1号店服务化之路

裴华－1号店

目录

- 1号店服务化历程
- 服务化过程中遇到典型的问题
- 持续改进之路

一号店整体服务化过程

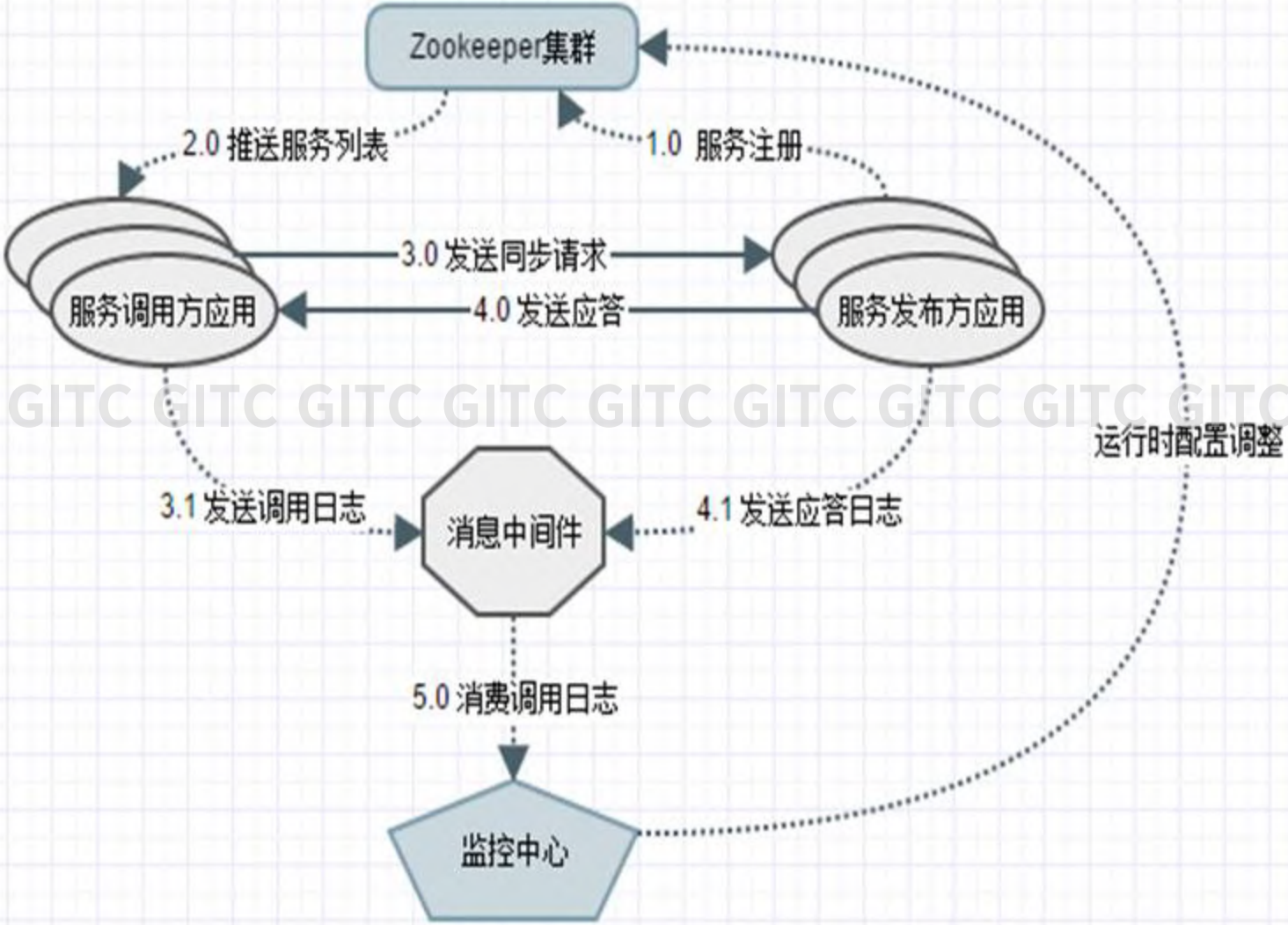
2013年
- 服务化启动
- 库存服务
- 价格服务
- 拆单服务

2014年
- 用户服务
- SSO服务
- 下单服务
- 订单查询服务
- 购物车服务
- 促销服务

2015年
- 产品服务
- 账户服务
- 支付服务
- 登录服务
- 会员服务

服务化之前的痛点：
- 数据库表管理薄弱
- 新增修改公共表结构字段每次都费时费力
- 逻辑散在各个模块之间
- 数据库连接数无法控制
- 互相依赖，甚至造成死锁
- ……

