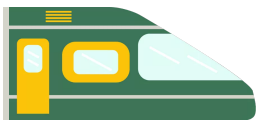
途牛抢票系统架构演进

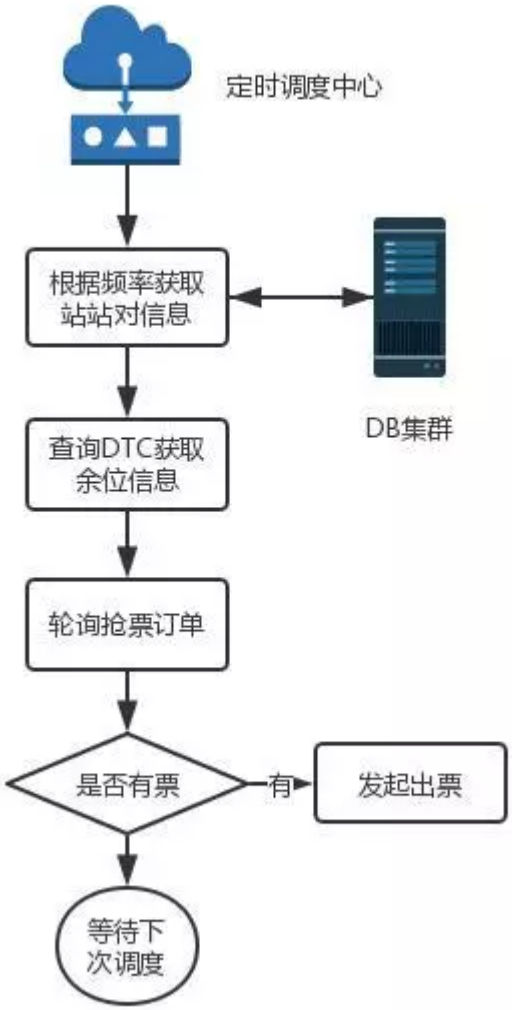
原创：丁然 途牛技术中心 1月24日

> 铁路作为中国最重要的交通工具，每一年春运都要面对浩浩荡荡的返乡潮，虽然铁路蓬勃发展，但依然无法支持如此庞大的返乡人群 — 2018年全国春运旅客发送量近30亿人次，2019年预计会更多，每一年春运，一票难求都是一个热门话题。
>
> 途牛抢票监控系统成立于2016年6月，通过借助抢票系统余位监控、线下票台供应商渠道自动发起占位出票，提高用户出票成功率。

一、系统成立



抢票监控系统起初作为一个模块嵌在OCS补偿系统内部，用不到10个类、几百行代码实现了余位监控的基本功能，部署在2台实例上，通过公共定时调度中心发起定时任务，根据频率获取车次站站对进行余票监控，如下图所示：



为了提升成功率，在2016年11月上线备选抢票车次功能，订单数量增加了50%以上，由定时调度中心发起的定时任务不断请求DB站站对数据，对DB造成极大的压力，导致查询缓慢延迟10s以上。此外由于实例较少，大量订单加载到JVM内存，虽然通过多线程提升订单处理速度，但由于订单数量较大，线程池在处理过程中需要等待DTC抓取余位判断是否有票，这种IO密集型的消息处理，造成线程处理时间也开始延长，导致队列堆积，触发线程池丢失策略，所有的订单不能在一个频率周期内全部监控一次即产生有的订单有监控有的订单没有监控到的问题，无法实现高频余票监控，只能低频运转。

二、系统拆分

2017年，根据微服务及解耦合思想，将现有的DTC数据抓取查询系统拆分为DTC_LEFT余位查询 和DTC_GRAB抢票查询 2个独立系统，并将系统间的代理IP等资源根据业务进行场景细分隔离使用，同时为了提高资源利用率，抢票抓取到的余位数据通过Tubing同步更新到余位查询系统，以保持余位数据最佳新鲜度。

在数据性能方面，DTC_GRAB抢票查询系统返回数据去除出发到达等抢票弱相关字段，减少字段赋值极大的缩小缓存大value情况，改造后缓存带宽发送流量精简80%以上，大幅提升余位查

