如何准备系统设计面试?

系统设计在面试中一定是最让面试者头疼的事情之一。 因为系统设计相关的问题通常是开放式的，所以没有标准答案。你在和面试官思想的交流碰撞中会慢慢优化自己的系统设计方案。理论上来说，系统设计面试也是和面试官一起一步一步改进原有系统设计方案的过程。

系统设计题往往也非常能考察出面试者的综合能力，回答好的话，很容易就能在面试中脱颖而出。不论是对于参加社招还是校招的小伙伴，都很有必要重视起来。

接下来，我会带着小伙伴们从我的角度出发来谈谈：**如何准备面试中的系统设计部分。**

由于文章篇幅有限，就不列举实际例子了，可能会在后面的文章中单独提一些具体的例子。

个人能力有限。如果文章有任何需要改善和完善的地方，欢迎在评论区指出，共同进步！

系统设计面试一般怎么问?

我简单总结了一下系统设计面试相关问题的问法：

1. 设计一个某某系统比如秒杀系统、微博系统、抢红包系统、短网址系统。
2. 设计某某系统中的一个功能比如哔哩哔哩的点赞功能。
3. 设计一个框架比如 RPC 框架、消息队列、缓存框架、分布式文件系统等等。
4. 某某系统的技术选型比如缓存用 `Redis` 还是 `Memcached` 、网关用 `Spring Cloud Gateway` 还是 `Netflix Zuul2` 。

系统设计怎么做?

我们将步骤总结成了以下 4 步。

Step1:问清楚系统具体要求

当面试官给出了系统设计题目之后，一定不要立即开始设计解决方案。 你需要先理解系统设计的需求：功能性需求和非功能性需求。

为了避免自己曲解题目所想要解决的问题，你可以先简要地给面试官说说自己的理解，

为啥要询问清楚系统的功能性需求也就是说系统包含哪些功能呢?

毕竟，如果面试官冷不丁地直接让你设计一个微博系统，你不可能把微博系统涵盖的功能比如推荐信息流、会员机制等一个一个都列举出来，然后再去设计吧！你需要筛选出系统所提供的核心功能（**缩小边界范围**）！

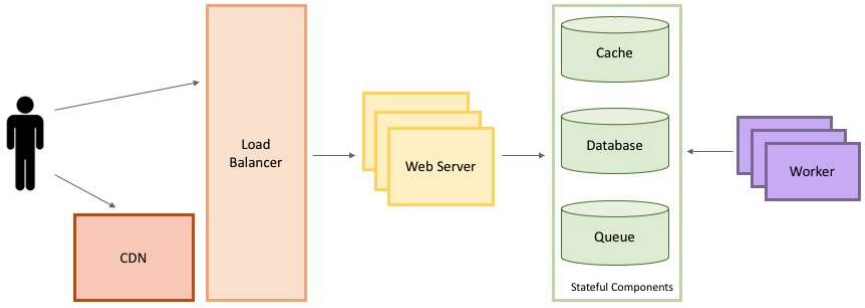
为啥要询问清楚系统的非功能性需求或者说约束条件比如系统需要达到多少QPS呢?

让你设计一个1w人用的微博系统和100w人用的微博系统能一样么？不同的约束系统对应的系统设计方案肯定是不一样的。

Step2:对系统进行抽象设计

我们需要在一个 High Level 的层面对系统进行设计。

你可以画出系统的抽象架构图，这个抽象架构图中包含了系统的一些组件以及这些组件之间的连接。



Step3:考虑系统目前需要优化的点

对系统进行抽象设计之后，你需要思考当前抽象的系统设计有哪些需要优化的点，比如说：

1. 当前系统部署在一台机器够吗？是否需要部署在多台机器然后进行负载均衡呢?
2. 数据库处理速度能否支撑业务需求？是否需要给指定字段加索引？是否需要读写分离？是否需要缓存?
3. 数据量是否大到需要分库分表?
4. 是否存在安全隐患?

5. 系统是否需要分布式文件系统？
6. .....