服务中间件 - Hedwig



服务化过程中的问题

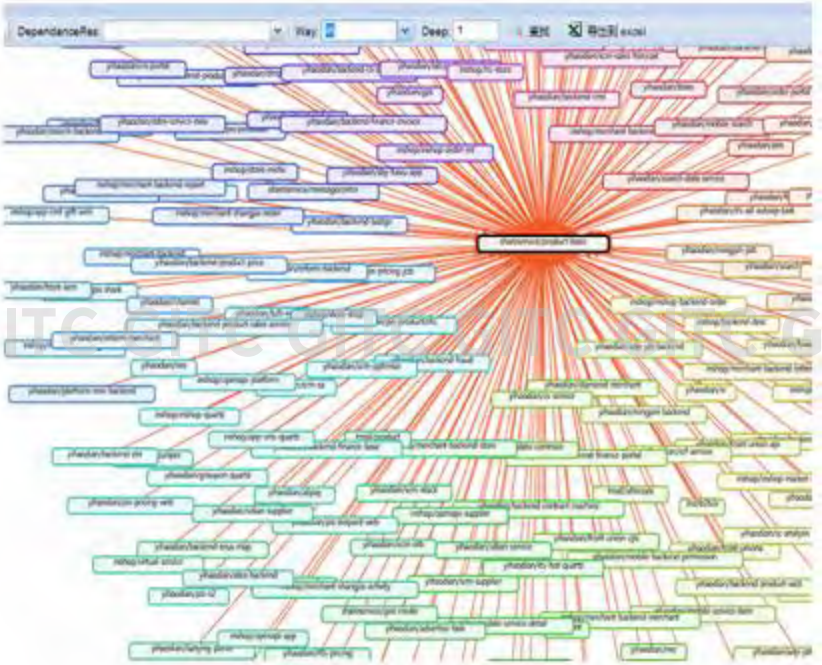
- 服务的规划
  - 分层结构
  - 依赖治理
  - 服务粒度的抽象
  - 异步和同步的切分
  - 在线数据和离线数据分离
- 服务的平衡
  - SQL效率的折损和平衡
  - 后端服务和前端服务的分而治之
- 服务过程控制
  - 服务接入控制
  - 快速测试工具
- 服务化带来的额外工作
  - 服务化之后引发的分布式事务
  - 服务缓存的时效控制
  - 服务轻量级的授权控制
- 双机房下的服务调用控制

服务分层结构 -服务的规划（1/5）

调用方向

依赖治理 -服务的规划（2/5）

- 治理方法
  - 利用Hedwig服务日志分析
  - 确保依赖层次顺序
  - 定位层次，合理拆分和合并
  - 服务功能高内聚
  - 打破循环依赖，即使非同一方法的调用
  - 控制服务调用链长度
  - 定期检查



