二

02 研发工程师如何用架构师视角回答架构设计方案?

今天这一讲，我想跟你聊一聊如何用架构师的视角进行技术面试。

面试时你是否常被问到这样的问题：“你之前是如何设计这个系统（或子系统/模块/功能）的？请介绍你的思路。”

很多研发同学在听到类似的面试题时，往往忽略“系统设计思路”关键词，而是陷入某个技术点细节里，让面试官听得一头雾水。这样即使技术再好，面试官也很难给你打高分，更可能认为你的设计能力不足，没有全局思维。

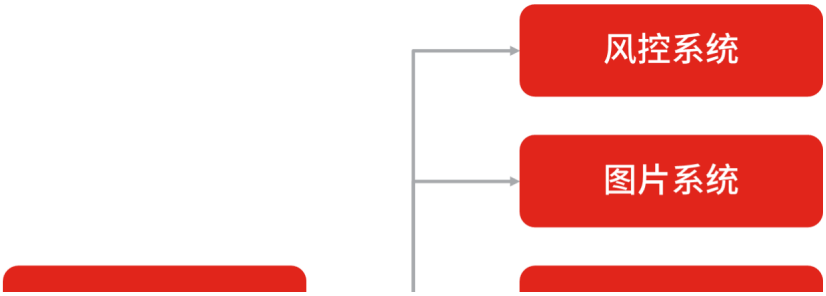
而要想答得更好，你要用架构师的视角回答，即从全局技术视角阐述设计的过程。接下来我会通过一个案例，讲解如何从全局技术视角介绍自己的技术方案。

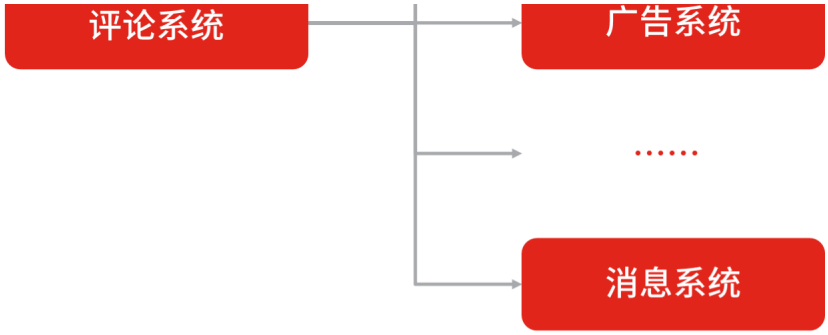
案例背景

在电商中，当用户发表一条商品评论，后台的逻辑是点评系统会调用一系列的远程 API 接口，如调用风控系统、广告系统、消息系统……几个甚至十几个系统的接口。

在业务建设之初，考虑到快速开发与上线，商品评论发布是通过同步 RPC（Remote Procedure Call，远程过程调用）远程调用各系统接口完成的。这种方式在系统少、逻辑简单的阶段很符合实际情况的设计。

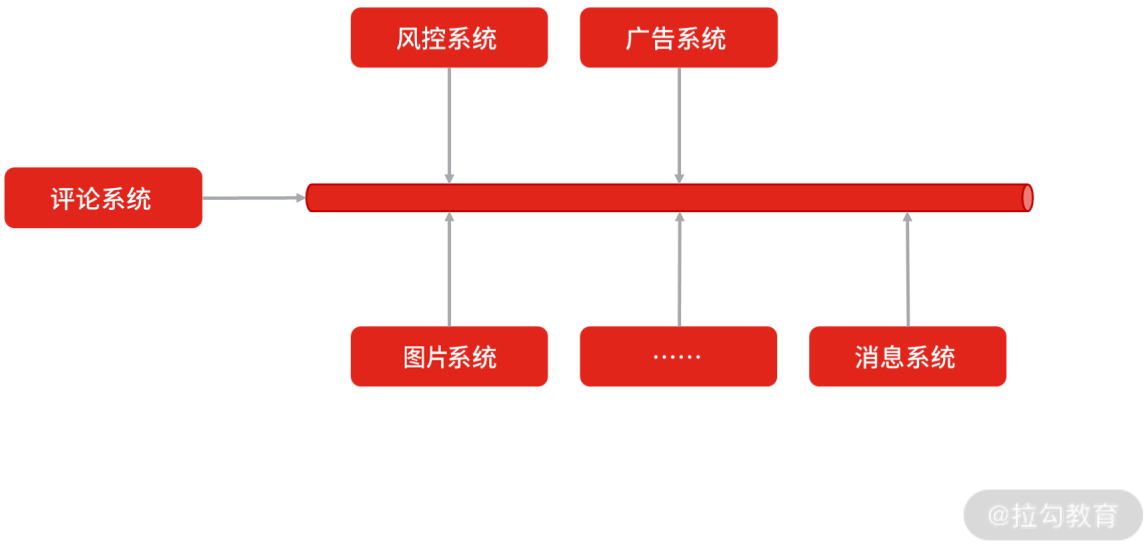
但随着业务快速发展，通过 RPC 同步调用的问题逐渐暴露出来。由于过多地依赖其他系统，导致评论发布的接口性能很低，可用性也容易受到其他系统影响。而且每当点评系统需求上线时，其他系统都需要跟着进行联调测试，导致需求迭代速度缓慢。





