电商平台海量服务架构探索

蘑菇街七公

蘑菇街
中国最大的女性时尚社交电商平台

提纲

01 蘑菇街初期架构及15年初面临的紧迫问题

02 系统拆分&服务化历程

03 购买链路大促流量洪峰的挑战及服务化架构应对

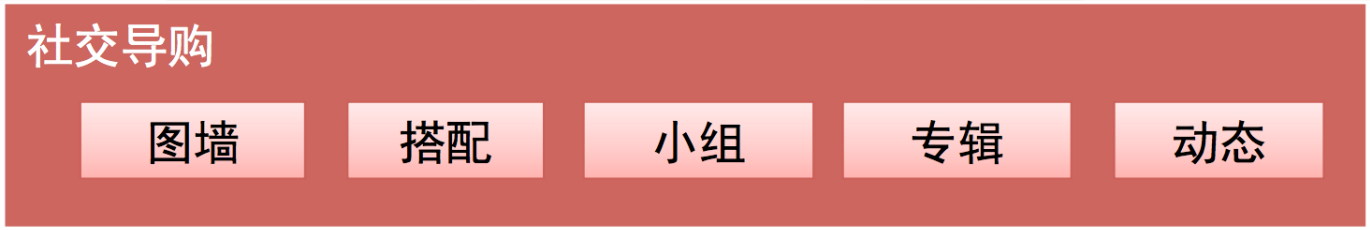
04 可用性的挑战与服务SLA保障

05 全局的蘑菇街服务架构

06 总结及下一步展望

蘑菇街导购时期的电商系统 业务架构

前台业务：



后台系统：

基础应用：

蘑菇街导购时期的电商系统 系统架构

