大厂面试（二）系统架构设计

原创 胖当当技术 胖当当技术 2023-11-21 10:07 上海

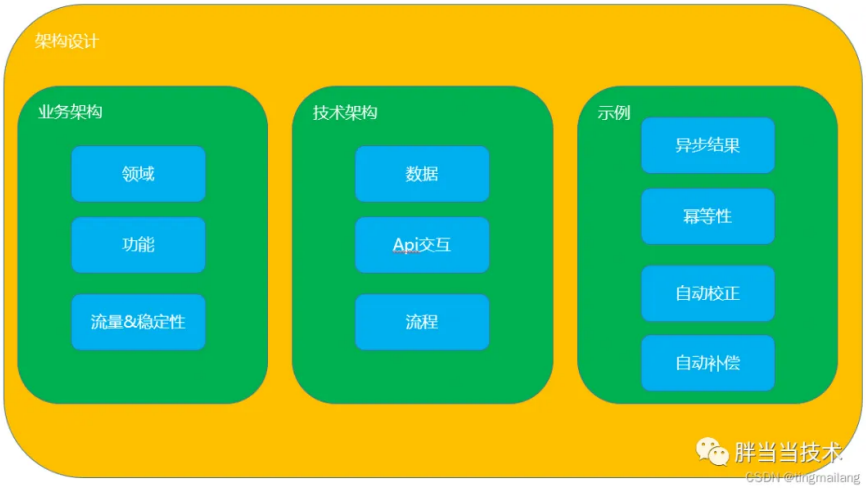
一、引言

面试过程中架构设计一般是三年以上经验会被问到的，初级人员经验不足、系统整体结构理解较浅，一般是不会被问到这种问题的。

作者在这里聊聊架构设计的理念经验，方便各位同学像聊天一样和面试官交流这个问题。这里结合了携程一位大牛：Gavin的文章干货 | 携程商旅订单系统架构设计和优化实践，里面谈及许多架构分层的理念，用以作为文章一部分的例子。（这里已经征得Gavin的同意）

二、架构设计

架构可以分为技术上的架构和业务上的架构，业务架构注重于边界分层、流程统一复用，着重于解耦以及敏捷开发。



技术架构着重于系统交互、工具开发使用，目的是为了达到更快的处理效果，更自动化的处理异常、系统交互合理减负，以及工作中技术调研解决问题。

1、业务架构

业务架构的划分首先区分于领域，将不同领域的系统功能分开，每个大的领域下又会有一些子域，领域划分之后需要根据功能特性进行划分，例如着重于数据计算、用户交互、中间流程等，需要区分业务面向的对象。

当面向对象相同时，还需要做一些额外的分界，比如作者手中有个安全风控的系统，但是其中一部分功能涉及到大量流量不是很稳定，虽然作者通过自研的计算限流GitHub - SongTing0711/count-limit、redis回滚GitHub - SongTing0711/redis-transaction、分布式代理锁GitHub - SongTing0711/distributed-proxy-lock等工具控制了这种不稳定的流量和计算，但是做系统不能寄希望于没有更加异常的情况。

另外一部分功能涉及到B端用户上千万的资金流转，不能出任何问题，因此将这两部分功能再次划分成为两个微服务。

而在Gavin的文章中提到了B端系统C端化，这其实站在了一个比较高的维度看系统分层，逻辑和服务与产品、产线解耦，博主认为是领域、功能划分的理念，功能可配置、可扩展、可插拔的方式可以用于敏捷开发、快速迭代。

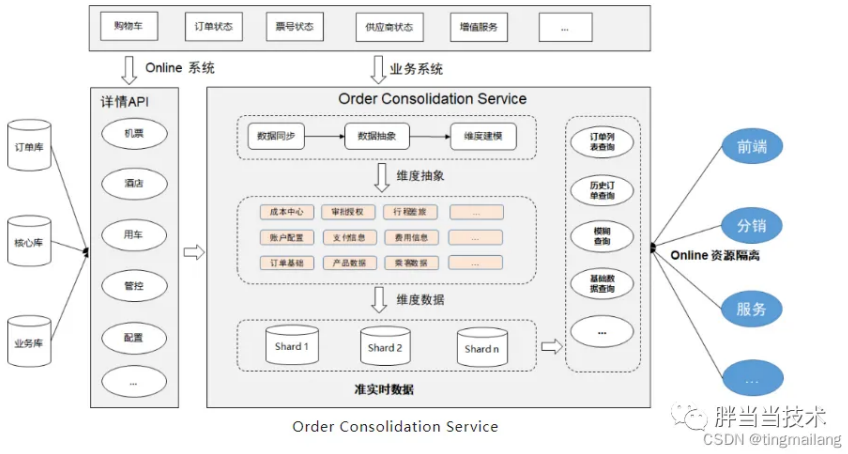
2、技术架构

2.1、数据

数据架构的区分主要在于实时性要求，对于强一致的实时性数据无论是直接查询DB还是业务侧处理到redis再对外都会额外增加一份风险，比如数据量过大—超时、查询较多—CPU、内存飙升、redis与DB数据一致性问题，会导致实时数据的风险随着系统体量增大成几何系数上升。

