**更多技术干货**

**关注公众号：IT技术思维**

****

**数据库&事务**

1.什么样的场景会产生数据库死锁，如何解决？

2.SQL如何优化

3.Oracle和mysql分页的实现原理

4.唯一索引能否插入空

5.数据库索引的算法原理

6.乐观锁，悲观锁

7.SQL怎样判断字段是否为空

**WEB**

1.post和get区别是什么？

2.cookie/session有什么区别？

3.web集群中登录态如何共享？

4.xss、csrf等如何防范？

5.sql注入如何防范？

6.浏览器缓存策略

7.Ajax如何解决跨域问题？

8.什么是ajax，为什么要使用ajax？

9.浅谈你对ajax的认识？

10.Cookie和Session以及Servlet的生命周期？

11.说一下你熟悉的常用linux命令？

12.后台传过的json数据前台怎么接收？

13.后台传过来一个集合前台怎么接收？

14.tomcat的优化方式？

15.http协议有哪些部分组成？

**框架**

1.Spring的AOP和IOC原理，类内部调用是否会触发AOP的逻辑？为什么？

2.Spring的事务传播属性或数据库的事务隔离级别，实现原理。

3.velocity如何避免引用空属性的报错？

4.Spring的单例和原型怎么配置，还用过其它什么scope?

5.log4j或者logback的日记记录文件在哪里？（来访问题，怎么确定日志打到哪里了）

**工具**

1.maven中怎么排除包冲突

2.IDEA合并代码怎么做？

**分布式**

1.如何实现分布式锁

2.分布式Session怎样实现

3.如何消除分布式事务

4.MetaQ，Zookeeper的原理和适用场景

5.Redis3.0的分布式集群架构

**高并发**

1.原子计数器AtomicInteger的技术实现

2.volatile的内存模型

3.接口服务如何防止重复调用

4.自己使用过的多线程场景是什么？线程安全如何保证？java的并发框架是否用过？线程池是否用过？RejectedExecutionHandler是否知道？

**设计**

1.对设计模式的了解，在项目中的实际应用。

2.排序算法描述

3.在项目中是否采用分层的结构，是怎样划分的，各层之间采用了哪些相关技术？

**Java内存模型和GC**

1. 描述下Java的内存模型。区分Young Generation、Tenured Generation、Permanent Generation(Java8以下）。Eden Area、S0、S1
2. 什么是Minor GC?什么是Major GC(Full GC)？
3. 何时会触发Full GC？
4. 常见的垃圾回收算法有哪些。
5. 如何优化JVM性能？JVM的性能取决于其配置是否与应用的功能相匹配。如何设置JVM相关参数?-Xms、-Xmx,-XX:NewRatio、-Xmn(设定初始化和最大新生代大小)、-Xss(设置线程栈大小)
6. 从Efficiency、Throughput、Concurrency、Overhead、JVM version等角度考量，如何选择垃圾收集器的实现？

**综合：**

1、面向对象的特征有哪些方面？  
2、访问修饰符public,private,protected,以及不写（默认）时的区别？  
3、String 是最基本的数据类型吗？  
4、float f=3.4;是否正确？  
5、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？  
6、Java有没有goto？  
7、int和Integer有什么区别？  
8、&amp;和&amp;&amp;的区别？  
9、解释内存中的栈(stack)、堆(heap)和静态区(static area)的用法。  
10、Math.round(11.5) 等于多少？Math.round(-11.5)等于多少？  
11、switch 是否能作用在byte 上，是否能作用在long 上，是否能作用在String上？  
12、用最有效率的方法计算2乘以8？  
13、数组有没有length()方法？String有没有length()方法？  
14、在Java中，如何跳出当前的多重嵌套循环？  
15、构造器（constructor）是否可被重写（override）？  
16、两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对？  
17、是否可以继承String类？  
18、当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递？  
19、String和StringBuilder、StringBuffer的区别？  
20、重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？  
21、描述一下JVM加载class文件的原理机制？  
22、char 型变量中能不能存贮一个中文汉字，为什么？  
23、抽象类（abstract class）和接口（interface）有什么异同？  
24、静态嵌套类(Static Nested Class)和内部类（Inner Class）的不同？  
25、Java 中会存在内存泄漏吗，请简单描述。  
26、抽象的（abstract）方法是否可同时是静态的（static）,是否可同时是本地方法（native），是否可同时被synchronized修饰？  
27、阐述静态变量和实例变量的区别。  
28、是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？  
29、如何实现对象克隆？  
31、String s = new String("xyz");创建了几个字符串对象？  
32、接口是否可继承（extends）接口？抽象类是否可实现（implements）接口？抽象类是否可继承具体类（concrete class）？  
33、一个".java"源文件中是否可以包含多个类（不是内部类）？有什么限制？  
34、Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？  
35、内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？  
36、Java 中的final关键字有哪些用法？  
38、数据类型之间的转换：  
39、如何实现字符串的反转及替换？  
40、怎样将GB2312编码的字符串转换为ISO-8859-1编码的字符串？  
41、日期和时间：  
42、打印昨天的当前时刻。  
43、比较一下Java和JavaSciprt。  
44、什么时候用断言（assert）？  
45、Error和Exception有什么区别？  
46、try{}里有一个return语句，那么紧跟在这个try后的finally{}里的代码会不会被执行，什么时候被执行，在return前还是后?  
47、Java语言如何进行异常处理，关键字：throws、throw、try、catch、finally分别如何使用？  
48、运行时异常与受检异常有何异同？  
49、列出一些你常见的运行时异常？  
50、阐述final、finally、finalize的区别。  
51、类ExampleA继承Exception，类ExampleB继承ExampleA。  
请问执行此段代码的输出是什么？  
52、List、Set、Map是否继承自Collection接口？  
53、阐述ArrayList、Vector、LinkedList的存储性能和特性。  
54、Collection和Collections的区别？  
55、List、Map、Set三个接口存取元素时，各有什么特点？  
56、TreeMap和TreeSet在排序时如何比较元素？Collections工具类中的sort()方法如何比较元素？  
57、Thread类的sleep()方法和对象的wait()方法都可以让线程暂停执行，它们有什么区别?  
58、线程的sleep()方法和yield()方法有什么区别？  
59、当一个线程进入一个对象的synchronized方法A之后，其它线程是否可进入此对象的synchronized方法B？  
60、请说出与线程同步以及线程调度相关的方法。  
61、编写多线程程序有几种实现方式？  
62、synchronized关键字的用法？  
63、举例说明同步和异步。  
64、启动一个线程是调用run()还是start()方法？  
65、什么是线程池（thread pool）？  
66、线程的基本状态以及状态之间的关系？  
67、简述synchronized 和java.util.concurrent.locks.Lock的异同？  
68、Java中如何实现序列化，有什么意义？  
69、Java中有几种类型的流？  
70、写一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的次数。  
71、如何用Java代码列出一个目录下所有的文件？  
72、用Java的套接字编程实现一个多线程的回显（echo）服务器。  
73、XML文档定义有几种形式？它们之间有何本质区别？解析XML文档有哪几种方式？  
74、你在项目中哪些地方用到了XML？  
75、阐述JDBC操作数据库的步骤。  
76、Statement和PreparedStatement有什么区别？哪个性能更好？  
77、使用JDBC操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能？  
78、在进行数据库编程时，连接池有什么作用？  
79、什么是DAO模式？  
80、事务的ACID是指什么？  
81、JDBC中如何进行事务处理？  
82、JDBC能否处理Blob和Clob？  
83、简述正则表达式及其用途。  
84、Java中是如何支持正则表达式操作的？  
85、获得一个类的类对象有哪些方式？  
86、如何通过反射创建对象？  
87、如何通过反射获取和设置对象私有字段的值？  
88、如何通过反射调用对象的方法？  
89、简述一下面向对象的"六原则一法则"。  
90、简述一下你了解的设计模式。  
91、用Java写一个单例类。  
92、什么是UML？  
93、UML中有哪些常用的图？  
94、用Java写一个冒泡排序。  
95、用Java写一个折半查找。

96、阐述Servlet和CGI的区别?  
97、Servlet接口中有哪些方法？  
98、转发（forward）和重定向（redirect）的区别？  
99、JSP有哪些内置对象？作用分别是什么？  
100、get和post请求的区别？  
101、常用的Web服务器有哪些？  
102、JSP和Servlet是什么关系？  
103、讲解JSP中的四种作用域。  
104、如何实现JSP或Servlet的单线程模式？  
105、实现会话跟踪的技术有哪些？  
106、过滤器有哪些作用和用法？  
107、监听器有哪些作用和用法？  
108、web.xml文件中可以配置哪些内容？  
109、你的项目中使用过哪些JSTL标签？  
110、使用标签库有什么好处？如何自定义JSP标签？  
111、说一下表达式语言（EL）的隐式对象及其作用。  
112、表达式语言（EL）支持哪些运算符？  
113、Java Web开发的Model 1和Model 2分别指的是什么？  
114、Servlet 3中的异步处理指的是什么？  
115、如何在基于Java的Web项目中实现文件上传和下载？  
116、服务器收到用户提交的表单数据，到底是调用Servlet的doGet()还是doPost()方法？  
117、JSP中的静态包含和动态包含有什么区别？  
118、Servlet中如何获取用户提交的查询参数或表单数据？  
119、Servlet中如何获取用户配置的初始化参数以及服务器上下文参数？  
120、如何设置请求的编码以及响应内容的类型？  
121、解释一下网络应用的模式及其特点。  
122、什么是Web Service（Web服务）？  
123、概念解释：SOAP、WSDL、UDDI。  
124、Java规范中和Web Service相关的规范有哪些？  
125、介绍一下你了解的Java领域的Web Service框架。

126、什么是ORM？  
127、持久层设计要考虑的问题有哪些？你用过的持久层框架有哪些？  
128、Hibernate中SessionFactory是线程安全的吗？Session是线程安全的吗（两个线程能够共享同一个Session吗）？  
129、Hibernate中Session的load和get方法的区别是什么？  
130、Session的save()、update()、merge()、lock()、saveOrUpdate()和persist()方法分别是做什么的？有什么区别？  
131、阐述Session加载实体对象的过程。  
132、Query接口的list方法和iterate方法有什么区别？  
133、Hibernate如何实现分页查询？  
134、锁机制有什么用？简述Hibernate的悲观锁和乐观锁机制。  
135、阐述实体对象的三种状态以及转换关系。  
136、如何理解Hibernate的延迟加载机制？在实际应用中，延迟加载与Session关闭的矛盾是如何处理的？  
137、举一个多对多关联的例子，并说明如何实现多对多关联映射。  
138、谈一下你对继承映射的理解。  
139、简述Hibernate常见优化策略。  
140、谈一谈Hibernate的一级缓存、二级缓存和查询缓存。  
141、Hibernate中DetachedCriteria类是做什么的？  
142、@OneToMany注解的mappedBy属性有什么作用？  
143、MyBatis中使用<code>#</code>和<code>$</code>书写占位符有什么区别？  
144、解释一下MyBatis中命名空间（namespace）的作用。  
145、MyBatis中的动态SQL是什么意思？  
146、什么是IoC和DI？DI是如何实现的？  
147、Spring中Bean的作用域有哪些？  
148、解释一下什么叫AOP（面向切面编程）？  
149、你是如何理解"横切关注"这个概念的？  
150、你如何理解AOP中的连接点（Joinpoint）、切点（Pointcut）、增强（Advice）、引介（Introduction）、织入（Weaving）、切面（Aspect）这些概念？  
151、Spring中自动装配的方式有哪些？  
152、Spring中如何使用注解来配置Bean？有哪些相关的注解？  
153、Spring支持的事务管理类型有哪些？你在项目中使用哪种方式？  
154、如何在Web项目中配置Spring的IoC容器？  
155、如何在Web项目中配置Spring MVC？  
156、Spring MVC的工作原理是怎样的？  
157、如何在Spring IoC容器中配置数据源？  
158、如何配置配置事务增强？  
159、选择使用Spring框架的原因（Spring框架为企业级开发带来的好处有哪些）？  
160、Spring IoC容器配置Bean的方式？  
161、阐述Spring框架中Bean的生命周期？  
162、依赖注入时如何注入集合属性？  
163、Spring中的自动装配有哪些限制？  
164、在Web项目中如何获得Spring的IoC容器？  
165. 大型网站在架构上应当考虑哪些问题？  
166、你用过的网站前端优化的技术有哪些？  
167、你使用过的应用服务器优化技术有哪些？  
168、什么是XSS攻击？什么是SQL注入攻击？什么是CSRF攻击？  
169. 什么是领域模型(domain model)？贫血模型(anaemic domain model)和充血模型(rich domain model)有什么区别？  
170. 谈一谈测试驱动开发（TDD）的好处以及你的理解。

171..整数数组按照绝对值排序

172.ConcurrentHashMap的数据结构

173.内存管理机制，如何回收过期对象，判断依据，内存溢出场景，如何排查

174.equals和==的区别

175.final,finally, finalize

176.常用的集合类有哪些？区别是什么？HashMap和HashTable的区别，那些事线程安全的哪些不是？

177.Object中常见的方法

178.java中的异常体系，异常出现的情况如：ClassNotFound、空指针等。

179.Jvm的内存模型是怎么样的？应用突然变得很慢，如何定位？

180.java的异常类层次结构

181.对jdk8新特征有什么了解？

182.java的类加载机制

183.原子类的实现

184.Error,Exception和RuntimeException的区别，作用又是什么？

185.sleep()和wait()区别是什么？

**Springboot(含答案)**

**1.开启 Spring Boot 特性有哪几种方式？**

1）继承spring-boot-starter-parent项目

2）导入spring-boot-dependencies项目依赖

**2.Spring Boot 需要独立的容器运行吗？**

可以不需要，内置了 Tomcat/ Jetty 等容器。

**3. 运行 Spring Boot 有哪几种方式？**

1）打包用命令或者放到容器中运行

2）用 Maven/ Gradle 插件运行

3）直接执行 main 方法运行

**4. Spring Boot 自动配置原理是什么？**

注解 @EnableAutoConfiguration, @Configuration, @ConditionalOnClass 就是自动配置的核心，首先它得是一个配置文件，其次根据类路径下是否有这个类去自动配置。

**5. Spring Boot 的目录结构是怎样的？**

cn  
 +- javastack  
     +- MyApplication.java  
     |  
     +- customer  
     |   +- Customer.java  
     |   +- CustomerController.java  
     |   +- CustomerService.java  
     |   +- CustomerRepository.java  
     |  
     +- order  
         +- Order.java  
         +- OrderController.java  
         +- OrderService.java  
         +- OrderRepository.java

这个目录结构是主流及推荐的做法，而在主入口类上加上 @SpringBootApplication 注解来开启 Spring Boot 的各项能力，如自动配置、组件扫描等。

1. **你如何理解 Spring Boot 中的 Starters？**

Starters可以理解为启动器，它包含了一系列可以集成到应用里面的依赖包，你可以一站式集成 Spring 及其他技术，而不需要到处找示例代码和依赖包。如你想使用 Spring JPA 访问数据库，只要加入 spring-boot-starter-data-jpa 启动器依赖就能使用了。

Starters包含了许多项目中需要用到的依赖，它们能快速持续的运行，都是一系列得到支持的管理传递性依赖。

1. **如何在 Spring Boot 启动的时候运行一些特定的代码**？

可以实现接口 ApplicationRunner 或者 CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个 run 方法。

