TOP1
技术型企业案例研究智库

618大促开放网关承载海量调用量背后的架构实践

王栋

摘要

每年618大促，我们的网关承载了海量级别的流量；

在这种情况下，网关系统必须保证整个系统的稳定性和高可用，保证高性能和可靠，以支撑业务；

我们面临的是一个非常复杂的问题，并不是一种技术可以解决的；

基于这种复杂问题，怎样做到很好地提高它的性能和稳定性、复杂技术之间怎么整合保证整体网关的高可用，是本案例的重点。

技术升级改造

业界对于如何确保网关高可用的方法都比较单一，限流降级，线程/进程隔离等；

我们所面对的场景复杂，单一方法没有效果。

我们通过分析遇到的问题，逐一击破，在流量管控、统一接入、安全防护、协议适配方面摸索出了一套适合我们的自有的技术实践。

网关系统主要有两种：

第一种，叫客户端网关主要用来接收一些客户端的请求，也就是服务端；

第二种，叫开放网关，主要是公司（比如京东）对于第三方合作伙伴提供接口。

这两种不同网关所使用的技术非常类似。我们以开放网关案例进行分析。

面临的难点：

第一，网关系统需要扛几十亿的流量调用，接口的平稳运行、每一个接口在透传后端服务时的性能耗损都非常重要。

处理思路：

- 我们采用了一些多级缓存技术，或者更前置的Nginx+lua+Redis技术，让这种大流量应用能够脱离开JVM的依赖；
- 梳理各个接口，通过降级的策略把一些弱依赖的接口进行降级，从而保证核心应用的可用。

面临的难点：

第二，网关系统其实就是一个把Http请求透传到后端服务的过程。

我们的网关承接了一千以上的后端服务接口，面对这种情况，怎样做到服务与服务之间相互不影响？架构层面怎样能够杜绝蝴蝶效应、防止雪崩？

每个接口性能都不一致，而且每个接口所依赖的外部资源、数据库缓存等都不一样，几乎每天都会出现各种各样的问题，我们怎样通过一些隔离技术、治理技术等，保证当这些接口出现问题的时候，不会影响到全局？

处理思路：

隔离、流控、降级和熔断等

面临的难点：

第三，暴露的一千多个服务接口，就意味着后面有几十个甚至上百个团队，每天都可能上线新的需求。

面对这么复杂的情况，我们不可能每次后端服务器有任何修改，都需要有网关的修改或上线，这样网关会变得非常脆弱，稳定性极低。

处理思路：

动态接入的技术，让后端的网关能够通过一种接入的协议进行无缝接入，之后通过一些动态代理的方式，直接让后端的接口，不管做任何修改或上线，都可以通过后端管理平台从网关上对外进行透传发布。

有什么曲折、经验教训是什么

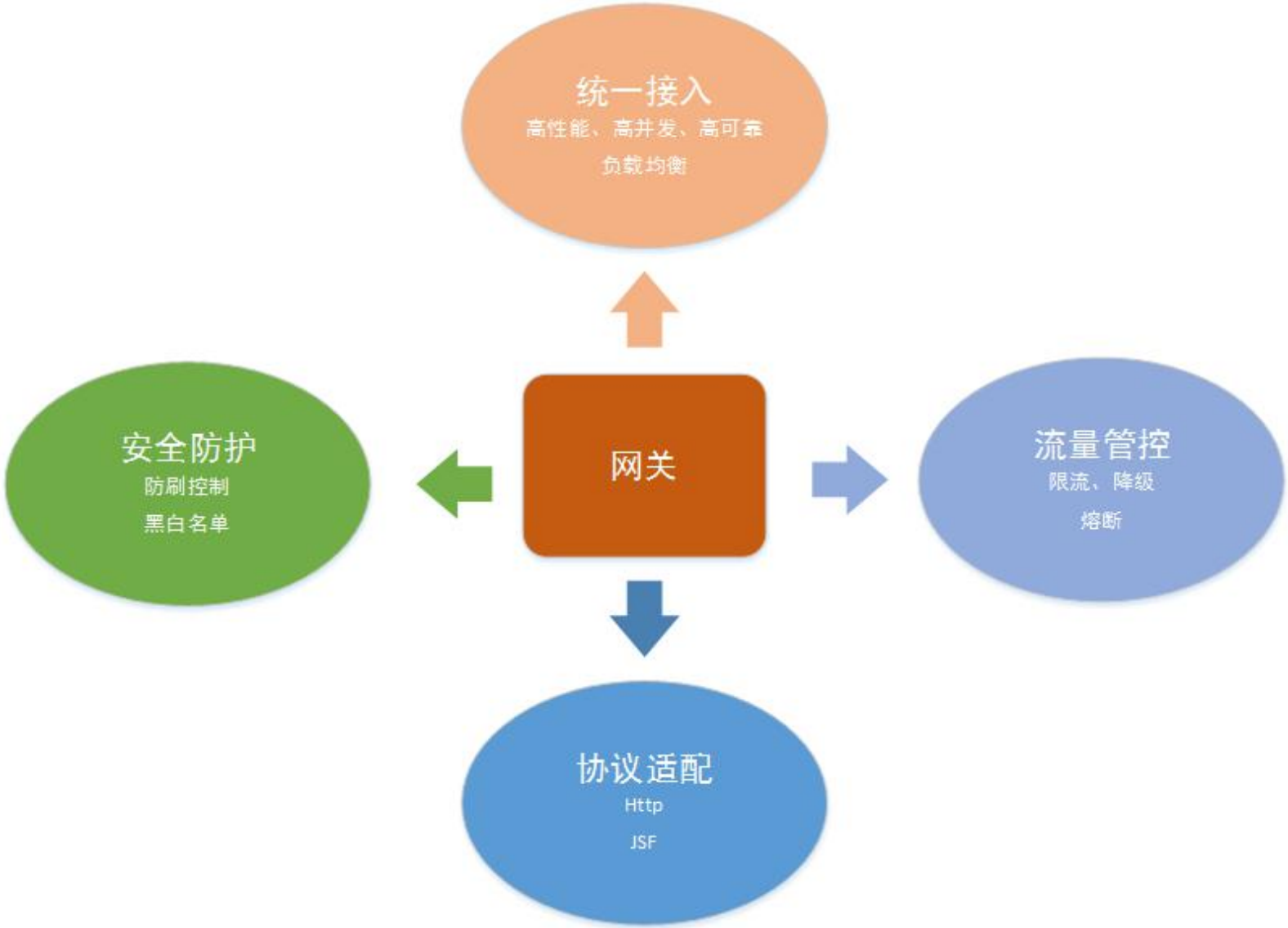
目前网关内有1000+个服务接口，要做到接口服务之间互不影响，架构之内杜绝蝴蝶效应，防止雪崩。起初理想很丰满奔着高可用的美好方向，

但过程中随着每个接口的服务性能不一致，某一个接口服务所依赖的外部资源出现延迟，此时恰好访问量突然飙高，导致线程堆积，致使整个网关服务不可用。

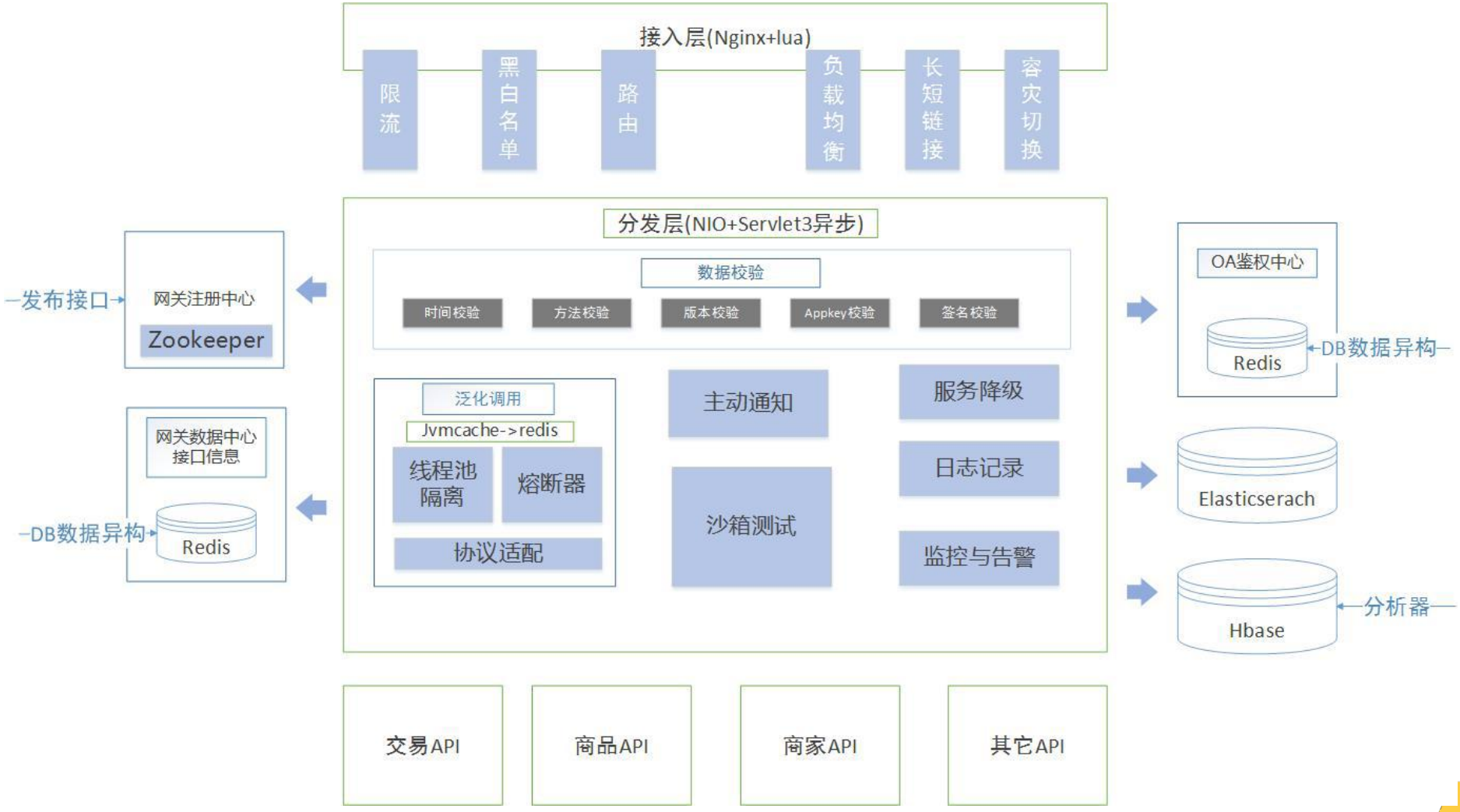
在这种情况下，我们开始逐步治理线程池的合理分配，另外增加熔断机制，保证单一依赖遇到突发问题可自行暂时切断访问，待一段指定的时间过后又可自动恢复。