70%

30%

代理层

Nginx Proxy

WEB层

Nginx+PHP-FPM

Nginx+PHP-FPM

基础设施层

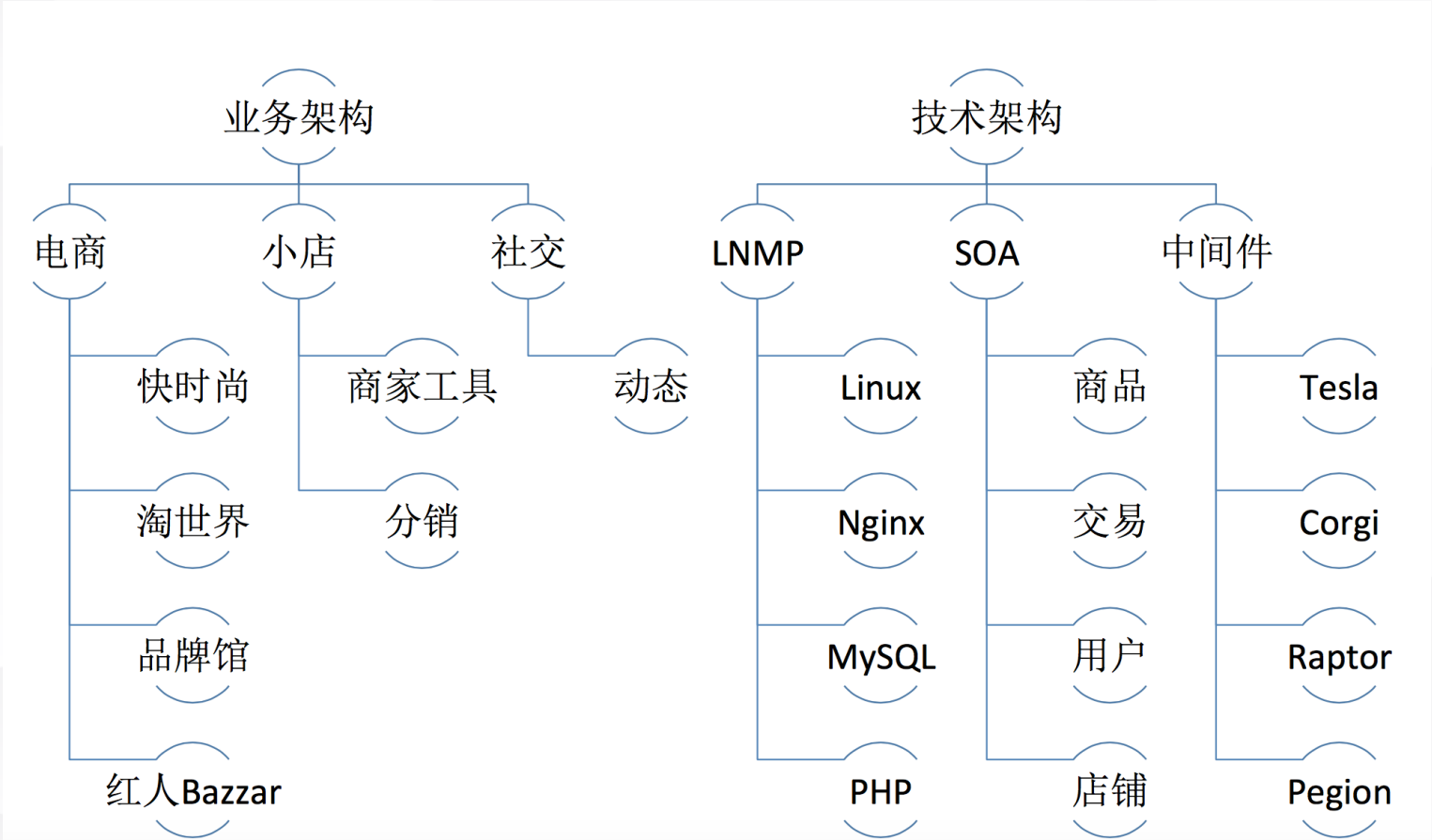
MySQL

Master → Slaver

Redis

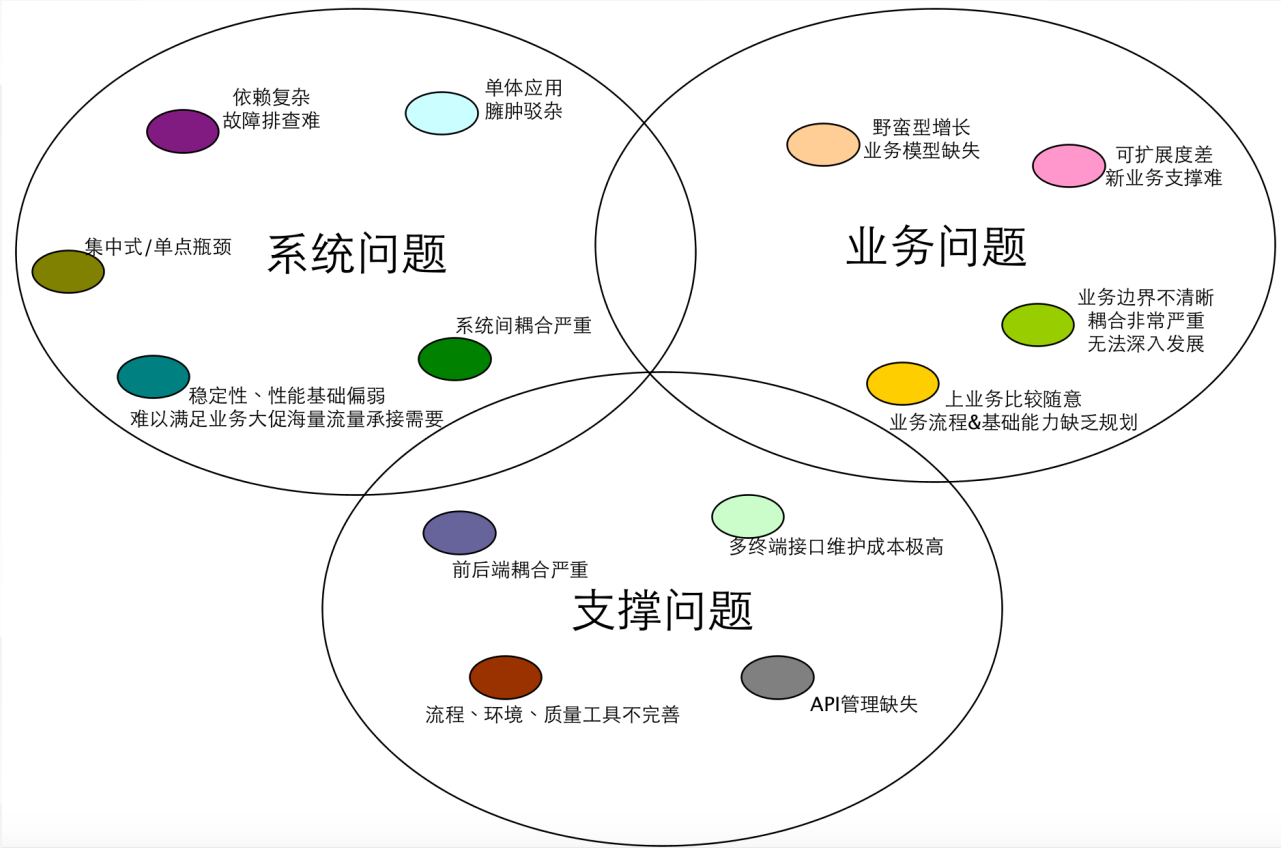
Lucene

蘑菇街转型初期的电商系统



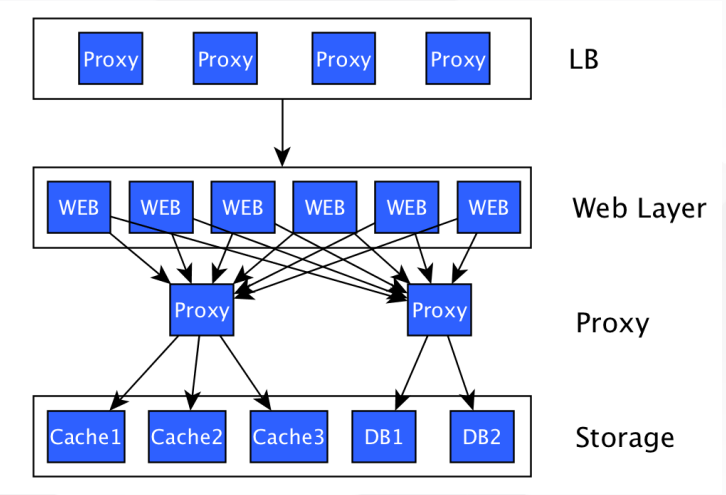