1. **Spring Boot 有哪几种读取配置的方式？**

Spring Boot 可以通过 @PropertySource,@Value,@Environment, @ConfigurationProperties 来绑定变量

1. **Spring Boot 支持哪些日志框架？推荐和默认的日志框架是哪个？**

Spring Boot 支持 Java Util Logging, Log4j2, Lockback 作为日志框架，如果你使用 Starters 启动器，Spring Boot 将使用 Logback 作为默认日志框架

1**0.SpringBoot 实现热部署有哪几种方式？**

主要有两种方式：

* Spring Loaded
* Spring-boot-devtools

Spring-boot-devtools

**11.你如何理解 Spring Boot 配置加载顺序？**

在 Spring Boot 里面，可以使用以下几种方式来加载配置。

1）properties文件；

2）YAML文件；

3）系统环境变量；

4）命令行参数；

等等……

**12. Spring Boot 如何定义多套不同环境配置？**

提供多套配置文件，如：

applcation.properties  
  
application-dev.properties  
  
application-test.properties  
  
application-prod.properties

**13. Spring Boot 可以兼容老 Spring 项目吗，如何做？**

可以兼容，使用 @ImportResource 注解导入老 Spring 项目配置文件。

**14. 保护 Spring Boot 应用有哪些方法？**

* 在生产中使用HTTPS
* 使用Snyk检查你的依赖关系
* 升级到最新版本
* 启用CSRF保护
* 使用内容安全策略防止XSS攻击
* …

**15. Spring Boot 2.X 有什么新特性？与 1.X 有什么区别？**

* 配置变更
* JDK 版本升级
* 第三方类库升级
* 响应式 Spring 编程支持
* HTTP/2 支持
* 配置属性绑定
* 更多改进与加强…

**多线程：(含答案)**

**1、多线程有什么用？**

一个可能在很多人看来很扯淡的一个问题：我会用多线程就好了，还管它有什么用？在我看来，这个回答更扯淡。所谓"知其然知其所以然"，"会用"只是"知其然"，"为什么用"才是"知其所以然"，只有达到"知其然知其所以然"的程度才可以说是把一个知识点运用自如。OK，下面说说对这个问题的看法：

1）发挥多核CPU的优势

随着工业的进步，现在的笔记本、台式机乃至商用的应用服务器至少也都是双核的，4核、8核甚至16核的也都不少见，如果是单线程的程序，那么在双核CPU上就浪费了50%，在4核CPU上就浪费了75%。单核CPU上所谓的"多线程"那是假的多线程，同一时间处理器只会处理一段逻辑，只不过线程之间切换得比较快，看着像多个线程"同时"运行罢了。多核CPU上的多线程才是真正的多线程，它能让你的多段逻辑同时工作，多线程，可以真正发挥出多核CPU的优势来，达到充分利用CPU的目的。

2）防止阻塞

从程序运行效率的角度来看，单核CPU不但不会发挥出多线程的优势，反而会因为在单核CPU上运行多线程导致线程上下文的切换，而降低程序整体的效率。但是单核CPU我们还是要应用多线程，就是为了防止阻塞。试想，如果单核CPU使用单线程，那么只要这个线程阻塞了，比方说远程读取某个数据吧，对端迟迟未返回又没有设置超时时间，那么你的整个程序在数据返回回来之前就停止运行了。多线程可以防止这个问题，多条线程同时运行，哪怕一条线程的代码执行读取数据阻塞，也不会影响其它任务的执行。

3）便于建模

这是另外一个没有这么明显的优点了。假设有一个大的任务A，单线程编程，那么就要考虑很多，建立整个程序模型比较麻烦。但是如果把这个大的任务A分解成几个小任务，任务B、任务C、任务D，分别建立程序模型，并通过多线程分别运行这几个任务，那就简单很多了。

**2、创建线程的方式**

比较常见的一个问题了，一般就是两种：

1）继承Thread类

2）实现Runnable接口

至于哪个好，不用说肯定是后者好，因为实现接口的方式比继承类的方式更灵活，也能减少程序之间的耦合度，面向接口编程也是设计模式6大原则的核心。

其实还有第3种，点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247483859&idx=1&sn=a46418cc67a1f724c3bff681cb628353&chksm=eb5384e5dc240df3f666ab02c1ddce07b3c34917e433124b0925d8390a54cf7b87ae1d5ebd54&scene=21#wechat_redirect)了解更多。

**3、start()方法和run()方法的区别**

只有调用了start()方法，才会表现出多线程的特性，不同线程的run()方法里面的代码交替执行。如果只是调用run()方法，那么代码还是同步执行的，必须等待一个线程的run()方法里面的代码全部执行完毕之后，另外一个线程才可以执行其run()方法里面的代码。

**4、Runnable接口和Callable接口的区别**

有点深的问题了，也看出一个Java程序员学习知识的广度。

Runnable接口中的run()方法的返回值是void，它做的事情只是纯粹地去执行run()方法中的代码而已；Callable接口中的call()方法是有返回值的，是一个泛型，和Future、FutureTask配合可以用来获取异步执行的结果。

这其实是很有用的一个特性，因为多线程相比单线程更难、更复杂的一个重要原因就是因为多线程充满着未知性，某条线程是否执行了？某条线程执行了多久？某条线程执行的时候我们期望的数据是否已经赋值完毕？无法得知，我们能做的只是等待这条多线程的任务执行完毕而已。而Callable+Future/FutureTask却可以获取多线程运行的结果，可以在等待时间太长没获取到需要的数据的情况下取消该线程的任务，真的是非常有用。

**5、CyclicBarrier和CountDownLatch的区别**

两个看上去有点像的类，都在java.util.concurrent下，都可以用来表示代码运行到某个点上，二者的区别在于：

1）CyclicBarrier的某个线程运行到某个点上之后，该线程即停止运行，直到所有的线程都到达了这个点，所有线程才重新运行；CountDownLatch则不是，某线程运行到某个点上之后，只是给某个数值-1而已，该线程继续运行。

2）CyclicBarrier只能唤起一个任务，CountDownLatch可以唤起多个任务。

3) CyclicBarrier可重用，CountDownLatch不可重用，计数值为0该CountDownLatch就不可再用了。

**6、volatile关键字的作用**

**一个非常重要的问题，是每个学习、应用**多线程的Java程序员都必须掌握的。理解volatile关键字的作用的前提是要理解Java内存模型，这里就不讲Java内存模型了，可以参见第31点，volatile关键字的作用主要有两个：

1）多线程主要围绕可见性和原子性两个特性而展开，使用volatile关键字修饰的变量，保证了其在多线程之间的可见性，即每次读取到volatile变量，一定是最新的数据。

2）代码底层执行不像我们看到的高级语言----Java程序这么简单，它的执行是Java代码-->字节码-->根据字节码执行对应的C/C++代码-->C/C++代码被编译成汇编语言-->和硬件电路交互，现实中，为了获取更好的性能JVM可能会对指令进行重排序，多线程下可能会出现一些意想不到的问题。使用volatile则会对禁止语义重排序，当然这也一定程度上降低了代码执行效率。

从实践角度而言，volatile的一个重要作用就是和CAS结合，保证了原子性，详细的可以参见java.util.concurrent.atomic包下的类，比如AtomicInteger，更多详情请点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247483916&idx=1&sn=89daf388da0d6fe40dc54e9a4018baeb&chksm=eb53873adc240e2cf55400f3261228d08fc943c4f196566e995681549c47630b70ac01b75031&scene=21#wechat_redirect)进行学习。

**7、什么是线程安全**

又是一个理论的问题，各式各样的答案有很多，我给出一个个人认为解释地最好的：如果你的代码在多线程下执行和在单线程下执行永远都能获得一样的结果，那么你的代码就是线程安全的。

这个问题有值得一提的地方，就是线程安全也是有几个级别的：

1）不可变

像String、Integer、Long这些，都是final类型的类，任何一个线程都改变不了它们的值，要改变除非新创建一个，因此这些不可变对象不需要任何同步手段就可以直接在多线程环境下使用

2）绝对线程安全

不管运行时环境如何，调用者都不需要额外的同步措施。要做到这一点通常需要付出许多额外的代价，Java中标注自己是线程安全的类，实际上绝大多数都不是线程安全的，不过绝对线程安全的类，Java中也有，比方说CopyOnWriteArrayList、CopyOnWriteArraySet

3）相对线程安全

相对线程安全也就是我们通常意义上所说的线程安全，像Vector这种，add、remove方法都是原子操作，不会被打断，但也仅限于此，如果有个线程在遍历某个Vector、有个线程同时在add这个Vector，99%的情况下都会出现ConcurrentModificationException，也就是fail-fast机制。

4）线程非安全

这个就没什么好说的了，ArrayList、LinkedList、HashMap等都是线程非安全的类，点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486446&idx=2&sn=cb4f3aff0427c5ac3ffe5b61e150f506&chksm=eb538ed8dc2407ceb91fffe3c3bd559d9b15537446f84eb3bfb1a80e67f5efee176ca468a07b&scene=21#wechat_redirect)了解为什么不安全。

**8、Java中如何获取到线程dump文件**

死循环、死锁、阻塞、页面打开慢等问题，打线程dump是最好的解决问题的途径。所谓线程dump也就是线程堆栈，获取到线程堆栈有两步：

1）获取到线程的pid，可以通过使用jps命令，在Linux环境下还可以使用ps -ef | grep java

2）打印线程堆栈，可以通过使用jstack pid命令，在Linux环境下还可以使用kill -3 pid

另外提一点，Thread类提供了一个getStackTrace()方法也可以用于获取线程堆栈。这是一个实例方法，因此此方法是和具体线程实例绑定的，每次获取获取到的是具体某个线程当前运行的堆栈。

**9、一个线程如果出现了运行时异常会怎么样**

如果这个异常没有被捕获的话，这个线程就停止执行了。另外重要的一点是：如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被立即释放

1**0、如何在两个线程之间共享数据**

通过在线程之间共享对象就可以了，然后通过wait/notify/notifyAll、await/signal/signalAll进行唤起和等待，比方说阻塞队列BlockingQueue就是为线程之间共享数据而设计的

**11、sleep方法和wait方法有什么区别**

这个问题常问，sleep方法和wait方法都可以用来放弃CPU一定的时间，不同点在于如果线程持有某个对象的监视器，sleep方法不会放弃这个对象的监视器，wait方法会放弃这个对象的监视器

**12、生产者消费者模型的作用是什么**

这个问题很理论，但是很重要：

1）通过平衡生产者的生产能力和消费者的消费能力来提升整个系统的运行效率，这是生产者消费者模型最重要的作用

2）解耦，这是生产者消费者模型附带的作用，解耦意味着生产者和消费者之间的联系少，联系越少越可以独自发展而不需要收到相互的制约

**13、ThreadLocal有什么用**

简单说ThreadLocal就是一种以空间换时间的做法，在每个Thread里面维护了一个以开地址法实现的ThreadLocal.ThreadLocalMap，把数据进行隔离，数据不共享，自然就没有线程安全方面的问题了

**14、为什么wait()方法和notify()/notifyAll()方法要在同步块中被调用**

这是JDK强制的，wait()方法和notify()/notifyAll()方法在调用前都必须先获得对象的锁

**15、wait()方法和notify()/notifyAll()方法在放弃对象监视器时有什么区别**

wait()方法和notify()/notifyAll()方法在放弃对象监视器的时候的区别在于：wait()方法立即释放对象监视器，notify()/notifyAll()方法则会等待线程剩余代码执行完毕才会放弃对象监视器。

**16、为什么要使用线程池**

避免频繁地创建和销毁线程，达到线程对象的重用。另外，使用线程池还可以根据项目灵活地控制并发的数目。点击[这里](https://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247483824&idx=1&sn=7e34a3944a93d649d78d618cf04e0619&scene=21#wechat_redirect)学习线程池详解。

**17、怎么检测一个线程是否持有对象监视器**

我也是在网上看到一道多线程面试题才知道有方法可以判断某个线程是否持有对象监视器：Thread类提供了一个holdsLock(Object obj)方法，当且仅当对象obj的监视器被某条线程持有的时候才会返回true，注意这是一个static方法，这意味着"某条线程"指的是当前线程。

**18、synchronized和ReentrantLock的区别**

synchronized是和if、else、for、while一样的关键字，ReentrantLock是类，这是二者的本质区别。既然ReentrantLock是类，那么它就提供了比synchronized更多更灵活的特性，可以被继承、可以有方法、可以有各种各样的类变量，ReentrantLock比synchronized的扩展性体现在几点上：

（1）ReentrantLock可以对获取锁的等待时间进行设置，这样就避免了死锁

（2）ReentrantLock可以获取各种锁的信息

（3）ReentrantLock可以灵活地实现多路通知

另外，二者的锁机制其实也是不一样的。ReentrantLock底层调用的是Unsafe的park方法加锁，synchronized操作的应该是对象头中mark word，这点我不能确定。

**19、ConcurrentHashMap的并发度是什么**

ConcurrentHashMap的并发度就是segment的大小，默认为16，这意味着最多同时可以有16条线程操作ConcurrentHashMap，这也是ConcurrentHashMap对Hashtable的最大优势，任何情况下，Hashtable能同时有两条线程获取Hashtable中的数据吗？

**20、ReadWriteLock是什么**

首先明确一下，不是说ReentrantLock不好，只是ReentrantLock某些时候有局限。如果使用ReentrantLock，可能本身是为了防止线程A在写数据、线程B在读数据造成的数据不一致，但这样，如果线程C在读数据、线程D也在读数据，读数据是不会改变数据的，没有必要加锁，但是还是加锁了，降低了程序的性能。

因为这个，才诞生了读写锁ReadWriteLock。ReadWriteLock是一个读写锁接口，ReentrantReadWriteLock是ReadWriteLock接口的一个具体实现，实现了读写的分离，读锁是共享的，写锁是独占的，读和读之间不会互斥，读和写、写和读、写和写之间才会互斥，提升了读写的性能。

**21、FutureTask是什么**

这个其实前面有提到过，FutureTask表示一个异步运算的任务。FutureTask里面可以传入一个Callable的具体实现类，可以对这个异步运算的任务的结果进行等待获取、判断是否已经完成、取消任务等操作。当然，由于FutureTask也是Runnable接口的实现类，所以FutureTask也可以放入线程池中。

**22、Linux环境下如何查找哪个线程使用CPU最长**

这是一个比较偏实践的问题，这种问题我觉得挺有意义的。可以这么做：

（1）获取项目的pid，jps或者ps -ef | grep java，这个前面有讲过

（2）top -H -p pid，顺序不能改变

这样就可以打印出当前的项目，每条线程占用CPU时间的百分比。注意这里打出的是LWP，也就是操作系统原生线程的线程号，我笔记本山没有部署Linux环境下的Java工程，因此没有办法截图演示，网友朋友们如果公司是使用Linux环境部署项目的话，可以尝试一下。

使用"top -H -p pid"+"jps pid"可以很容易地找到某条占用CPU高的线程的线程堆栈，从而定位占用CPU高的原因，一般是因为不当的代码操作导致了死循环。

最后提一点，"top -H -p pid"打出来的LWP是十进制的，"jps pid"打出来的本地线程号是十六进制的，转换一下，就能定位到占用CPU高的线程的当前线程堆栈了。