使得整个网关实现自主治理。一定程度上具备了“自我治愈”的能力。

网关涵盖技术


统一接入
高性能、高并发、高可靠
负载均衡
安全防护
防刷控制
黑白名单
网关
流量管控
限流、降级
熔断
协议适配
Http
JSF

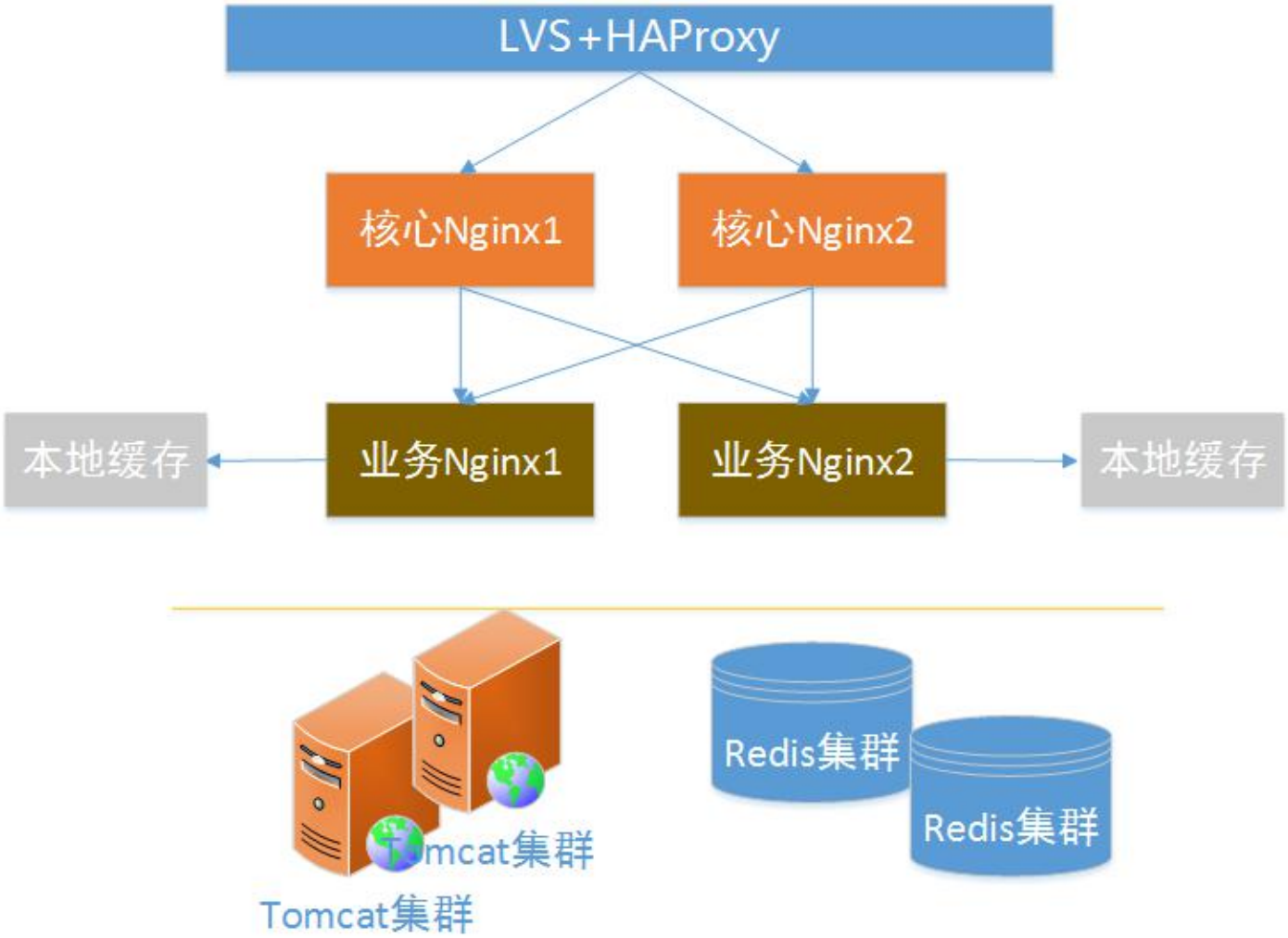
自研网关架构


接入层(Nginx+lua)
限流
黑白名单
路由
负载均衡
长短链接
容灾切换
分发层(NIO+Servlet3异步)
数据校验
时间校验
方法校验
版本校验
Appkey校验
签名校验
泛化调用
Jvmcache->redis
线程池隔离
熔断器
协议适配
主动通知
沙箱测试
服务降级
日志记录
监控与告警
发布接口
网关注册中心
Zookeeper
网关数据中心
接口信息
DB数据异构
Redis
OA鉴权中心
Redis
DB数据异构
Elasticserach
Hbase
分析器
交易API
商品API
商家API
其它API

基本思路或过程改进点

- Ngnix + Lua
- 引入NIO
- 分离之术
- 降级限流
- 熔断
- 缓存银弹
- 数据异构
- 快速失败
- 监控统计

实践 1 Nginx层统一接入

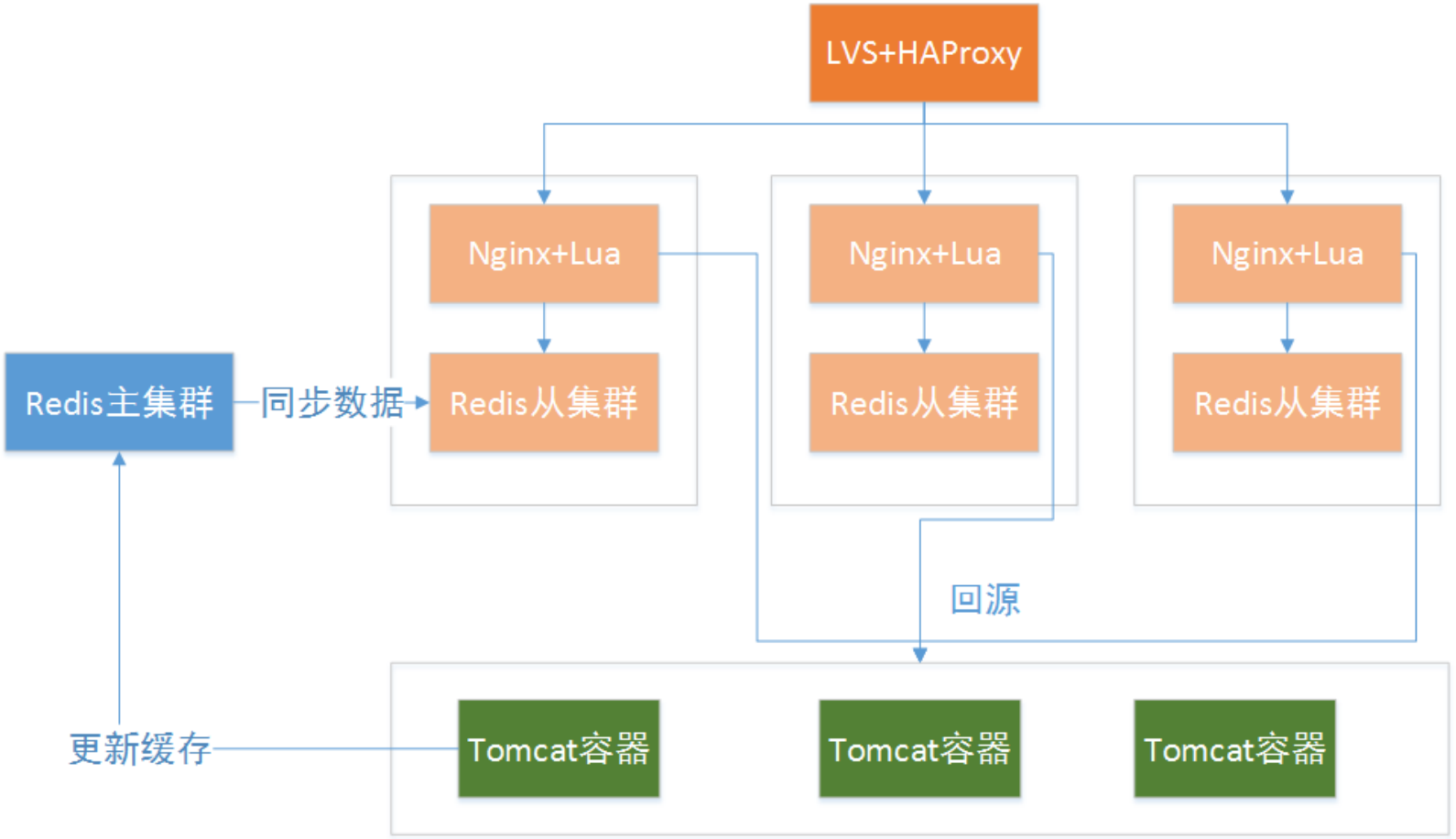


核心Nginx层确保是无状态的，在这一次实现流量切组，以及限流等功能

业务Nginx确保是无状态的，可以实现比如内容压缩需求，减轻核心Nginx的cpu压力

业务Nginx把请求直接转发给Tomcat这一层，当有的Tomcat出现问题可以在这一次摘掉

实践 1 Nginx层统一接入

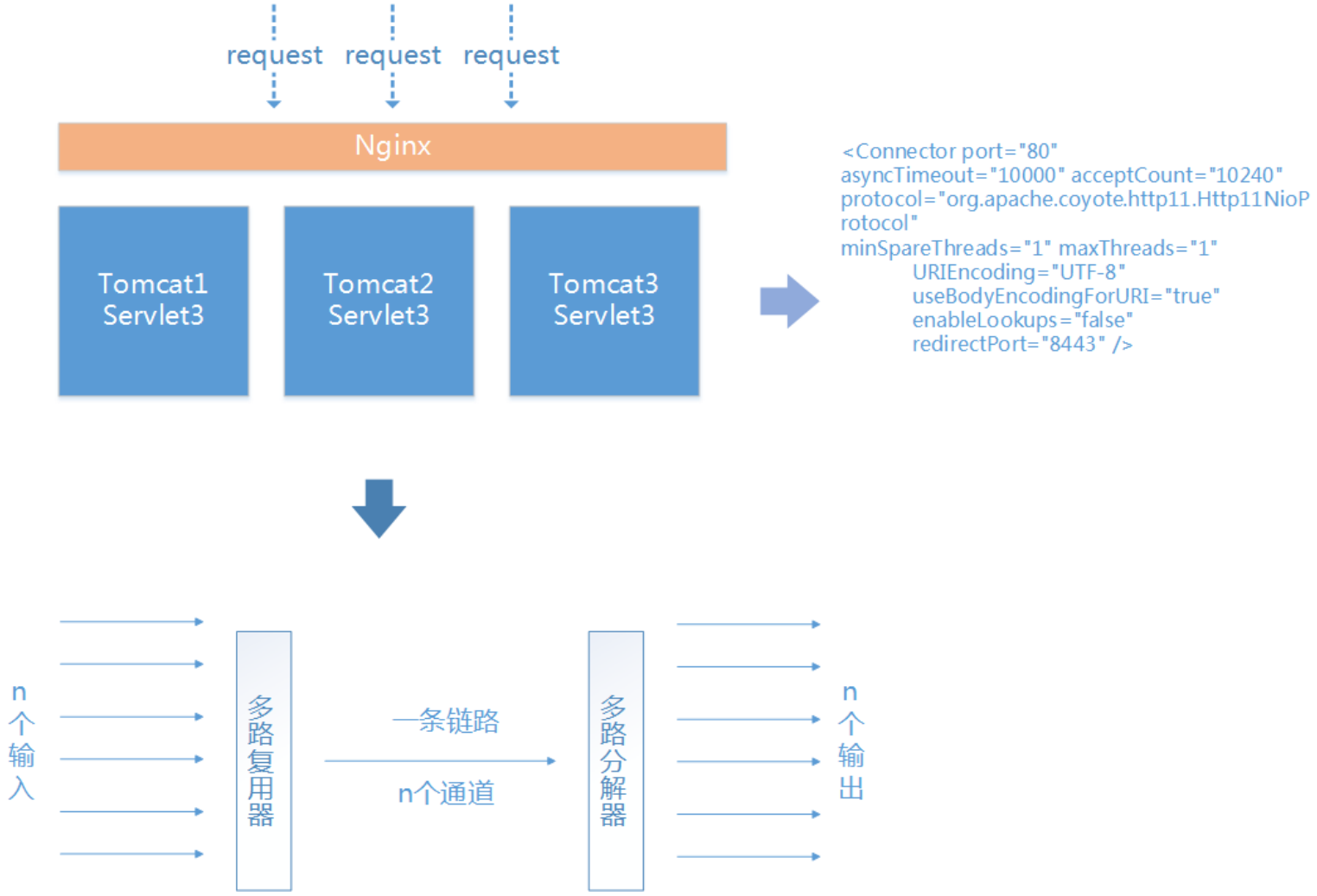


这里的redis从集群是于nginx层放在同一服务器，省去网络请求

数据首先读本机，如果没有数据，则回源到对应的tomcat应用，从其它数据源拉去数据(极少情况下才会走tomcat)