电商系统发展中期面临的一般性问题

中期界定：用户破千万、PV过亿



蘑菇街2015年初面临的紧迫问题

蘑菇街2015：用户过亿、PV过10亿

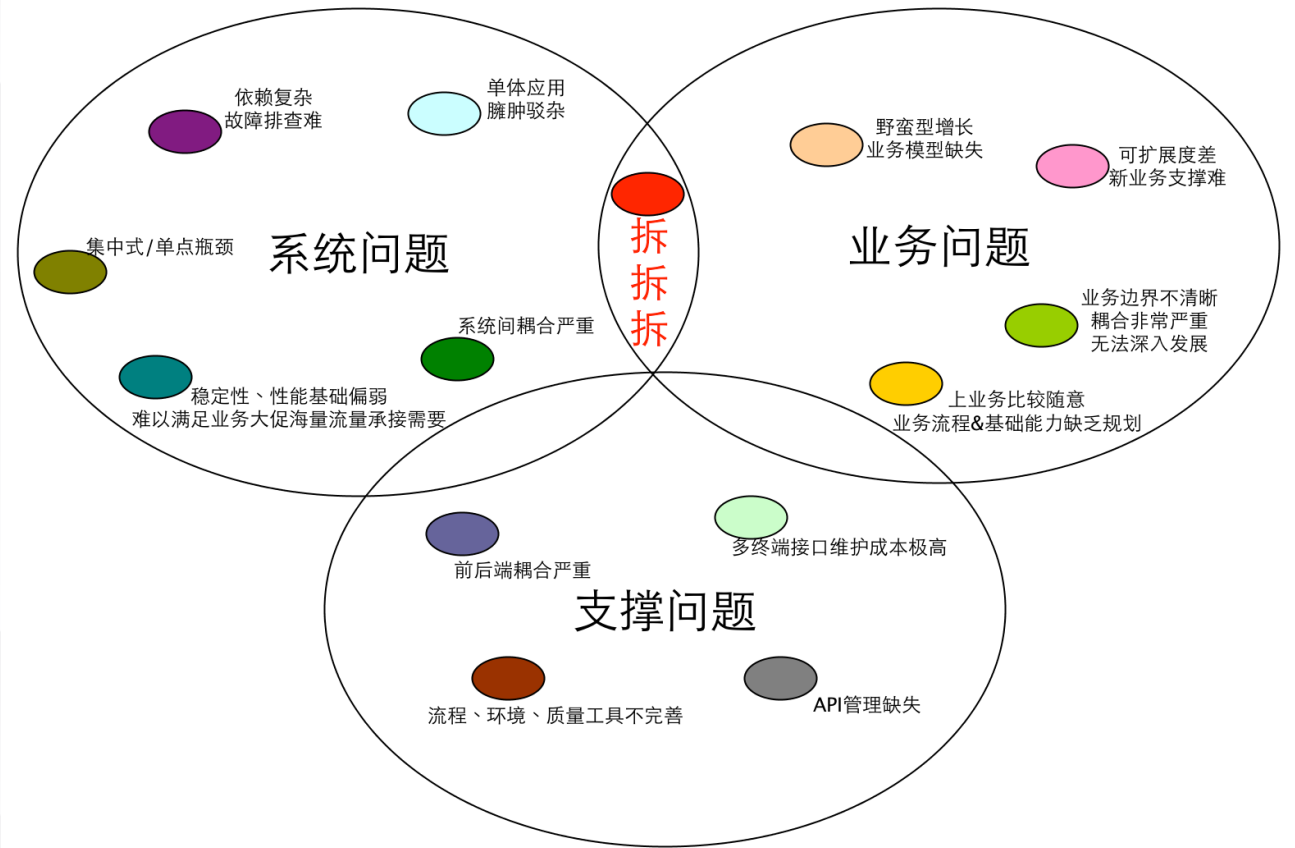


- 业务在超高速发展，每年保持3倍以上的增长
- 购买链路大促峰值流量是日常的数百倍
- 15年初状况是只能支持400单/s的交易创建
- 历史包袱沉重、系统耦合非常严重
- 业务异常复杂，且电商商品、库存、促销、交易、支付等业务形态都在快速膨胀