系统

在做系统架构升级改造时，如果你有互联网设计理念，会很容易想到问题在于系统间的耦合度太高。解决办法就是采用异步化解耦，从而通过引入 MQ 消息管道，在架构上进行系统业务逻辑拆分，将原本强依赖的系统间的同步 RPC 调用变成异步消息触发，如下面图片中的架构所示：



架构

案例分析

对于上面的案例，假设你是应聘者，当被问“如何做这个点评系统的改造？”时，你会怎么回答？你会不会直截了当地说“我引入了 MQ 消息队列，做了系统解耦，采用异步消息通知的方式来触发系统调用”呢？

在互联网系统设计方案如此透明的今天，随便在网上搜一下都会有大量类似的解决方案。以上回答不但不会让面试官满意，甚至有可能令人怀疑你的项目经历的真实性。

作为研发工程师，正确的回答方式是要让面试官知道你解决问题的思维。相比一上来就说用了什么技术，阐述解决思维更能证明你的能力，**因为解决技术问题的方法有很多，这是“术”，但解决技术问题的底层思维逻辑是一样的，这是“道”。**

面对此类问题，我总结了如下四个层面的答案：

1. 谈复杂来源；
2. 谈解决方案；
3. 谈评估标准；
4. 说技术实现。

问题解答

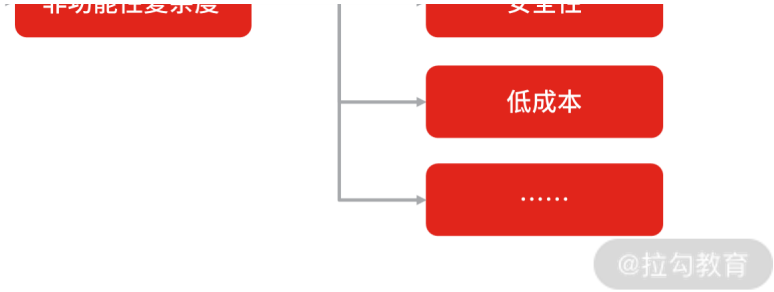
我还是拿上面的例子来分析如何回答此类问题。

复杂来源

之所以要先分析系统的复杂度，是因为只有正确分析后才能明确设计原则，进而设计架构方案，整体项目才不会找错方向。

如果一个系统本来因业务逻辑复杂导致功能耦合严重，你却设计了一个 TPS（Transactions Per Second，每秒事务处理量）达到 10000/秒 的高性能架构，那么即使架构性能再优秀，也没有现实意义，因为技术设计没有解决主要问题的复杂度。这是很多研发工程师的通病，设计偏离了方向，只是为了设计而设计。

那么如何正确评估系统的复杂度呢？ 互联网软件通常分为功能性的复杂度和非功能性的复杂度两种。我将分析过程制作成了一张图片。



复杂度评估

从功能性复杂度方面来看，你可以从案例中得知，产品业务发展快速、系统越来越多、协作效率越来越低。作为系统负责人，你敏锐地发现问题根源在架构上各业务子系统强耦合。于是你引入消息队列解耦各系统，这是系统业务领域带来的本质上的复杂度，也就是功能性的复杂度，解决的是系统效率的问题。