**23、Java编程写一个会导致死锁的程序**

第一次看到这个题目，觉得这是一个非常好的问题。很多人都知道死锁是怎么一回事儿：线程A和线程B相互等待对方持有的锁导致程序无限死循环下去。当然也仅限于此了，问一下怎么写一个死锁的程序就不知道了，这种情况说白了就是不懂什么是死锁，懂一个理论就完事儿了，实践中碰到死锁的问题基本上是看不出来的。

真正理解什么是死锁，这个问题其实不难，几个步骤：

1）两个线程里面分别持有两个Object对象：lock1和lock2。这两个lock作为同步代码块的锁；

2）线程1的run()方法中同步代码块先获取lock1的对象锁，Thread.sleep(xxx)，时间不需要太多，50毫秒差不多了，然后接着获取lock2的对象锁。这么做主要是为了防止线程1启动一下子就连续获得了lock1和lock2两个对象的对象锁

3）线程2的run)(方法中同步代码块先获取lock2的对象锁，接着获取lock1的对象锁，当然这时lock1的对象锁已经被线程1锁持有，线程2肯定是要等待线程1释放lock1的对象锁的

这样，线程1"睡觉"睡完，线程2已经获取了lock2的对象锁了，线程1此时尝试获取lock2的对象锁，便被阻塞，此时一个死锁就形成了。代码就不写了，占的篇幅有点多，Java多线程7：死锁这篇文章里面有，就是上面步骤的代码实现。

点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247483887&idx=1&sn=ab85965995bde2fb539aaa4b47e7e008&chksm=eb5384d9dc240dcfa20ce9ceedcc8f6174337f74482b103d0b9fc363b5e6b6366e7ddc4590c7&scene=21#wechat_redirect)提供了一个死锁的案例。

**24、怎么唤醒一个阻塞的线程**

如果线程是因为调用了wait()、sleep()或者join()方法而导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，无能为力，因为IO是操作系统实现的，Java代码并没有办法直接接触到操作系统。

**25、不可变对象对多线程有什么帮助**

前面有提到过的一个问题，不可变对象保证了对象的内存可见性，对不可变对象的读取不需要进行额外的同步手段，提升了代码执行效率。

**26、什么是多线程的上下文切换**

多线程的上下文切换是指CPU控制权由一个已经正在运行的线程切换到另外一个就绪并等待获取CPU执行权的线程的过程。

**27、如果你提交任务时，线程池队列已满，这时会发生什么**

这里区分一下：

1）如果使用的是无界队列LinkedBlockingQueue，也就是无界队列的话，没关系，继续添加任务到阻塞队列中等待执行，因为LinkedBlockingQueue可以近乎认为是一个无穷大的队列，可以无限存放任务

2）如果使用的是有界队列比如ArrayBlockingQueue，任务首先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，会根据maximumPoolSize的值增加线程数量，如果增加了线程数量还是处理不过来，ArrayBlockingQueue继续满，那么则会使用拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy

**28、Java中用到的线程调度算法是什么**

抢占式。一个线程用完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。

**29、Thread.sleep(0)的作用是什么**

这个问题和上面那个问题是相关的，我就连在一起了。由于Java采用抢占式的线程调度算法，因此可能会出现某条线程常常获取到CPU控制权的情况，为了让某些优先级比较低的线程也能获取到CPU控制权，可以使用Thread.sleep(0)手动触发一次操作系统分配时间片的操作，这也是平衡CPU控制权的一种操作。

**30、什么是自旋**

很多synchronized里面的代码只是一些很简单的代码，执行时间非常快，此时等待的线程都加锁可能是一种不太值得的操作，因为线程阻塞涉及到用户态和内核态切换的问题。既然synchronized里面的代码执行得非常快，不妨让等待锁的线程不要被阻塞，而是在synchronized的边界做忙循环，这就是自旋。如果做了多次忙循环发现还没有获得锁，再阻塞，这样可能是一种更好的策略。

**31、什么是Java内存模型**

Java内存模型定义了一种多线程访问Java内存的规范。Java内存模型要完整讲不是这里几句话能说清楚的，我简单总结一下Java内存模型的几部分内容：

1）Java内存模型将内存分为了主内存和工作内存。类的状态，也就是类之间共享的变量，是存储在主内存中的，每次Java线程用到这些主内存中的变量的时候，会读一次主内存中的变量，并让这些内存在自己的工作内存中有一份拷贝，运行自己线程代码的时候，用到这些变量，操作的都是自己工作内存中的那一份。在线程代码执行完毕之后，会将最新的值更新到主内存中去

2）定义了几个原子操作，用于操作主内存和工作内存中的变量

3）定义了volatile变量的使用规则

4）happens-before，即先行发生原则，定义了操作A必然先行发生于操作B的一些规则，比如在同一个线程内控制流前面的代码一定先行发生于控制流后面的代码、一个释放锁unlock的动作一定先行发生于后面对于同一个锁进行锁定lock的动作等等，只要符合这些规则，则不需要额外做同步措施，如果某段代码不符合所有的happens-before规则，则这段代码一定是线程非安全的

**32、什么是CAS**CAS，全称为Compare and Swap，即比较-替换。假设有三个操作数：内存值V、旧的预期值A、要修改的值B，当且仅当预期值A和内存值V相同时，才会将内存值修改为B并返回true，否则什么都不做并返回false。当然CAS一定要volatile变量配合，这样才能保证每次拿到的变量是主内存中最新的那个值，否则旧的预期值A对某条线程来说，永远是一个不会变的值A，只要某次CAS操作失败，永远都不可能成功。更多CAS详情请点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247483728&idx=1&sn=3d734dc972a244891406cfbc443eabed&chksm=eb538466dc240d7033b889665b579a490266b8c8f1e7a08da35f67ca484dad19503e8b230e05&scene=21#wechat_redirect)学习。

**33、什么是乐观锁和悲观锁**

1）乐观锁：就像它的名字一样，对于并发间操作产生的线程安全问题持乐观状态，乐观锁认为竞争不总是会发生，因此它不需要持有锁，将比较-替换这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，那么就应该有相应的重试逻辑。

2）悲观锁：还是像它的名字一样，对于并发间操作产生的线程安全问题持悲观状态，悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像synchronized，不管三七二十一，直接上了锁就操作资源了。

点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247483753&idx=1&sn=fd09b268d8f374095c4e9e2d627f3305&chksm=eb53845fdc240d4938023e75a207cb29c615d1b5891ae340e85d0754e134af5a8b183b01ae52&scene=21#wechat_redirect)了解更多乐观锁与悲观锁详情。

**34、什么是AQS**

简单说一下AQS，AQS全称为AbstractQueuedSychronizer，翻译过来应该是抽象队列同步器。

如果说java.util.concurrent的基础是CAS的话，那么AQS就是整个Java并发包的核心了，ReentrantLock、CountDownLatch、Semaphore等等都用到了它。AQS实际上以双向队列的形式连接所有的Entry，比方说ReentrantLock，所有等待的线程都被放在一个Entry中并连成双向队列，前面一个线程使用ReentrantLock好了，则双向队列实际上的第一个Entry开始运行。

AQS定义了对双向队列所有的操作，而只开放了tryLock和tryRelease方法给开发者使用，开发者可以根据自己的实现重写tryLock和tryRelease方法，以实现自己的并发功能。

**35、单例模式的线程安全性**

老生常谈的问题了，首先要说的是单例模式的线程安全意味着：某个类的实例在多线程环境下只会被创建一次出来。单例模式有很多种的写法，我总结一下：

1）饿汉式单例模式的写法：线程安全

2）懒汉式单例模式的写法：非线程安全

3）双检锁单例模式的写法：线程安全

**36、Semaphore有什么作用**

Semaphore就是一个信号量，它的作用是限制某段代码块的并发数。Semaphore有一个构造函数，可以传入一个int型整数n，表示某段代码最多只有n个线程可以访问，如果超出了n，那么请等待，等到某个线程执行完毕这段代码块，下一个线程再进入。由此可以看出如果Semaphore构造函数中传入的int型整数n=1，相当于变成了一个synchronized了。

**37、Hashtable的size()方法中明明只有一条语句"return count"，为什么还要做同步？**

这是我之前的一个困惑，不知道大家有没有想过这个问题。某个方法中如果有多条语句，并且都在操作同一个类变量，那么在多线程环境下不加锁，势必会引发线程安全问题，这很好理解，但是size()方法明明只有一条语句，为什么还要加锁？

关于这个问题，在慢慢地工作、学习中，有了理解，主要原因有两点：

1）同一时间只能有一条线程执行固定类的同步方法，但是对于类的非同步方法，可以多条线程同时访问。所以，这样就有问题了，可能线程A在执行Hashtable的put方法添加数据，线程B则可以正常调用size()方法读取Hashtable中当前元素的个数，那读取到的值可能不是最新的，可能线程A添加了完了数据，但是没有对size++，线程B就已经读取size了，那么对于线程B来说读取到的size一定是不准确的。而给size()方法加了同步之后，意味着线程B调用size()方法只有在线程A调用put方法完毕之后才可以调用，这样就保证了线程安全性

2）CPU执行代码，执行的不是Java代码，这点很关键，一定得记住。Java代码最终是被翻译成机器码执行的，机器码才是真正可以和硬件电路交互的代码。即使你看到Java代码只有一行，甚至你看到Java代码编译之后生成的字节码也只有一行，也不意味着对于底层来说这句语句的操作只有一个。一句"return count"假设被翻译成了三句汇编语句执行，一句汇编语句和其机器码做对应，完全可能执行完第一句，线程就切换了。

**38、线程类的构造方法、静态块是被哪个线程调用的**

这是一个非常刁钻和狡猾的问题。请记住：线程类的构造方法、静态块是被new这个线程类所在的线程所调用的，而run方法里面的代码才是被线程自身所调用的。

如果说上面的说法让你感到困惑，那么我举个例子，假设Thread2中new了Thread1，main函数中new了Thread2，那么：

1）Thread2的构造方法、静态块是main线程调用的，Thread2的run()方法是Thread2自己调用的

2）Thread1的构造方法、静态块是Thread2调用的，Thread1的run()方法是Thread1自己调用的

**39、同步方法和同步块，哪个是更好的选择**

同步块，这意味着同步块之外的代码是异步执行的，这比同步整个方法更提升代码的效率。请知道一条原则：同步的范围越小越好。

借着这一条，我额外提一点，虽说同步的范围越少越好，但是在Java虚拟机中还是存在着一种叫做锁粗化的优化方法，这种方法就是把同步范围变大。这是有用的，比方说StringBuffer，它是一个线程安全的类，自然最常用的append()方法是一个同步方法，我们写代码的时候会反复append字符串，这意味着要进行反复的加锁->解锁，这对性能不利，因为这意味着Java虚拟机在这条线程上要反复地在内核态和用户态之间进行切换，因此Java虚拟机会将多次append方法调用的代码进行一个锁粗化的操作，将多次的append的操作扩展到append方法的头尾，变成一个大的同步块，这样就减少了加锁-->解锁的次数，有效地提升了代码执行的效率。

**40、高并发、任务执行时间短的业务怎样使用线程池？并发不高、任务执行时间长的业务怎样使用线程池？并发高、业务执行时间长的业务怎样使用线程池？**

这是我在并发编程网上看到的一个问题，把这个问题放在最后一个，希望每个人都能看到并且思考一下，因为这个问题非常好、非常实际、非常专业。关于这个问题，个人看法是：

1）高并发、任务执行时间短的业务，线程池线程数可以设置为CPU核数+1，减少线程上下文的切换

2）并发不高、任务执行时间长的业务要区分开看：

a）假如是业务时间长集中在IO操作上，也就是IO密集型的任务，因为IO操作并不占用CPU，所以不要让所有的CPU闲下来，可以加大线程池中的线程数目，让CPU处理更多的业务

b）假如是业务时间长集中在计算操作上，也就是计算密集型任务，这个就没办法了，和（1）一样吧，线程池中的线程数设置得少一些，减少线程上下文的切换

c）并发高、业务执行时间长，解决这种类型任务的关键不在于线程池而在于整体架构的设计，看看这些业务里面某些数据是否能做缓存是第一步，增加服务器是第二步，至于线程池的设置，设置参考其他有关线程池的文章。最后，业务执行时间长的问题，也可能需要分析一下，看看能不能使用中间件对任务进行拆分和解耦。

**DUBBO**

**1、Dubbo是什么？**

Dubbo是阿里巴巴开源的基于 Java 的高性能 RPC 分布式服务框架，现已成为 Apache 基金会孵化项目。

面试官问你如果这个都不清楚，那下面的就没必要问了。

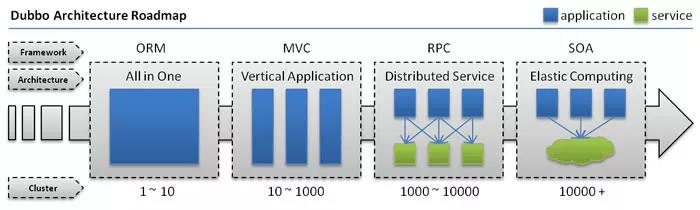
官网：http://dubbo.apache.org

**2、为什么要用Dubbo？**

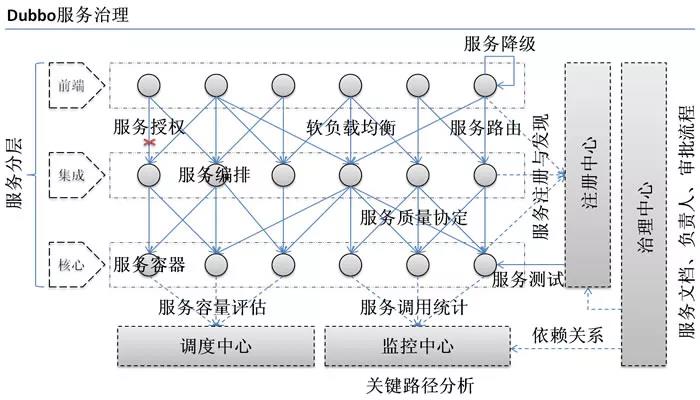
因为是阿里开源项目，国内很多互联网公司都在用，已经经过很多线上考验。内部使用了 Netty、Zookeeper，保证了高性能高可用性。

使用 Dubbo 可以将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，可用于提高业务复用灵活扩展，使前端应用能更快速的响应多变的市场需求。

下面这张图可以很清楚的诠释，最重要的一点是，分布式架构可以承受更大规模的并发流量。



下面是 Dubbo 的服务治理图。



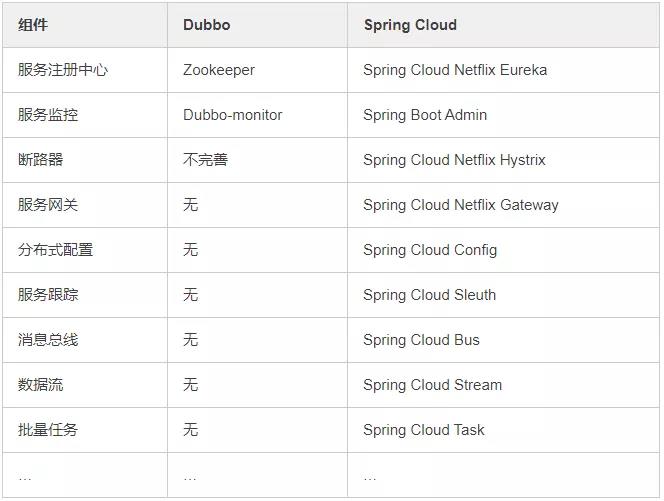
**3、Dubbo 和 Spring Cloud 有什么区别？**

两个没关联，如果硬要说区别，有以下几点。

1）通信方式不同

Dubbo 使用的是 RPC 通信，而 Spring Cloud 使用的是 HTTP RESTFul 方式。

2）组成部分不同



**4、dubbo都支持什么协议，推荐用哪种？**

* dubbo://（推荐）
* rmi://
* hessian://
* http://
* webservice://
* thrift://
* memcached://
* redis://
* rest://