亟需在支持业务快速发展的同时，做好系统改造和容量&性能提升工作！

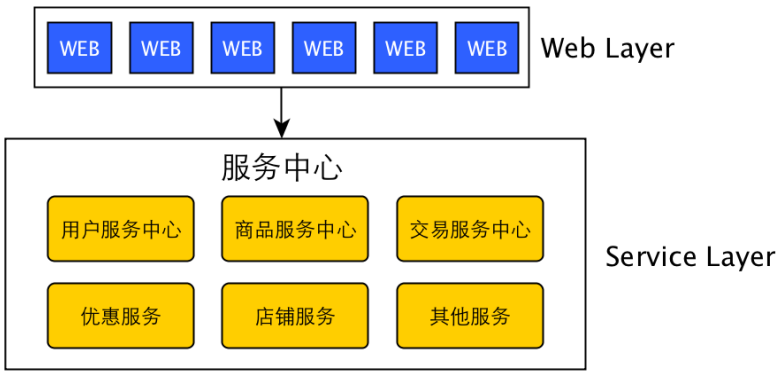
电商系统发展中期面临的一般性问题

破解之道：系统拆分&服务化



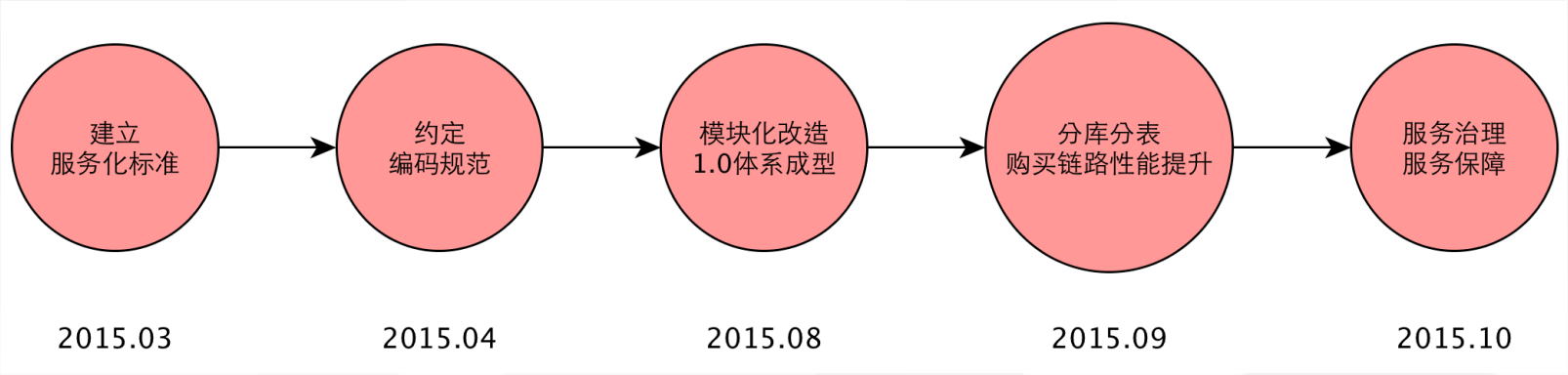
蘑菇街系统拆分&服务化历程

系统拆分&服务化 如何做?

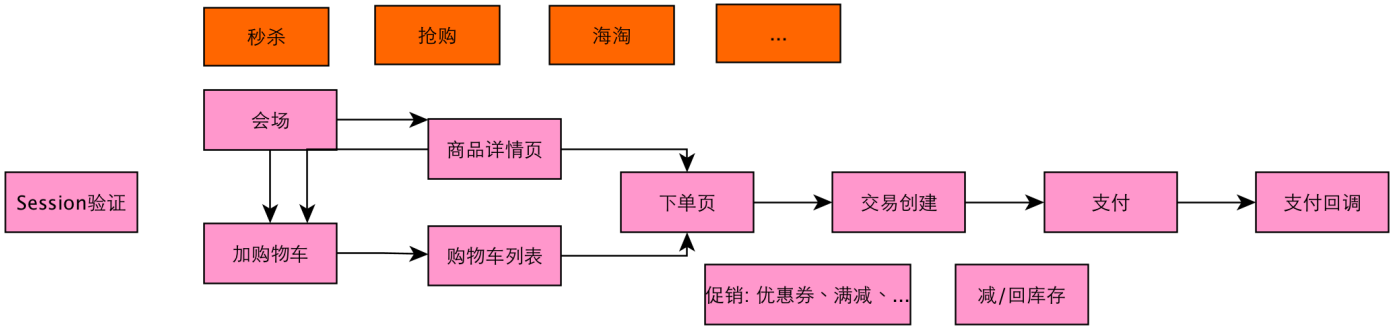


- DB垂直拆分
- 业务系统垂直拆分：交易、资金等等
- 数据建模&业务建模，确定逻辑清晰、粒度合适的服务API
- 基础业务逻辑下沉到服务，web层专注展示逻辑和编排
- 基于SOA中间件建设服务化系统
- 服务治理

蘑菇街系统拆分&服务化历程



购买链路大促流量洪峰的挑战及服务化架构应对



- 按业务域对数据库进行垂直拆分
- 读写分离，读可任意扩展
- 分库分表，提升中心服务写入容量 自研分库分表中间件TSharding
- 异步化&最终一致性：大事务拆解为多个本地事务
- 预处理、缓存&静态化
- 单机异步执行、并行优化
- 任务多机器

自研分库分表中间件，突破单点写瓶颈

分库分表业界方案对比

	taobao TDDL	b2b Cobar	Google Vitess	Mybatis Plugin
proxy/ smart client	smart client	proxy	proxy	SQL改写 通过对statement、parameter、rs处理进行包装来扩展
分表支持	支持	不支持单库分多表	支持	支持
分库支持	支持	支持	支持	mybatis无法管理数据源，不支持。
协议	jdbc	mysql protocol	mysql protocol	无
语法支持	jdbc规范	部分语法	部分语法	全语法
结果集合并	支持	支持	支持	不支持
读写分离支持	支持	不支持	支持	不支持
事务支持	支持单库事务	支持单库事务	支持单库事务	N/A
在线扩容（秒级内只读）	支持	支持	支持	不支持
接入成本（使用方）	一般	简单	简单	简单
研发成本（中间件）	复杂	复杂	复杂	极低
运维成本	简单	复杂。经常出问题	复杂。需要一套平台支撑	0成本
多机房支持	支持	支持	支持	不支持

自研分库分表中间件，突破单点写瓶颈

分库分表组件TSharding

https://github.com/baihui212/tsharding

TSharding组件特点

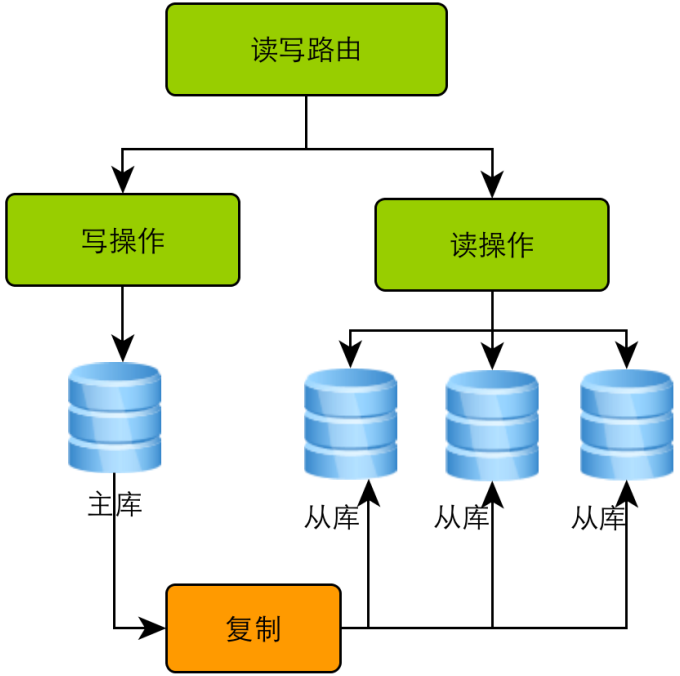
- 简单、易扩展、易移植
- 支持各类业务表的Sharding需求，分库又分表
- 支持数据源路由
- 支持事务
- 支持结果集合并
- 支持读写分离

