# 苏宁易购亿万级商品评价系统的架构演进之路和实现细节

周健 极客时间 2016-03-28

苏宁易购评价系统跟随着易购商城的业务发展，经历了从Commerce系统拆分再到系统全面重构的整个历程。如何满足系统流量的日益增长，在提升系统性能和满足稳定性和可扩展性的要求的同时，向目标系统架构一步步平滑靠近，成为系统面临的最大挑战。本次分享的内容包括：

1、评价系统架构演变

2、评价系统架构设计

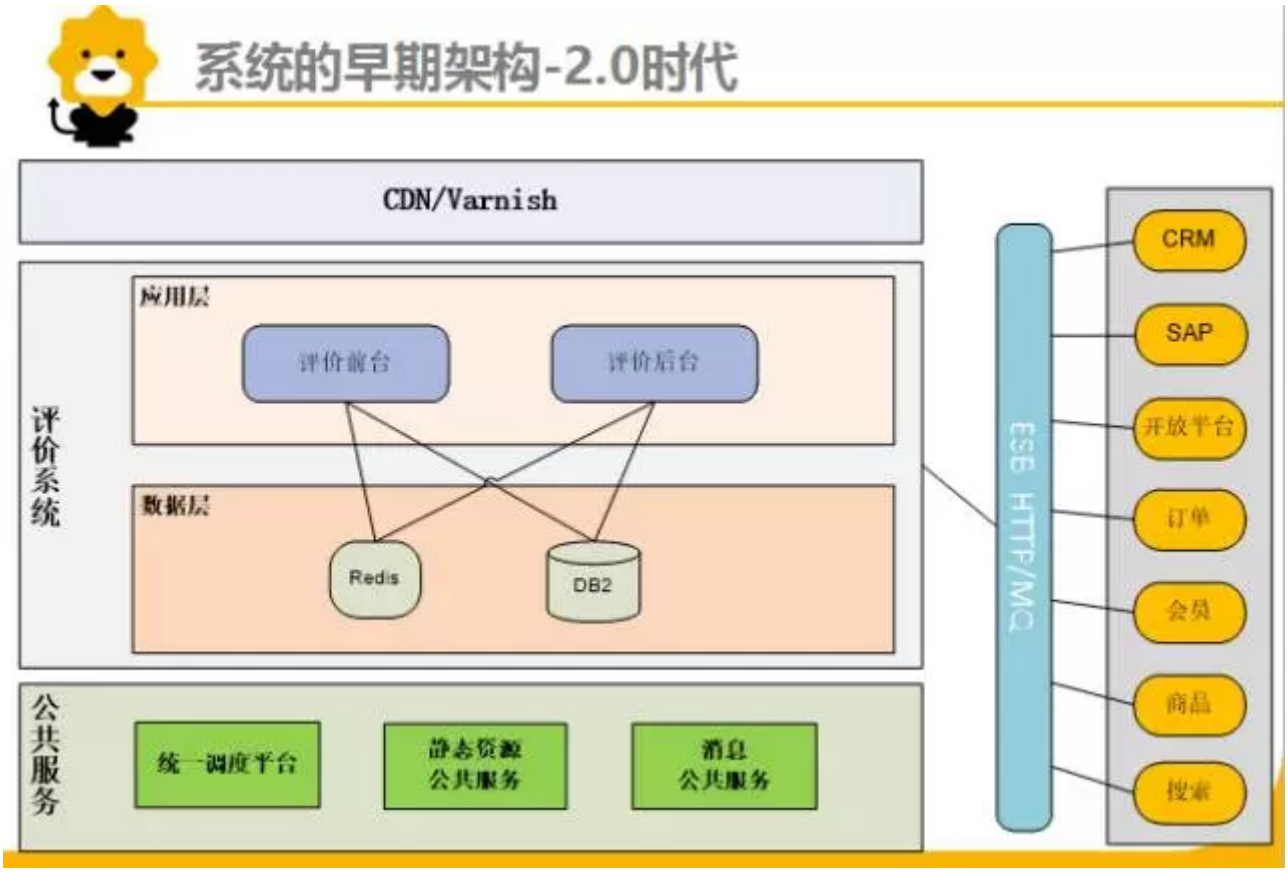
3、技术实现

4、曾经踩过的坑

## 评价系统架构演变


苏宁易购早期的电商平台是基于IBM Commerce为核心，与SAP等后台系统进行交互的套件组装系统；评价系统作为IBM Commerce系统中的一个功能模块，与订单、会员、商品、库存等功能模块耦合在一个庞大的系统中。因此**系统开发和维护成本很高**：