**5、Dubbo需要 Web 容器吗？**

不需要，如果硬要用 Web 容器，只会增加复杂性，也浪费资源。

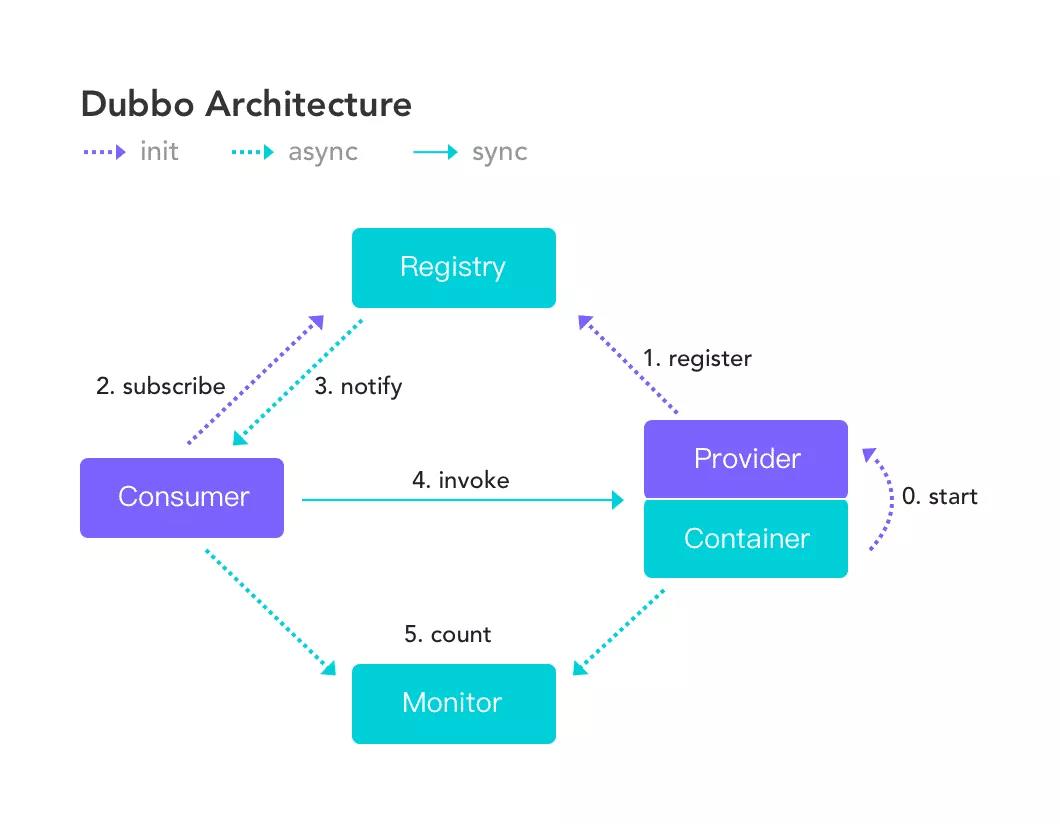
**6、Dubbo内置了哪几种服务容器？**

* Spring Container
* Jetty Container
* Log4j Container
* Dubbo 的服务容器只是一个简单的 Main 方法，并加载一个简单的 Spring 容器，用于暴露服务。

**7、Dubbo里面有哪几种节点角色？**

****

**8、画一画服务注册与发现的流程图**



该图来自 Dubbo 官网，供你参考，如果你说你熟悉 Dubbo, 面试官经常会让你画这个图，记好了。

**9、Dubbo默认使用什么注册中心，还有别的选择吗？**

推荐使用 Zookeeper 作为注册中心，还有 Redis、Multicast、Simple 注册中心，但不推荐。

**10、Dubbo有哪几种配置方式？**

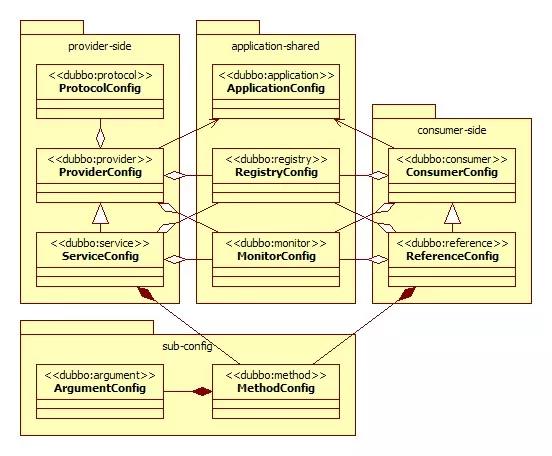
1）Spring 配置方式  
2）Java API 配置方式

**11、Dubbo 核心的配置有哪些？**

我曾经面试就遇到过面试官让你写这些配置，我也是蒙逼。



配置之间的关系见下图。



**12、在 Provider 上可以配置的 Consumer 端的属性有哪些？**

1）timeout：方法调用超时  
2）retries：失败重试次数，默认重试 2 次  
3）loadbalance：负载均衡算法，默认随机  
4）actives 消费者端，最大并发调用限制

**13、Dubbo启动时如果依赖的服务不可用会怎样？**

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，默认 check="true"，可以通过 check="false" 关闭检查。

**14、Dubbo推荐使用什么序列化框架，你知道的还有哪些？**

**推荐使用Hessian序列化，还有Duddo、FastJson、Java**自带序列化。

**15、Dubbo默认使用的是什么通信框架，还有别的选择吗？**

Dubbo 默认使用 Netty 框架，也是推荐的选择，另外内容还集成有Mina、Grizzly。

**16、Dubbo有哪几种集群容错方案，默认是哪种？**



**17、Dubbo有哪几种负载均衡策略，默认是哪种？**



**18、注册了多个同一样的服务，如果测试指定的某一个服务呢？**

可以配置环境点对点直连，绕过注册中心，将以服务接口为单位，忽略注册中心的提供者列表。

**19、Dubbo支持服务多协议吗？**

Dubbo 允许配置多协议，在不同服务上支持不同协议或者同一服务上同时支持多种协议。

**20、当一个服务接口有多种实现时怎么做？**

当一个接口有多种实现时，可以用 group 属性来分组，服务提供方和消费方都指定同一个 group 即可。

**21、服务上线怎么兼容旧版本？**

可以用版本号（version）过渡，多个不同版本的服务注册到注册中心，版本号不同的服务相互间不引用。这个和服务分组的概念有一点类似。

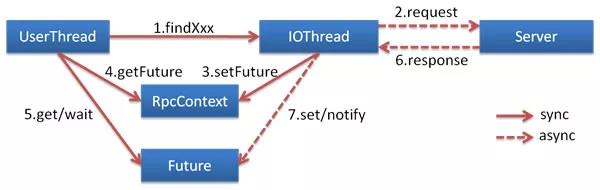
**22、Dubbo可以对结果进行缓存吗？**

可以，Dubbo 提供了声明式缓存，用于加速热门数据的访问速度，以减少用户加缓存的工作量。

**23、Dubbo服务之间的调用是阻塞的吗？**

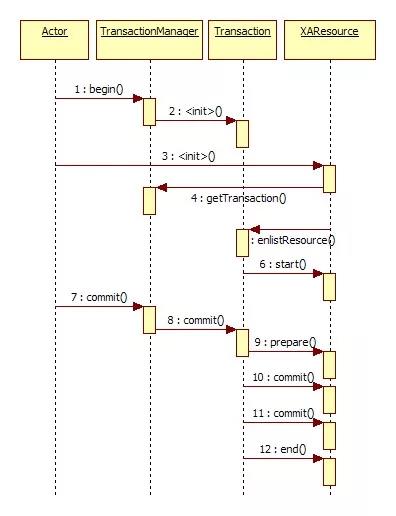
默认是同步等待结果阻塞的，支持异步调用。

Dubbo 是基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小，异步调用会返回一个 Future 对象。

异步调用流程图如下。

**24、Dubbo支持分布式事务吗？**

目前暂时不支持，后续可能采用基于 JTA/XA 规范实现，如以图所示。



**25、Dubbo telnet 命令能做什么？**

dubbo 通过 telnet 命令来进行服务治理

telnet localhost 8090

**26、Dubbo支持服务降级吗？**

Dubbo 2.2.0 以上版本支持。

**27、Dubbo如何优雅停机？**

Dubbo 是通过 JDK 的 ShutdownHook 来完成优雅停机的，所以如果使用 kill -9 PID 等强制关闭指令，是不会执行优雅停机的，只有通过 kill PID 时，才会执行。

**28、服务提供者能实现失效踢出是什么原理？**

服务失效踢出基于 Zookeeper 的临时节点原理。

**29、如何解决服务调用链过长的问题？**

Dubbo 可以使用 Pinpoint 和 Apache Skywalking(Incubator) 实现分布式服务追踪，当然还有其他很多方案。

**30、服务读写推荐的容错策略是怎样的？**

读操作建议使用 Failover 失败自动切换，默认重试两次其他服务器。

写操作建议使用 Failfast 快速失败，发一次调用失败就立即报错。

**31、Dubbo必须依赖的包有哪些？**

Dubbo 必须依赖 JDK，其他为可选。

**32、Dubbo的管理控制台能做什么？**

管理控制台主要包含：路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡，等管理功能。

**33、说说 Dubbo 服务暴露的过程。**

Dubbo 会在 Spring 实例化完 bean 之后，在刷新容器最后一步发布 ContextRefreshEvent 事件的时候，通知实现了 ApplicationListener 的 ServiceBean 类进行回调 onApplicationEvent 事件方法，Dubbo 会在这个方法中调用 ServiceBean 父类 ServiceConfig 的 export 方法，而该方法真正实现了服务的（异步或者非异步）发布。

**34、Dubbo 停止维护了吗？**

2014 年开始停止维护过几年，17 年开始重新维护，并进入了 Apache 项目。

**35、Dubbo 和 Dubbox 有什么区别？**

Dubbox 是继 Dubbo 停止维护后，当当网基于 Dubbo 做的一个扩展项目，如加了服务可 Restful 调用，更新了开源组件等。

**36、你还了解别的分布式框架吗？**

别的还有 Spring cloud、Facebook 的 Thrift、Twitter 的 Finagle 等。

**37、Dubbo 能集成 Spring Boot 吗？**

可以的，项目地址如下。

https://github.com/apache/incubator-dubbo-spring-boot-project

**38、在使用过程中都遇到了些什么问题？**

Dubbo 的设计目的是为了满足高并发小数据量的 rpc 调用，在大数据量下的性能表现并不好，建议使用 rmi 或 http 协议。

**39、你读过 Dubbo 的源码吗？**

要了解 Dubbo 就必须看其源码，了解其原理，花点时间看下吧，网上也有很多教程。

**40、你觉得用 Dubbo 好还是 Spring Cloud 好？**

扩展性的问题，没有好坏，只有适合不适合，不过我好像更倾向于使用 Dubbo, Spring Cloud 版本升级太快，组件更新替换太频繁，配置太繁琐，还有很多我觉得是没有 Dubbo 顺手的地方……

**Redis（含答案）：**

**1、什么是Redis？**

Redis本质上是一个Key-Value类型的内存数据库，很像memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据flush到硬盘上进行保存。因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作，是已知性能最快的Key-Value DB。

Redis的出色之处不仅仅是性能，Redis最大的魅力是支持保存多种数据结构，此外单个value的最大限制是1GB，不像 memcached只能保存1MB的数据，因此Redis可以用来实现很多有用的功能，比方说用他的List来做FIFO双向链表，实现一个轻量级的高性 能消息队列服务，用他的Set可以做高性能的tag系统等等。另外Redis也可以对存入的Key-Value设置expire时间，因此也可以被当作一 个功能加强版的memcached来用。

Redis的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此Redis适合的场景主要局限在较小数据量的高性能操作和运算上。

**2、Redis相比memcached有哪些优势？**

(1) memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型

(2) redis的速度比memcached快很多

(3) redis可以持久化其数据

**3、Redis支持哪几种数据类型？**

String、List、Set、Sorted Set、hashes

**4、Redis主要消耗什么物理资源？**

内存。

**5、Redis的全称是什么？**

Remote Dictionary Server。

**6、Redis有哪几种数据淘汰策略？**

noeviction:返回错误当内存限制达到并且客户端尝试执行会让更多内存被使用的命令（大部分的写入指令，但DEL和几个例外）

allkeys-lru: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。

volatile-lru: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键,使得新添加的数据有空间存放。

allkeys-random: 回收随机的键使得新添加的数据有空间存放。

volatile-random: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。

volatile-ttl: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键,使得新添加的数据有空间存放。

**7、Redis官方为什么不提供Windows版本？**

因为目前Linux版本已经相当稳定，而且用户量很大，无需开发windows版本，反而会带来兼容性等问题。

**8、一个字符串类型的值能存储最大容量是多少？**

512M

**9、为什么Redis需要把所有数据放到内存中？**

Redis为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以redis具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘I/O速度为严重影响redis的性能。在内存越来越便宜的今天，redis将会越来越受欢迎。

如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

**10、Redis集群方案应该怎么做？都有哪些方案？**

1.twemproxy，大概概念是，它类似于一个代理方式，使用方法和普通redis无任何区别，设置好它下属的多个redis实例后，使用时在本需要连接redis的地方改为连接twemproxy，它会以一个代理的身份接收请求并使用一致性hash算法，将请求转接到具体redis，将结果再返回twemproxy。使用方式简便(相对redis只需修改连接端口)，对旧项目扩展的首选。 问题：twemproxy自身单端口实例的压力，使用一致性hash后，对redis节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。

2.codis，目前用的最多的集群方案，基本和twemproxy一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新hash节点。

3.redis cluster3.0自带的集群，特点在于他的分布式算法不是一致性hash，而是hash槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

4.在业务代码层实现，起几个毫无关联的redis实例，在代码层，对key 进行hash计算，然后去对应的redis实例操作数据。 这种方式对hash层代码要求比较高，考虑部分包括，节点失效后的替代算法方案，数据震荡后的自动脚本恢复，实例的监控，等等。

**11、Redis集群方案什么情况下会导致整个集群不可用？**

有A，B，C三个节点的集群,在没有复制模型的情况下,如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用。

**12、MySQL里有2000w数据，redis中只存20w的数据，**

如何保证redis中的数据都是热点数据？

redis内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

**13、Redis有哪些适合的场景？**

（1）、会话缓存（Session Cache）

最常用的一种使用Redis的情景是会话缓存（session cache）。用Redis缓存会话比其他存储（如Memcached）的优势在于：Redis提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？

幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用Redis来缓存会话的文档。甚至广为人知的商业平台Magento也提供Redis的插件。

（2）、全页缓存（FPC）

除基本的会话token之外，Redis还提供很简便的FPC平台。回到一致性问题，即使重启了Redis实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似PHP本地FPC。