服务粒度的抽象 - 服务的规划（3/5）

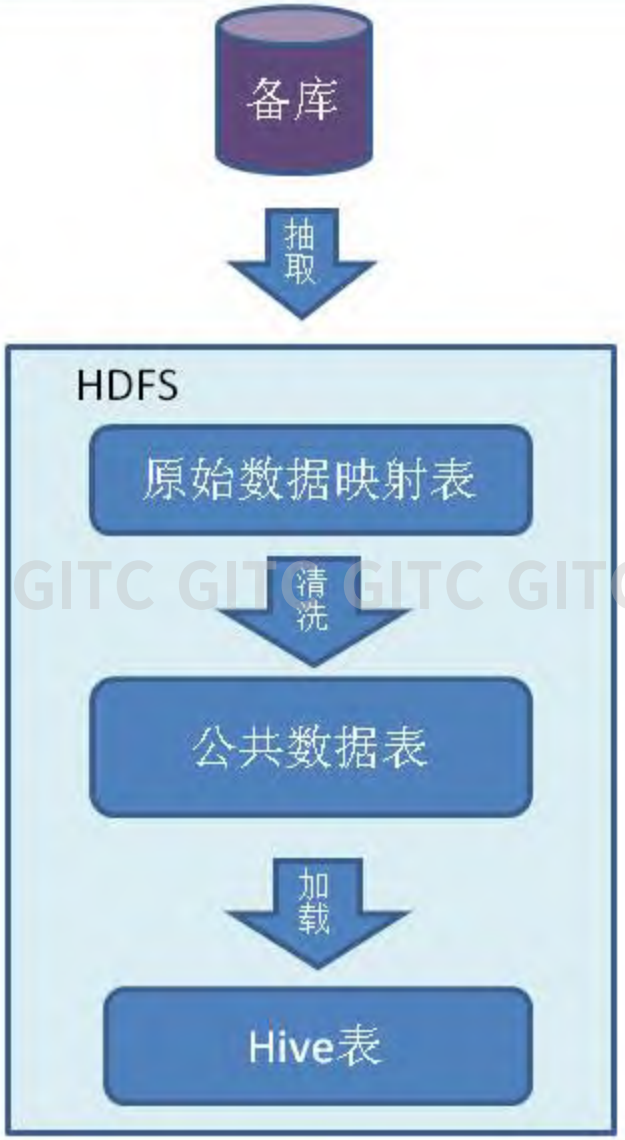
- 服务粒度要求
  - 基础服务
    - 服务粒度比较细
    - 实现简单功能
    - 共享程度很高
    - 定义三级对象（VO）：
      - 基本属性
      - 半属性（最难定义）
      - 全属性
    - 举例：产品服务
  - 聚合服务
    - 服务粒度粗
    - 功能相对复杂
    - 共享程度高，但低于基础服务
    - 举例：购物车服务

同步服务和异步服务拆分 - 服务的规划（4/5）

- 拆分往往发生在聚合服务和应用层中
- 拆分的原因
  - 降低系统耦合度，提升处理效能
    - 例如：
      - 详情页根据商品变化消息清理squid缓存
      - 搜索实时索引根据库存变化消息进行排序调整
      - 下单服务中限购数量的控制，改为在前端购物流程控制
  - 时效性要求不高，提高主流程响应时效
    - 例如：在购物车服务中的网盟佣金处理
  - 次要服务，没必要影响主流程
    - 例如：精准化栏位推荐的商品变更
- 拆分的实现
  - 前端应用的懒加载
  - 通过异步线程池的方式进行请求
  - 通过消息机制进行处理

在线数据和离线数据分离（5/5）

- 原有的状况
  - 直连数据库抽取数据
    - 曾出现过抽数据影响线上应用的情况
  - 通过服务化的接口抽取
    - 接口屏蔽表结构的逻辑之后，只有通过接口取数才准确
  - 基本上都是周期性的定时操作
- 改进：
  - 搭建离线数据平台，降低在线服务的功能复杂度和负载
    - Hive提供复杂条件查询
    - 例如： SEO、网盟、搜索、财务等场景



提供给前后端服务分而治之 -服务的平衡（1/2）

	前端对服务的要求	后端对服务的要求
响应速度	50ms以内	秒级，甚至几十秒
时效性	可以缓存，关键点实时	实时数据
查询角度	点查询为主	批量关联查询

	提供给前端的服务处理方式	提供给后端的服务处理方式
缓存	几乎全部缓存，并且按照不同的SLA要求，缓存时效不同	缓存命中率低，缓存效果不明显
接口	单独提供	可以复用前端接口 复杂查询单独提供接口
服务分组	前端分组	后端分组