1. 系统启动、部署复杂度特别高，严重影响效率；
2. 系统某一个模块需要新增功能时，需要整站系统发布；
3. 如果其中某一个模块出现问题时，会影响整个易购系统的可用性；




评价系统从易购网站2.0时代开始，从IBM Commerce系统中拆分出来，建立了一套**独立的系统**。评价系统2.0基于**开源框架**构建，系统开发维护成本降低，与外部系统之间通过ESB HTTP/MQ进行数据交互，系统之间松耦合。但是随着易购线上业务的不断发展，系统瓶颈也逐步暴露出来：

- 系统基于DB2商业数据库提供存储和查询服务，聚合运算逻辑依赖于DB，高并发性能和可扩展性受到限制；
- SOA服务化不彻底，服务职责划分不清，服务治理迫在眉睫；
- 监控、告警、日志不完善，系统可维护性差；

## 2.0时代系统面临的挑战

和周边其他系统关系复杂，商品评价依赖于商品、订单、物流、会员等基础数据，还需要对外提供各种评价数据服务，例如和订单中心之间的订单数据服务；系统之间这些依赖关系需要合理的规划；

多年累积的用户评价数据量非常庞大，应用中包含有对评价数据的各种聚合、排序、多维度查询等复杂业务场景，例如商品详情页查询好、中、差评个数，按商品、店铺、标签、通子码等多个维度查询评价列表，如何选择合适的存储满足应用场景的要求?

在网站流量较大的商品详情页会展示商品的评价，促销活动期间并发量非常高；如果单纯的依靠数据库查询，可扩展性不够，性能瓶颈无法根本解决。

随着公司多端融合战略的落地，系统也要能满足易购PC端、移动端、门店、WAP多渠道融合展示的特点；例如对于商品评价列表的查询服务，在不同的渠道都需要调用该服务获取数据，但是在展示层会体现出多端之间的差异；系统还面临着恶意爬虫、机器攻击等安全问题的干扰。

## 评价系统架构设计

评价系统从应用层面上设计了三层——前台展示层、中台服务层、后台数据管理层。

**前台展示层**：为多渠道的终端评价展示提供个性化的展示逻辑。

**中台服务层**： 为评价系统的前台展示层及周边系统提供统一的原子服务，例如评价列表查询接口、发表评价接口等等。

**后台数据管理层**： 提供后台数据处理逻辑，包括数据后台管理、数据消息的接收和推送（接收来自OMS、PCMS等系统的订单、商品基础数据，对BI、搜索等系统提供商品评价数量等信息） 、定时任务的调度处理等。

易购网站3.0时代，重点就是解决**系统之间的服务治理**问题，同时提升系统的性能、可靠性、可维护性等能力：

1. 结合苏宁自身业务的特点，将多渠道的系统服务进行融合；
2. 设置了Redis、Solr、DB、 DFS等多级存储，合理利用最宝贵的资源；
3. 通过风控、流控、降级等多种手段提升系统的可靠性和安全性；
4. 推广RSF远程服务框架，提升服务效率和服务治理能力；
5. 完善系统的监控和告警；

核心的评价列表查询服务由查询DB**重构为Redis+Solr方案**，Solr可水平扩展来提高系统总体吞吐量；商品评价的增量数据通过异步方式提交至Solr 。

数据库由商业数据库DB2**切换为开源的MySQL**，设置为一主多从，多个分库模式，满足数据库的可扩展性需求。对系统服务重新进行**梳理合并**，多个渠道进行整合，同时服务调用由ESB HTTP**切换为远程服务框架RSF。**通过接入WAF、流控、防爬拦截、准备降级方案等多种手段，提高系统的**安全性和可靠性**；系统的每个分层层面保证都是**可扩展的。**