再次以Magento为例，Magento提供一个插件来使用Redis作为全页缓存后端。

此外，对WordPress的用户来说，Pantheon有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

（3）、队列

Reids在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得Redis能作为一个很好的消息队列平台来使用。Redis作为队列使用的操作，就类似于本地程序语言（如Python）对 list 的 push/pop 操作。

如果你快速的在Google中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用Redis创建非常好的后端工具，以满足各种队列需求。例如，Celery有一个后台就是使用Redis作为broker，你可以从这里去查看。

（4），排行榜/计数器

Redis在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变的非常简单，Redis只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的10个用户–我们称之为“user\_scores”，我们只需要像下面一样执行即可：

当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行：

ZRANGE user\_scores 0 10 WITHSCORES

Agora Games就是一个很好的例子，用Ruby实现的，它的排行榜就是使用Redis来存储数据的，你可以在这里看到。

（5）、发布/订阅

最后（但肯定不是最不重要的）是Redis的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用Redis的发布/订阅功能来建立聊天系统！（不，这是真的，你可以去核实）。

**14、Redis支持的Java客户端都有哪些？官方推荐用哪个？**

Redisson、Jedis、lettuce等等，官方推荐使用Redisson。

**15、Redis和Redisson有什么关系？**

Redisson是一个高级的分布式协调Redis客服端，能帮助用户在分布式环境中轻松实现一些Java的对象 (Bloom filter, BitSet, Set, SetMultimap, ScoredSortedSet, SortedSet, Map, ConcurrentMap, List, ListMultimap, Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, ReadWriteLock, AtomicLong, CountDownLatch, Publish / Subscribe, HyperLogLog)。

**16、Jedis与Redisson对比有什么优缺点？**

Jedis是Redis的Java实现的客户端，其API提供了比较全面的Redis命令的支持；Redisson实现了分布式和可扩展的Java数据结构，和Jedis相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等Redis特性。Redisson的宗旨是促进使用者对Redis的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

**17、Redis如何设置密码及验证密码？**

设置密码：config set requirepass 123456

授权密码：auth 123456

**18、说说Redis哈希槽的概念？**

Redis集群没有使用一致性hash,而是引入了哈希槽的概念，Redis集群有16384个哈希槽，每个key通过CRC16校验后对16384取模来决定放置哪个槽，集群的每个节点负责一部分hash槽。

**19、Redis集群的主从复制模型是怎样的？**

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有N-1个复制品.

**20、Redis集群会有写操作丢失吗？为什么？**

Redis并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

**21、Redis集群之间是如何复制的？**

异步复制

**22、Redis集群最大节点个数是多少？**

16384个。

**23、Redis集群如何选择数据库？**

Redis集群目前无法做数据库选择，默认在0数据库。

**24、怎么测试Redis的连通性？**

Ping

**25、Redis中的管道有什么用？**

一次请求/响应服务器能实现处理新的请求即使旧的请求还未被响应。这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。

这就是管道（pipelining），是一种几十年来广泛使用的技术。例如许多POP3协议已经实现支持这个功能，大大加快了从服务器下载新邮件的过程。

**26、怎么理解Redis事务？**

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

**27、Redis事务相关的命令有哪几个？**

MULTI、EXEC、DISCARD、WATCH

**28、Redis key的过期时间和永久有效分别怎么设置？**

EXPIRE和PERSIST命令。

**29、Redis如何做内存优化？**

尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的web系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的key,而是应该把这个用户的所有信息存储到一张散列表里面.

**30、Redis回收进程如何工作的？**

1. 一个客户端运行了新的命令，添加了新的数据。
2. Redi检查内存使用情况，如果大于maxmemory的限制, 则根据设定好的策略进行回收。
3. 一个新的命令被执行，等等。
4. 所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地回收回到边界以下。

如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

**31、Redis回收使用的是什么算法？**

LRU算法

**32、Redis如何做大量数据插入？**

Redis2.6开始redis-cli支持一种新的被称之为pipe mode的新模式用于执行大量数据插入工作。

**33、为什么要做Redis分区？**

分区可以让Redis管理更大的内存，Redis将可以使用所有机器的内存。如果没有分区，你最多只能使用一台机器的内存。分区使Redis的计算能力通过简单地增加计算机得到成倍提升,Redis的网络带宽也会随着计算机和网卡的增加而成倍增长。

**34、你知道有哪些Redis分区实现方案？**

* 客户端分区就是在客户端就已经决定数据会被存储到哪个redis节点或者从哪个redis节点读取。大多数客户端已经实现了客户端分区。
* 代理分区 意味着客户端将请求发送给代理，然后代理决定去哪个节点写数据或者读数据。代理根据分区规则决定请求哪些Redis实例，然后根据Redis的响应结果返回给客户端。redis和memcached的一种代理实现就是Twemproxy
* 查询路由(Query routing) 的意思是客户端随机地请求任意一个redis实例，然后由Redis将请求转发给正确的Redis节点。Redis Cluster实现了一种混合形式的查询路由，但并不是直接将请求从一个redis节点转发到另一个redis节点，而是在客户端的帮助下直接redirected到正确的redis节点。

**35、Redis分区有什么缺点？**

* 涉及多个key的操作通常不会被支持。例如你不能对两个集合求交集，因为他们可能被存储到不同的Redis实例（实际上这种情况也有办法，但是不能直接使用交集指令）。
* 同时操作多个key,则不能使用Redis事务.
* 分区使用的粒度是key，不能使用一个非常长的排序key存储一个数据集（The partitioning granularity is the key, so it is not possible to shard a dataset with a single huge key like a very big sorted set）.
* 当使用分区的时候，数据处理会非常复杂，例如为了备份你必须从不同的Redis实例和主机同时收集RDB / AOF文件。
* 分区时动态扩容或缩容可能非常复杂。Redis集群在运行时增加或者删除Redis节点，能做到最大程度对用户透明地数据再平衡，但其他一些客户端分区或者代理分区方法则不支持这种特性。然而，有一种预分片的技术也可以较好的解决这个问题。

**36、Redis持久化数据和缓存怎么做扩容？**

* 如果Redis被当做缓存使用，使用一致性哈希实现动态扩容缩容。
* 如果Redis被当做一个持久化存储使用，必须使用固定的keys-to-nodes映射关系，节点的数量一旦确定不能变化。否则的话(即Redis节点需要动态变化的情况），必须使用可以在运行时进行数据再平衡的一套系统，而当前只有Redis集群可以做到这样。

**37、分布式Redis是前期做还是后期规模上来了再做好？为什么？**

既然Redis是如此的轻量（单实例只使用1M内存）,为防止以后的扩容，最好的办法就是一开始就启动较多实例。即便你只有一台服务器，你也可以一开始就让Redis以分布式的方式运行，使用分区，在同一台服务器上启动多个实例。

一开始就多设置几个Redis实例，例如32或者64个实例，对大多数用户来说这操作起来可能比较麻烦，但是从长久来看做这点牺牲是值得的。

这样的话，当你的数据不断增长，需要更多的Redis服务器时，你需要做的就是仅仅将Redis实例从一台服务迁移到另外一台服务器而已（而不用考虑重新分区的问题）。一旦你添加了另一台服务器，你需要将你一半的Redis实例从第一台机器迁移到第二台机器。

**38、Twemproxy是什么？**

Twemproxy是Twitter维护的（缓存）代理系统，代理Memcached的ASCII协议和Redis协议。它是单线程程序，使用c语言编写，运行起来非常快。它是采用Apache 2.0 license的开源软件。 Twemproxy支持自动分区，如果其代理的其中一个Redis节点不可用时，会自动将该节点排除（这将改变原来的keys-instances的映射关系，所以你应该仅在把Redis当缓存时使用Twemproxy)。 Twemproxy本身不存在单点问题，因为你可以启动多个Twemproxy实例，然后让你的客户端去连接任意一个Twemproxy实例。 Twemproxy是Redis客户端和服务器端的一个中间层，由它来处理分区功能应该不算复杂，并且应该算比较可靠的。

**39、支持一致性哈希的客户端有哪些？**

Redis-rb、Predis等。

**40、Redis与其他key-value存储有什么不同？**

1. Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
2. Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，应为数据量不能大于硬件内存。在内存数据库方面的另一个优点是， 相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。 同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

**41、Redis的内存占用情况怎么样？**

给你举个例子： 100万个键值对（键是0到999999值是字符串“hello world”）在我的32位的Mac笔记本上 用了100MB。同样的数据放到一个key里只需要16MB， 这是因为键值有一个很大的开销。 在Memcached上执行也是类似的结果，但是相对Redis的开销要小一点点，因为Redis会记录类型信息引用计数等等。

当然，大键值对时两者的比例要好很多。

64位的系统比32位的需要更多的内存开销，尤其是键值对都较小时，这是因为64位的系统里指针占用了8个字节。 但是，当然，64位系统支持更大的内存，所以为了运行大型的Redis服务器或多或少的需要使用64位的系统。

**42、都有哪些办法可以降低Redis的内存使用情况呢？**

如果你使用的是32位的Redis实例，可以好好利用Hash,list,sorted set,set等集合类型数据，因为通常情况下很多小的Key-Value可以用更紧凑的方式存放到一起。

**43、查看Redis使用情况及状态信息用什么命令？**

Info

**44、Redis的内存用完了会发生什么？**

如果达到设置的上限，Redis的写命令会返回错误信息（但是读命令还可以正常返回。）或者你可以将Redis当缓存来使用配置淘汰机制，当Redis达到内存上限时会冲刷掉旧的内容。

**45、Redis是单线程的，如何提高多核CPU的利用率？**

可以在同一个服务器部署多个Redis的实例，并把他们当作不同的服务器来使用，在某些时候，无论如何一个服务器是不够的， 所以，如果你想使用多个CPU，你可以考虑一下分片（shard）。

**46、一个Redis实例最多能存放多少的keys？List、Set、**Sorted Set他们最多能存放多少元素？

理论上Redis可以处理多达232的keys，并且在实际中进行了测试，每个实例至少存放了2亿5千万的keys。我们正在测试一些较大的值。

任何list、set、和sorted set都可以放232个元素。

换句话说，Redis的存储极限是系统中的可用内存值。

**47、Redis常见性能问题和解决方案？**

(1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

(2) 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

**48、Redis提供了哪几种持久化方式？**

1. RDB持久化方式能够在指定的时间间隔能对你的数据进行快照存储.
2. AOF持久化方式记录每次对服务器写的操作,当服务器重启的时候会重新执行这些命令来恢复原始的数据,AOF命令以redis协议追加保存每次写的操作到文件末尾.Redis还能对AOF文件进行后台重写,使得AOF文件的体积不至于过大.
3. 如果你只希望你的数据在服务器运行的时候存在,你也可以不使用任何持久化方式.
4. 你也可以同时开启两种持久化方式, 在这种情况下, 当redis重启的时候会优先载入AOF文件来恢复原始的数据,因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整.
5. 最重要的事情是了解RDB和AOF持久化方式的不同,让我们以RDB持久化方式开始。

**49、如何选择合适的持久化方式？**

一般来说， 如果想达到足以媲美PostgreSQL的数据安全性， 你应该同时使用两种持久化功能。如果你非常关心你的数据， 但仍然可以承受数分钟以内的数据丢失，那么你可以只使用RDB持久化。

有很多用户都只使用AOF持久化，但并不推荐这种方式：因为定时生成RDB快照（snapshot）非常便于进行数据库备份， 并且 RDB 恢复数据集的速度也要比AOF恢复的速度要快，除此之外， 使用RDB还可以避免之前提到的AOF程序的bug。

**50、修改配置不重启Redis会实时生效吗？**

针对运行实例，有许多配置选项可以通过 CONFIG SET 命令进行修改，而无需执行任何形式的重启。 从 Redis 2.2 开始，可以从 AOF 切换到 RDB 的快照持久性或其他方式而不需要重启 Redis。检索 ‘CONFIG GET \*’ 命令获取更多信息。

但偶尔重新启动是必须的，如为升级 Redis 程序到新的版本，或者当你需要修改某些目前 CONFIG 命令还不支持的配置参数的时候。

**Java泛型**

**1.Java中的泛型是什么 ? 使用泛型的好处是什么?**

这是在各种Java泛型面试中，一开场你就会被问到的问题中的一个，主要集中在初级和中级面试中。那些拥有Java1.4或更早版本的开发背景的人都知道，在集合中存储对象并在使用前进行类型转换是多么的不方便。

泛型防止了那种情况的发生。它提供了编译期的类型安全，确保你只能把正确类型的对象放入集合中，避免了在运行时出现ClassCastException。

**2. Java的泛型是如何工作的 ? 什么是类型擦除 ?**

这是一道更好的泛型面试题。泛型是通过类型擦除来实现的，编译器在编译时擦除了所有类型相关的信息，所以在运行时不存在任何类型相关的信息。例如List<String>在运行时仅用一个List来表示。这样做的目的，是确保能和Java 5之前的版本开发二进制类库进行兼容。

你无法在运行时访问到类型参数，因为编译器已经把泛型类型转换成了原始类型。根据你对这个泛型问题的回答情况，你会得到一些后续提问，比如为什么泛型是由类型擦除来实现的或者给你展示一些会导致编译器出错的错误泛型代码。

**3. 什么是泛型中的限定通配符和非限定通配符 ?**

这是另一个非常流行的Java泛型面试题。限定通配符对类型进行了限制。有两种限定通配符，一种是<? extends T>它通过确保类型必须是T的子类来设定类型的上界，另一种是<? super T>它通过确保类型必须是T的父类来设定类型的下界。泛型类型必须用限定内的类型来进行初始化，否则会导致编译错误。另一方面<?>表示了非限定通配符，因为<?>可以用任意类型来替代。

**4. List<? extends T>和List <? super T>之间有什么区别 ?**

这和上一个面试题有联系，有时面试官会用这个问题来评估你对泛型的理解，而不是直接问你什么是限定通配符和非限定通配符。这两个List的声明都是限定通配符的例子，List<? extends T>可以接受任何继承自T的类型的List，而List<? super T>可以接受任何T的父类构成的List。例如List<? extends Number>可以接受List<Integer>或List<Float>。在本段出现的连接中可以找到更多信息。

**5. 如何编写一个泛型方法，让它能接受泛型参数并返回泛型类型?**

编写泛型方法并不困难，你需要用泛型类型来替代原始类型，比如使用T, E or K,V等被广泛认可的类型占位符。泛型方法的例子请参阅Java集合类框架。最简单的情况下，一个泛型方法可能会像这样:

public V put(K key, V value) {

　return cache.put(key, value);

}

**6. Java中如何使用泛型编写带有参数的类?**

这是上一道面试题的延伸。面试官可能会要求你用泛型编写一个类型安全的类，而不是编写一个泛型方法。关键仍然是使用泛型类型来代替原始类型，而且要使用JDK中采用的标准占位符。

**7. 编写一段泛型程序来实现LRU缓存?**

对于喜欢Java编程的人来说这相当于是一次练习。给你个提示，LinkedHashMap可以用来实现固定大小的LRU缓存，当LRU缓存已经满了的时候，它会把最老的键值对移出缓存。

LinkedHashMap提供了一个称为removeEldestEntry()的方法，该方法会被put()和putAll()调用来删除最老的键值对。当然，如果你已经编写了一个可运行的JUnit测试，你也可以随意编写你自己的实现代码。