这里Gavin对于数据问题的感受和博主是一致的，Gavin的解决方案参考了数仓建模，形成一种准实时的业务数据处理流程，暴露给上下游的各个领域。

这里基本是可以满足大部分业务场景，准实时的数据一般博主是使用es进行存储的，但是受限于话语权和业务很难要求一部分场景也使用准实时数据。博主比较推荐的是京东的HotKey--热点探测工具，通过简单的api调用将热点数据缓存在本地，但是内存占用会变高，需要权衡。



（引用1）

2.2、api交互

微服务的交互反复、链路过长、分布式事务一直是服务拆分被诟病的点，但是大量的系统功能又不得不拆分，作者遇到过的很多情况不单单是开发、维护的复杂，还有一些情况是有风险的。

举个例子，门店库存变化之后推出mq消息，但是有的下游系统一收到mq就来查数据，这时候事务都还没提交，查到就是不对的。有的同学说我提交之后再发mq就好了，但是一般暴露出去的api都是查从库的，从库跟主库的同步也是需要时间的，这种风险依然存在。一般会尽量要求下游不要这样查，实在不行的会在mq把他需要的信息查出来直接带出去，但是这种做法也不是最佳的。

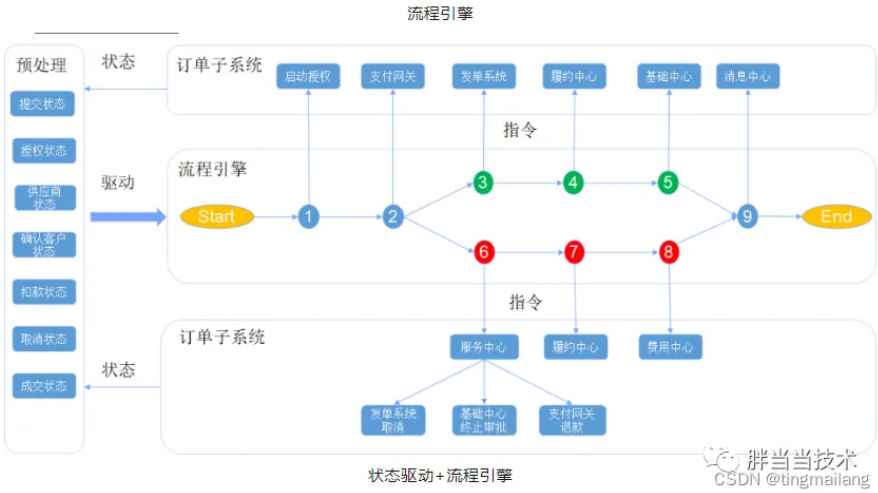
这种时候往往要配合重试机制，和时间序列，比如我收到的mq的多少毫秒发出的，如果查到的数据是更新时间在这个时间之前，那就应该停顿再重试。

Gavin提到需要避免各个服务之间相互直接调用，相互订阅广播通知导致系统的耦合和边界不清晰，其实也是在反映交互的问题。

2.3、流程

做软件就是想要把统一复用的工作由人转移到机器，如果说架构上单一特殊的点太多，完全没法抽象出统一的技术流程，再牛的软件也会成为码山，前面的人设计的时候不考虑，后面的人不敢改。