读写分离

读写分离

- 优点：
  消除读单点
  缩小DB故障时影响范围
  可以任意水平扩展，支撑高并发请求

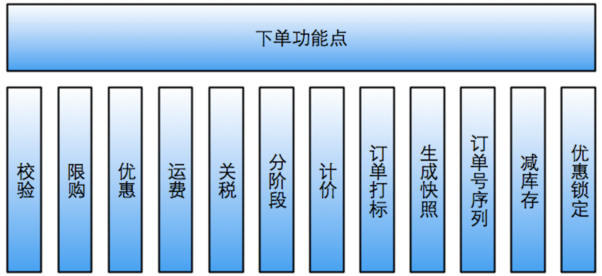
- 举例：
  用户DB、商品DB

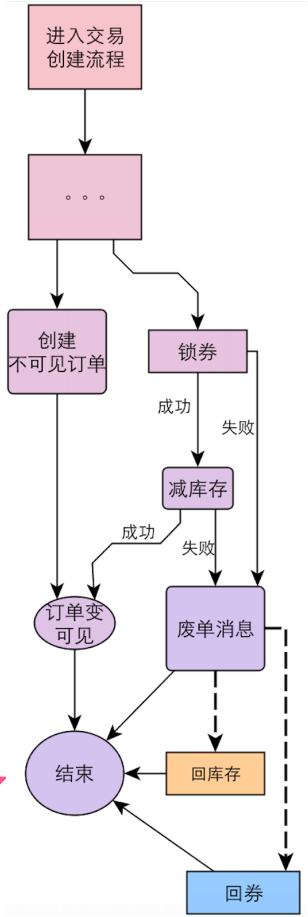


异步化&最终一致性

拆分大的流程为多个小的本地事务

- 优点：
  非常适合非实时、非强一致性的关联业务写入
  事务消息/DB事件变化通知+MQ ACK机制保证一致
  消除二阶段提交等分布式事务框架侵入性影响
- 举例：
  交易创建流程中的功能点管理、分布式事务一致性



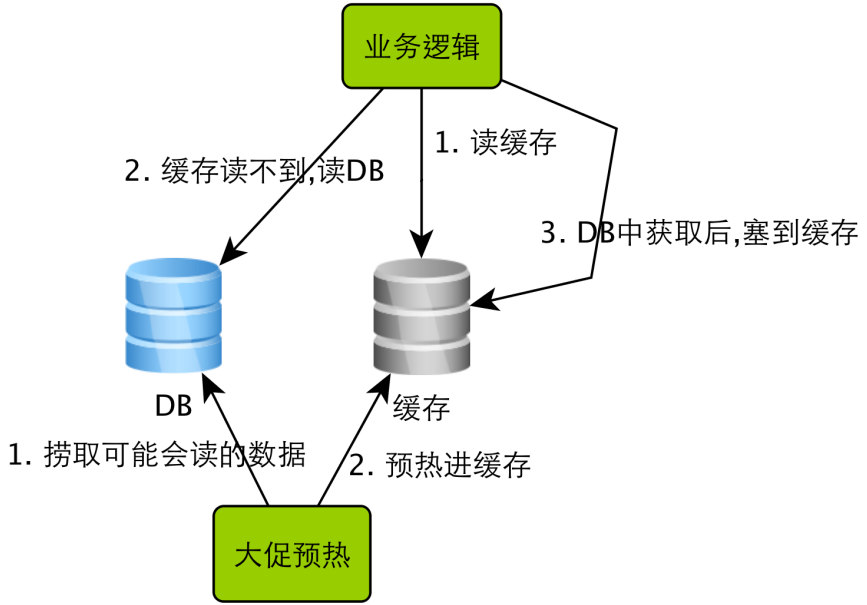


预处理、缓存&静态化

预处理、缓存&静态化

- 优点：
  有效提升关键时间段的读取效率
  减少瞬间热点数据的竞争
  极大减少DB请求量
  多场景适合，支撑高并发请求

- 举例：
  促销大促券数据预热
  商品详情页静态化预热



单机异步执行、并行优化

单机异步执行、并行优化

- 优点：
  异步化并行执行、提高IO总体效率
  能够有效降低RT，提升单机QPS
  消除了线程池管理的复杂性工作
  多场景支持，支撑高并发请求

- 举例：
  购物车列表查询
  促销优惠计算

```
//提交查询
Map<String, CommonParam> paramMap = paramBuilder.getParamMap();
for (QueryNode queryNode : queryPlans) {
    QueryPlanSupport plan = queryNode.getQueryPlanSupport();
    if (null != plan) {
        //得到各个外部服务的查询结果, add降级开关
        if (!CartWebMonitorUtil.isCartBizSwitchOn(CartWebMonitorUtil.getMonitorServiceMap().get(queryNode.getNodeName()))) {
            Object result = plan.queryAsPlan(paramMap.get(queryNode.getNodeName()));
            bizResult.put(queryNode.getNodeName(), result);
        }
    }
}

//获取结果
MethodMonitor.start(CartWebMonitorUtil.CART_LIST_ASYNC_QUERY);
Map<String, Object> futureBizResult = new HashMap<>();
for (Map.Entry<String, Object> kv : bizResult.entrySet()) {
    Object result = kv.getValue();
    //处理异步查询
    if (result instanceof Future) {
        Future futureResult = (Future) result;
        try {
            result = futureResult.get(asyncRpcTimeOut, TimeUnit.MILLISECONDS);
        } catch (Exception e) {
            logger.error("node: " + kv.getKey() + " get futureResult failed! exception:{}", e);
            result = null;
        }
    }
    futureBizResult.put(kv.getKey(), result);
}
```