**8. 你可以把List<String>传递给一个接受List<Object>参数的方法吗？**

对任何一个不太熟悉泛型的人来说，这个Java泛型题目看起来令人疑惑，因为乍看起来String是一种Object，所以List<String>应当可以用在需要List<Object>的地方，但是事实并非如此。真这样做的话会导致编译错误。

如果你再深一步考虑，你会发现Java这样做是有意义的，因为List<Object>可以存储任何类型的对象包括String, Integer等等，而List<String>却只能用来存储Strings。

List<Object> objectList;

List<String> stringList;

objectList = stringList;

**9. Array中可以用泛型吗?**

这可能是Java泛型面试题中最简单的一个了，当然前提是你要知道Array事实上并不支持泛型，这也是为什么Joshua Bloch在Effective Java一书中建议使用List来代替Array，因为List可以提供编译期的类型安全保证，而Array却不能。

**10. 如何阻止Java中的类型未检查的警告?**

如果你把泛型和原始类型混合起来使用，例如下列代码，Java 5的javac编译器会产生类型未检查的警告，例如

 List<String> rawList = new ArrayList();

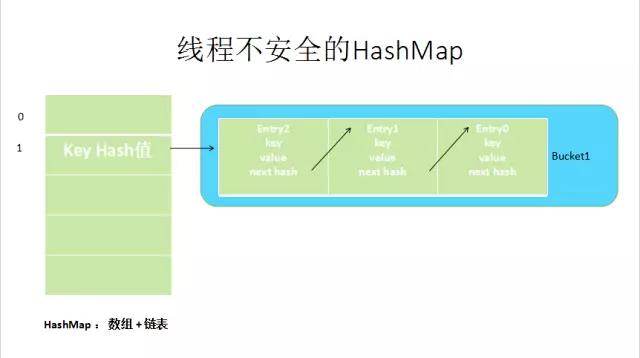
**HashMap线程安全问题：（含答案）**

Java中平时用的最多的Map集合就是HashMap了，它是线程不安全的。

看下面两个场景：

1、当用在方法内的局部变量时，局部变量属于当前线程级别的变量，其他线程访问不了，所以这时也不存在线程安全不安全的问题了。

2、当用在单例对象成员变量的时候呢？这时候多个线程过来访问的就是同一个HashMap了，对同个HashMap操作这时候就存在线程安全的问题了。



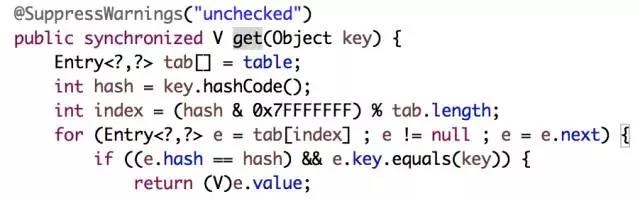
**线程安全的Map**

为了避免出现场景2的线程安全的问题，不能使用HashMap作为成员变量，要寻求使用线程安全的Map，下面来总结下有哪些线程安全的Map呢？

**1、HashTable**

private Map<String, Object> map = new Hashtable<>();

来看看HashTable的源码



HashTable的get/put方法都被synchronized关键字修饰，说明它们是方法级别阻塞的，它们占用共享资源锁，所以导致同时只能一个线程操作get或者put，而且get/put操作不能同时执行，所以这种同步的集合效率非常低，一般不建议使用这个集合。

**2、SynchronizedMap**

private Map<String, Object> map = Collections.synchronizedMap(newHashMap<String, Object>());

这种是直接使用工具类里面的方法创建SynchronizedMap，把传入进行的HashMap对象进行了包装同步而已，来看看它的源码。

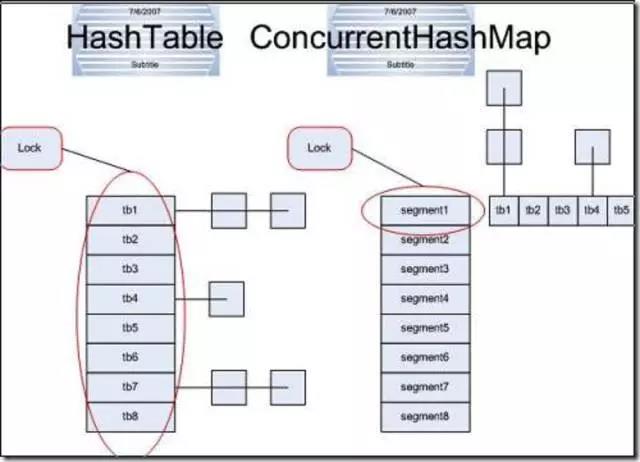


这个同步方式实现也比较简单，看出SynchronizedMap的实现方式是加了个对象锁，每次对HashMap的操作都要先获取这个mutex的对象锁才能进入，所以性能也不会比HashTable好到哪里去，也不建议使用。

**3、ConcurrentHashMap - 推荐**

private Map<String, Object> map = new ConcurrentHashMap<>();

这个也是最推荐使用的线程安全的Map，也是实现方式最复杂的一个集合，每个版本的实现方式也不一样，在jdk8之前是使用分段加锁的一个方式，分成16个桶，每次只加锁其中一个桶，而在jdk8又加入了红黑树和CAS算法来实现。



虽然实现起来很复杂，但使用起来也是非常简单的，在java面试中问的频率也非常高，最重要的是性能要比上面两种同步方式要快太多，推荐使用。

**计算机网络问题：**

**GET**

请注意，查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：/test/demo\_form.asp?name1=value1&name2=value2

1. GET 请求可被缓存
2. GET 请求保留在浏览器历史记录中
3. GET 请求可被收藏为书签
4. GET 请求不应在处理敏感数据时使用
5. GET 请求有长度限制
6. GET 请求只应当用于取回数据POST 方法**（POST）**请注意，查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：POST /test/demo\_form.asp HTTP/1.1Host: w3schools.comname1=value1&name2=value2
7. POST 请求不会被缓存
8. POST 请求不会保留在浏览器历史记录中
9. POST 不能被收藏为书签
10. POST 请求对数据长度没有要求

## dns使用的协议：

## 既使用TCP又使用UDP **首先了解一下TCP与UDP传送字节的长度限制：**

UDP报文的最大长度为512字节，而TCP则允许报文长度超过512字节。当DNS查询超过512字节时，协议的TC标志出现删除标志，这时则使用TCP发送。通常传统的UDP报文一般不会大于512字节。

**区域传送时使用TCP，主要有一下两点考虑：**

1. 辅域名服务器会定时（一般时3小时）向主域名服务器进行查询以便了解数据是否有变动。如有变动，则会执行一次区域传送，进行数据同步。区域传送将使用TCP而不是UDP，因为数据同步传送的数据量比一个请求和应答的数据量要多得多。
2. TCP是一种可靠的连接，保证了数据的准确性。

**域名解析时使用UDP协议：**

客户端向DNS服务器查询域名，一般返回的内容都不超过512字节，用UDP传输即可。不用经过TCP三次握手，这样DNS服务器负载更低，响应更快。虽然从理论上说，客户端也可以指定向DNS服务器查询的时候使用TCP，但事实上，很多DNS服务器进行配置的时候，仅支持UDP查询包。

## 幂等

一个幂等操作的特点是其任意多次执行所产生的影响均与一次执行的影响相同。幂等函数，或幂等方法，是指可以使用相同参数重复执行，并能获得相同结果的函数。这些函数不会影响系统状态，也不用担心重复执行会对系统造成改变。例如，“getUsername()和setTrue()”函数就是一个幂等函数.

## Cookies和session区别

1. Cookies是一种能够让网站服务器把少量数据储存到客户端的硬盘或内存，或是从客户端的硬盘读取数据的一种技术。Cookies是当你浏览某网站时，由Web服务器置于你硬盘上的一个非常小的文本文件，它可以记录你的用户ID、密码、浏览过的网页、停留的时间等信息。session: 当用户请求来自应用程序的 Web 页时，如果该用户还没有会话，则 Web 服务器将自动创建一个 Session 对象。当会话过期或被放弃后，服务器将终止该会话。cookie机制：采用的是在客户端保持状态的方案，而session机制采用的是在服务端保持状态的方案。同时我们看到由于服务器端保持状态的方案在客户端也需要保存一个标识，所以session机制可能需要借助cookie机制来达到保存标识的目的。
2. Session是服务器用来跟踪用户的一种手段，每个Session都有一个唯一标识：session ID。当服务器创建了Session时，给客户端发送的响应报文包含了Set-cookie字段，其中有一个名为sid的键值对，这个键值Session ID。客户端收到后就把Cookie保存浏览器，并且之后发送的请求报表都包含SessionID。HTTP就是通过Session和Cookie这两个发送一起合作来实现跟踪用户状态，Session用于服务端，Cookie用于客户端

## TCP粘包和拆包产生的原因

1. 应用程序写入数据的字节大小大于套接字发送缓冲区的大小
2. 进行MSS大小的TCP分段。MSS是最大报文段长度的缩写。MSS是TCP报文段中的数据字段的最大长度。数据字段加上TCP首部才等于整个的TCP报文段。所以MSS并不是TCP报文段的最大长度，而是：MSS=TCP报文段长度-TCP首部长度
3. 以太网的payload大于MTU进行IP分片。MTU指：一种通信协议的某一层上面所能通过的最大数据包大小。如果IP层有一个数据包要传，而且数据的长度比链路层的MTU大，那么IP层就会进行分片，把数据包分成托干片，让每一片都不超过MTU。注意，IP分片可以发生在原始发送端主机上，也可以发生在中间路由器上。

## TCP粘包和拆包的解决策略

1. 消息定长。例如100字节。
2. 在包尾部增加回车或者空格符等特殊字符进行分割，典型的如FTP协议
3. 将消息分为消息头和消息尾。
4. 其它复杂的协议，如RTMP协议等。

## 三次握手

第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据

## ****四次挥手****

1. 客户端先发送FIN，进入FIN\_WAIT1状态
2. 服务端收到FIN，发送ACK，进入CLOSE\_WAIT状态，客户端收到这个ACK，进入FIN\_WAIT2状态
3. 服务端发送FIN，进入LAST\_ACK状态
4. 客户端收到FIN，发送ACK，进入TIME\_WAIT状态，服务端收到ACK，进入CLOSE状态

TIME\_WAIT的状态就是主动断开的一方（这里是客户端），发送完最后一次ACK之后进入的状态。并且持续时间还挺长的。客户端TIME\_WAIT持续2倍MSL时长，在linux体系中大概是60s，转换成CLOSE状态

### TIME\_WAIT

TIME\_WAIT 是主动关闭链接时形成的，等待2MSL时间，约4分钟。主要是防止最后一个ACK丢失。 由于TIME\_WAIT 的时间会非常长，因此server端应尽量减少主动关闭连接

### CLOSE\_WAIT

CLOSE\_WAIT是被动关闭连接是形成的。根据TCP状态机，服务器端收到客户端发送的FIN，则按照TCP实现发送ACK，因此进入CLOSE\_WAIT状态。但如果服务器端不执行close()，就不能由CLOSE\_WAIT迁移到LAST\_ACK，则系统中会存在很多CLOSE\_WAIT状态的连接。此时，可能是系统忙于处理读、写操作，而未将已收到FIN的连接，进行close。此时，recv/read已收到FIN的连接socket，会返回0。

### 为什么需要 TIME\_WAIT 状态？

假设最终的ACK丢失，server将重发FIN，client必须维护TCP状态信息以便可以重发最终的ACK，否则会发送RST，结果server认为发生错误。TCP实现必须可靠地终止连接的两个方向(全双工关闭)，client必须进入 TIME\_WAIT 状态，因为client可能面 临重发最终ACK的情形。

### 为什么 TIME\_WAIT 状态需要保持 2MSL 这么长的时间？

如果 TIME\_WAIT 状态保持时间不足够长(比如小于2MSL)，第一个连接就正常终止了。第二个拥有相同相关五元组的连接出现，而第一个连接的重复报文到达，干扰了第二个连接。TCP实现必须防止某个连接的重复报文在连接终止后出现，所以让TIME\_WAIT状态保持时间足够长(2MSL)，连接相应方向上的TCP报文要么完全响应完毕，要么被 丢弃。建立第二个连接的时候，不会混淆。

### TIME\_WAIT 和CLOSE\_WAIT状态socket过多

如果服务器出了异常，百分之八九十都是下面两种情况：

1.服务器保持了大量TIME\_WAIT状态

2.服务器保持了大量CLOSE\_WAIT状态，简单来说CLOSE\_WAIT数目过大是由于被动关闭连接处理不当导致的。

## 一次完整的HTTP请求过程

域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

## 讲一下长连接

一、基于http协议的长连接

在HTTP1.0和HTTP1.1协议中都有对长连接的支持。其中HTTP1.0需要在request中增加”Connection： keep-alive“ header才能够支持，而HTTP1.1默认支持.

**http1.0请求与服务端的交互过程:**

1. 客户端发出带有包含一个header：”Connection： keep-alive“的请求
2. 服务端接收到这个请求后,根据http1.0和”Connection： keep-alive“判断出这是一个长连接,就会在response的header中也增加”Connection： keep-alive“,同是不会关闭已建立的tcp连接.
3. 客户端收到服务端的response后,发现其中包含”Connection： keep-alive“，就认为是一个长连接，不关闭这个连接。并用该连接再发送request.转到a)，点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486710&idx=2&sn=2ebb8532a404f8d7bc295641f253f1d7&chksm=eb5389c0dc2400d68a8adbedb86e30c487261dd4346a7a3c36c15cb75846c3187c42688aff50&scene=21#wechat_redirect)了解 http 1.0 vs 2.0 区别。

二、发心跳包。每隔几秒就发一个数据包过去

## TCP如何保证可靠传输？

1. 三次握手。
2. 将数据截断为合理的长度。应用数据被分割成 TCP 认为最适合发送的数据块（按字节编号，合理分片）
3. 超时重发。当 TCP 发出一个段后，它启动一个定时器，如果不能及时收到一个确认就重发
4. 对于收到的请求，给出确认响应
5. 校验出包有错，丢弃报文段，不给出响应
6. 对失序数据进行重新排序，然后才交给应用层
7. 对于重复数据 ， 能够丢弃重复数据
8. 流量控制。TCP 连接的每一方都有固定大小的缓冲空间。TCP 的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。
9. 拥塞控制。当网络拥塞时，减少数据的发送。

## 详细介绍http

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486710&idx=2&sn=2ebb8532a404f8d7bc295641f253f1d7&chksm=eb5389c0dc2400d68a8adbedb86e30c487261dd4346a7a3c36c15cb75846c3187c42688aff50&scene=21#wechat_redirect)了解 http 1.0 vs 2.0 区别。

**特点**

1. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。
2. 灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。
3. 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
4. 无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。
5. 支持B/S及C/S模式。

**请求消息Request**

1. 请求行，用来说明请求类型,要访问的资源以及所使用的HTTP版本.
2. 请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息从第二行起为请求头部，HOST将指出请求的目的地.User-Agent,服务器端和客户端脚本都能访问它,它是浏览器类型检测逻辑的重要基础.该信息由你的浏览器来定义,并且在每个请求中自动发送等等
3. 空行，请求头部后面的空行是必须的
4. 请求数据也叫主体，可以添加任意的其他数据。