## 技术实现

### 评价系统典型场景1：商品评价数量

商品评价数量是评价系统中**访问量最大**的请求，包括商品/供应商好、中、差评数量、标签数量、个性化评价项数量等，会在商品详情页、店铺详情页、搜索列表页等访问量较大的页面展示。因此使用缓存，相比之前查询DB性能大幅度提高，每次评价操作需**同步更新缓存**，对应的场景较多：包括发表评价、追评、修改评价、删除评价、放弃评价、评价审核、回复等等，每个操作对各个评价数量的操作各不相同，包括商品维度的好中差评数量、商品+供应商维度的好中差评数量缓存都要同步进行更新。

### 三段式缓存设计

针对商品评价数量缓存的生命周期特点，设计了三段缓存的实现方案，通过控制缓存的生命周期，避免缓存失效后集中回源造成系统压力过大。

### 评价系统典型场景2：商品评价列表

商品评价列表也是**应用最多**的一类场景：目前的评价数据总量接近亿级，在查询时需按标签、供应商、商品等多个维度进行筛选，尤其是查询大页码的评价列表对数据库的查询性能是个挑战，所以我们重构后改为**缓存+Solr方案，不再需要读数据库**：

1. 用户发表的评价数据，我们在数据入库的同时同步写入缓存，可以在第一时间展示在商品评价页；
2. 缓存里的评价数据再通过异步方式提交到Solr存储，提交至Solr的评价数据则从缓存清除；

3. 按商品、店铺维度展示的评价数据，在系统中台通过缓存 + Solr的数据聚合方式对外提供查询服务；

**Solr异步提交**

Solr采用读写分离，缓存里的评价数据通过异步方式提交到Solr写服务器，这样的好处是：

1. 降低Solr snapshot的频率，避免高并发的写操作带来的Solr索引同步，导致的Solr读效率降低；
2. 评价操作逻辑尽可能简单，提高评价操作类接口的性能；
3. Solr的评价数据处理和评价操作接口分离，即是索引数据的缓冲，也能保证数据处理不会影响到评价的主流程；
4. Solr的索引数据处理需要提供完善的监控和管理功能，能实时看到系统当前堆积的数据情况和处理情况。

**Solr/SolrCloud**

**Solr：多维度查询利器**

评价数据量达到亿级，评价数据展示的场景包括按商品通子码、店铺、商品特征值、标签等多个维度查询，并按日期或推荐算法排序。商品评价数量存储在缓存，而当缓存时间到达设定的阀值后，则会回源到Solr通过Facet实现对商品评价数量的汇集计算功能。规划的按评价内容全文检索、亮显、智能匹配等功能都可以基于Solr搜索实现。Solr读写分离，读库可支持水平扩展；

**Solr性能**

通过实际压测，在亿级数据量的Solr（两台4C/8G配置的服务器）环境下，多维度（按商品、供应商、等维度）大页码查询的场景TPS可以达到2000左右。

通过水平扩展Replication节点，可以显著提升Solr/SolrCloud的整体吞吐量。

随着评价数据量的一直增大，Solr的索引文件越来越大，这也将导致Solr的查询效率也会逐步下降。这时候就需要采用**Solr的分布式方案**：Solr4.0以后的版本已经提供了基于Solr和Zookeeper的分布式搜索方案SolrCloud。

### DB读写分离/分库分表

采用苏宁自研的DAL数据库组件，数据库读写分离和分库分表。

1. 评价数据面向用户展示时，例如待评订单都是和用户相关的场景。所以把待评订单数据按照会员作为Sharding Key进行分库分表，保证每个分表的数据量尽量控制在百万级别的数据范围内。
2. 一主多从实现读写分离， 读操作尽量通过从库执行，减轻写库的压力。
3. 对于失效的订单历史数据，及时进行数据归档，减少数据库压力。
4. 只有用户级别的待评订单查询和后台运营评价管理两类场景下才查询数据库，其他场景都改为查缓存和Solr，尽量降低对数据库的查询访问。