如果本机数据过期或者丢失都需worker进行数据推送更新

实践 2 引入NIO 利用Servlet3异步化



JDK7+TOMCAT7

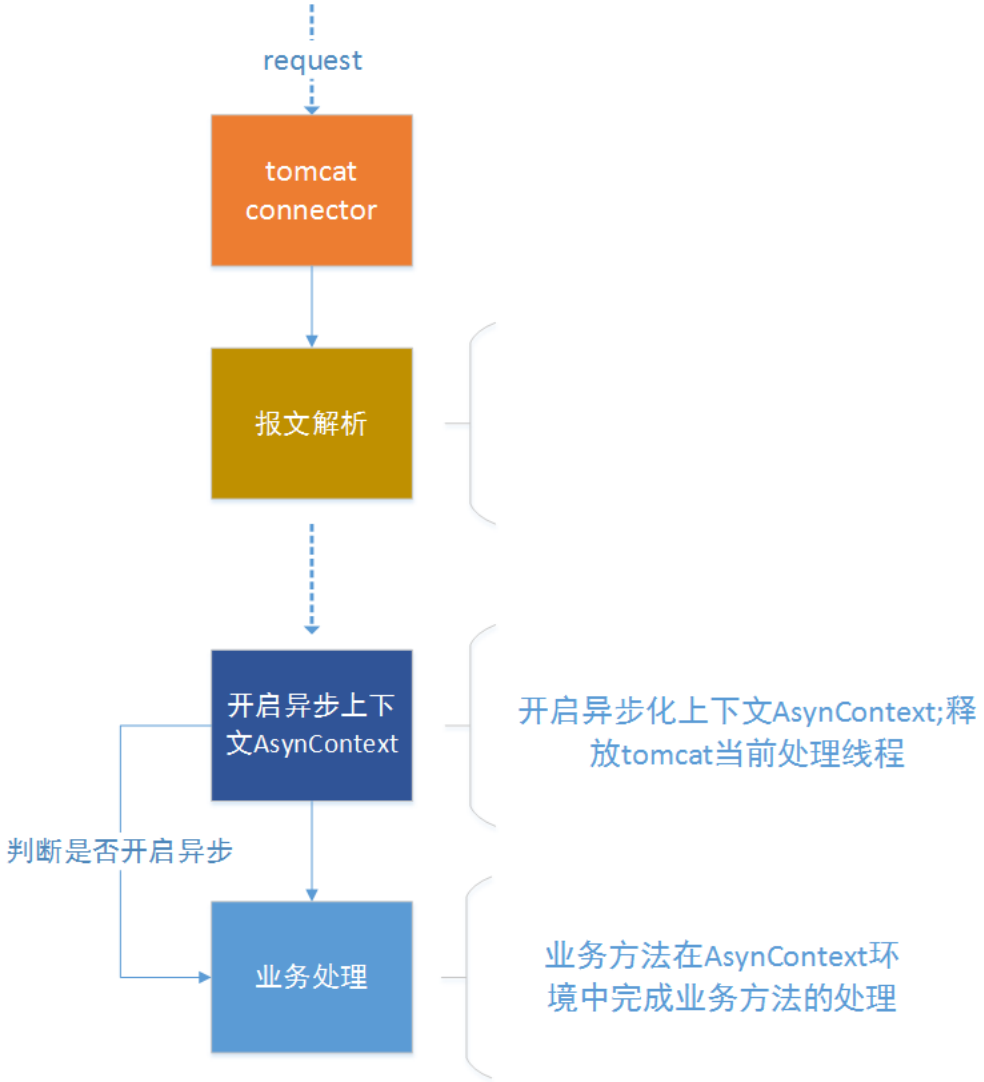
利用NIO的多路复用器处理更高的并发连接数

使用Servlet3让请求异步化

AsyncContext asyncContext =req.startAsync()