**响应消息Response**

1. 状态行，由HTTP协议版本号， 状态码， 状态消息 三部分组成。
2. 消息报头，用来说明客户端要使用的一些附加信息
3. 空行，消息报头后面的空行是必须的
4. 响应正文，服务器返回给客户端的文本信息。

**状态码**

* 200 OK //客户端请求成功
* 301 Moved Permanently //永久重定向,使用域名跳转
* 302 Found // 临时重定向,未登陆的用户访问用户中心重定向到登录页面
* 400 Bad Request //客户端请求有语法错误，不能被服务器所理解
* 401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用
* 403 Forbidden //服务器收到请求，但是拒绝提供服务
* 404 Not Found //请求资源不存在，eg：输入了错误的URL
* 500 Internal Server Error //服务器发生不可预期的错误
* 503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

**http的方法**

1. get:客户端向服务端发起请求，获得资源。请求获得URL处所在的资源。
2. post:向服务端提交新的请求字段。请求URL的资源后添加新的数据。
3. head:请求获取URL资源的响应报告，即获得URL资源的头部
4. patch：请求局部修改URL所在资源的数据项
5. put：请求修改URL所在资源的数据元素。
6. delete：请求删除url资源的数据

## URI和URL的区别

URI，是uniform resource identifier，统一资源标识符，用来唯一的标识一个资源。Web上可用的每种资源如HTML文档、图像、视频片段、程序等都是一个来URI来定位的

URI一般由三部组成：

1. 访问资源的命名机制
2. 存放资源的主机名
3. 资源自身的名称，由路径表示，着重强调于资源。

URL是uniform resource locator，统一资源定位器，它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何locate这个资源。URL是Internet上用来描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上，特别是著名的Mosaic。采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。

URL一般由三部组成：

1. 协议(或称为服务方式)
2. 存有该资源的主机IP地址(有时也包括端口号)
3. 主机资源的具体地址。如目录和文件名等

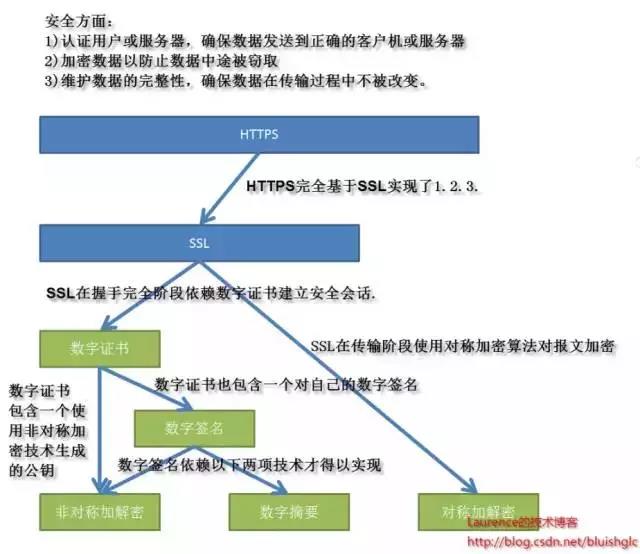
## HTTPS和HTTP的区别

1. https协议需要到CA申请证书，一般免费证书很少，需要交费。
2. http是超文本传输协议，信息是明文传输；https 则是具有安全性的ssl加密传输协 议。
3. http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
4. http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。
5. http默认使用80端口，https默认使用443端口

## https是如何保证数据传输的安全

https实际就是在TCP层与http层之间加入了SSL/TLS来为上层的安全保驾护航，主要用到对称加密、非对称加密、证书，等技术进行客户端与服务器的数据加密传输，最终达到保证整个通信的安全性。点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486325&idx=1&sn=fe28eb5cd4bb79b0d97df33176f1d0b2&chksm=eb538e43dc2407559e97a425438b2c779ef68c65d55984949f08f465f6608ac9289286926921&scene=21#wechat_redirect)弄懂 https 的 9 个问题。

**SSL/TLS协议作用：**

1. 认证用户和服务器，确保数据发送到正确的客户机和服务器；
2. 加密数据以防止数据中途被窃取；
3. 维护数据的完整性，确保数据在传输过程中不被改变。

**10大算法问题**

**字符串**

如果IDE没有代码自动补全功能，所以你应该记住下面的这些方法。

toCharArray() // 获得字符串对应的char数组

Arrays.sort() // 数组排序

Arrays.toString(char[] a) // 数组转成字符串

charAt(int x) // 获得某个索引处的字符

length() // 字符串长度

length // 数组大小

**2链表**

在Java中，链表的实现非常简单，每个节点Node都有一个值val和指向下个节点的链接next。

class Node {

int val;

Node next;

Node(int x) {

val = x;

next = null;

}

}

链表两个著名的应用是栈Stack和队列Queue。

**栈：**

class Stack{

Node top;

public Node peek(){

if(top != null){

return top;

}

return null;

}

public Node pop(){

if(top == null){

return null;

}else{

Node temp = new Node(top.val);

top = top.next;

return temp;

}

}

public void push(Node n){

if(n != null){

n.next = top;

top = n;

}

}

}

**队列：**

class Queue{

Node first, last;

public void enqueue(Node n){

if(first == null){

first = n;

last = first;

}else{

last.next = n;

last = n;

}

}

public Node dequeue(){

if(first == null){

return null;

}else{

Node temp = new Node(first.val);

first = first.next;

return temp;

}

}

}

**树**

这里的树通常是指二叉树，每个节点都包含一个左孩子节点和右孩子节点，像下面这样：

class TreeNode{

int value;

TreeNode left;

TreeNode right;

}

下面是与树相关的一些概念：

1）平衡 vs. 非平衡：平衡二叉树中，每个节点的左右子树的深度相差至多为1（1或0）。

2）满二叉树（Full Binary Tree）：除叶子节点以为的每个节点都有两个孩子。

3）完美二叉树（Perfect Binary Tree）：是具有下列性质的满二叉树：所有的叶子节点都有相同的深度或处在同一层次，且每个父节点都必须有两个孩子。

4）完全二叉树（Complete Binary Tree）：二叉树中，可能除了最后一个，每一层都被完全填满，且所有节点都必须尽可能想左靠。

译者注：完美二叉树也隐约称为完全二叉树。完美二叉树的一个例子是一个人在给定深度的祖先图，因为每个人都一定有两个生父母。完全二叉树可以看成是可以有若干额外向左靠的叶子节点的完美二叉树。疑问：完美二叉树和满二叉树的区别？（参考：http://xlinux.nist.gov/dads/HTML/perfectBinaryTree.html）

**4图**

图相关的问题主要集中在深度优先搜索（depth first search）和广度优先搜索（breath first search）。

下面是一个简单的图广度优先搜索的实现。

**1) 定义GraphNode**

class GraphNode{

int val;

GraphNode next;

GraphNode[] neighbors;

boolean visited;

GraphNode(int x) {

val = x;

}

GraphNode(int x, GraphNode[] n){

val = x;

neighbors = n;

}

public String toString(){

return "value: "+ this.val;

}

}

**2) 定义一个队列Queue**

class Queue{

GraphNode first, last;

public void enqueue(GraphNode n){

if(first == null){

first = n;

last = first;

}else{

last.next = n;

last = n;

}

}

public GraphNode dequeue(){

if(first == null){

return null;

}else{

GraphNode temp = new GraphNode(first.val, first.neighbors);

first = first.next;

return temp;

}

}

}

**3) 用队列Queue实现广度优先搜索**

public class GraphTest {

public static void main(String[] args) {

GraphNode n1 = new GraphNode(1);

GraphNode n2 = new GraphNode(2);

GraphNode n3 = new GraphNode(3);

GraphNode n4 = new GraphNode(4);

GraphNode n5 = new GraphNode(5);

n1.neighbors = new GraphNode[]{n2,n3,n5};

n2.neighbors = new GraphNode[]{n1,n4};

n3.neighbors = new GraphNode[]{n1,n4,n5};

n4.neighbors = new GraphNode[]{n2,n3,n5};

n5.neighbors = new GraphNode[]{n1,n3,n4};

breathFirstSearch(n1, 5);

}

public static void breathFirstSearch(GraphNode root, int x){

if(root.val == x)

System.out.println("find in root");

Queue queue = new Queue();

root.visited = true;

queue.enqueue(root);

while(queue.first != null){

GraphNode c = (GraphNode) queue.dequeue();

for(GraphNode n: c.neighbors){

if(!n.visited){

System.out.print(n + " ");

n.visited = true;

if(n.val == x)

System.out.println("Find "+n);

queue.enqueue(n);

}

}

}

}

}

Output:

|  |  |
| --- | --- |
| 1  2 | value: 2 value: 3 value: 5 Find value: 5  value: 4 |

5排序

下面是不同排序算法的时间复杂度，你可以去wiki看一下这些算法的基本思想。

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Average Time | Worst Time | Space |
| 冒泡排序 | n^2 | n^2 | 1 |
| 选择排序 | n^2 | n^2 | 1 |
| Counting Sort | n+k | n+k | n+k |
| Insertion sort | n^2 | n^2 |  |
| Quick sort | n log(n) | n^2 |  |
| Merge sort | n log(n) | n log(n) | depends |

另外，这里有一些实现/演示：: Counting sort、Mergesort、 Quicksort、 InsertionSort。

6**递归VS迭代**

对程序员来说，递归应该是一个与生俱来的思想（a built-in thought），可以通过一个简单的例子来说明。

问题： 有n步台阶，一次只能上1步或2步，共有多少种走法。

步骤1:找到走完前n步台阶和前n-1步台阶之间的关系。

为了走完n步台阶，只有两种方法：从n-1步台阶爬1步走到或从n-2步台阶处爬2步走到。如果f(n)是爬到第n步台阶的方法数，那么f(n) = f(n-1) + f(n-2)。

步骤2: 确保开始条件是正确的。

f(0) = 0;  
f(1) = 1;

public static int f(int n){

if(n <= 2) return n;

int x = f(n-1) + f(n-2);

return x;

}

递归方法的时间复杂度是n的指数级，因为有很多冗余的计算，如下：

f(5)  
f(4) + f(3)  
f(3) + f(2) + f(2) + f(1)  
f(2) + f(1) + f(1) + f(0) + f(1) + f(0) + f(1)  
f(1) + f(0) + f(1) + f(1) + f(0) + f(1) + f(0) + f(1)

直接的想法是将递归转换为迭代：

public static int f(int n) {

if (n <= 2){

return n;

}

int first = 1, second = 2;

int third = 0;

for (int i = 3; i <= n; i++) {

third = first + second;

first = second;

second = third;

}

return third;

}

对这个例子而言，迭代花费的时间更少，你可能也想看看Recursion vs Iteration。

7**动态规划**

动态规划是解决下面这些性质类问题的技术：

1）一个问题可以通过更小子问题的解决方法来解决（译者注：即问题的最优解包含了其子问题的最优解，也就是最优子结构性质）。

2）有些子问题的解可能需要计算多次（译者注：也就是子问题重叠性质）。

3）子问题的解存储在一张表格里，这样每个子问题只用计算一次。

4）需要额外的空间以节省时间。

爬台阶问题完全符合上面的四条性质，因此可以用动态规划法来解决。

public static int[] A = new int[100];

public static int f3(int n) {

if (n <= 2)

A[n]= n;

if(A[n] > 0)

return A[n];

else

A[n] = f3(n-1) + f3(n-2);//store results so only calculate once!

return A[n];

}

**位操作**

位操作符：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OR (|) | AND (&) | XOR (^) | Left Shift (<<) | Right Shift (>>) | Not (~) |
| 1|0=1 | 1&0=0 | 1^0=1 | 0010<<2=1000 | 1100>>2=0011 | ~1=0 |

获得给定数字n的第i位：(i从0计数并从右边开始)

public static boolean getBit(int num, int i){

int result = num & (1<<i);

if(result == 0){

return false;

}else{

return true;

}

例如，获得数字10的第2位：

i=1, n=10  
1<<1= 10  
1010&10=10  
10 is not 0, so return true;

**9概率问题**

**解决概率相关的问题通常**需要很好的规划了解问题（formatting the problem），这里刚好有一个这类问题的简单例子：

一个房间里有50个人，那么至少有两个人生日相同的概率是多少？（忽略闰年的事实，也就是一年365天）

计算某些事情的概率很多时候都可以转换成先计算其相对面。在这个例子里，我们可以计算所有人生日都互不相同的概率，也就是：365/365 \* 364/365 \* 363/365 \* … \* (365-49)/365，这样至少两个人生日相同的概率就是1 – 这个值。

public static double caculateProbability(int n){

double x = 1;

for(int i=0; i<n; i++){

x \*= (365.0-i)/365.0;

}

double pro = Math.round((1-x) \* 100);

return pro/100;

calculateProbability(50) = 0.97

**排列组合**

组合和排列的区别在于次序是否关键。

**JAVA基础**

1. JAVA中的几种基本数据类型是什么，各自占用多少字节。
2. String类能被继承吗，为什么。
3. String，Stringbuffer，StringBuilder的区别。
4. ArrayList和LinkedList有什么区别。
5. 讲讲类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数，字  
   段，当new的时候，他们的执行顺序。
6. 用过哪些Map类，都有什么区别，HashMap是线程安全的吗,并发下使用的Map是什么，他们  
   内部原理分别是什么，比如存储方式，hashcode，扩容，默认容量等。
7. JAVA8的ConcurrentHashMap为什么放弃了分段锁，有什么问题吗，如果你来设计，你如何  
   设计。
8. 有没有有顺序的Map实现类，如果有，他们是怎么保证有序的。
9. 抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口么,类可以实现多个接口  
   么。
10. 继承和聚合的区别在哪。
11. IO模型有哪些，讲讲你理解的nio ，他和bio，aio的区别是啥，谈谈reactor模型。
12. 反射的原理，反射创建类实例的三种方式是什么。
13. 反射中，Class.forName和ClassLoader区别 。
14. 描述动态代理的几种实现方式，分别说出相应的优缺点。
15. 动态代理与cglib实现的区别。
16. 为什么CGlib方式可以对接口实现代理。
17. final的用途。
18. 写出三种单例模式实现 。
19. 如何在父类中为子类自动完成所有的hashcode和equals实现？这么做有何优劣。
20. 请结合OO设计理念，谈谈访问修饰符public、private、protected、default在应用设  
    计中的作用。
21. 深拷贝和浅拷贝区别。
22. 数组和链表数据结构描述，各自的时间复杂度。
23. error和exception的区别，CheckedException，RuntimeException的区别。
24. 请列出5个运行时异常。
25. 在自己的代码中，如果创建一个java.lang.String类，这个类是否可以被类加载器加  
    载？为什么。
26. 说一说你对java.lang.Object对象中hashCode和equals方法的理解。在什么场景下需  
    要重新实现这两个方法。
27. 在jdk1.5中，引入了泛型，泛型的存在是用来解决什么问题。
28. 这样的a.hashcode() 有什么用，与a.equals(b)有什么关系。
29. 有没有可能2个不相等的对象有相同的hashcode。
30. Java中的HashSet内部是如何工作的。
31. 什么是序列化，怎么序列化，为什么序列化，反序列化会遇到什么问题，如何解决。
32. java8的新特性。

