高可用：如何设计一个高可用系统？

一篇短小的文章，面试经常遇到的这个问题。本文主要包括下面这些内容：

1. 高可用的定义
2. 哪些情况可能会导致系统不可用？
3. 有些提高系统可用性的方法？只是简单的提一嘴，更具体内容在后续的文章中介绍，就拿限流来说，你需要搞懂：何为限流？如何限流？为什么要限流？如何做呢？说一下原理？。

什么是高可用？可用性的判断标准是啥？

高可用描述的是一个系统在大部分时间都是可用的，可以为我们提供服务的。高可用代表系统即使在发生硬件故障或者系统升级的时候，服务仍然是可用的。

一般情况下，我们使用多少个 9 来评判一个系统的可用性，比如 99.9999% 就是代表该系统在所有的运行时间中只有 0.0001% 的时间是不可用的，这样的系统就是非常非常高可用的了！当然，也会有系统如果可用性不太好的话，可能连 9 都上不了。

哪些情况会导致系统不可用？

1. 黑客攻击；
2. 硬件故障，比如服务器坏掉。
3. 并发量/用户请求量激增导致整个服务宕掉或者部分服务不可用。
4. 代码中的坏味道导致内存泄漏或者其他问题导致程序挂掉。
5. 网站架构某个重要的角色比如 Nginx 或者数据库突然不可用。
6. 自然灾害或者人为破坏。
7. ……

有哪些提高系统可用性的方法？

1. 注重代码质量，测试严格把关

我觉得这个是最最最重要的，代码质量有问题比如比较常见的内存泄漏、循环依赖都是对系统可用性极大的损害。大家都喜欢谈限流、降级、熔断，但是我觉得从代码质量这个源头把关是首先要做好的一件很重要的事情。如何提高代码质量？比较实际可用的就是 CodeReview，不要在乎每天多花的那 1 个小时左右的时间，作用可大着呢！

另外，安利这个对提高代码质量有实际效果的宝贝：

1. sonarqube ： 保证你写出更安全更干净的代码！（ps: 目前所在的项目基本都会用到这个插件）。
2. Alibaba 开源的 Java 诊断工具 Arthas 也是很不错的选择。
3. IDEA 自带的代码分析等工具进行代码扫描也是非常非常棒的。

2.使用集群，减少单点故障

先拿常用的 Redis 举个例子！我们如何保证我们的 Redis 缓存高可用呢？答案就是使用集群，避免单点故障。当我们使用一个 Redis 实例作为缓存的时候，这个 Redis 实例挂了之后，整个缓存服务可能就挂了。使用了集群之后，即使一台 Redis 实例，不到一秒就会有另外一台 Redis 实例顶上。

3.限流

流量控制（flow control），其原理是监控应用流量的 QPS 或并发线程数等指标，当达到指定的阈值时对流量进行控制，以避免被瞬时的流量高峰冲垮，从而保障应用的高可用性。——来自 alibaba-Sentinel <https://github.com/alibaba/Sentinel> 的 wiki。

4.超时和重试机制设置

一旦用户请求超过某个时间的得不到响应，就抛出异常。这个是非常重要的，很多线上系统故障都是因为没有进行超时设置或者超时设置的方式不对导致的。我们在读取第三方服务的时候，尤其适合设置超时和重试机制。一般我们使用一些 RPC 框架的时候，这些框架都自带的超时重试的配置。如果不进行超时设置可能会导致请求响应速度慢，甚至导致请求堆积进而让系统无法在处理请求。重试的次数一般设为 3 次，再多次的重试没有好处，反而会加重服务器压力（部分场景使用失败重试机制会不太适合）。

5.熔断机制

超时和重试机制设置之外，熔断机制也是很重要的。 熔断机制说的是系统自动收集所依赖服务的资源使用情况和性能指标，当所依赖的服务恶化或者调用失败次数达到某个阈值的时候就迅速失败，让当前系统立即切换依赖其他备用服务。 比较常用的是流量控制和熔断降级框架是 Netflix 的 Hystrix 和 alibaba 的 Sentinel。

6.异步调用

异步调用的话我们不需要关心最后的结果，这样我们就可以用户请求完成之后就立即返回结果，具体处理我们可以后续再做，秒杀场景用这个还是蛮多的。但是，使用异步之后我们可能需要 **适当修改业务流程进行配合**，比如**用户在提交订单之后，不能立即返回用户订单提交成功，需要在消息队列的订单消费者进程真正处理完该订单之后，甚至出库后，再通过电子邮件或短信通知用户订单成功。** 除了可以在程序中实现异步之外，我们常常还使用消息队列，消息队列可以通过异步处理提高系统性能（削峰、减少响应所需时间）并且可以降低系统耦合性。

7.使用缓存

如果我们的系统属于并发量比较高的话，如果我们单纯使用数据库的话，当大量请求直接落到数据库可能数据库就会直接挂掉。使用缓存缓存热点数据，因为缓存存储在内存中，所以速度相当地快！

8.其他

1. **核心应用和服务优先使用更好的硬件**
2. **监控系统资源使用情况增加报警设置。**
3. **注意备份，必要时候回滚。**
4. **灰度发布：** 将服务器集群分成若干部分，每天只发布一部分机器，观察运行稳定没有故障，第二天继续发布一部分机器，持续几天才把整个集群全部发布完毕，期间如果发现问题，只需要回滚已发布的一部分服务器即可
5. **定期检查/更换硬件：** 如果不是购买的云服务的话，定期还是需要对硬件进行一波检查的，对于一些需要更换或者升级的硬件，要及时更换或者升级。
6. .....(想起来再补充！也欢迎各位欢迎补充！)