同时Servlet3还可以实现请求隔离，后面还会重点描述隔离技术

实践 2 引入NIO 利用Servlet3异步化

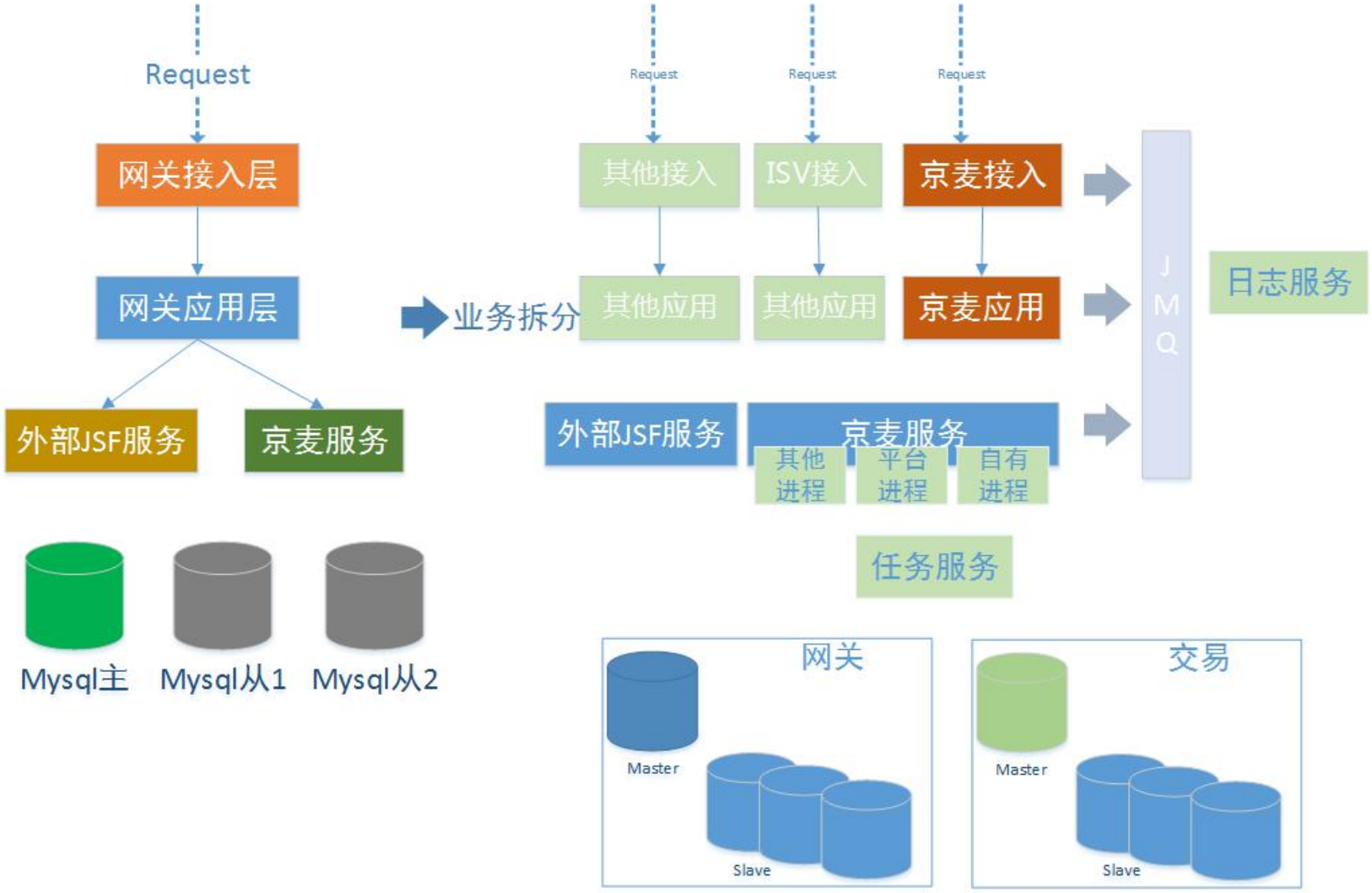


异步化之后可以提升吞吐量和灵活性，但相应时间会变长。

业务方法开启异步化上下文AsynContext;释放tomcat当前处理线程，从而提高其吞吐量

业务方法在AsynContext环境中完成业务方法的处理，调用其complete方法，将响应写回响应流

实践 3 分离之术-业务进程分离

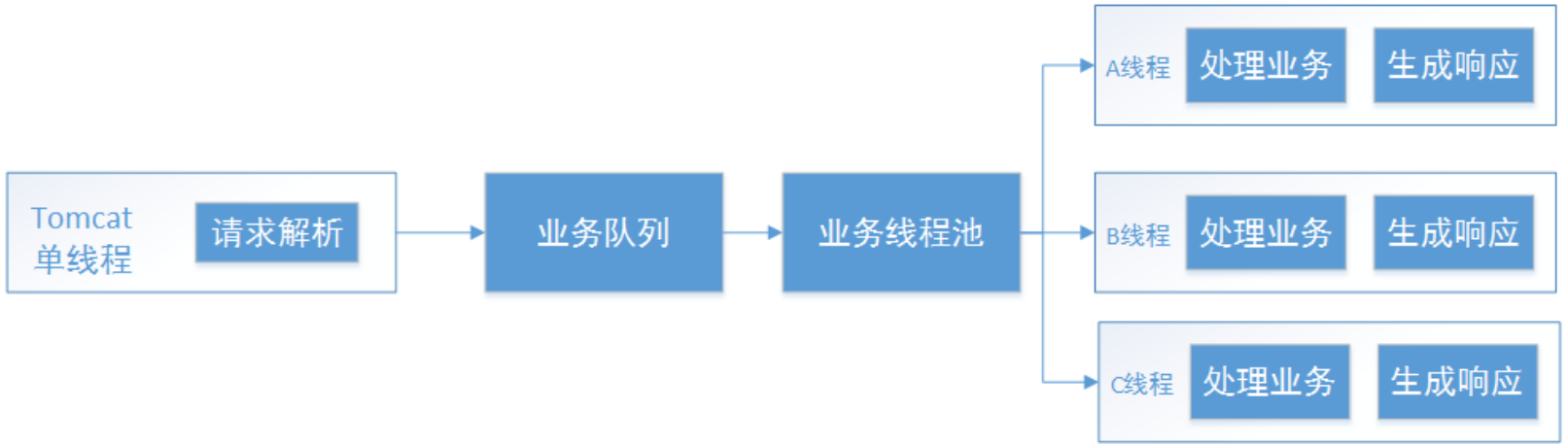


进行业务区分，不同的业务物理上隔离部署

在线/离线区分，比如离线统计、跑定时任务的统统隔离部署

生产/监控区分，比如监控统计类异步走MQ有专门的服务器和数据库来处理

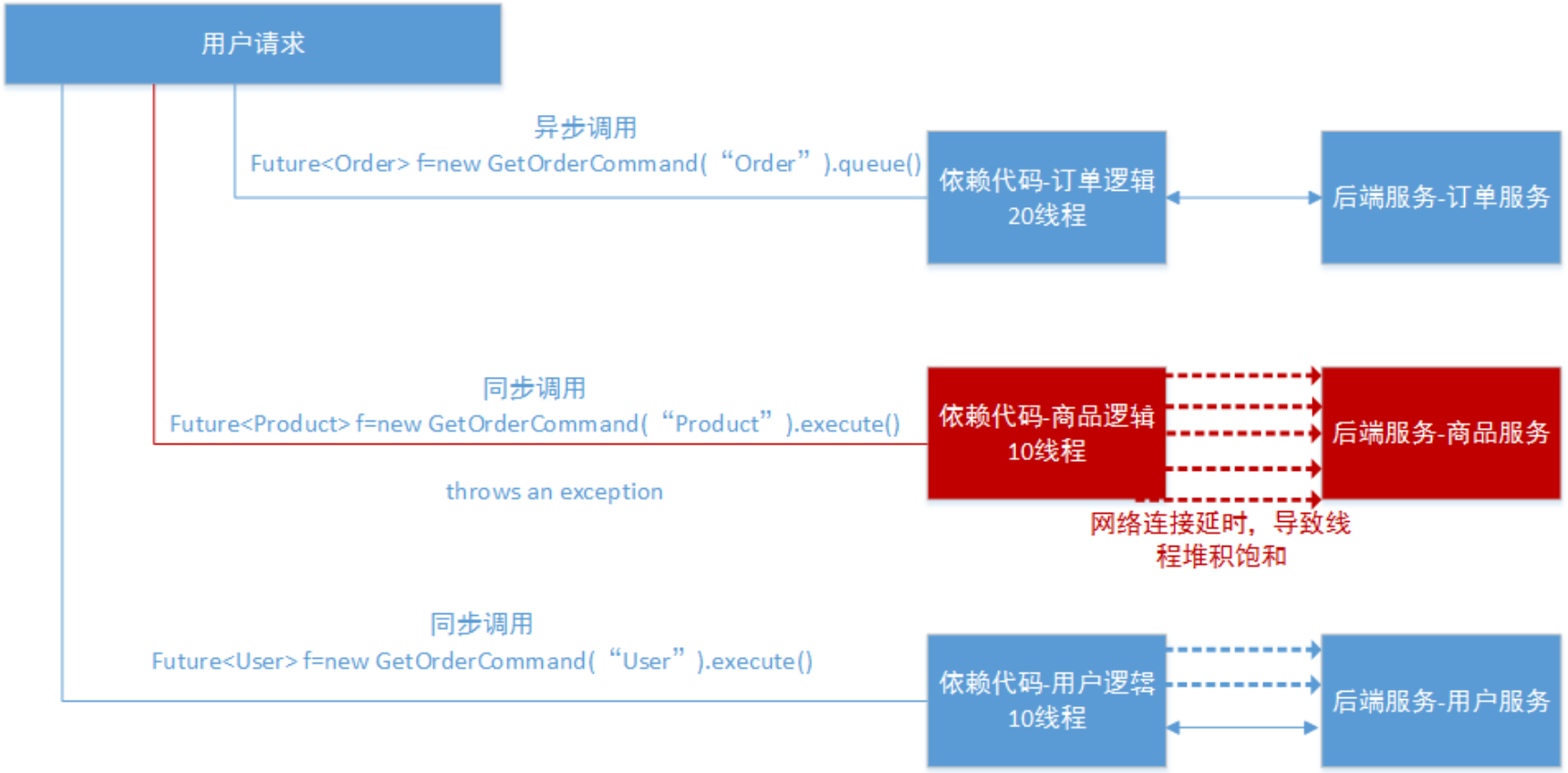
实践 3 分离之术-请求解析和业务处理分离



请求由Tomcat线程处理，在NIO模式下可以用非常少量线程处理大量链接情况

这里的业务线程池还可以进一步隔离，不同业务设置不同的线程池

实践 3 分离之术-业务线程池分离

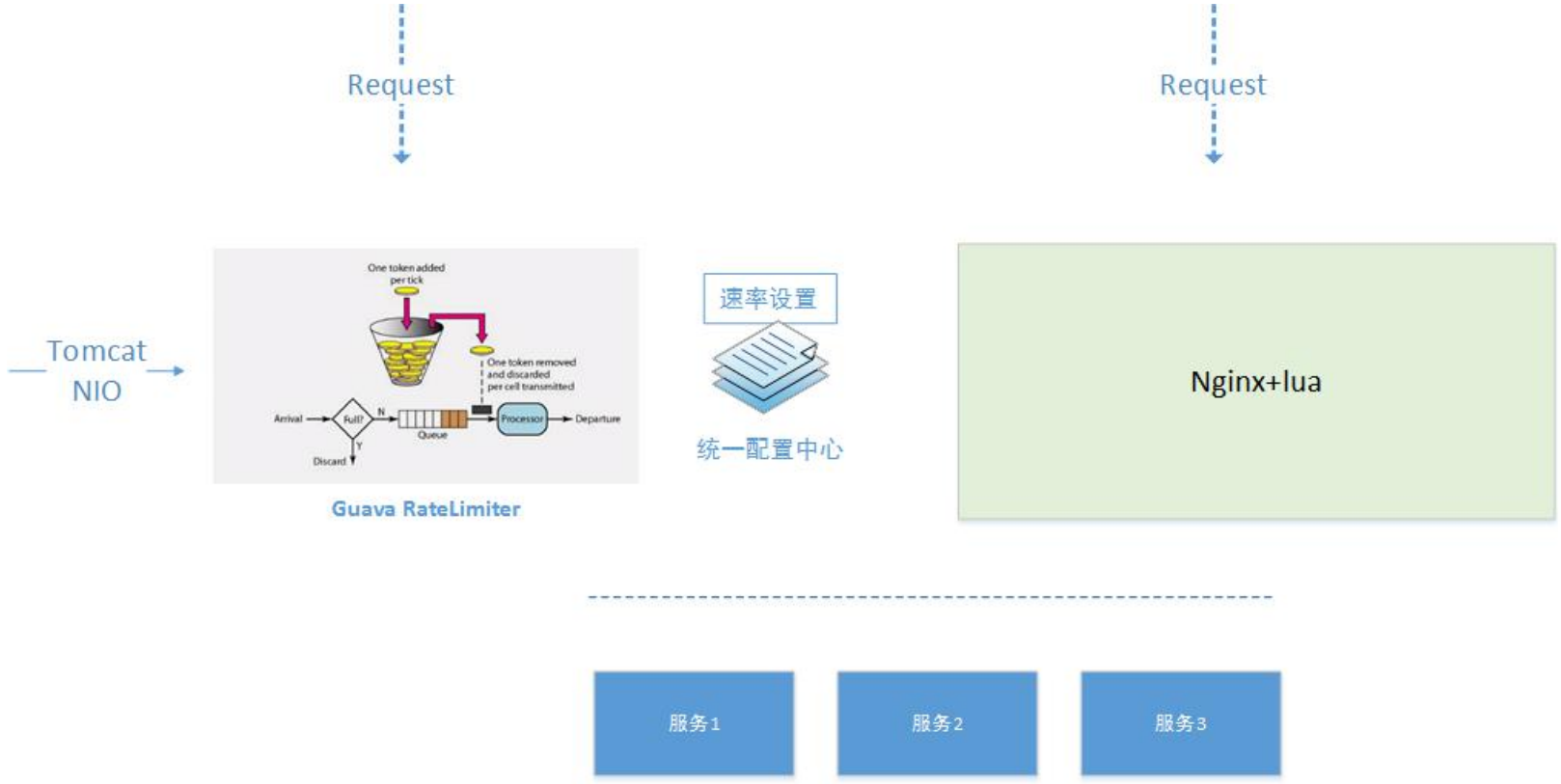


线程隔离互不影响，线程隔离可以采用线程池隔离和信号量隔离

