**关于大型网站系统架构你不得不懂的10个问题**

下面这些问题都是一线大厂的真实面试问题，不论是对你面试还是说拓宽知识面都很有帮助。之前发过一篇8 张图读懂大型网站技术架构 可以作为不太了解大型网站系统技术架构朋友的入门文章。

# 1. 你使用过哪些组件或者方法来提升网站性能,可用性以及并发量

提高硬件能力、增加系统服务器。（当服务器增加到某个程度的时候系统所能提供的并发访问量几乎不变，所以不能根本解决问题）

使用缓存（本地缓存：本地可以使用JDK自带的 Map、Guava Cache.分布式缓存：Redis、Memcache.本地缓存不适用于提高系统并发量，一般是用处用在程序中。比如Spring是如何实现单例的呢？大家如果看过源码的话，应该知道，Spiring把已经初始过的变量放在一个Map中，下次再要使用这个变量的时候，先判断Map中有没有，这也就是系统中常见的单例模式的实现。）

消息队列 （解耦+削峰+异步）

采用分布式开发 （不同的服务部署在不同的机器节点上，并且一个服务也可以部署在多台机器上，然后利用 Nginx 负载均衡访问。这样就解决了单点部署(All In)的缺点，大大提高的系统并发量）

数据库分库（读写分离）、分表（水平分表、垂直分表）

采用集群 （多台机器提供相同的服务）

CDN 加速 (将一些静态资源比如图片、视频等等缓存到离用户最近的网络节点)

浏览器缓存

使用合适的连接池（数据库连接池、线程池等等）

适当使用多线程进行开发。

# 2. 设计高可用系统的常用手段

降级： 服务降级是当服务器压力剧增的情况下，根据当前业务情况及流量对一些服务和页面有策略的降级，以此释放服务器资源以保证核心任务的正常运行。降级往往会指定不同的级别，面临不同的异常等级执行不同的处理。根据服务方式：可以拒接服务，可以延迟服务，也有时候可以随机服务。根据服务范围：可以砍掉某个功能，也可以砍掉某些模块。总之服务降级需要根据不同的业务需求采用不同的降级策略。主要的目的就是服务虽然有损但是总比没有好；

限流： 防止恶意请求流量、恶意攻击，或者防止流量超出系统峰值；

缓存： 避免大量请求直接落到数据库，将数据库击垮；

超时和重试机制： 避免请求堆积造成雪崩；

回滚机制： 快速修复错误版本。

# 3. 现代互联网应用系统通常具有哪些特点?

高并发，大流量；

高可用：系统7×24小时不间断服务；

海量数据：需要存储、管理海量数据，需要使用大量服务器；

用户分布广泛，网络情况复杂：许多大型互联网都是为全球用户提供服务的，用户分布范围广，各地网络情况千差万别；

安全环境恶劣：由于互联网的开放性，使得互联网更容易受到攻击，大型网站几乎每天都会被黑客攻击；

需求快速变更，发布频繁：和传统软件的版本发布频率不同，互联网产品为快速适应市场，满足用户需求，其产品发布频率是极高的；

渐进式发展：与传统软件产品或企业应用系统一开始就规划好全部的功能和非功能需求不同，几乎所有的大型互联网网站都是从一个小网站开始，渐进地发展起来。

# 4. 谈谈你对微服务领域的了解和认识

现在大公司都在用并且未来的趋势都是 Spring Cloud，而阿里开源的 Spring Cloud Alibaba 也是 Spring Cloud 规范的实现 。

我们通常把 Spring Cloud 理解为一系列开源组件的集合，但是 Spring Cloud并不是等同于 Spring Cloud Netflix 的 Ribbon、Feign、Eureka（停止更新）、Hystrix 这一套组件，而是抽象了一套通用的开发模式。它的目的是通过抽象出这套通用的模式，让开发者更快更好地开发业务。但是这套开发模式运行时的实际载体，还是依赖于 RPC、网关、服务发现、配置管理、限流熔断、分布式链路跟踪等组件的具体实现。

Spring Cloud Alibaba 是官方认证的新一套 Spring Cloud 规范的实现,Spring Cloud Alibaba 是一套国产开源产品集合，后续还会有中文 reference 和一些原理分析文章，所以，这对于国内的开发者是非常棒的一件事。阿里的这一举动势必会推动国内微服务技术的发展，因为在没有 Spring Cloud Alibaba 之前，我们的第一选择是 Spring Cloud Netflix，但是它们的文档都是英文的，出问题后排查也比较困难， 在国内并不是有特别多的人精通。Spring Cloud Alibaba 由阿里开源组件和阿里云产品组件两部分组成，其致力于提供微服务一站式解决方案，方便开发者通过 Spring Cloud 编程模型轻松开发微服务应用。

