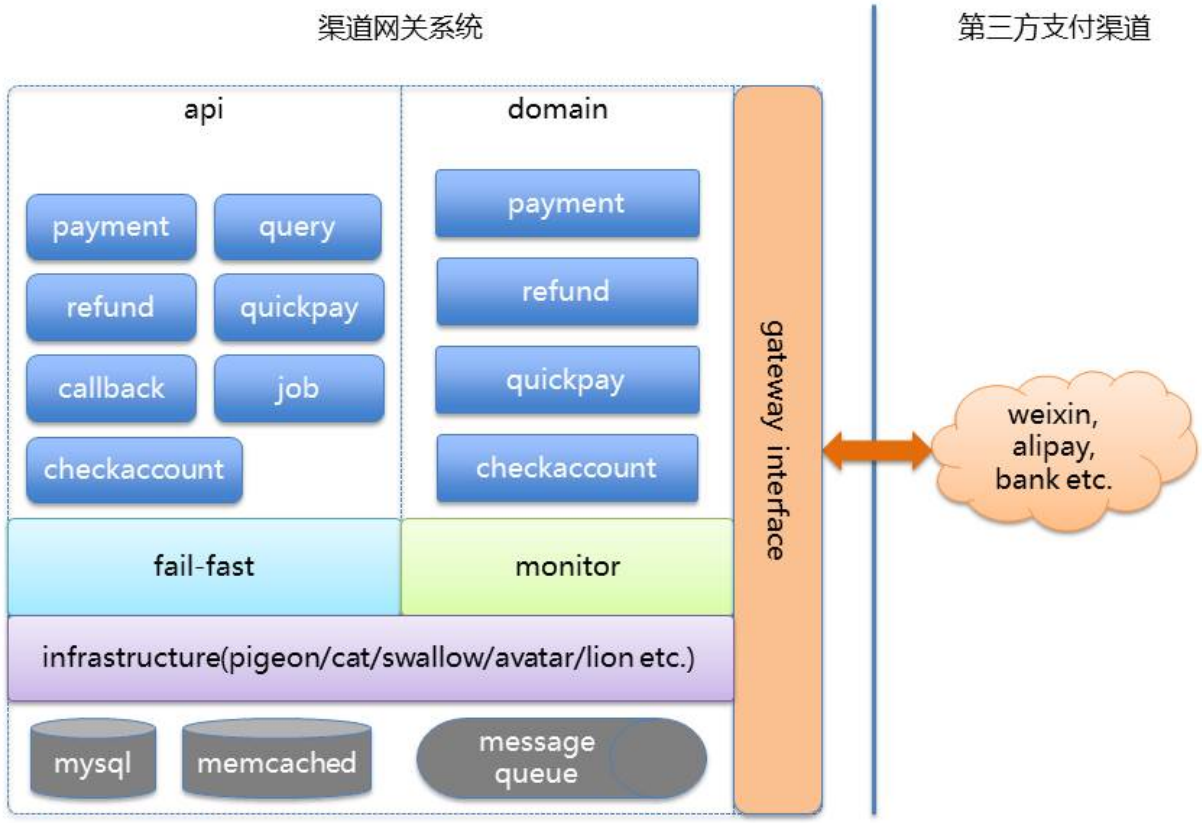
为解决问题(3)中支付链路可用性监测，依赖公司内部的监控平台上报，实时监控支付成功通知趋势曲线；同时渠道网关系统内部从业务层面自行实现了支付链路端到端的监控。秒级监控支付链路端到端支付成功总量及支付成功率，并基于这2个指标的历史统计信息，提供实时的支付链路邮件或短信报警。而在流量高峰时，该监控还可通过人工手动降级（异步化或关闭）。这在很大程度上提高了开发人员的核心支付链路故障响应速度。

为解决问题(4)中的“绿帽子”，渠道网关系统配合DBA回收所有外部系统的DB直接访问权限，提供替换的API以供外部系统访问，这给后续的提升DB稳定性、DB容量规划以及后续可能的异步多机房部署打下基础。

针对问题(5)中退款case，渠道网关系统配合退款链路上的其他交易、支付系统，从源头上对第三方渠道退款异常case进行统一收集、整理并分类，并形成退款链路核心指标（退款当日成功率/次日成功率/7日成功率）监控，该部分的系统实践会随着后续的“退款链路统一优化”一起进行分享；

随着上述实践的逐步完成，渠道网关系统的可用性得到显著提高，核心链路的API接口可用性达到**99.99%**，在公司的917大促中，渠道网关系统平稳度过流量高峰，并迎来了新的记录：提交第

到99.99%，在公司的917大促中，渠道网关系统平稳度过流量高峰，并迎来了新的记录：提交第三方渠道支付请求的TPS达到历史新高。且在部分渠道接口发生故障时，能保证核心支付API接口的稳定性，并做到故障渠道的自动检测、恢复，实现收银台对应渠道的联动上下线切换。同时，通过核心支付链路支付成功率监控，实现第三方渠道内部故障时，渠道上下线的手动切换。至此，基本保证了在部分第三方渠道有损的情况下，渠道网关系统的柔性可用。演化后的此阶段系统架构如图6。



6

四、经验与总结

在整个渠道网关系统一步步的完善过程中，踩过很多坑，吃过很多教训，几点小的收获：

1. 坚持核心思想，拆分、解耦，大系统做小，做简单；
2. 系统总会有出问题的时候，重要的是如何快速定位、恢复、解决问题，这是一个长期而又艰巨的任务；
3. 高可用性的最大敌人不仅是技术，还是使用技术实现系统的人，如何在业务、系统快速迭代的过程中，保证自我驱动，不掉队；
4. 高流量，大并发对每一个工程师既是挑战，更是机遇。

支付 渠道网关 高可用 服务隔离 fail-fast