询接口的响应速度，时间由原来共用DTC的4s提升至300ms左右。

在机器实例方面，将余位监控系统SGT部署到云平台实现机器实例的动态扩展，并通过定时调度中心发起调度到一台机器上，通过Tubing将订单数据分发到所有的实例上，有效降低单实例的线程池并发的压力，如下图所示：

此后接入京东抢票等多家大型分销商，订单数再次增加5~6倍，在不断提升抢票成功率的同时，抢票频率由30秒提升到5秒左右，由于订单数据存在DB中，调度中心每5秒进行一次DB订单读取，由于订单量太大，采用翻页循环读取数据方式，频繁翻页导致DB压力增大，开始出现

上一次调度的数据还没有读取完，下一次的翻页查询又开始，恶性循环之下产生数据查询堆积，导致DB连接池很快突破上限，出现无法获取连接等异常，致使所有订单无法在规定时间周期内完成有效的全量监控，余位监控能力依然存在性能问题。

三、系统重构

DB性能未彻底解决，基础公共定时调度中心规则改变(限制了最快调度频率)，面对2大影响余位监控性能问题，决定进行架构重构，寻找脱离公共中心调度的监控方案。

经过查阅分析，借鉴了淘宝tbschedule开源分布式定时调度分片框架思想，对于原框架每一个分片和实例IP固定绑定弊端，无法满足抢票业务通过字母进行动态随机分片到所有实例上的问题，自行开发一套符合抢票的定时调度随机平均分片的框架，实现了将N个字母随机平均分配到所有的实例上，每个实例根据字母站站对分片从DB获取一小部分站站对数据，再通过Tubing均匀分发到所有的实例，摆脱了大量数据DB翻页查询对DB不断查询的压力，同时我们也对抓取订单数据、分片数据增加Redis缓存，进一步降低DB压力。此外在每个实例上配置定时调度模块，通过配置页面灵活进行所有实例的频率调整等。

整体架构如下图所示：

配置分片如下图所示：

配置调度时间如下图所示：

据不完全统计，2019年春运期间，系统总抢票订单数已突破500万以上，单日查询量超过3000多亿次。由于订单量再次暴增，大量站站队数据导致相关Redis缓存value值不断加大，开始出现Redis获取连接失败、Redis读取时间延迟等问题，于是我们在每台分布式实例引入本地二级缓存机制，将DB的一部分数据存入实例本地缓存，抢票监控系统在获取站站队数据时先查询本地缓存，再查Redis缓存，最后查询DB，充分发挥分布式本地缓存负载均衡减压，大幅降低Redis和DB压力，整体监控效率提升30%以上，DB慢查询降低如下图所示：

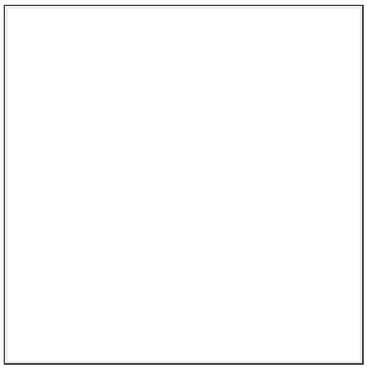
监控容灾

系统初始化，每个实例都会自动注册到zookeeper上，选举一台实例作为leader，其他节点作为Follower，当Leader宕机，follower会自动触发再次选举leader。当系统发布或者某个实例出现异常，问题节点会被自动剔除掉，分片不会再次分到这个节点上。如果zookeeper宕机，每个实例会根据最后一次的分片结果保证业务继续进行，每个实例会自动进行zookeeper重新连接注册，连接成功后，重新选举leader，重新进行分片操作。

至此服务器、DB、缓存及相关性能问题逐步得到根治解决。

写在最后

通过抢票监控系统，将大流量、高并发、分布式、缓存、限流、降级、数据一致性等进行了项目技术落地，为后续相关高并发项目奠定基础，获得了宝贵经验。在日常系统维护中也犯过错、踩过坑，经历过痛苦绝望，但静下心来细思品味就会发现这是历史的机遇，又是不可多得的挑战，正所谓业务推动技术，技术服务业务。把握机会，再接再厉，相信一切都会迎刃而解。



扫二维码，加入途牛技术粉丝微信群，牛牛等着你哦~

敲门砖：“途牛技术”