另外，Apache Dubbo Ecosystem 是围绕 Apache Dubbo 打造的微服务生态，是经过生产验证的微服务的最佳实践组合。在阿里巴巴的微服务解决方案中，Dubbo、Nacos 和 Sentinel，以及后续将开源的微服务组件，都是 Dubbo EcoSystem 的一部分。阿里后续也会将 Dubbo EcoSystem 集成到 Spring Cloud 的生态中。

# 5. 谈谈你对 Dubbo 和 Spring Cloud 的认识(两者关系)

具体可以看公众号-阿里巴巴中间件的这篇文章:独家解读：Dubbo Ecosystem - 从微服务框架到微服务生态

Dubbo 与 Spring Cloud 并不是竞争关系，Dubbo 作为成熟的 RPC 框架，其易用性、扩展性和健壮性已得到业界的认可。未来 Dubbo 将会作为 Spring Cloud Alibaba 的 RPC 组件，并与 Spring Cloud 原生的 Feign 以及 RestTemplate 进行无缝整合，实现“零”成本迁移。

在阿里巴巴的微服务解决方案中，Dubbo、Nacos 和 Sentinel，以及后续将开源的微服务组件，都是 Dubbo EcoSystem 的一部分。我们后续也会将 Dubbo EcoSystem 集成到 Spring Cloud 的生态中。

# 6. 性能测试了解吗?说说你知道的性能测试工具?

性能测试指通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。性能测试是总称，通常细分为：

基准测试： 在给系统施加较低压力时，查看系统的运行状况并记录相关数做为基础参考

\*\*负载测试：\*\*是指对系统不断地增加压力或增加一定压力下的持续时间，直到系统的某项或多项性能指标达到安全临界值，例如某种资源已经达到饱和状态等 。此时继续加压，系统处理能力会下降。

压力测试： 超过安全负载情况下，不断施加压力（增加并发请求），直到系统崩溃或无法处理任何请求，依此获得系统最大压力承受能力。

稳定性测试： 被测试系统在特定硬件、软件、网络环境下，加载一定业务压力（模拟生产环境不同时间点、不均匀请求，呈波浪特性）运行一段较长时间，以此检测系统是否稳定。

后端程序员或者测试平常比较常用的测试工具是 JMeter（官网：https://jmeter.apache.org/）。Apache JMeter 是一款基于Java的压力测试工具(100％纯Java应用程序)，旨在加载测试功能行为和测量性能。它最初被设计用于 Web 应用测试但后来扩展到其他测试领域。

# 7. 对于一个单体应用系统,随着产品使用的用户越来越多,网站的流量会增加,最终单台服务器无法处理那么大的流量怎么办?

这个时候就要考虑扩容了。《亿级流量网站架构核心技术》这本书上面介绍到我们可以考虑下面几步来解决这个问题：

第一步，可以考虑简单的扩容来解决问题。比如增加系统的服务器，提高硬件能力等等。

第二步，如果简单扩容搞不定，就需要水平拆分和垂直拆分数据／应用来提升系统的伸缩性，即通过扩容提升系统负载能力。

第三步，如果通过水平拆分／垂直拆分还是搞不定，那就需要根据现有系统特性，架构层面进行重构甚至是重新设计，即推倒重来。

对于系统设计，理想的情况下应支持线性扩容和弹性扩容，即在系统瓶颈时，只需要增加机器就可以解决系统瓶颈，如降低延迟提升吞吐量，从而实现扩容需求。

如果你想扩容，则支持水平/垂直伸缩是前提。在进行拆分时，一定要清楚知道自己的目的是什么，拆分后带来的问题如何解决，拆分后如果没有得到任何收益就不要为了

拆而拆，即不要过度拆分，要适合自己的业务。

# 8. 大表优化的常见手段

当MySQL单表记录数过大时，数据库的CRUD性能会明显下降，一些常见的优化措施如下：

限定数据的范围： 务必禁止不带任何限制数据范围条件的查询语句。比如：我们当用户在查询订单历史的时候，我们可以控制在一个月的范围内。；

读/写分离： 经典的数据库拆分方案，主库负责写，从库负责读；

垂直分区： 根据数据库里面数据表的相关性进行拆分。 例如，用户表中既有用户的登录信息又有用户的基本信息，可以将用户表拆分成两个单独的表，甚至放到单独的库做分库。简单来说垂直拆分是指数据表列的拆分，把一张列比较多的表拆分为多张表。 如下图所示，这样来说大家应该就更容易理解了。垂直拆分的优点： 可以使得行数据变小，在查询时减少读取的Block数，减少I/O次数。此外，垂直分区可以简化表的结构，易于维护。垂直拆分的缺点： 主键会出现冗余，需要管理冗余列，并会引起Join操作，可以通过在应用层进行Join来解决。此外，垂直分区会让事务变得更加复杂；