SQL效率的折损和平衡 -服务的平衡（2/2）

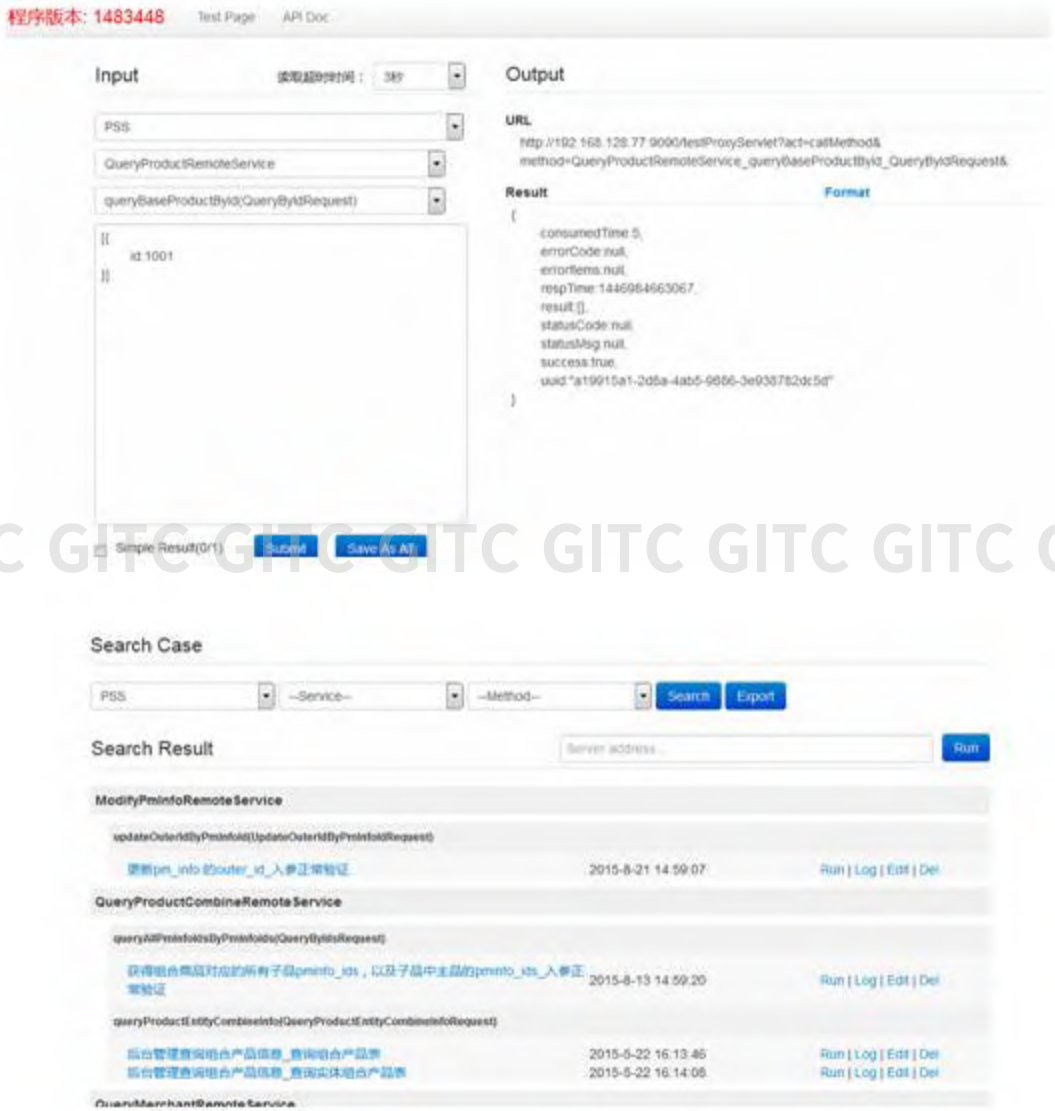
- 可能引发效率下降的因素（特别是后端）
  - 接口会限定返回数量，增加多次IO请求，每次情况中还会有序列化和反序列化的开销
    - 接受现实
    - 合理的定义接口返回数量的大小
  - 本来若干表是可以直接关联查询的，因为拆分，可能要在内存进行关联计算
    - 合理整理业务逻辑，去掉不必要的关联查询
    - 采用合理的逻辑，降低关联计算的复杂难度
    - 同步部分数据表用于关联查询
  - Oracle分页查询，服务端每次通过入参控制分页，会导致分页后面的查询效率越来越慢
    - 优化查询，例如将分页换为id的范围查询

服务接入控制 -服务过程控制（1/2）

- 问题：
  - 新功能的快速开发和接入的平衡要求
  - 新员工的不断增加新的sql查询
  - 老的代码缺少维护
- 解决方案
  - 控制DB访问用户，控制权限
  - 独立分库
    - 同义词
  - DAO层切面检查，分析异常中表名