可用性的挑战

- 99.99% 故障时间52分钟以内
- 每年业务3倍以上增长
- 每周数百次发布、两个月一次大促
- 如何达到日常&大促高可用？——服务SLA保障

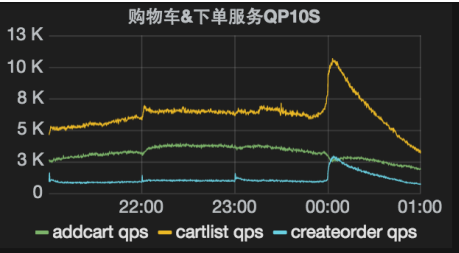
服务SLA保障

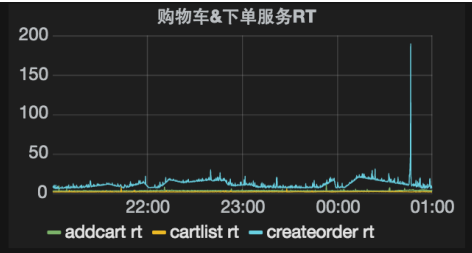
- SLA：Service Level Agreement 是对服务提供者的要求
- SLA体现在对容量（QPS）、性能（RT）、程度（可用性、出错率）的约束
- 基础监控先行，把关键指标监控起来
- 依赖治理、逻辑优化：

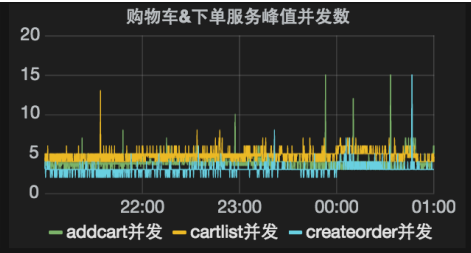
  减少不必要的依赖；强依赖转弱依赖

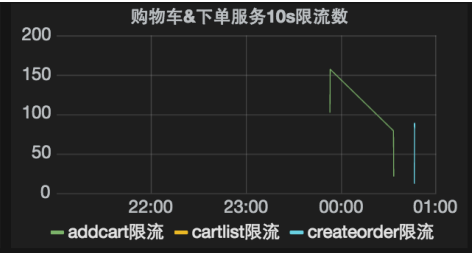
  弱依赖埋好开关；强依赖做好SLA保障

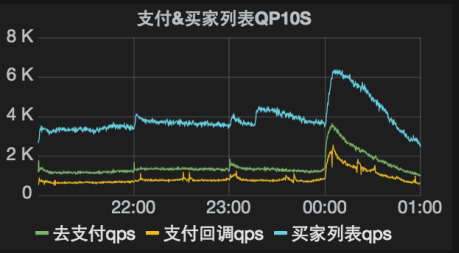
服务SLA保障

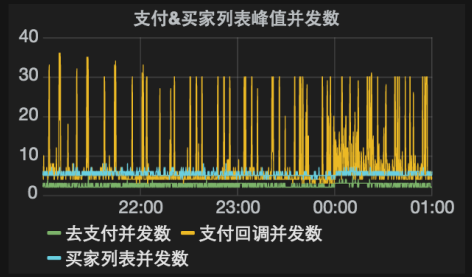

购物车&下单服务QP10S
13 K
10 K
8 K
5 K
3 K
0
22:00
23:00
00:00
01:00
addcart qps
cartlist qps
createorder qps

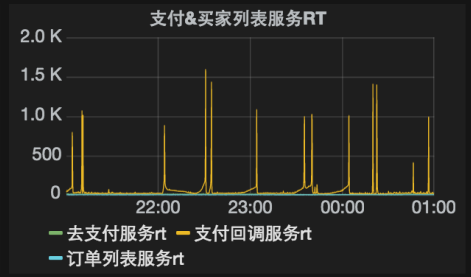

购物车&下单服务RT
200
150
100
50
0
22:00
23:00
00:00
01:00
addcart rt
cartlist rt
createorder rt


购物车&下单服务峰值并发数
20
15
10
5
0
22:00
23:00
00:00
01:00
addcart并发
cartlist并发
createorder并发

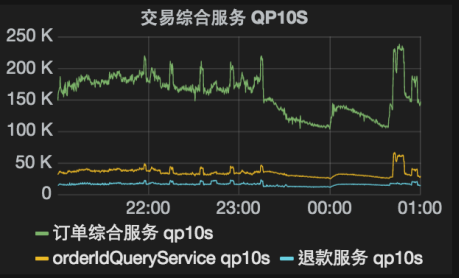

购物车&下单服务10s限流数
200
150
100
50
0
22:00
23:00
00:00
01:00
addcart限流
cartlist限流
createorder限流