### 数据访问原则



数据访问原则：**最宝贵的数据资源留给最需要的场景**。

访问量最大的评价数量请求通过访问Redis缓存返回数据；
用户查询商品评价列表内容请求通过访问Redis + Solr返回数据，不依赖于数据库；

用户维度的评价数据展示及为运营提供的少量数据管理功能，则直接访问最宝贵的DB资源。

**多终端融合服务化**

结合苏宁自身一体两翼三云四端的发展战略，围绕线上线下两翼平台，实现移动端、PC、TV、门店多端的融合。

系统中台对外提供应用服务，统一了前台展示、周边系统的服务依赖。而系统前台则按不同渠道之间的差异实现各自的展示逻辑，这样既做到了数据服务的统一，又实现了展示服务间的隔离。

评价系统对外提供的数据接口，统一整合为异步消息接口。这样评价系统只需要计算一次数据，就可通过消息发布至订阅的相关系统。

不同终端的智能自动适配：评价晒单图片在不同类型的终端上智能展示不同的大小，避免在移动端上出现因图片大导致查询慢、卡死。

**性能指标监控**

接入统一监控平台，实时监控服务TP90、TP99、调用量等关键性能指标的变化情况，尤其是每次版本发布前后的对比。

**私有云服务**

新评价系统基于私有云平台创建，实现弹性伸缩，大促前可以快速扩容上线。

为了保障系统的安全性和稳定性，需要重点考虑如何防止恶意爬虫攻击，出现异常流量时如何应对：在安全性、流量控制、大促降级等方面也需要做很多细致的基础工作，在此就不一一介绍了。

## 曾经踩过的坑

### 索引预热

系统上线后，发现每次在索引更新并同步至Read Solr的短时间内，Solr性能会下降非常明显。

Solr对外提供的查询服务即SolrIndexSearcher，每次索引更新后即生成一个新的SolrIndexSearcher，之前运行的Cache对象都将失效。所以我们可以做的事情就是在重新构建的SolrIndexSearcher返回之前，需要对Cache进行预热，这样暴露出去的SolrIndexSearcher就是预热后准备好缓存内容的Cache。

### 索引优化

系统在上线之初，面临着要选择索引的定期优化到底是在Index Solr上，还是在Read Solr做优化？实际测试后发现，索引在优化的过程中对查询的性能影响还是比较大，而且索引文件越大，优化时间越长。综合考虑：选择在Index Solr上优化，再同步至Read Solr，这样在索引优化的过程中不会影响系统的查询性能。

优化过程中索引文件需占用两倍的磁盘空间，但是优化完成后索引文件所占空间会变小。另外，JVM参数指定的堆内存设置建议比索引文件大；每个Shard上的索引文件不要过大，建议不要超过10G；

### 系统平滑升级

系统每经过一次大的重构，做到平滑升级都是系统架构设计时必须重点考虑的。老系统的很多接口服务都历史久远，和周边系统关系错综复杂，已经很难一步完成切换，风险太高。

因此总体上按照在老系统增加适配层的原则进行迁移。通过适配层，完成接口服务的新旧迁移。适配层的开关在统一配置平台配置，如果发现问题则可以快速回滚。周边系统则按计划逐步切换至新的服务，直至老系统服务的退出。

切换过程中，通过灰度发布降低发布风险是非常必要的。另外需要关注的是，任何接口的升级都不能依赖于系统版本的升级，这样可以有效地避免版本的回滚。

### 数据分片工具

系统重构的过程中，需要考虑系统的切换实现方案。其中面临一个问题就是：如何对海量历史数据进行批量处理。怎么才能既顺利自动完成对数据的处理，又不影响生产系统的正

常运行?

我们设计了一个对海量数据自动切片的工具，通过工具切片后把数据切分为一个一个小的数据块，数据块可以切分得足够小以保证每次影响的数据范围足够小。后台再通过分布式任务调度，对一个个切分好的数据块通过任务调度执行数据处理。工具切片和任务调度都可以通过配置完成自动化的数据更新。