此外，对于互联网系统设计，还需要考虑非功能性的复杂度，例如高性能、高可用和扩展性等的复杂度的设计。

从非功能性复杂度方面来看，我们假设系统用户每天发送 100 万条点评，那么点评的消息管道一天会产生 100 万条消息，再假设平均一条消息有 10 个子系统读取，那么每秒的处理数据，即点评消息队列系统的 TPS 和 QPS（Queries Per Second，每秒查询次数）就分别是 11（1000000/60*60*24）和 115（10000000/60*60*24）。

不过系统的读写不是完全平均的，设计的目标应该以峰值来计算，即取平均值的 4 倍。于是点评消息队列系统的 TPS 变成了 44，QPS 变成了 460，这个量级的数据意味着并不需要设计高性能架构方案。

接着还要考虑业务规模发展。架构设计的目标应该满足未来业务增长，我们把未来业务增长的预估峰值设定为目前峰值的 4 倍，这样最终的性能要求分别是：TPS 为 176，QPS 是 1840。这样的读写指标还达不到系统压测的性能基线，所以可以确定的是点评系统的复杂度并不在高性能问题上。

对于点评系统来说，还需要考虑高可用的问题。假设点评系统的消息队列挂掉，将导致用户评论发送失败，当然在用户体验层面，解决方式可以在页面端提示用户重新操作，但如果问题影响到了点评消息的读取，导致评论没有走风控策略，就会造成严重的影响。所以高可用性是点评系统的设计复杂度之一，包括点评写入、点评存储，以及点评消息的读取，都需要保证高可用性。

为了方便理解非功能性的复杂度，我只分析了“高性能”和“高可用”这两点，在实际应用中，不同的公司或者团队可能还有其他方面的复杂度分析。例如有的公司会考虑安全性，有的公司会考虑成本等。

所以综合分析来看，点评系统改造的复杂度来源于两点。

- **功能性复杂度**：要解决业务发展带来的系统耦合、开发效率缓慢问题。
- **非功能性复杂度**：要保证系统的高可用性。

解决方案

在确定了系统面临的主要复杂度问题后，就有了明确的方案设计目标，这时就可以开始进行架构方案设计了。我同样会结合本文的案例场景，谈谈点评系统消息管道的架构设计解决方案。

1. **采用开源的 MQ 消息管道。**目前 MQ 消息管道有很多开源解决方案，比如 Kafka、RocketMQ、RabbitMQ 等。在实际项目中，你可以根据不同的应用场景选择合适的成熟开源消息队列方案，这是很多公司常用的做法。
2. **采用开源的 Redis 实现消息队列。**方案 1 虽然应用了开源 MQ 实现点评消息的通信，但是因为引入一个消息中间件就会带来运维成本，所以方案 2 可以基于轻量级的 Redis 实现，以降低系统的维护成本和实现复杂度。
3. **采用内存队列 + MySQL 来实现。**方案 2 中虽然应用了较为轻量级的 Redis 来实现，但是还需要引入一个缓存系统，同样也会带来运维成本，所以方案 3 是直接基于 MySQL 实现，即基于内存队列的方式，异步持久化到数据库，然后通过定时任务读取 MySQL 中的消息并处理。

一般情况，你至少要设计两到三套备选方案，考虑通过不同的技术方式来解决问题。方案设计不用过于详细，而是要确定技术的可行性和优缺点。

评估标准

设计完三套解决方案之后，摆在眼前的问题就是需要选择最合适的一个。**这就需要一套评估标准了。**

在互联网软件架构中，架构师常常会把一些通用的设计原则写到设计文档中，比如设计松耦合、系统可监控，这些原则似乎不常用，但好的架构师会通过设计原则来控制项目的技术风险。比如系统无单点，限制了系统技术方案不可出现单点服务的设计；再如系统可降级，限制了系统有具备降级的能力，进而约束了开发人员需要设计数据兜底的技术方案。