线程池隔离是一种异步的方式，请求线程和业务逻辑线程是分开的

信号量隔离是在同一个线程下进行的，所以如果业务逻辑中含有RPC调用则不能使用这种方式

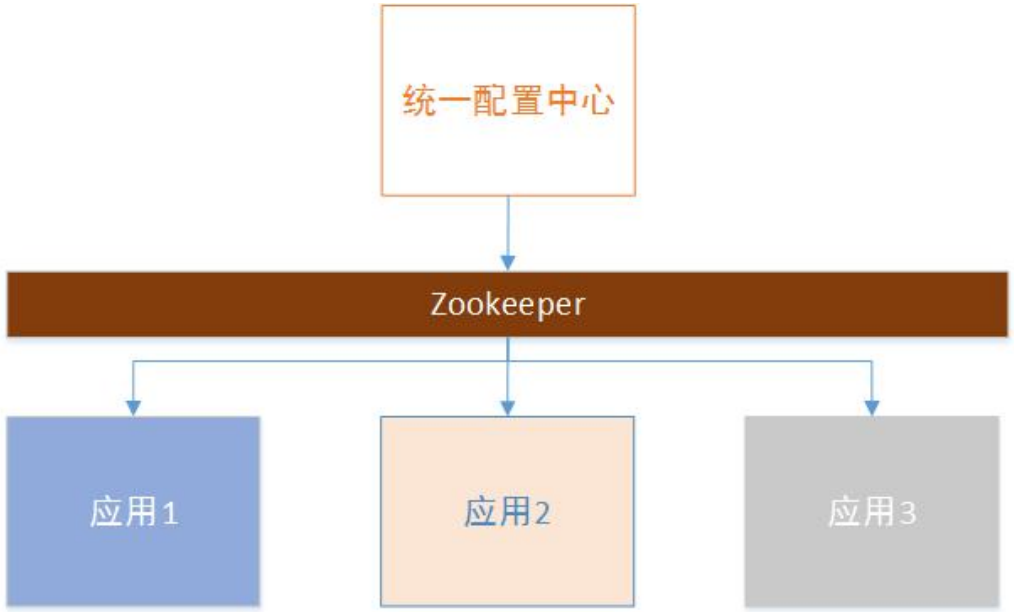
实践 4 限流

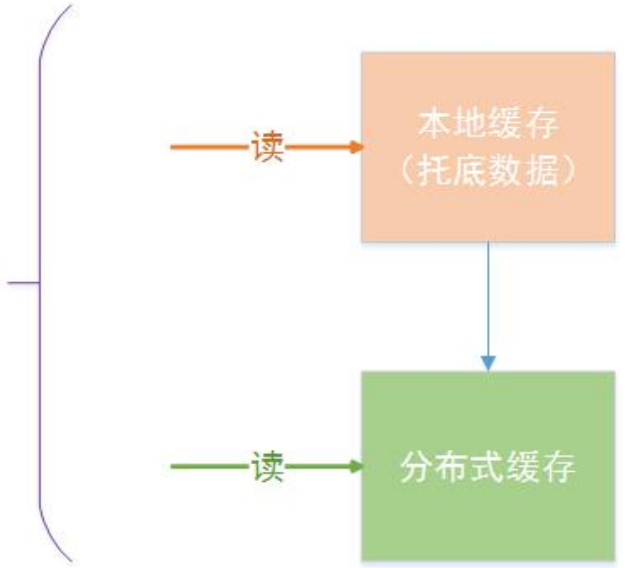


恶意请求、恶意攻击，恶意的请求流量可以只访问cache，恶意的IP可以在Nginx层进行屏蔽

防止流程超出系统的承载能力，虽然会预估但总有意外，如果没有限流，当超过系统承载峰值的时候，整个系统就会打垮

实践 5 降级



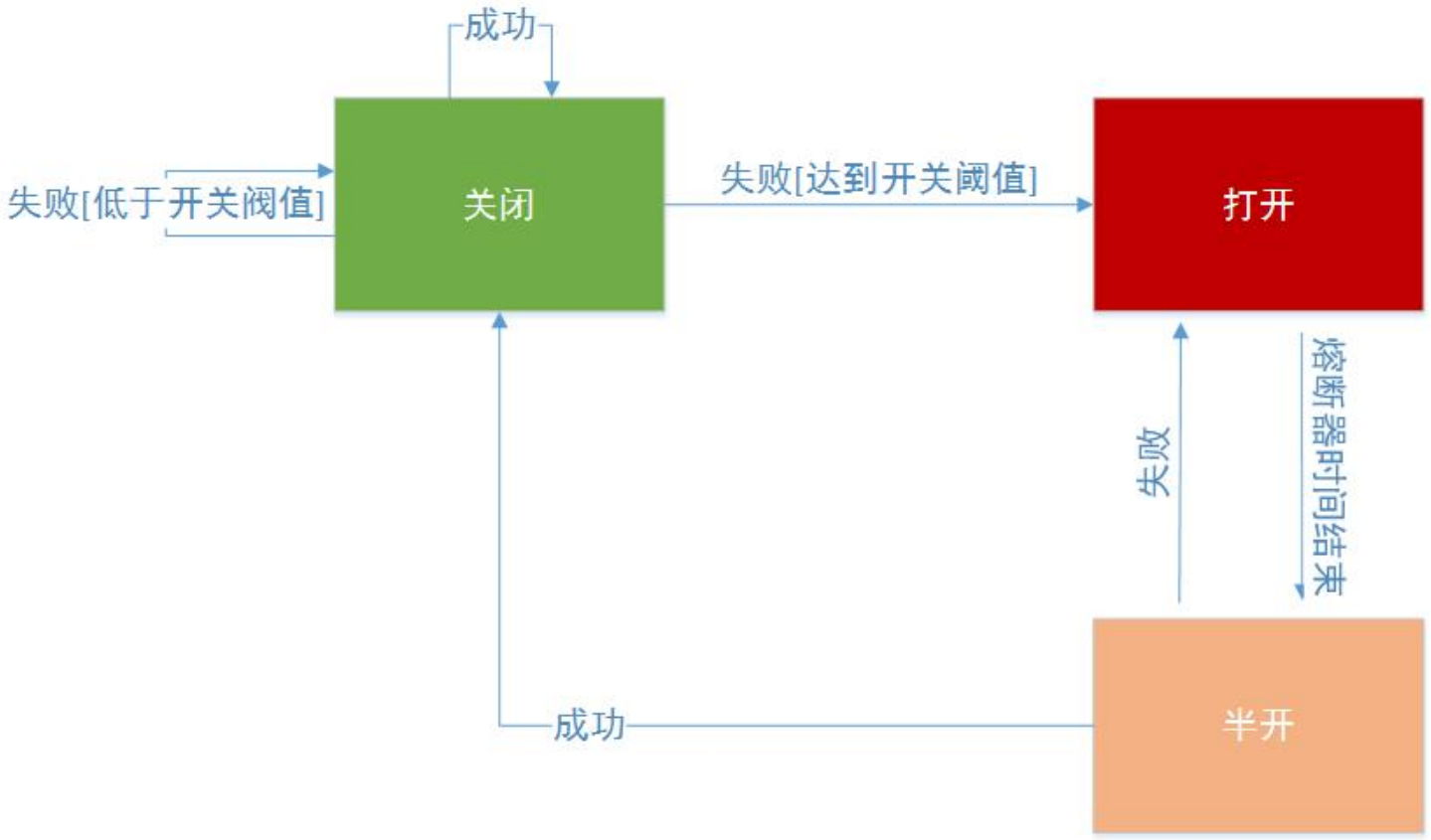


降级开关要集中化管理，比如通过zookeeper推送到各个应用服务

读服务要支持多级降级，比如降级只读本地缓存、只读分布式缓存、只读托底数据等等

开关最好前置到第一层，比如Nginx这一层，这样请求就不会达到Tomcat

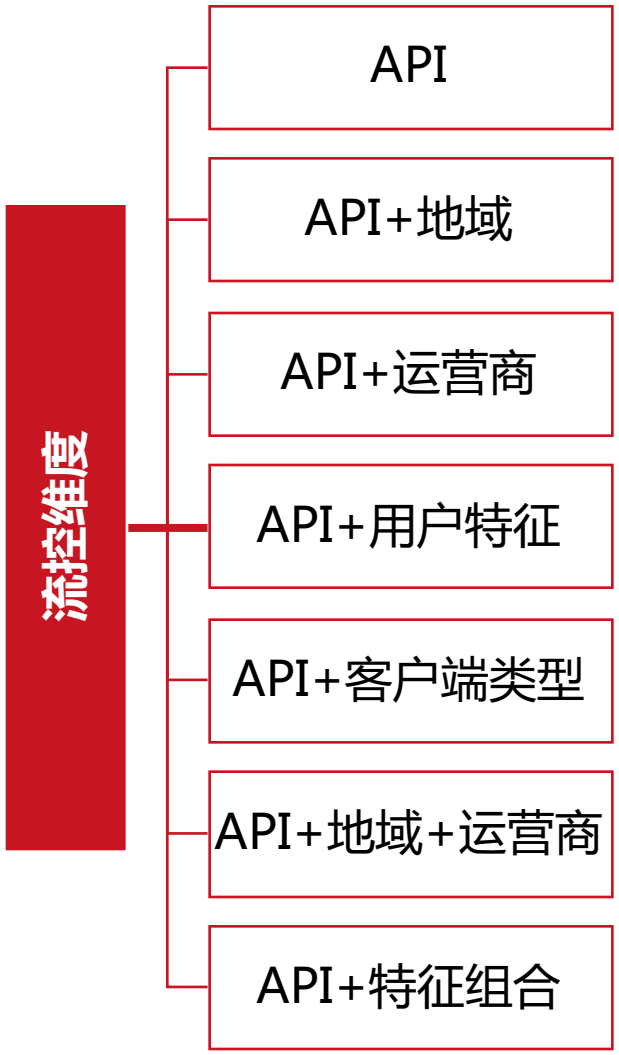
实践 6 熔断

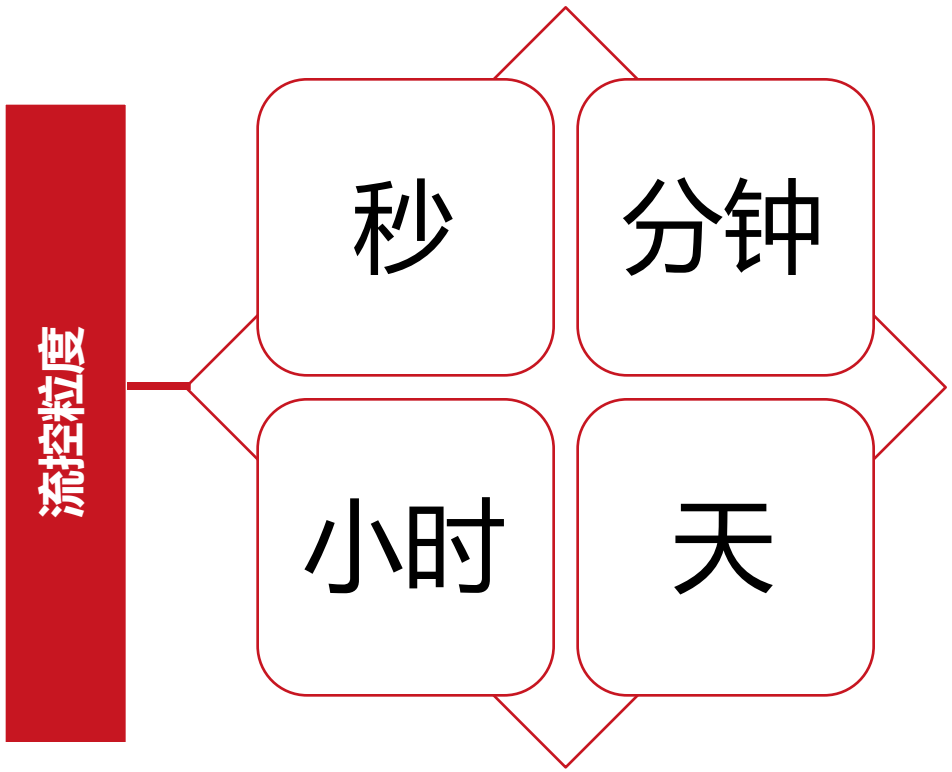


可以直接引入Hystrix来实现熔断，也可借助Hystrix的思想自主实现