Step4:优化你的系统抽象设计

根据 Step 3 中的“系统需要优化的点” 对系统的抽象设计做进一步完善。

系统设计该如何准备？

知识储备

系统设计面试非常考察你的知识储备，系统设计能力的提高需要大量的理论知识储备。比如说你要知道大型网站架构设计必备的三板斧：

1. **高性能架构设计**：熟悉系统常见性能优化手段比如引入 **读写分离**、**缓存**、负载均衡、**异步** 等等。
2. **高可用架构设计**：CAP理论和BASE理论、通过集群来提高系统整体稳定性、超时和重试机制、应对接口级故障：**降级**、**熔断**、**限流**、排队。
3. **高扩展架构设计**：说白了就是懂得如何拆分系统。你按照不同的思路来拆分软件系统，就会得到不同的架构。

实战

虽然懂得了理论，但是自己没有进行实践的话，很多东西是无法体会到的！

因此，你还要 **不断通过实战项目锻炼自己的系统设计能力。**

保持好奇心

多思考自己经常浏览的网站是怎么做的。比如：

1. 你刷微博的时候可以思考一下微博是如何记录点赞数量的？
2. 你看哔哩哔哩的时候可以思考一下消息提醒系统是如何做的？
3. 你使用短链系统的时候可以考虑一下短链系统是如何做的？
4. .....

技术选型

实现同样的功能，一般会有多种技术选择方案，比如缓存用 `Redis` 还是 `Memcached`、网关用 `Spring Cloud Gateway` 还是 `Netflix Zuul2`。很多时候，面试官在系统设计面过程中会具体到技术的选型，因而，你需要区分不同技术的优缺点。

系统设计面试必知

> 系统设计的时候必然离不开描述性能相关的指标比如 QPS。

性能相关的指标

响应时间

响应时间RT(Response-time)就是用户发出请求到用户收到系统处理结果所需要的时间。

RT是一个非常重要且直观的指标，RT数值大小直接反应了系统处理用户请求速度的快慢。

并发数

并发数可以简单理解为系统能够同时供多少人访问使用也就是说系统同时能处理的请求数量。

并发数反应了系统的负载能力。

QPS 和 TPS

- **QPS（Query Per Second）**：服务器每秒可以执行的查询次数；
- **TPS（Transaction Per Second）**：服务器每秒处理的事务数（这里的一个事务可以理解为客户发出请求到收到服务器的过程）；

书中是这样描述 QPS 和 TPS 的区别的。

> QPS vs TPS：QPS 基本类似于 TPS，但是不同的是，对于一个页面的一次访问，形成一个TPS；但一次页面请求，可能产生多次对服务器的请求，服务器对这些请求，就可计入“QPS”之中。如，访问一个页面会请求服务器2次，一次访问，产生一个“T”，产生2个“Q”。

吞吐量

吞吐量指的是系统单位时间内系统处理的请求数量。

一个系统的吞吐量与请求对系统的资源消耗等紧密关联。请求对系统资源消耗越多，系统吞吐能力越低，反之则越高。

TPS、QPS都是吞吐量的常用量化指标。

- **QPS（TPS）** = 并发数/平均响应时间(RT)
- **并发数** = QPS * 平均响应时间(RT)

系统活跃度

介绍几个描述系统活跃度的常见名词，建议牢牢记住。你不光会在回答系统设计面试题的时候碰到，日常工作中你也会经常碰到这些名词。

PV(Page View)

访问量, 即页面浏览量或点击量，衡量网站用户访问的网页数量；在一定统计周期内用户每打开或刷新一个页面就记录1次，多次打开或刷新同一页面则浏览量累计。UV 从网页打开的数量/刷新的次数的角度来统计的。

UV(Unique Visitor)

独立访客，统计1天内访问某站点的用户数。1天内相同访客多次访问网站，只计算为1个独立访客。UV 是从用户个体的角度来统计的。

DAU(Daily Active User)

日活跃用户数量。

MAU(monthly active users)