**JVM知识**

1. 什么情况下会发生栈内存溢出。
2. JVM的内存结构，Eden和Survivor比例。
3. JVM内存为什么要分成新生代，老年代，持久代。新生代中为什么要分为Eden和Survivor。
4. JVM中一次完整的GC流程是怎样的，对象如何晋升到老年代，说说你知道的几种主要的JVM参  
   数。
5. 你知道哪几种垃圾收集器，各自的优缺点，重点讲下cms和G1，包括原理，流程，优缺点。
6. 垃圾回收算法的实现原理。
7. 当出现了内存溢出，你怎么排错。
8. JVM内存模型的相关知识了解多少，比如重排序，内存屏障，happen-before，主内存，工作  
   内存等。
9. 简单说说你了解的类加载器，可以打破双亲委派么，怎么打破。
10. 讲讲JAVA的反射机制。
11. 你们线上应用的JVM参数有哪些。
12. g1和cms区别,吞吐量优先和响应优先的垃圾收集器选择。
13. 怎么打出线程栈信息。
14. 请解释如下jvm参数的含义：  
    -server -Xms512m -Xmx512m -Xss1024K  
    -XX:PermSize=256m -XX:MaxPermSize=512m -  
    XX:MaxTenuringThreshold=20XX:CMSInitiatingOccupancyFraction=80 -  
    XX:+UseCMSInitiatingOccupancyOnly。

**开源框架知识**

1. 简单讲讲tomcat结构，以及其类加载器流程，线程模型等。
2. tomcat如何调优，涉及哪些参数 。
3. 讲讲Spring加载流程。
4. Spring AOP的实现原理。
5. 讲讲Spring事务的传播属性。
6. Spring如何管理事务的。
7. Spring怎么配置事务（具体说出一些关键的xml 元素）。
8. 说说你对Spring的理解，非单例注入的原理？它的生命周期？循环注入的原理，aop的实现原  
   理，说说aop中的几个术语，它们是怎么相互工作的。
9. Springmvc 中DispatcherServlet初始化过程。
10. netty的线程模型，netty如何基于reactor模型上实现的。
11. 为什么选择netty。
12. 什么是TCP粘包，拆包。解决方式是什么。
13. netty的fashwheeltimer的用法，实现原理，是否出现过调用不够准时，怎么解决。
14. netty的心跳处理在弱网下怎么办。
15. netty的通讯协议是什么样的。
16. springmvc用到的注解，作用是什么，原理。
17. springboot启动机制。
18. 点击[这里](https://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486678&idx=1&sn=2a5e38e67c3d267d6c58d963adb24ccc&scene=21#wechat_redirect)有一套答案版的Spring试题。

**操作系统**

1. Linux系统下你关注过哪些内核参数，说说你知道的。
2. Linux下IO模型有几种，各自的含义是什么。
3. epoll和poll有什么区别。
4. 平时用到哪些Linux命令。
5. 用一行命令查看文件的最后五行。
6. 用一行命令输出正在运行的java进程。
7. 介绍下你理解的操作系统中线程切换过程。
8. 进程和线程的区别。
9. top 命令之后有哪些内容，有什么作用。
10. 线上CPU爆高，请问你如何找到问题所在。

**多线程**

1. 多线程的几种实现方式，什么是线程安全。
2. volatile的原理，作用，能代替锁么。
3. 画一个线程的生命周期状态图。
4. sleep和wait的区别。
5. sleep和sleep(0)的区别。
6. Lock与Synchronized的区别 。
7. synchronized的原理是什么，一般用在什么地方(比如加在静态方法和非静态方法的区别，静  
   态方法和非静态方法同时执行的时候会有影响吗)，解释以下名词：重排序，自旋锁，偏向锁，轻  
   量级锁，可重入锁，公平锁，非公平锁，乐观锁，悲观锁。
8. 用过哪些原子类，他们的原理是什么。
9. JUC下研究过哪些并发工具，讲讲原理。
10. 用过线程池吗，如果用过，请说明原理，并说说newCache和newFixed有什么区别，构造函  
    数的各个参数的含义是什么，比如coreSize，maxsize等。
11. 线程池的关闭方式有几种，各自的区别是什么。
12. 假如有一个第三方接口，有很多个线程去调用获取数据，现在规定每秒钟最多有10个线程同  
    时调用它，如何做到。
13. spring的controller是单例还是多例，怎么保证并发的安全。
14. 用三个线程按顺序循环打印abc三个字母，比如abcabcabc。
15. ThreadLocal用过么，用途是什么，原理是什么，用的时候要注意什么。
16. 如果让你实现一个并发安全的链表，你会怎么做。
17. 有哪些无锁数据结构，他们实现的原理是什么。
18. 讲讲java同步机制的wait和notify。
19. CAS机制是什么，如何解决ABA问题。
20. 多线程如果线程挂住了怎么办。
21. countdowlatch和cyclicbarrier的内部原理和用法，以及相互之间的差别(比如  
    countdownlatch的await方法和是怎么实现的)。
22. 对AbstractQueuedSynchronizer了解多少，讲讲加锁和解锁的流程，独占锁和公平所  
    加锁有什么不同。
23. 使用synchronized修饰静态方法和非静态方法有什么区别。
24. 简述ConcurrentLinkedQueue和LinkedBlockingQueue的用处和不同之处。
25. 导致线程死锁的原因？怎么解除线程死锁。
26. 非常多个线程（可能是不同机器），相互之间需要等待协调，才能完成某种工作，问怎么设计这种协调方案。
27. 用过读写锁吗，原理是什么，一般在什么场景下用。
28. 开启多个线程，如果保证顺序执行，有哪几种实现方式，或者如何保证多个线程都执行完  
    再拿到结果。
29. 延迟队列的实现方式，delayQueue和时间轮算法的异同。
30. 点击[这里](https://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486721&idx=2&sn=c2058d5ddd7453eb9c39732c114879d5&scene=21#wechat_redirect)有一套答案版的多线程试题。

**TCP与HTTP**

1. http1.0和http1.1有什么区别。
2. TCP三次握手和四次挥手的流程，为什么断开连接要4次,如果握手只有两次，会出现什么。
3. TIME\_WAIT和CLOSE\_WAIT的区别。
4. 说说你知道的几种HTTP响应码，比如200, 302, 404。
5. 当你用浏览器打开一个链接（如：http://www.javastack.cn）的时候，计算机做了哪些工作步骤。
6. TCP/IP如何保证可靠性，说说TCP头的结构。
7. 如何避免浏览器缓存。
8. 如何理解HTTP协议的无状态性。
9. 简述Http请求get和post的区别以及数据包格式。
10. HTTP有哪些method
11. 简述HTTP请求的报文格式。
12. HTTP的长连接是什么意思。
13. HTTPS的加密方式是什么，讲讲整个加密解密流程。
14. Http和https的三次握手有什么区别。
15. 什么是分块传送。
16. Session和cookie的区别。
17. 点击[这里](https://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486746&idx=2&sn=7e27c0c278e7ffa2af4b18b1d928db83&scene=21#wechat_redirect)有一套答案版的试题。

**架构设计与分布式**

**用java自己实现一个LRU。**

1. 分布式集群下如何做到唯一序列号。
2. 设计一个秒杀系统，30分钟没付款就自动关闭交易。
3. 如何使用redis和zookeeper实现分布式锁？有什么区别优缺点，会有什么问题，分别适用什么  
   场景。（延伸：如果知道redlock，讲讲他的算法实现，争议在哪里）
4. 如果有人恶意创建非法连接，怎么解决。
5. 分布式事务的原理，优缺点，如何使用分布式事务，2pc 3pc 的区别，解决了哪些问题，还有  
   哪些问题没解决，如何解决，你自己项目里涉及到分布式事务是怎么处理的。
6. 什么是一致性hash。
7. 什么是restful，讲讲你理解的restful。
8. 如何设计一个良好的API。
9. 如何设计建立和保持100w的长连接。
10. 解释什么是MESI协议(缓存一致性)。
11. 说说你知道的几种HASH算法，简单的也可以。
12. 什么是paxos算法， 什么是zab协议。
13. 一个在线文档系统，文档可以被编辑，如何防止多人同时对同  
    一份文档进行编辑更新。
14. 线上系统突然变得异常缓慢，你如何查找问题。
15. 说说你平时用到的设计模式。
16. Dubbo的原理，有看过源码么，数据怎么流转的，怎么实现集群，负载均衡，服务注册  
    和发现，重试转发，快速失败的策略是怎样的 。
17. 一次RPC请求的流程是什么。
18. 自己实现过rpc么，原理可以简单讲讲。Rpc要解决什么问题。
19. 异步模式的用途和意义。
20. 编程中自己都怎么考虑一些设计原则的，比如开闭原则，以及在工作中的应用。
21. 设计一个社交网站中的“私信”功能，要求高并发、可扩展等等。 画一下架构图。
22. MVC模式，即常见的MVC框架。
23. 聊下曾经参与设计的服务器架构并画图，谈谈遇到的问题，怎么解决的。
24. 应用服务器怎么监控性能，各种方式的区别。
25. 如何设计一套高并发支付方案，架构如何设计。
26. 如何实现负载均衡，有哪些算法可以实现。
27. Zookeeper的用途，选举的原理是什么。
28. Zookeeper watch机制原理。
29. Mybatis的底层实现原理。
30. 请思考一个方案，实现分布式环境下的countDownLatch。
31. 后台系统怎么防止请求重复提交。
32. 描述一个服务从发布到被消费的详细过程。
33. 讲讲你理解的服务治理。
34. 如何做到接口的幂等性。
35. 如何做限流策略，令牌桶和漏斗算法的使用场景。
36. 什么叫数据一致性，你怎么理解数据一致性。
37. 分布式服务调用方，不依赖服务提供方的话，怎么处理服务方挂掉后，大量无效资源请求  
    的浪费，如果只是服务提供方吞吐不高的时候该怎么做，如果服务挂了，那么一会重启，该怎  
    么做到最小的资源浪费，流量半开的实现机制是什么。
38. dubbo的泛化调用怎么实现的，如果是你，你会怎么做。
39. 远程调用会有超时现象，如果做到优雅的控制，JDK自带的超时机制有哪些，怎么实现的。

**算法**

1. 10亿个数字里里面找最小的10个。
2. 有1亿个数字，其中有2个是重复的，快速找到它，时间和空间要最优。
3. 2亿个随机生成的无序整数,找出中间大小的值。
4. 给一个不知道长度的（可能很大）输入字符串，设计一种方案，将重复的字符排重。
5. 遍历二叉树。
6. 有3n+1个数字，其中3n个中是重复的，只有1个是不重复的，怎么找出来。
7. 写一个字符串（如：www.javastack.cn）反转函数。
8. 常用的排序算法，快排，归并、冒泡。 快排的最优时间复杂度，最差复杂度。冒泡排序的  
   优化方案。
9. 二分查找的时间复杂度，优势。
10. 一个已经构建好的TreeSet，怎么完成倒排序。
11. 什么是B+树，B-树，列出实际的使用场景。
12. 一个单向链表，删除倒数第N个数据。
13. 200个有序的数组，每个数组里面100个元素，找出top20的元素。
14. 单向链表，查找中间的那个元素。

**数据库知识**

1. 数据库隔离级别有哪些，各自的含义是什么，MYSQL默认的隔离级别是是什么。
2. 什么是幻读。
3. MYSQL有哪些存储引擎，各自优缺点。
4. 高并发下，如何做到安全的修改同一行数据。
5. 乐观锁和悲观锁是什么，INNODB的标准行级锁有哪2种，解释其含义。
6. SQL优化的一般步骤是什么，怎么看执行计划，如何理解其中各个字段的含义。
7. 数据库会死锁吗，举一个死锁的例子，mysql怎么解决死锁。
8. MYsql的索引原理，索引的类型有哪些，如何创建合理的索引，索引如何优化。
9. 聚集索引和非聚集索引的区别。
10. select for update 是什么含义，会锁表还是锁行或是其他。
11. 为什么要用Btree实现，它是怎么分裂的，什么时候分裂，为什么是平衡的。
12. 数据库的ACID是什么。
13. 某个表有近千万数据，CRUD比较慢，如何优化。
14. Mysql怎么优化table scan的。
15. 如何写sql能够有效的使用到复合索引。
16. mysql中in 和exists 区别。
17. 数据库自增主键可能的问题。
18. MVCC的含义，如何实现的。
19. 你做过的项目里遇到分库分表了吗，怎么做的，有用到中间件么，比如sharding jdbc等,他  
    们的原理知道么。
20. MYSQL的主从延迟怎么解决。

**消息队列**

1. 消息队列的使用场景。
2. 消息的重发，补充策略。
3. 如何保证消息的有序性。
4. 用过哪些MQ，和其他mq比较有什么优缺点，MQ的连接是线程安全的吗，你们公司的MQ服务  
   架构怎样的。
5. MQ系统的数据如何保证不丢失。
6. rabbitmq如何实现集群高可用。
7. kafka吞吐量高的原因。
8. kafka 和其他消息队列的区别，kafka 主从同步怎么实现。
9. 利用mq怎么实现最终一致性。
10. 使用kafka有没有遇到什么问题，怎么解决的。
11. MQ有可能发生重复消费，如何避免，如何做到幂等。
12. MQ的消息延迟了怎么处理，消息可以设置过期时间么，过期了你们一般怎么处理。

**缓存**

1. 常见的缓存策略有哪些，如何做到缓存(比如redis)与DB里的数据一致性，你们项目中用到了  
   什么缓存系统，如何设计的。
2. 如何防止缓存击穿和雪崩。
3. 缓存数据过期后的更新如何设计。
4. redis的list结构相关的操作。
5. Redis的数据结构都有哪些。
6. Redis的使用要注意什么，讲讲持久化方式，内存设置，集群的应用和优劣势，淘汰策略等。
7. redis2和redis3的区别，redis3内部通讯机制。
8. 当前redis集群有哪些玩法，各自优缺点，场景。
9. Memcache的原理，哪些数据适合放在缓存中。
10. redis和memcached 的内存管理的区别。
11. Redis的并发竞争问题如何解决，了解Redis事务的CAS操作吗。
12. Redis的选举算法和流程是怎样的。
13. redis的持久化的机制，aof和rdb的区别。
14. redis的集群怎么同步的数据的。
15. 知道哪些redis的优化操作。
16. Reids的主从复制机制原理。
17. Redis的线程模型是什么。
18. 请思考一个方案，设计一个可以控制缓存总体大小的自动适应的本地缓存。
19. 如何看待缓存的使用（本地缓存，集中式缓存），简述本地缓存和集中式缓存和优缺点。
20. 本地缓存在并发使用时的注意事项。
21. 点击[这里](http://mp.weixin.qq.com/s?__biz=MzI3ODcxMzQzMw==&mid=2247486734&idx=2&sn=d8454c6cbd09ab60ef5a728a36c19e8c&chksm=eb538838dc24012e15b813df90a115803c243eb6a242c052d684c6c3819b5f3300f3a2d482be&scene=21#wechat_redirect)有一套答案版的Redis试题。

**搜索**

1. elasticsearch了解多少，说说你们公司es的集群架构，索引数据大小，分片有多少，以及一些  
   调优手段 。elasticsearch的倒排索引是什么。
2. elasticsearch 索引数据多了怎么办，如何调优，部署。
3. elasticsearch是如何实现master选举的。
4. 详细描述一下Elasticsearch索引文档的过程。
5. 详细描述一下Elasticsearch搜索的过程。
6. Elasticsearch在部署时，对Linux的设置有哪些优化方法？
7. lucence内部结构是什么。