## 零碎的

HTTPS了解吗？说一下加密过程？

对称加密：简单说就是有一个密钥，它可以加密一段信息，也可以对加密后的信息进行解密，和我们日常生活中用的钥匙作用差不多。

什么是非对称加密：简单说就是有两把密钥，通常一把叫做公钥、一把叫私钥，用公钥加密的内容必须用私钥才能解开，同样，私钥加密的内容只有公钥能解开。

1. 双方都用非对称加密：
   1. 双方各自持有公钥和私钥。传输内容前，将公钥互相明文传输。
   2. 非对称加密算法非常耗时。
2. 用对称加密+非对称加密：
   1. 网站拥有公钥私钥。服务器向浏览器明文传输公钥A，浏览器随机(**用数据哈希**)生成一个用于对称加密的密钥X，用A加密后发给服务器，服务器用密钥A‘解密拿到X。这样双方都拿到了X。用X进行加密解密(对称)
   2. 漏洞：中间人攻击（**根本原因是浏览器无法确认收到的公钥是不是网站自己的，**）
3. 引入了CA数字签名+数字证书验证

HTTPS加密过程：

1. 客户端请求服务器获取证书公钥
2. 客户端(SSL/TLS)解析证书（无效会弹出警告）
3. 生成随机值
4. 用公钥加密随机值生成**密钥**
5. 客户端将秘钥发送给服务器
6. 服务端用私钥解密秘钥得到随机值
7. 将信息和随机值混合在一起进行对称加密
8. 将加密的内容发送给客户端
9. 客户端用秘钥解密信息

<https://zhuanlan.zhihu.com/p/43789231>

什么叫幂等？POST是幂等的吗？

* 一个 HTTP 方法是**幂等**的，指的是同样的请求被执行一次与连续执行多次的效果是一样的，服务器的状态也是一样的。换句话说就是，幂等方法不应该具有副作用（统计用途除外）。在正确实现的条件下， [GET](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/GET) ， [HEAD](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/HEAD) ， [PUT](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/PUT) 和 [DELETE](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/DELETE) 等方法都是**幂等**的，而 [POST](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/POST) 方法不是。所有的 [safe](https://developer.mozilla.org/zh-CN/docs/Glossary/Safe) 方法也都是幂等的。
* Post每次调用都会有新的资源产生，因此不满足幂等性
* 为什么使用幂等？

前端重复提交表单：

用户恶意进行刷单：

接口超时重复提交：

消息进行重复消费：

路由器如何进行路由转换？

路由：就是选择的路线

1. 汽车靠导航到达目的地，IP包，在每一个通往目的地的路由器上，查询下一条的目的地址by路由表。
2. 地图导航是怎么形成的？——根据路由协议(OSPF，IS-IS)，生成路由表。
3. 如果OSPF，IS-IS在同一台路由器上运行，对于**同一个目的地（网络地址相同，网络掩码相同）**它们都生成了路由表，那用谁的呢？——优先级高的进入路由表。
4. 如何路由查找，找到最快到达目的地的下一跳？——最长匹配原则。



1. 链路状态协议的比较：
   1. **OSPF** ，Open Shortest Path First，开放最短路径优先。OSPF其实并不开放，以OSPFv2来说，它只用来交换IPv4的路由信息，那如何交换IPv6路由信息呢？没办法！因为OSPFv2协议字段里没有这样的字段，来分辨交换得是IPv4，IPv6，CLNS，MAC，Label，这是很致命的，协议写的太死板、扩展性不好。
   2. 同为链路状态协议的IS-IS，不仅支持IP路由交换，还支持CLNS交换，目前被数据中心二层互联新技术，OTV，TRILL用来交换MAC地址，你没有看错，是二层的MAC地址，这太神奇了！