当熔断处于半打开状态，不能一直熔断，需要设置一个时间窗口默认5s，达到这个窗口后进行重试，用来判断是否关闭熔断

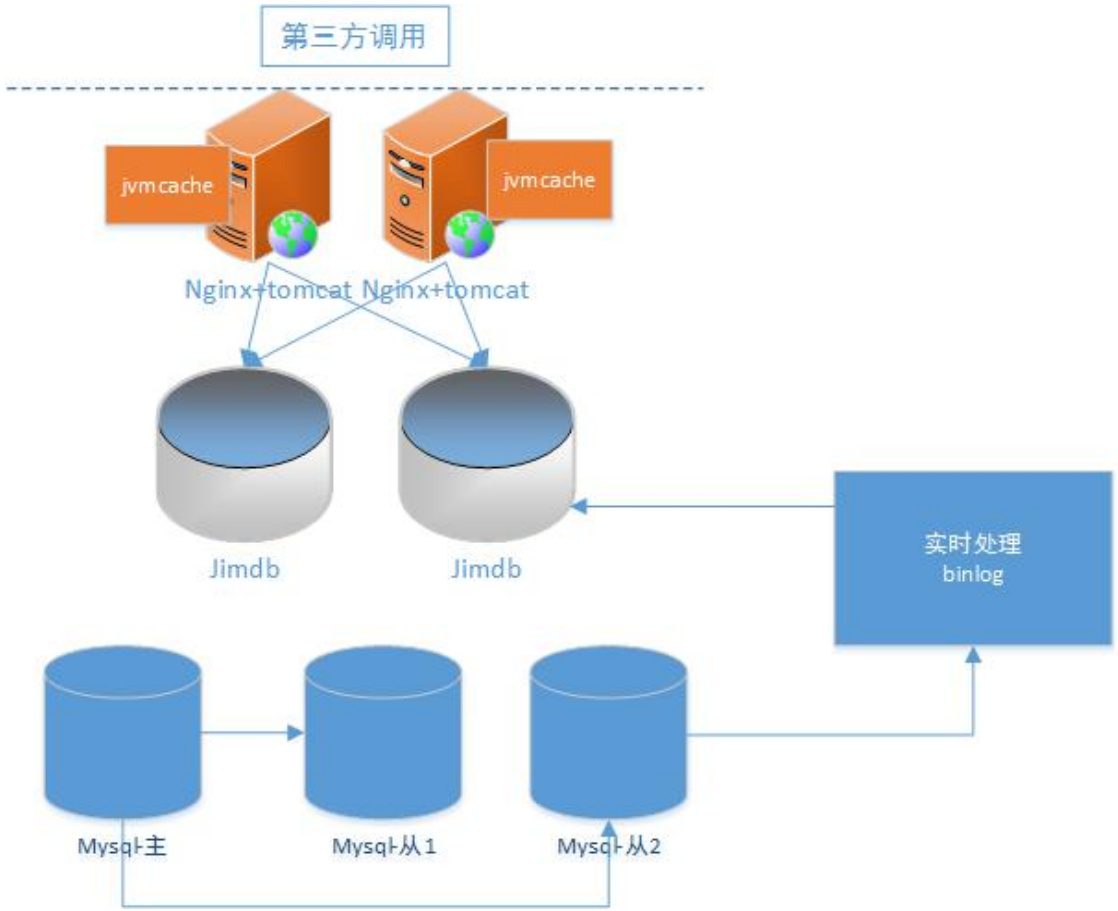
精细化流量控制





- 多个API
- 限流策略
- 分流策略

实践 7.2 持久化缓存



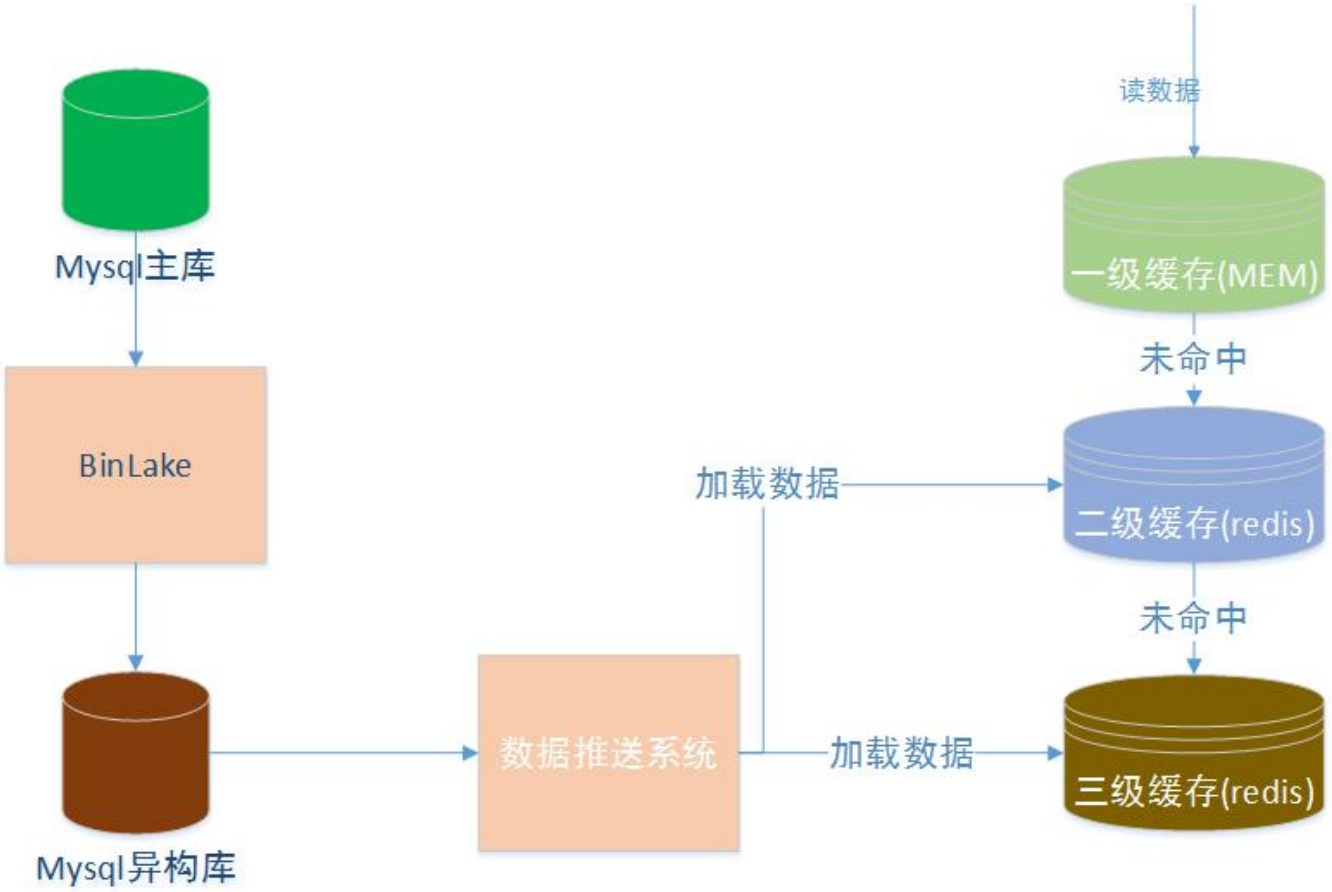
对外提供接口数据查询不能直接查库

引用jvmcache做第一层缓存

通过异构将数据持久化到缓存中永不过期

通过异构方法对缓存实时更新

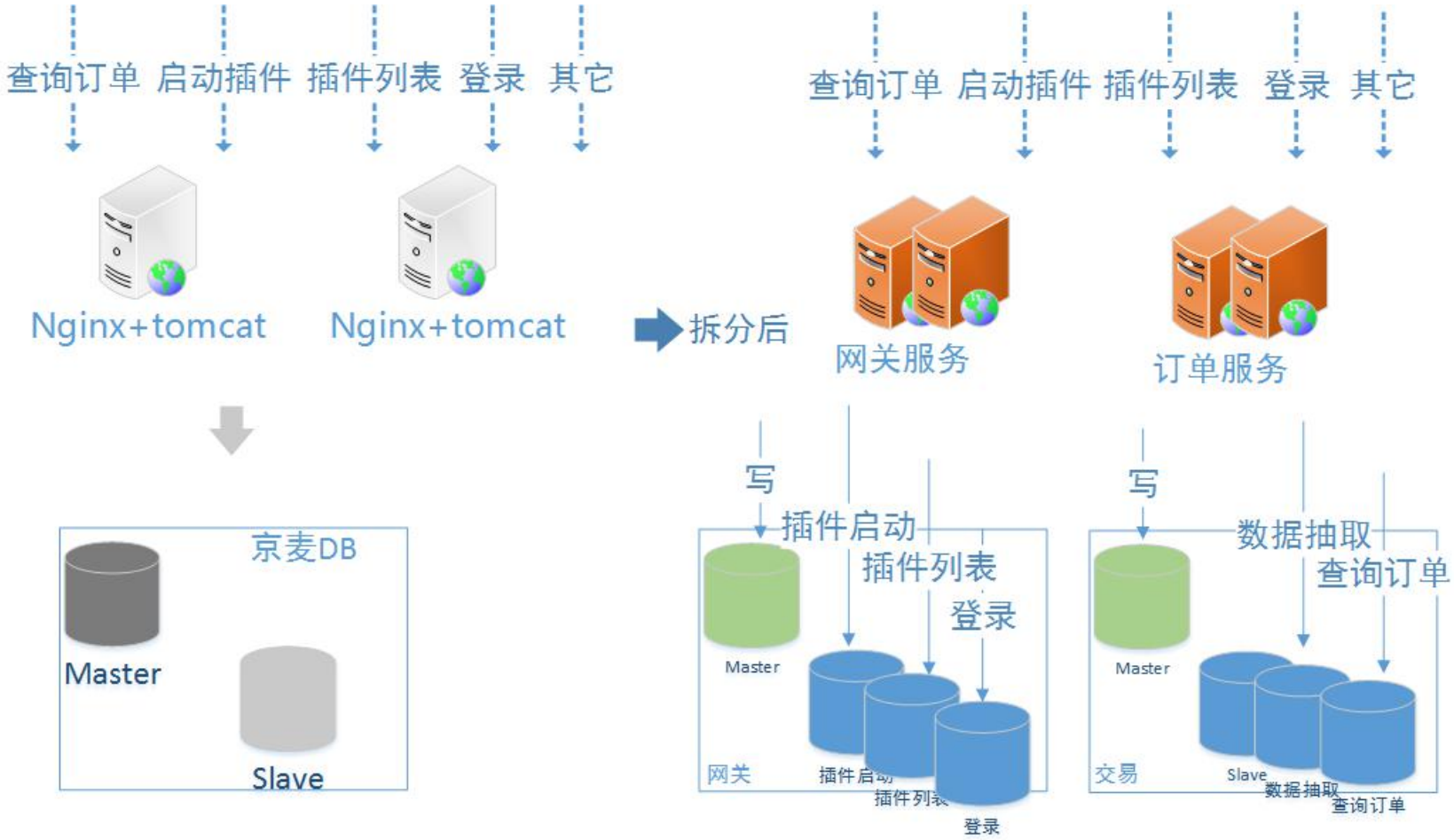
实践 7.3 多级缓存



一级缓存通过内存来实现，访问频度搞，数据体积小可以放入一级缓存，二级缓存在第一次为命中的前提下提供服务，三级缓存其实是一个用不过期的缓存

大value要进行压缩或者将大value拆分为多个小value

实践 7.1 数据库 – 按业务拆分 读写分离

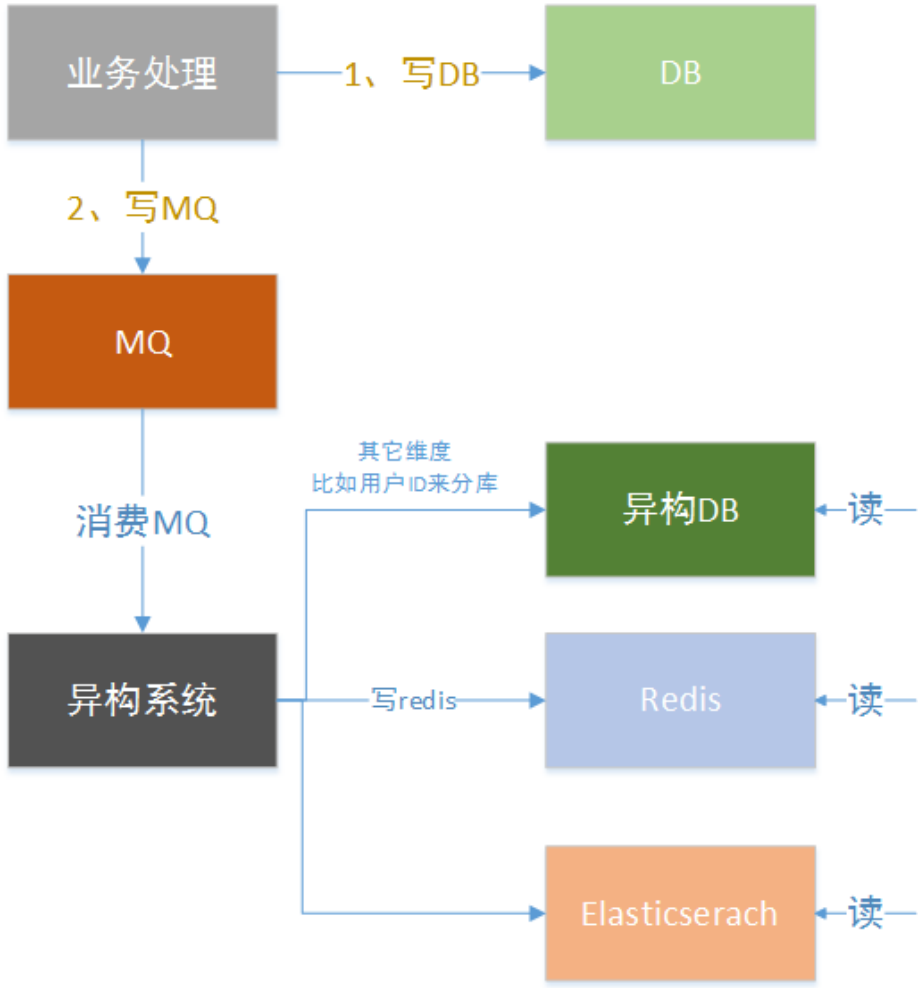


垂直拆分，按照业务类型分库目的是解决多个表之间的IO竞争、单机容量问题

分库分表的策略一般有取模、分区两种方法

分库分表后跨表查询的时候，要考虑join是否支持，另外按照不同维度来查询的时候如果不好处理可以借助异构功能，重新构建到其它维度的库或者elasticserach中