水平分区： 保持数据表结构不变，通过某种策略存储数据分片。这样每一片数据分散到不同的表或者库中，达到了分布式的目的。 水平拆分可以支撑非常大的数据量。 水平拆分是指数据表行的拆分，表的行数超过200万行时，就会变慢，这时可以把一张的表的数据拆成多张表来存放。举个例子：我们可以将用户信息表拆分成多个用户信息表，这样就可以避免单一表数据量过大对性能造成影响。数据库水平拆分水平拆分可以支持非常大的数据量。需要注意的一点是:分表仅仅是解决了单一表数据过大的问题，但由于表的数据还是在同一台机器上，其实对于提升MySQL并发能力没有什么意义，所以 水平拆分最好分库 。水平拆分能够 支持非常大的数据量存储，应用端改造也少，但 分片事务难以解决 ，跨界点Join性能较差，逻辑复杂。《Java工程师修炼之道》的作者推荐 尽量不要对数据进行分片，因为拆分会带来逻辑、部署、运维的各种复杂度 ，一般的数据表在优化得当的情况下支撑千万以下的数据量是没有太大问题的。如果实在要分片，尽量选择客户端分片架构，这样可以减少一次和中间件的网络I/O。

下面补充一下数据库分片的两种常见方案：

客户端代理： 分片逻辑在应用端，封装在jar包中，通过修改或者封装JDBC层来实现。 当当网的 Sharding-JDBC 、阿里的TDDL是两种比较常用的实现。

中间件代理： 在应用和数据中间加了一个代理层。分片逻辑统一维护在中间件服务中。 我们现在谈的 Mycat 、360的Atlas、网易的DDB等等都是这种架构的实现。

# 9. 在系统中使用消息队列能带来什么好处?

《大型网站技术架构》第四章和第七章均有提到消息队列对应用性能及扩展性的提升。

1) 通过异步处理提高系统性能

通过异步处理提高系统性能

如上图，在不使用消息队列服务器的时候，用户的请求数据直接写入数据库，在高并发的情况下数据库压力剧增，使得响应速度变慢。但是在使用消息队列之后，用户的请求数据发送给消息队列之后立即 返回，再由消息队列的消费者进程从消息队列中获取数据，异步写入数据库。由于消息队列服务器处理速度快于数据库（消息队列也比数据库有更好的伸缩性），因此响应速度得到大幅改善。

通过以上分析我们可以得出消息队列具有很好的削峰作用的功能——即通过异步处理，将短时间高并发产生的事务消息存储在消息队列中，从而削平高峰期的并发事务。 举例：在电子商务一些秒杀、促销活动中，合理使用消息队列可以有效抵御促销活动刚开始大量订单涌入对系统的冲击。如下图所示：

合理使用消息队列可以有效抵御促销活动刚开始大量订单涌入对系统的冲击

因为用户请求数据写入消息队列之后就立即返回给用户了，但是请求数据在后续的业务校验、写数据库等操作中可能失败。因此使用消息队列进行异步处理之后，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，订单数据写入消息队列，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，以免交易纠纷。这就类似我们平时手机订火车票和电影票。

2) 降低系统耦合性

我们知道模块分布式部署以后聚合方式通常有两种：1.分布式消息队列和2.分布式服务。

先来简单说一下分布式服务：

目前使用比较多的用来构建SOA（Service Oriented Architecture面向服务体系结构）的分布式服务框架是阿里巴巴开源的Dubbo.如果想深入了解Dubbo的可以看我写的关于Dubbo的这一篇文章：《高性能优秀的服务框架-dubbo介绍》：https://juejin.im/post/5acadeb1f265da2375072f9c

再来谈我们的分布式消息队列：

我们知道如果模块之间不存在直接调用，那么新增模块或者修改模块就对其他模块影响较小，这样系统的可扩展性无疑更好一些。

我们最常见的事件驱动架构类似生产者消费者模式，在大型网站中通常用利用消息队列实现事件驱动结构。如下图所示：

利用消息队列实现事件驱动结构

消息队列使利用发布-订阅模式工作，消息发送者（生产者）发布消息，一个或多个消息接受者（消费者）订阅消息。 从上图可以看到消息发送者（生产者）和消息接受者（消费者）之间没有直接耦合，消息发送者将消息发送至分布式消息队列即结束对消息的处理，消息接受者从分布式消息队列获取该消息后进行后续处理，并不需要知道该消息从何而来。对新增业务，只要对该类消息感兴趣，即可订阅该消息，对原有系统和业务没有任何影响，从而实现网站业务的可扩展性设计。