月活跃用户人数。

举例：某网站 DAU为 1200w， 用户日均使用时长 1 小时，RT为0.5s，求并发量和QPS。

平均并发量 = DAU（1200w）* 日均使用时长（1 小时，3600秒） /一天的秒数（86400）=1200w/24 = 50w

真实并发量（考虑到某些时间段使用人数比较少） = DAU（1200w）* 日均使用时长（1 小时，3600秒） /一天的秒数-访问量比较小的时间段假设为8小时（57600）=1200w/16 = 75w

峰值并发量 = 平均并发量 * 6 = 300w

QPS = 真实并发量/RT = 75W/0.5=150w/s

常用性能测试工具

后端常用

既然系统设计涉及到系统性能方面的问题，那在面试的时候，面试官就很可能会问：**你是如何进行性能测试的?**

推荐 4 个比较常用的性能测试工具：

1. **Jmeter**： Apache JMeter 是 JAVA 开发的性能测试工具。
2. **LoadRunner**：一款商业的性能测试工具。
3. **Galtling**： 一款基于Scala 开发的高性能服务器性能测试工具。
4. **ab**： 全称为 Apache Bench 。Apache 旗下的一款测试工具，非常实用。

没记错的话，除了 **LoadRunner** 其他几款性能测试工具都是开源免费的。

前端常用

1. **Fiddler**：抓包工具，它可以修改请求的数据，甚至可以修改服务器返回的数据，功能非常强大，是Web 调试的利器。
2. **HttpWatch**: 可用于录制HTTP请求信息的工具。

常见软件的QPS

这里给出的 QPS 仅供参考，实际项目需要进行压测来计算。

- **Nginx**： 一般情况下，系统的性能瓶颈基本不会是 Nginx。单机 Nginx 可以达到 30w +。
- **Redis**: Redis 官方的性能测试报告：https://redis.io/topics/benchmarks <https://redis.io/topics/benchmarks> 。从报告中，我们可以得出 Redis 的单机 QPS 可以达到 8w+（CPU性能有关系，也和执行的命令也有关系比如执行 SET 命令甚至可以达到10w+QPS）。
- **MySQL**: MySQL 单机的 QPS 为 大概在 4k 左右。

- **Tomcat**：单机 Tomcat 的QPS 在 2w左右。这个和你的 Tomcat 配置有很大关系，举个例子Tomcat 支持的连接器有 **NIO**、**NIO.2** 和 **APR**。 `AprEndpoint` 是通过 JNI 调用 APR 本地库而实现非阻塞 I/O 的，性能更好，Tomcat 配置 APR 为 连接器的话，QPS 可以达到 3w左右。更多相关内容可以自行搜索 Tomcat 性能优化。

系统设计原则

合适优于先进 > 演化优于一步到位 > 简单优于复杂

常见的性能优化策略

性能优化之前我们需要对请求经历的各个环节进行分析，排查出可能出现性能瓶颈的地方，定位问题。

下面是一些性能优化时，我经常拿来自问的一些问题：

1. 当前系统的SQL语句是否存在问题？
2. 当前系统是否需要升级硬件？
3. 系统是否需要缓存？
4. 系统架构本身是不是就有问题？
5. 系统是否存在死锁的地方？
6. 数据库索引使用是否合理？
7. 系统是否存在内存泄漏？（Java 的自动回收内存虽然很方便，但是，有时候代码写的不好真的会造成内存泄漏）
8. 系统的耗时操作进行了异步处理？
9. .....

性能优化必知法则

SQL优化，JVM、DB，Tomcat参数调优 > 硬件性能优化（内存升级、CPU核心数增加、机械硬盘—>固态硬盘等等） > 业务逻辑优化/缓存 > 读写分离、集群等 > 分库分表

系统设计面试的注意事项

想好再说

没必要面试官刚问了问题之后，你没准备好就开始回答。这样不会给面试官带来好印象的！系统设计本就需要面试者结合自己的以往的经验进行思考，这个过程是需要花费一些时间的。