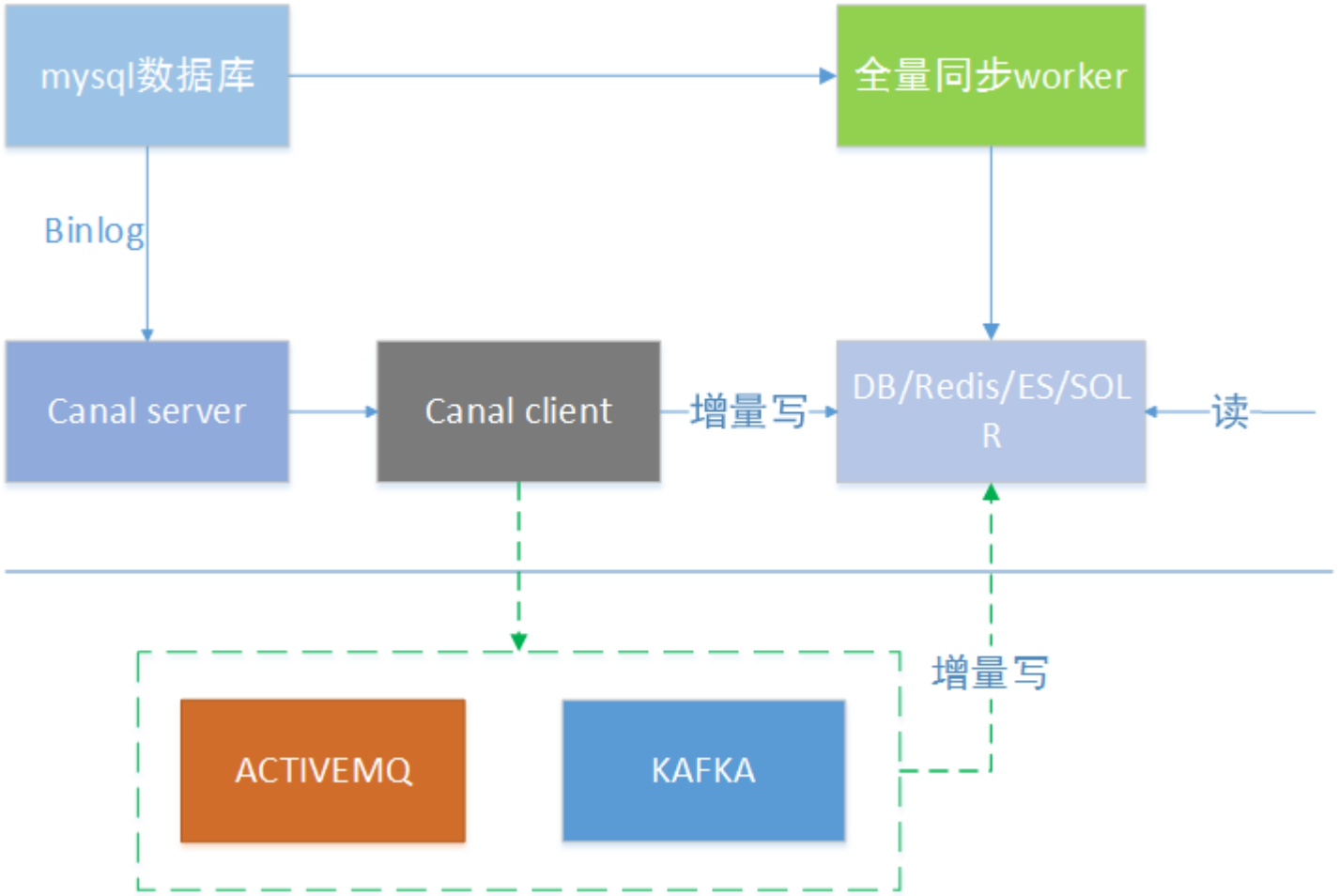
实践 8 数据异构-mq



Mq的方式的缺点是不能保证数据的一致性，但也一定不能强求把MQ的调用放在DB的事务中。

Mq的方式简单，适合于对一致性要求不高的场景

实践 8 数据异构-binlog



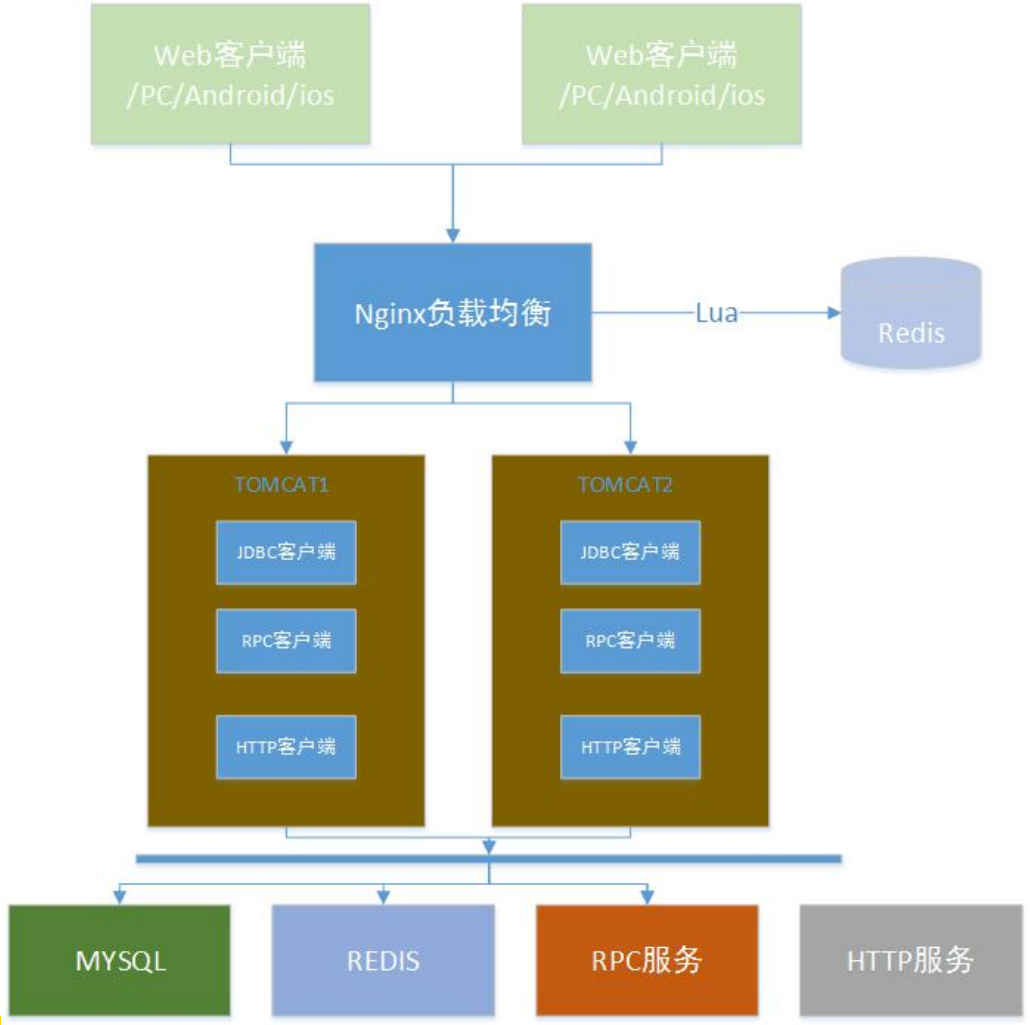
必须修改binglog为row模式

如果需要多消费客户端可以先写入ActiveMQ或者kafka

增量更新出现问题时候要保存消费位点历史快照确保回退机制

业界开源的是canal，京东内部使用的是自研的binlake

实践 9 快速失败-链路中的超时

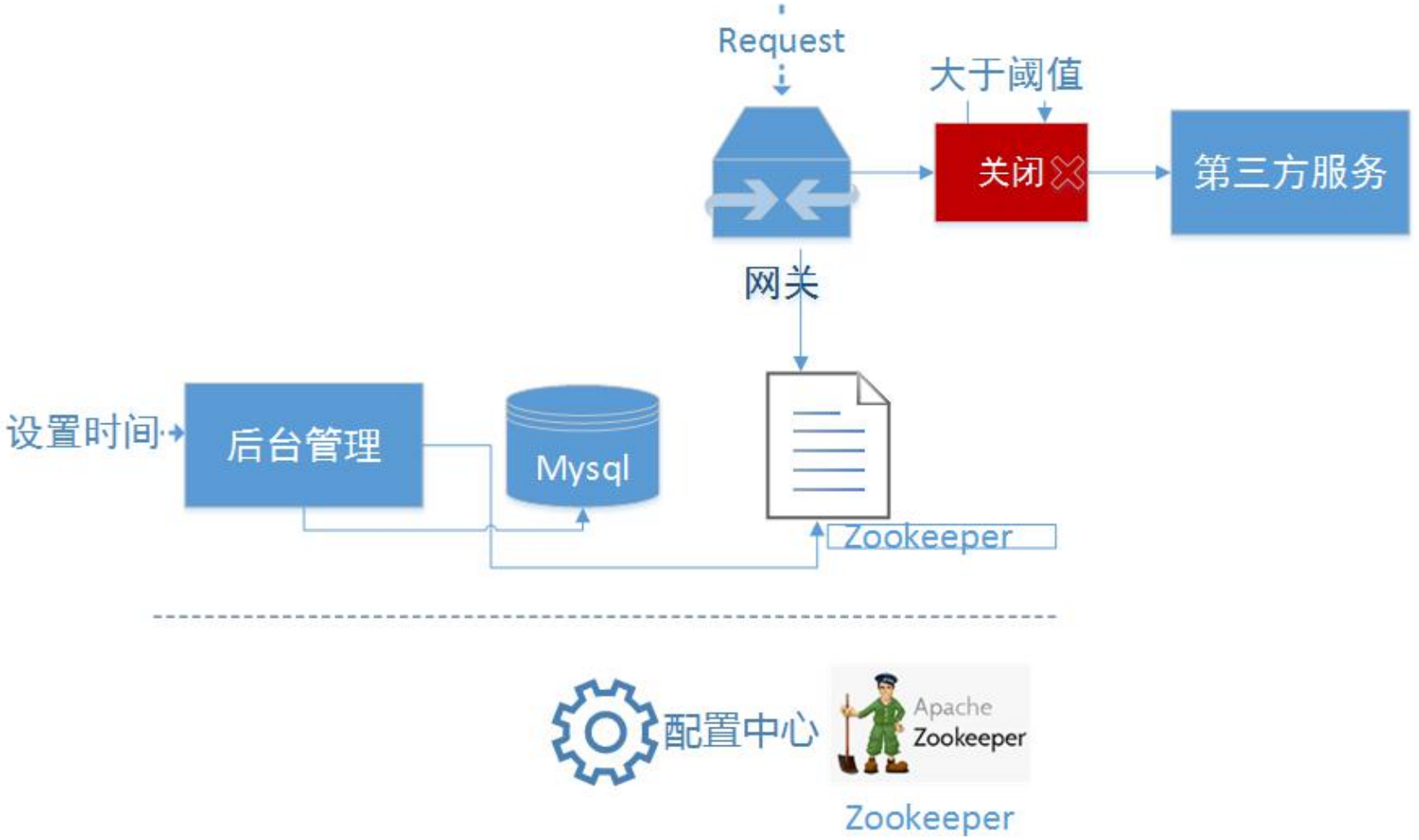


超时设置对一个分布式系统非常重要，没有设置超时，请求响应慢的情况下可能会累积导致连锁反应，甚至雪崩

左图是一个超时的应用链，web超时、RPC客户端超时、JDBC超时、还有业务超时等

最重要的超时是网络相关的超时

实践 9 快速失败-超时配置



通过配置中心对每个接口超时时间可动态设置，从而同步到每个应用机器

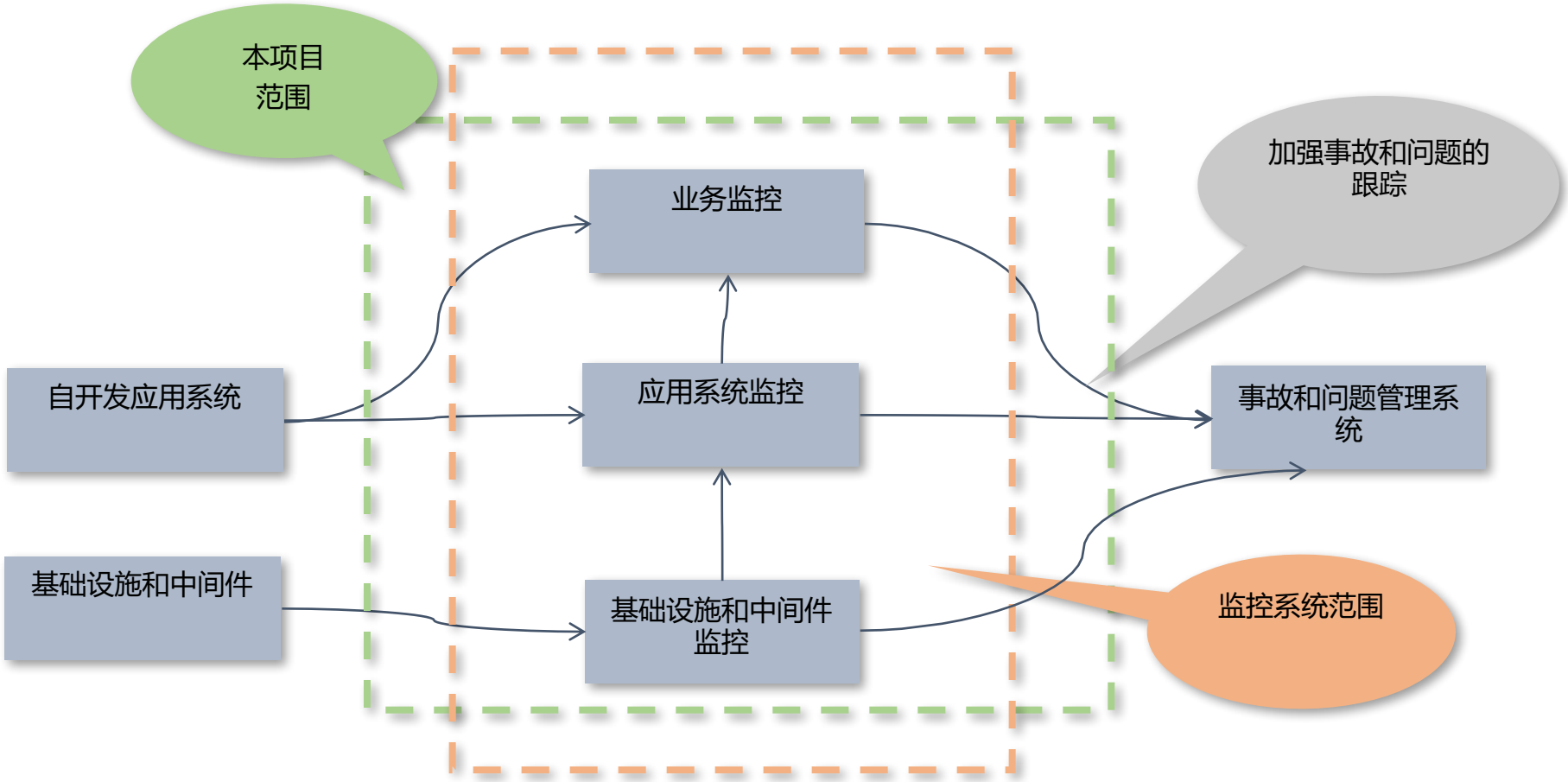
这里可以配合熔断机制、重试机制一起使用

在每次code review的时候资源的超时检查是最重要的一环，一定不要遗漏

实践 10 监控统计UMP - 监控目标

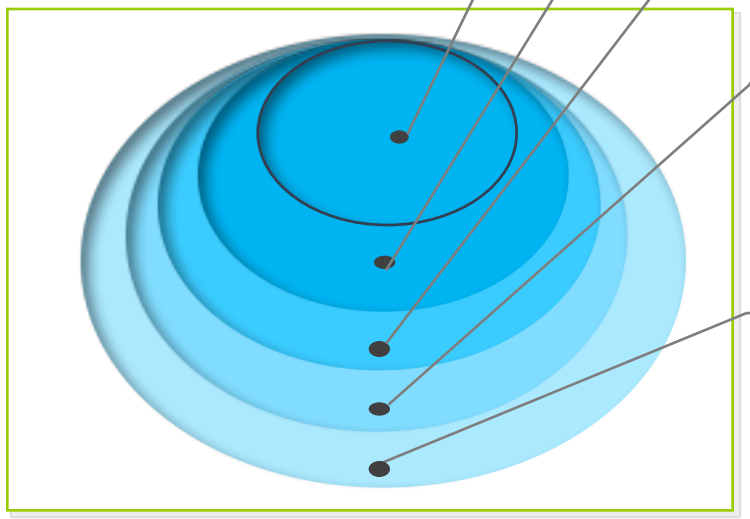
1. 24小时守护系统
2. 监视运行状态，实时控制
3. 统计数据，分析指标
4. 实时报警

实践 10 监控统计UMP - 监控范围