这些看似不重要的设计原则，其实是评估架构解决方案的重要手段。做系统架构，需要站在更高的层面考虑系统的全局性关注点，比如性能、可用性、IT 成本、投入资源、实现复杂度、安全性、后续扩展性等。这在不同场景的不同阶段会起到决定性作用。

那么针对案例中的点评系统来说，要如何评估方案呢？这要从点评系统的复杂度来源进行评估。

- **点评系统功能性复杂度**

点评系统的功能性复杂度问题，本质上是随着业务发展带来的系统开发效率问题。解决这个问题要试着站得更高一些，以部门负责人的视角，考虑现有研发团队的能力素质、IT 成本、资源投入周期等因素是否匹配上面三种架构解决方案。

- **点评系统非功能性复杂度**

为了解决系统的高可用，可以参考三个设计原则。

第一个是系统无单点原则。首先要保证系统各节点在部署的时候至少是冗余的，没有单点。很显然三种设计方案都支持无单点部署方式，都可以做到高可用。

第二个是可水平扩展原则。对于水平扩展，MQ 和 Redis 都具有先天的优势，但内存队列 + MySQL 的方式则需要做分库分表的开发改造，并且还要根据业务提前考虑未来的容量预估。

第三个是可降级原则。降级处理是当系统出现故障的时候，为了系统的可用性，选择有损的或者兜底的方式提供服务。

常用手段主要有三种。

- **限流**，即抛弃超出预估流量外的用户。
- **降级**，即抛弃部分不重要的功能，让系统提供有损服务，如商品详情页不展示宝贝收藏的数量，以确保核心功能不受影响。
- **熔断**，即抛弃对故障系统的调用。一般情况下熔断会伴随着降级处理，比如展示兜底数据。

针对案例场景中三个解决方案的降级策略，在一般的情况下，我们默认数据库是不可降级的，MQ 和 Redis 都可以通过降级到数据库的方式做容灾处理。所以案例中的三个解决方案，MQ 和 Redis 要考虑降级到 MySQL 或其他方式，这里就还需要根据情况投入降级的开发成本。

对于本节课的案例我不评价哪个更好，你在多个解决方案中做选择时，**不要陷入某个纯粹技术点的优劣之争，那样很难有结果，越大的项目越明显。** 通常来说，方案没有优劣之分，而是要看哪个更适合当下的问题，只要架构满足一定时期内的业务发展就可以。

你要知道，作为技术人，考虑问题的方式要比具体的选型结果更为重要，这是面试的加分

点。

技术实现

在确定了具体的架构解决方案之后，**需要进一步说明技术上的落地实现方式和深层原理**，如果你最终选择基于 Redis 来实现消息队列，那么可以有几种实现方式？各自的优缺点有哪些？对于这些问题，要做到心里有数。比如，基于 Redis List 的 LPUSH 和 RPOP 的实现方式、基于 Redis 的订阅或发布模式，或者基于 Redis 的有序集合（Sorted Set）的实现方式，你可以自行搜索，我不再赘述。

总结

最后，我把今天的“四步回答法”做个总结，加深每一步你需要掌握的注意点。

1. 在回答系统复杂度来源的时候，要注意结合具体的业务场景和业务发展阶段来阐述。业务场景表明了业务的独特性，发展阶段表明了业务的成熟度，因为同一业务场景在不同阶段产生的矛盾也是不同的。
2. 在回答解决方案的时候，有价值的解决方案一定是建立在明确复杂度来源基础之上的。所以在设计架构的时候才分主要问题和次要问题，主要问题是必须要解决的点，次要问题可以根据实际情况进行取舍。
3. 在回答如何评估架构方案时，至少要从功能性和非功能性两个角度出发判断方案的合理性。对于很难决策的方案，要从更高的视角（比如技术负责人、业务负责人的角度）进行考量。
4. 在技术实现的细节上，要尽量讲出技术的实现原理，不要浮于表面的框架组合。

到这里，我们已经知道了如何用架构师的视角进行技术面试，那么你现在理解**什么是架构师视角了吗？其实简单一句话，所谓的架构师视角就是全局的视角，这里的全局包括空间全局和时间全局，在空间全局上你要看到整个系统的领域边界，在时间全局上你要看到整个系统的发展周期。**

上一页 下一页