没有绝对的答案

系统设计没有标准答案。重要的是你和面试官一起交流的过程。

一般情况下，你会在和面试官的交流过程中，一步一步完成系统设计。这个过程中，你会在面试官的引导下不断完善自己的系统设计方案。

因此，你不必要在系统设计面试之前找很多题目，然后只是单纯记住他们的答案。

勿要绝对

系统设计没有最好的设计方案，只有最合适的设计方案。这就类比架构设计了：**软件开发没有银弹，架构设计的目的就是选择合适的解决方案。 何为银弹？** 狼人传说中，只有银弹(银质子弹)才能制服这些猛兽。对应到软件开发活动中，银弹特指开发者们寻求的一种克服软件开发这个难缠的猛兽的“万能钥匙🔑”。

权衡利弊

知道使用某个技术可能会为系统带来的利弊。比如使用消息队列的好处是解耦和削峰，但是，同样也让系统可用性降低、复杂性提高，同时还会存在一致性问题（消息丢失或者消息未被消费咋办）。

慢慢优化

刚开始设计的系统不需要太完美，可以慢慢优化。

不追新技术

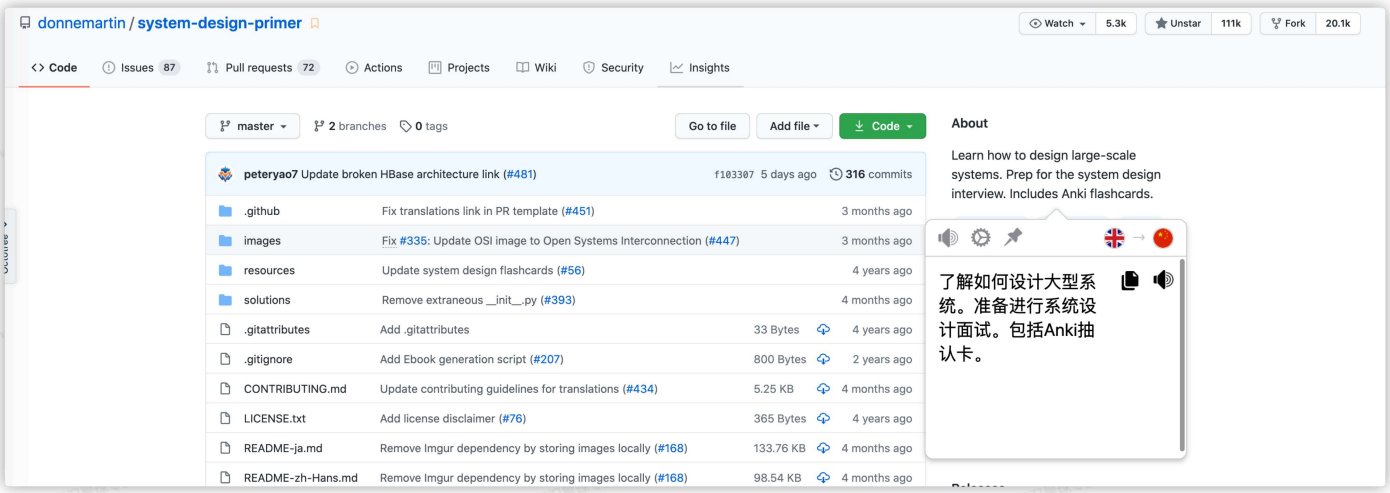
使用稳定的、适合业务的技术，不必要过于追求新技术。

追简避杂

系统设计应当追求简单避免复杂。KISS（Keep It Simple, Stupid）原则——保持简单，易于理解。

总结

这篇文章简单带着小伙伴们分析了一下系统设计面试。如果你还想要深入学习的话，可以参考： https://github.com/donnemartin/system-design-primer <https://github.com/donnemartin/system-design-primer> 。



参考

1. https://github.com/donnemartin/system-design-primer <https://github.com/donnemartin/system-design-primer>
2. https://www.acecodeinterview.com/intro/ <https://www.acecodeinterview.com/intro/>
3. https://gist.github.com/vasanthk/485d1c25737e8e72759f <https://gist.github.com/vasanthk/485d1c25737e8e72759f>