<https://zhuanlan.zhihu.com/p/21392419>

<https://zhuanlan.zhihu.com/p/21399671>

为什么三次握手，四次挥手？五次挥手行不行？

进程和线程区别

1. 背景：
   1. CPU速度太快，在CPU看来所有的任务都是一个个轮流执行的。
   2. 除了CPU以外所有的执行环境，主要是寄存器的一些内容，就构成了的进程的上下文环境。
2. 进程就是上下文切换之间的程序执行部分。
3. 线程
   1. 进程颗粒度太大，每一次执行都需要切换上下文。
   2. 线程共享了进程的上下文环境(实质上，线程共享了进程的地址空间。再实质，堆。)
   3. 那么这里具体的执行就可能变成：程序A得到CPU =》CPU加载上下文，开始执行程序A的a小段，然后执行A的b小段，然后再执行A的c小段，最后CPU保存A的上下文。
4. JVM角度
   1. 一个进程中可以有多个线程，多个线程共享进程的**堆**和**方法区 (JDK1.8 之后的元空间)资源，**
   2. **但是每个线程有自己的程序计数器**、**虚拟机栈** 和 **本地方法栈**。

<https://www.zhihu.com/question/25532384/answer/81152571>

<https://javaguide.cn/cs-basics/operating-system/operating-system-basic-questions-01.html#%E8%BF%9B%E7%A8%8B%E5%92%8C%E7%BA%BF%E7%A8%8B%E7%9A%84%E5%8C%BA%E5%88%AB>

如何避免死锁？

1. 死锁是什么？

多个进程/线程同时被阻塞，它们中的一个或者全部都在等待某个资源被释放。由于进程/线程被无限期地阻塞，因此程序不可能正常终止。

1. 死锁的出现必须满足以下四个必要条件：
   1. 互斥：**多个线程不能同时使用同一个资源**。
   2. 占有和等待：线程“贪心”，常常有了一把钥匙还没归还就要下一把。
   3. 不剥夺、非抢占：当线程已经持有了资源，**在自己使用完之前不能被其他线程获取**
   4. 环路等待

要避免死锁，只要破环其中一个条件即可。

1. 解决死锁的方法：
   1. 预防：限制并发进程对资源的请求
   2. 避免：
   3. 检测和解除：系统设有专门的机构
2. 常见的避免死锁的方法：
   1. 破坏互斥：系统拷贝一份钥匙的成本极高。No
   2. 破坏环路等待条件：要求任何线程获取钥匙的顺序必须先取完A再B。ok
   3. 破坏不剥夺条件：设置“最长占用时间”，之后释放。Ok
   4. 破坏占有和等待条件：规定先归还，才能申请新资源。Ok

<https://zhuanlan.zhihu.com/p/26945588>

可重入锁了解吗？

1. 可重入锁也叫递归锁，是在一个线程获取锁后，内部如果还需要获取锁，可以直接获取的锁（前提锁对象得是同一个对象或者class）。
2. 优点：可一定程度避免死锁。
3. Java中，可重入锁有**ReentrantLock**和**synchronized**；非可重入锁有**NonReentrantLock**。

<https://tech.meituan.com/2018/11/15/java-lock.html>

锁：[不可不说的Java“锁”事](https://tech.meituan.com/2018/11/15/java-lock.html)by美团技术团队

读写锁？

[**读写锁**](https://www.zhihu.com/search?q=%E8%AF%BB%E5%86%99%E9%94%81&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A246535792%7D)**（readers-writer lock）**，看英文可以顾名思义，在执行加锁操作时需要额外表明读写意图，复数读者之间并不互斥，而写者则要求与任何人互斥。读写锁不需要特殊支持就可以直接用之前提到的几个东西实现，比如可以直接用两个 spinlock 或者两个 mutex 实现：

如果整个场景中只有一个读者、一个写者，那么其实可以等价于直接使用互斥器。不过由于读写锁需要额外记录读者数量，花销要大一点。

函数调用时堆栈的变化

<https://cloud.tencent.com/developer/article/1920687>

[内存划分为什么要分为堆和栈，当初设计这两个的时候分别是要解决什么问题？](https://www.zhihu.com/question/447017261/answer/1759797973)