实践 10 监控统计 - 监控点

监控点组成



5 硬件监控：CPU、内存、磁盘、网络流量、 Swap等

4 自定义监控：直接报警

3 性能监控：TP999 TP99 TP90 TP50四种性 能指标作为SLA的参考标准、平 均值，最大、最小值

2 心跳监控：系统心跳、方法心跳

1 存活监控：URL存活性检测、端口检测

实践 10 监控统计UMP – 网关应用层自动接入

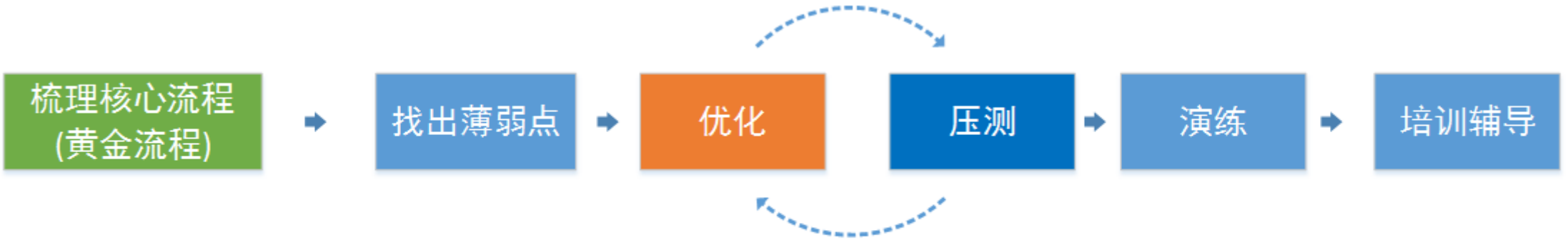
通过aop方式将mysql、es、hbase的操作统一规范化umpkey

执行Python脚本设置报警参数

按照业务特点合理设置报警阈值

要求：要早于用户发现问题！

终极输出：备战和优化经验



- 登录/扫码、免登、账号密码
- 交易下单
- 打单、出库
- 消息推送/价格门
- ...

- 应用服务器CPU/负载
- 慢sql
- UMP性能监控
- 系统耦合
- ...

- TP99
- Jvm
- Cpu load
- QPS/TPS
- ...

- 限流
- 降级
- 实战模拟
- ...

- 心理辅导
- 以往问题经验总结
- 工具使用
- ...

案例启示

- 面对突发流量和事故，采用降级、熔断和流控的手段可以有效防范，保护好系统。
- 话说天下大事“合久必分，分久必合”但在软件架构上，一直是一个分的趋势。通过分离、隔离技术提高可用性。
- 如果在抗量，大访问量方面有一种银弹的话，那就是缓存。

微信官方公众号：壹佰案例
关注查看更多往届实践案例

TOP1
技术型企业案例研究智库

100

TOP 100 CASE STUDIES OF THE YEAR

全球软件案例研究峰会