消息接受者对消息进行过滤、处理、包装后，构造成一个新的消息类型，将消息继续发送出去，等待其他消息接受者订阅该消息。因此基于事件（消息对象）驱动的业务架构可以是一系列流程。

另外为了避免消息队列服务器宕机造成消息丢失，会将成功发送到消息队列的消息存储在消息生产者服务器上，等消息真正被消费者服务器处理后才删除消息。在消息队列服务器宕机后，生产者服务器会选择分布式消息队列服务器集群中的其他服务器发布消息。

备注： 不要认为消息队列只能利用发布-订阅模式工作，只不过在解耦这个特定业务环境下是使用发布-订阅模式的，比如在我们的ActiveMQ消息队列中还有点对点工作模式，具体的会在后面的文章给大家详细介绍，这一篇文章主要还是让大家对消息队列有一个更透彻的了解。

这个问题一般会在上一个问题问完之后，紧接着被问到。“使用消息队列会带来什么问题？”这个问题要引起重视，一般我们都会考虑使用消息队列会带来的好处而忽略它带来的问题！

# 10. 说说自己对 CAP 定理,BASE 理论的了解

CAP 定理

CAP定理

在理论计算机科学中，CAP定理（CAP theorem），又被称作布鲁尔定理（Brewer’s theorem），它指出对于一个分布式计算系统来说，不可能同时满足以下三点：

一致性（Consistence） :所有节点访问同一份最新的数据副本

可用性（Availability）:每次请求都能获取到非错的响应——但是不保证获取的数据为最新数据

分区容错性（Partition tolerance） : 分布式系统在遇到某节点或网络分区故障的时候，仍然能够对外提供满足一致性和可用性的服务。

CAP仅适用于原子读写的NOSQL场景中，并不适合数据库系统。现在的分布式系统具有更多特性比如扩展性、可用性等等，在进行系统设计和开发时，我们不应该仅仅局限在CAP问题上。

注意：不是所谓的3选2（不要被网上大多数文章误导了）:

大部分人解释这一定律时，常常简单的表述为：“一致性、可用性、分区容忍性三者你只能同时达到其中两个，不可能同时达到”。实际上这是一个非常具有误导性质的说法，而且在CAP理论诞生12年之后，CAP之父也在2012年重写了之前的论文。

当发生网络分区的时候，如果我们要继续服务，那么强一致性和可用性只能2选1。也就是说当网络分区之后P是前提，决定了P之后才有C和A的选择。也就是说分区容错性（Partition tolerance）我们是必须要实现的。

我在网上找了很多文章想看一下有没有文章提到这个不是所谓的3选2，用百度半天没找到了一篇，用谷歌搜索找到一篇比较不错的，如果想深入学习一下CAP就看这篇文章把，我这里就不多BB了：《分布式系统之CAP理论》 ： http://www.cnblogs.com/hxsyl/p/4381980.html

BASE 理论

BASE 是 Basically Available（基本可用） 、Soft-state（软状态） 和 Eventually Consistent（最终一致性） 三个短语的缩写。BASE理论是对CAP中一致性和可用性权衡的结果，其来源于对大规模互联网系统分布式实践的总结，是基于CAP定理逐步演化而来的，它大大降低了我们对系统的要求。

BASE理论的核心思想： 即使无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。也就是牺牲数据的一致性来满足系统的高可用性，系统中一部分数据不可用或者不一致时，仍需要保持系统整体“主要可用”。

BASE理论三要素：

BASE理论三要素

基本可用： 基本可用是指分布式系统在出现不可预知故障的时候，允许损失部分可用性。但是，这绝不等价于系统不可用。 比如： ①响应时间上的损失:正常情况下，一个在线搜索引擎需要在0.5秒之内返回给用户相应的查询结果，但由于出现故障，查询结果的响应时间增加了1~2秒；②系统功能上的损失：正常情况下，在一个电子商务网站上进行购物的时候，消费者几乎能够顺利完成每一笔订单，但是在一些节日大促购物高峰的时候，由于消费者的购物行为激增，为了保护购物系统的稳定性，部分消费者可能会被引导到一个降级页面；

软状态： 软状态指允许系统中的数据存在中间状态，并认为该中间状态的存在不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步的过程存在延时；

最终一致性： 最终一致性强调的是系统中所有的数据副本，在经过一段时间的同步后，最终能够达到一个一致的状态。因此，最终一致性的本质是需要系统保证最终数据能够达到一致，而不需要实时保证系统数据的强一致性。