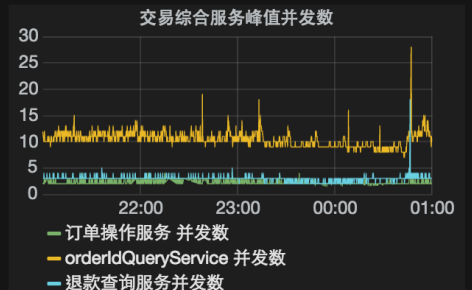

支付&买家列表QP10S
8 K
6 K
4 K
2 K
0
22:00
23:00
00:00
01:00
去支付qps
支付回调qps
买家列表qps

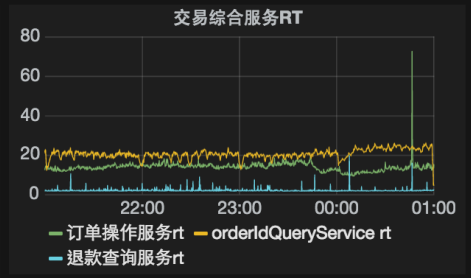

支付&买家列表峰值并发数
40
30
20
10
0
22:00
23:00
00:00
01:00
去支付并发数
支付回调并发数
买家列表并发数


支付&买家列表服务RT
2.0 K
1.5 K
1.0 K
500
0
22:00
23:00
00:00
01:00
去支付服务rt
支付回调服务rt
订单列表服务rt


支付&订单列表服务限流数
800
600
400
200
0
22:00
23:00
00:00
01:00
去支付服务限流数
支付回调服务限流数:距今天-1天
订单列表服务限流数:距今天-7天


交易综合服务 QP10S
250 K
200 K
150 K
100 K
50 K
0
22:00
23:00
00:00
01:00
订单综合服务 qp10s
orderIdQueryService qp10s
退款服务 qp10s


交易综合服务峰值并发数
30
25
20
15
10
5
0
22:00
23:00
00:00
01:00
订单操作服务 并发数
orderIdQueryService 并发数
退款查询服务并发数

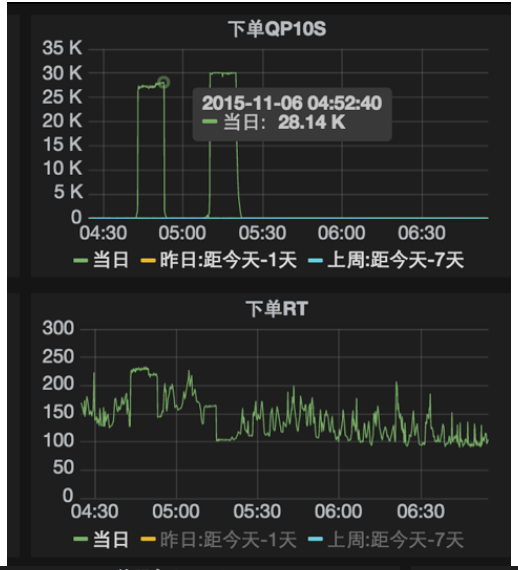

交易综合服务RT
80
60
40
20
0
22:00
23:00
00:00
01:00
订单操作服务rt
orderIdQueryService rt
退款查询服务rt


交易综合服务限流数
1.00
0.75
0.50
0.25
0
No datapoints
22:00
23:00
00:00
01:00
订单操作服务限流数
orderIdQueryService限流数
退款服务限流数

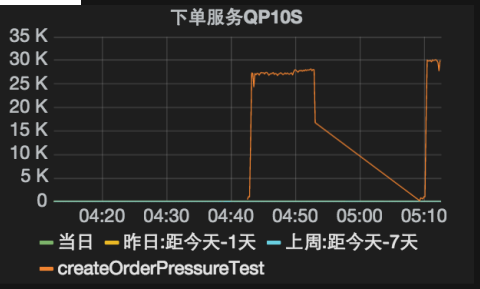
服务SLA保障-压测

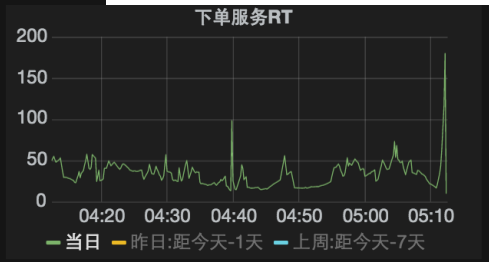
- 线下单机压测 识别应用单机性能瓶颈
- 线上交易写压测引入影子表，隔离脏数据业务影响
- 单链路压测 识别出瓶颈点和风险点；验证集群水位及各层核心系统容量配比；开关有效性等
- 全链路压测 验证流量预估准确性、识别系统在全局压力下的系统能力和稳定性水平

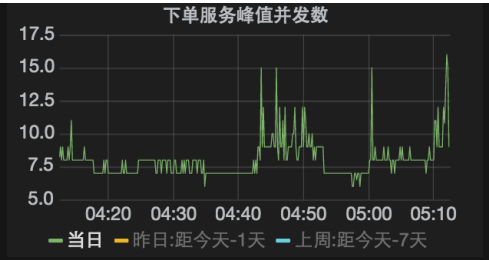
case：交易创建单链路压测确认系统容量

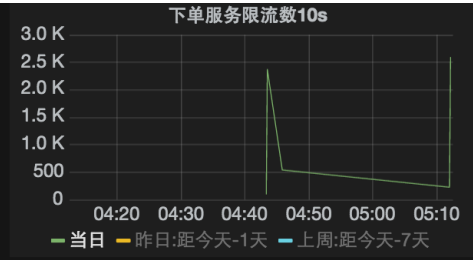



```
mysql> select from_unixtime(updated), count(*) as t f
50:27') group by updated order by t desc limit 5;
+-------------------------+------+
| from_unixtime(updated) | t    |
+-------------------------+------+
| 2015-11-06 04:50:13    | 4897 |
| 2015-11-06 04:49:47    | 4879 |
| 2015-11-06 04:49:50    | 4855 |
| 2015-11-06 04:50:00    | 4845 |
| 2015-11-06 04:49:58    | 4840 |
+-------------------------+------+
```