服务快速测试工具 - 服务过程控制（2/2）

- TestProxy
  - 模拟服务的端到端测试
  - 用与服务的快速测试构建
  - 保存为自动化测试用例，用于回归
  - 与性能测试和自动化平台无缝集成



- 分布式事务的产生场景
  - 产品生码流程
    - 产品信息处理
    - 价格处理
    - 供应商信息处理
    - ……
  - 礼品卡充值流程
    - 礼品卡状态变更
    - 用户账户余额变更
- 解决方案
  - 事务补偿
  - 请求幂等
  - 异步事务

缓存的失效控制 - 服务化带来的额外工作（2/3）

- 服务化之后对缓存的控制要求更高
  - By Key维度的缓存对象清理
    - 在写接口的时候清理
    - 由于可能一个key会有，基础、中、全三个缓存对象都要清理
  - 修改导致批量缓存失效
    - 例如：
      - 类目属性的变更
    - 解决方案
      - Key的对象尽可能对应单表，多次操作缓存对象
      - 批量清理缓存映射关联模块，通过key的版本更新，导致批量失效

```
Product:
{
  id:1,
  name: XXX,
  brand:
  {
    id: 1003,
    name: '名牌'
  }
}
```

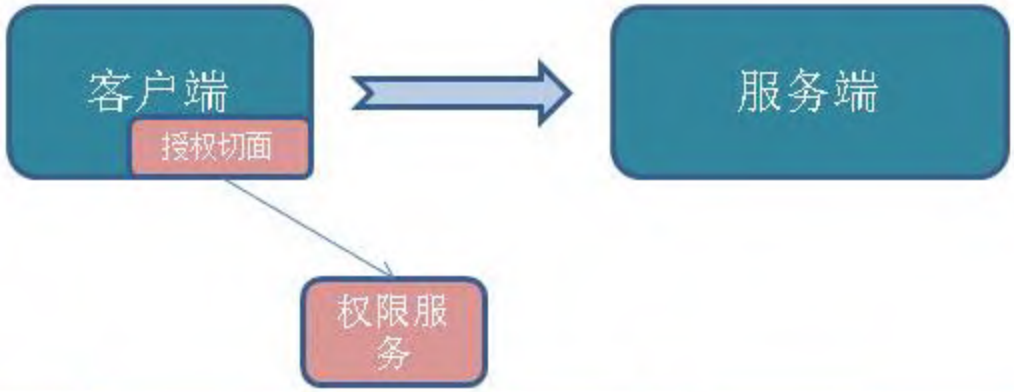
```
Product:
{
  id:1,
  name: XXX,
  brandId: 1003,
}

brand:
{
  id: 1003,
  name: '名牌'
}
```

```
<!-- prefix_queryFieldxx_filed1_v1_%1%_filed2_%2% -->
  <cacheKey id="aa1" prefix="queryFieldxx">
    <argument name="filed1" column="FIELD1" table="TEST" isUseNameVersion="true" />
    <argument name="filed2" column="FIELD2" table="TEST" />
    <values>
      <value column="filed1" table="brand"/>
      <value column="filed2" table="category"/>
    </values>
</cacheKey>
```

轻量级授权控制 - 服务化带来的额外工作（3/3）

- 问题：
  - 需要对敏感的接口进行保护，防止恶意和错误的调用
  - 例如：
    - 改价接口
    - 修改商品信息接口
    - 修改账户余额接口
- 解决方案：
  - 服务端的pool在提供的客户端内，进行切面权限控制
  - 按照pool维度，控制是否访问某些服务的权限，特别是写接口
  - 也可以控制到方法级别（很少用）



双机房环境下的服务调用控制

- 多机房部署对服务的要求
  - 尽可能的杜绝服务层的跨机房访问，所有的服务依赖都是在本地机房发生
  - 控制数据库的连接数，尤其是主库
  - 缓存和数据库在跨机房情况的同步一致性

持续改进之路

- 持续控制服务分层部署
- 持续的梳理依赖关系
- 持续优化服务粒度
- 构建更多合理的聚合服务，快速支持业务发展和调整
- 支持更多机房的部署，并且划分并且构建业务单元模块
- 支持云服务的部署

谢谢!

Q&A