抽象出流程引擎把每一个流程独立化，再由配置、事件进行排列组合是Gavin极力推荐的。



（引用2）

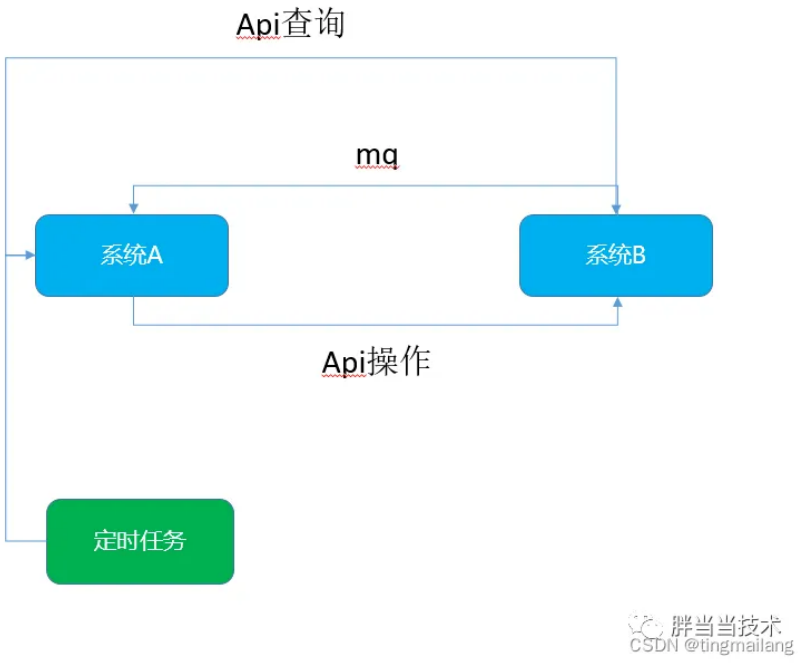
三、示例

博主在之前的分析中讲述了一些例子，这里再用一些实际示例介绍架构设计的复杂性。

1、异步结果

有时候调用下游，下游短时间是给不出一个结果的，这时候一般会采用异步通知，mq或者把结果通过接口暴露出去，上游定时去查，这两种各有优缺点，等mq会有处理失败的情况，异常被吞掉或者mq没有支持ack确认，会导致结果处理问题。

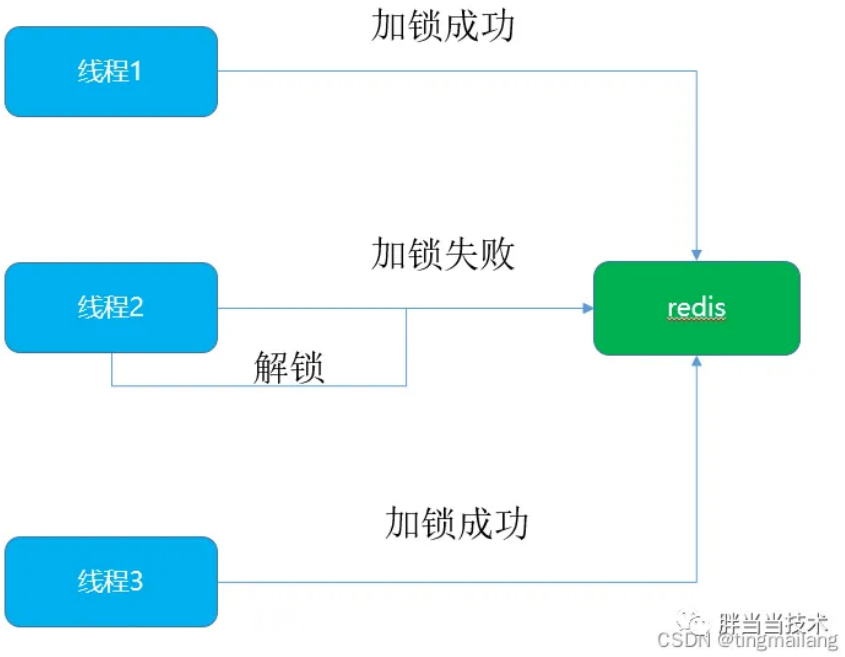
定时任务去查就是会有延迟，为了安全、实时，博主一般会两个一起用。



2、幂等性

2.1、并发

Api或者mq的幂等性是个老生常谈的问题，一般是要结合分布式锁防止并发的，博主遇到过一种情况是加锁的问题，开发人员没有判断是否本线程加锁就进行了解锁，导致第三个线程抢到锁，产生了并发问题，所以博主基于收拢规范和敏捷开发做了分布式代理锁工具GitHub - SongTing0711/distributed-proxy-lock。



2.2、顺序消费

但是只防止并发还不行，会存在重试或者乱序消费之类的问题，这时候就需要按照某一个key进行hash顺序消费，比如门店id、订单编号。

2.3、时间戳

还有时候需要看时间戳，比如机器心跳，因为即使是顺序消息，但是不同时间戳进入mq，如果当前消息的时间戳比上一条早，就应该被丢弃。

2.4、唯一键

还有一些处理情况需要在业务数据存储上游调用或者发布消息是的唯一键，用来防止重复处理，比如博主调用财务系统会给他一个唯一单号，根据这个唯一单号财务只处理一次，防止调用过程中超时熔断之类的情况导致调用成功但是上游却不知道。