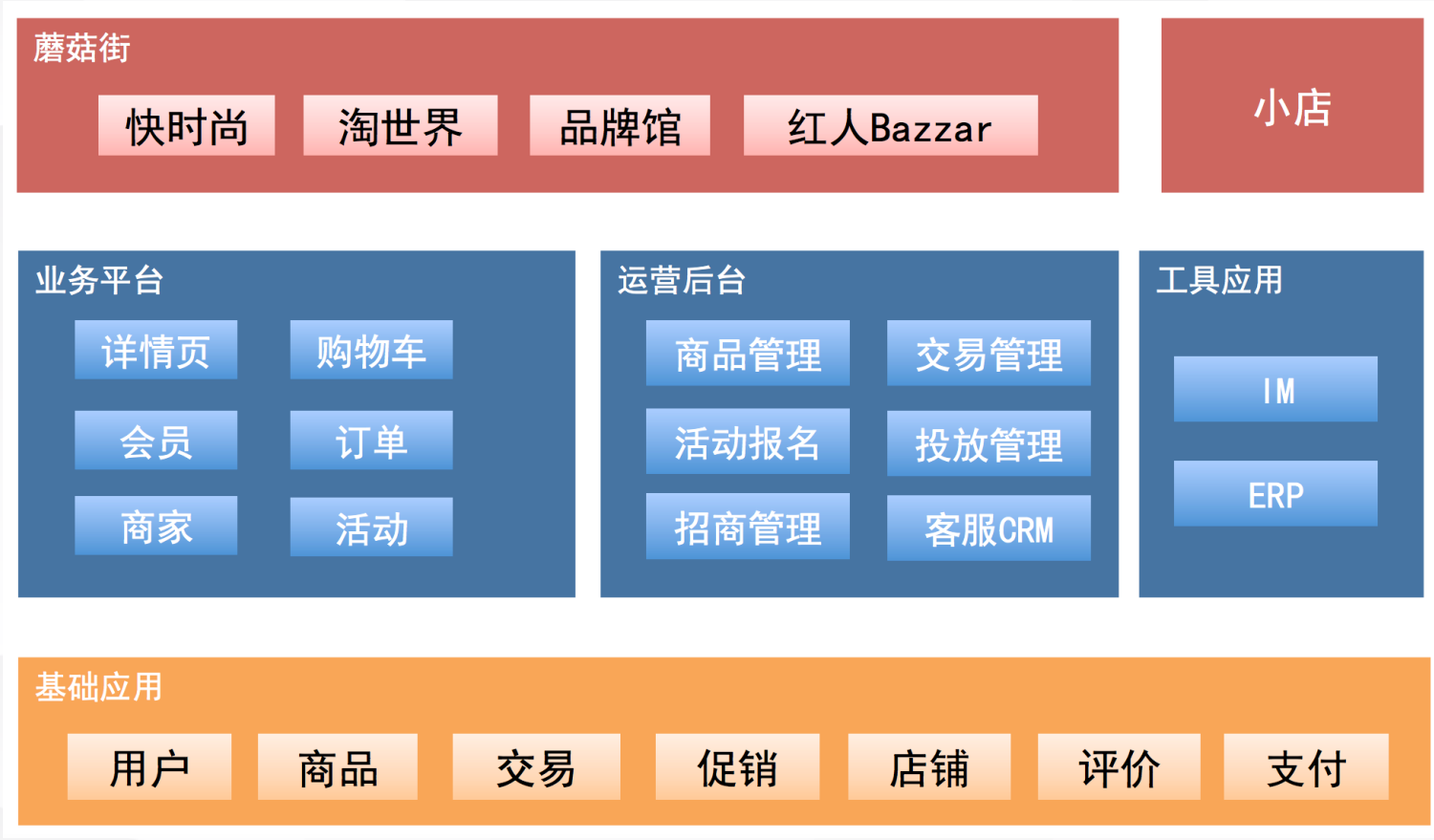








蘑菇街全局业务架构


蘑菇街
快时尚
淘世界
品牌馆
红人Bazzar
小店
业务平台
详情页
购物车
会员
订单
商家
活动
运营后台
商品管理
交易管理
活动报名
投放管理
招商管理
客服CRM
工具应用
IM
ERP
基础应用
用户
商品
交易
促销
店铺
评价
支付

蘑菇街全局系统架构

30%　　70%

负载均衡层　LVS

代理层　Nginx Proxy

WEB层　Nginx+PHP-FPM(pool)　Nginx+PHP-FPM(pool)

中间层　Tesla(RPC)

Raptor(数据中间层)　Corgi(消息中间件)　Furion(配置中心)

基础设施层　MySQL集群　Redis集群　Solr集群　文件存储

Sentry
监控系统

Chronos
定时任务

LogAgent
日志收集

Lurker
全链路监控

总结及下一步展望

- 服务化架构是随着业务的不断发展而不断演变的
- 要前瞻性地谋划实施、支撑业务快速发展
- 容量&性能关键字：一切可扩展、Cache、IO、压测
- 根据场景来选用性能优化方案，没有通用方案
- 正在做：服务治理、SLA保障系统化
- 下一步：同城/异地双活

Thanks!