3、自动矫正

由于分布式环境的数据隔离，数据存在不一致的情况，博主一般会用定时任务或者在流量不高的场景下进行数据比对，即比较DB和redis，也比较和其他服务的，发生不一致就清理掉生成一条正确的。

4、自动补偿

这种在上面很多示例中都有使用，定时任务补偿处理、下游异步结果补偿通知等等。

5、限流

流量是不稳定的，不管是物联网、互联网都一样，网络堆积、操作不规范、用户狂点等等，都会导致流量洪峰，所以哪家都逃不了限流这个问题。

5.1、并发限流

大多数成熟的工具都是针对并发的，比如谷歌的ratelimiter，各个大厂也基本都会自己包一个工具，嵌入自己的框架，更加符合工作环境的限流。

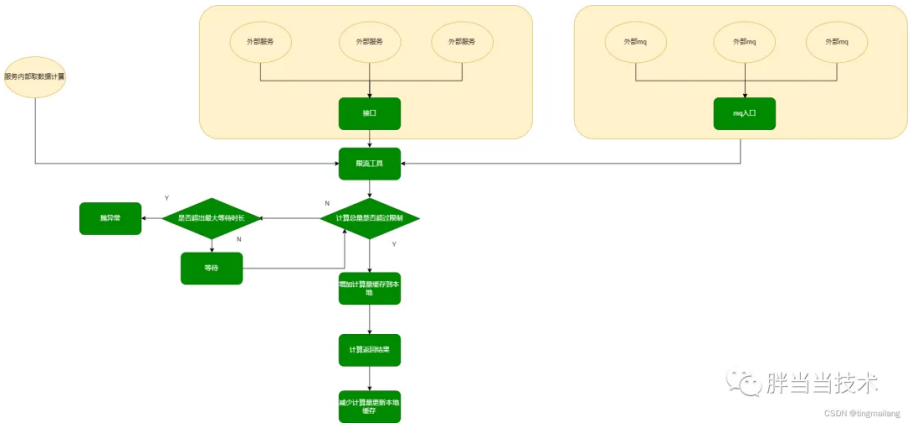
5.2、计算限流

这个是作者工作遇到问题发觉的限流需要，许多需求计算量都在扩大，比如合同下的门店会有三四千个，计算这些门店的数据在进行聚合，对于服务的内存和接口执行时间有着很大的影响。

针对越来越大容量、并发高的接口或者其他计算方法，同一时间在运行的计算维度进行限制，比如要计算门店设备，服务最多支持10000个门店同时在计算，相当于把资源到计算的对象维度。基于这个原因，作者编写了一个计算限流工具。

github地址：GitHub - SongTing0711/count-limit

其实这也可以说是资源维度的限流。



6、技术调研

到这里就很宽泛了，要求也比较高，那就是你在工作中发现了什么困境？然后调研市面上的中间件可以解决这个问题？部署起来代价大不大？有没有备选方案？我们自己能不能参照这个设计去研发工具？

而且这种调研基本只能是加班时间去搞，得有样例，不能是简单的文档调研。所以需要你加班的时候，把工具用起来，甚至麻烦的要在本地搭建服务端。

比如作者针对分布式事务搞得大家经常要去刷数据，所以调研阿里的seata。

针对机器跳变会导致预警、修理等业务场景的频繁无效处理，导致大量的物联网心跳被消费，浪费性能和资源。这里其实就可以使用京东武伟锋的HotKey，比如设置规则两秒接收五次以上的跳变就标记为问题，停止无用的处理，并且识别出之后交给硬件进行实际检测。

如果再深入的阅读源码，说出他的原理那就更好了，这就是另外一个面试题了，有没有阅读过什么源码，如果只是背书一样的把spring讲讲，面试官也没什么兴趣听。后面作者会结合自己阅读的源码讲讲面试经验。

四、总结

博主这里基于自己的经验结合对Gavin的文章做了系统架构设计的分析，这里特别感谢Gavin编写的干货｜携程商旅订单系统架构设计和优化实践，各位同学有兴趣可以去看看。

有了更多的了解，就可以像聊天一样跟面试官回答这个问题，另外要勇于肯定自己的观点，哪怕是错了，只要有自己的思考和依据，面试依然会给你高分。

设计 2　面试 5　架构 10

设计 · 目录

下一篇 · 开源工具（三）计算限流

个人观点，仅供参考