MySQL聚集索引和非聚集索引

* 聚集索引一个表只能有一个，而非聚集索引一个表可以存在多个
* 聚集索引存储记录是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储并不连续
* **聚集索引**：物理存储按照索引排序；聚集索引是一种索引组织形式，索引的键值逻辑顺序决定了表数据行的物理存储顺序
* **非聚集索引**：物理存储不按照索引排序；非聚集索引则就是普通索引了，仅仅只是对数据列创建相应的索引，不影响整个表的物理存储顺序.
* 索引是通过B+树的数据结构来描述的，我们可以这么理解聚簇索引：索引的叶节点就是数据节点。而非聚簇索引的叶节点仍然是索引节点，只不过有一个指针指向对应的数据块。

写个SQL吧，找出班级里排名在第3和第6之间的同学，这三名同学按照语文成绩的高低进行降序排列

事务和 MVCC

1. 事务：a原子性c一致性i隔离性d持久性
2. 从某个角度来说，我们可以控制的、或者说需要研究的只有隔离性这一个特性，而要控制隔离性，几乎只有调整隔离级别这一个手段
   1. 问题由重到轻：脏写、脏读、不可重复读、幻读
   2. 隔离级别由低到高：读未提交->读提交->可重复读(mysql默认)->串行化

MVCC:

1. Mutil-Version Concurrency Control，(多版本并发控制),mysql使用

2. MySQL会为每一行真实数据记录添加两三个隐藏的字段：事务id(只有执行insert/update/delete语句才能获得事务id)、回滚指针、(行id if没有主键的话)

<https://www.cnblogs.com/CodeBear/p/12710670.html>

LRU（我说了LinkedHashMap的实现）

Redis为什么快，为什么不需要多线程

1. 为什么快？

1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)；

2、数据结构简单，对数据操作也简单，Redis中的数据结构是专门进行设计的；

3、采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

4、使用多路I/O复用模型，非阻塞IO；

5、使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协议不一样，Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求；

\*多路I/O复用模型

多路I/O复用模型是利用 select、poll、epoll 可以同时监察多个流的 I/O 事件的能力，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有 I/O 事件时，就从阻塞态中唤醒，于是程序就会轮询一遍所有的流（epoll 是只轮询那些真正发出了事件的流），并且只依次顺序的处理就绪的流，这种做法就避免了大量的无用操作。

**这里“多路”指的是多个网络连接，“复用”指的是复用同一个线程。**采用多路 I/O 复用技术可以让单个线程高效的处理多个连接请求（尽量减少网络 IO 的时间消耗），且 Redis 在内存中操作数据的速度非常快，也就是说内存内的操作不会成为影响Redis性能的瓶颈，主要由以上几点造就了 Redis 具有很高的吞吐量。

2. 为什么单线程？

官方FAQ表示，因为Redis是基于内存的操作，CPU不是Redis的瓶颈，Redis的瓶颈最有可能是机器内存的大小或者网络带宽。既然单线程容易实现，而且CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了（毕竟采用多线程会有很多麻烦！）。

AOF是什么意思？

Append Only File。是Redis的一种持久化策略。是日志。

联想：RBD。Redis-database，另一种持久化策略。在conf文件中更改, 触发检查。

终端执行过程时 ctrl + c 发生了什么

<https://www.timiguo.com/archives/209/#:~:text=%E7%BB%88%E7%AB%AF%E9%A9%B1%E5%8A%A8%E7%A8%8B%E5%BA%8F%E5%B0%86Ctrl,SIGINT%E4%BF%A1%E5%8F%B7%E7%BB%99%E8%AF%A5%E8%BF%9B%E7%A8%8B%EF%BC%89%E3%80%82> 还没看

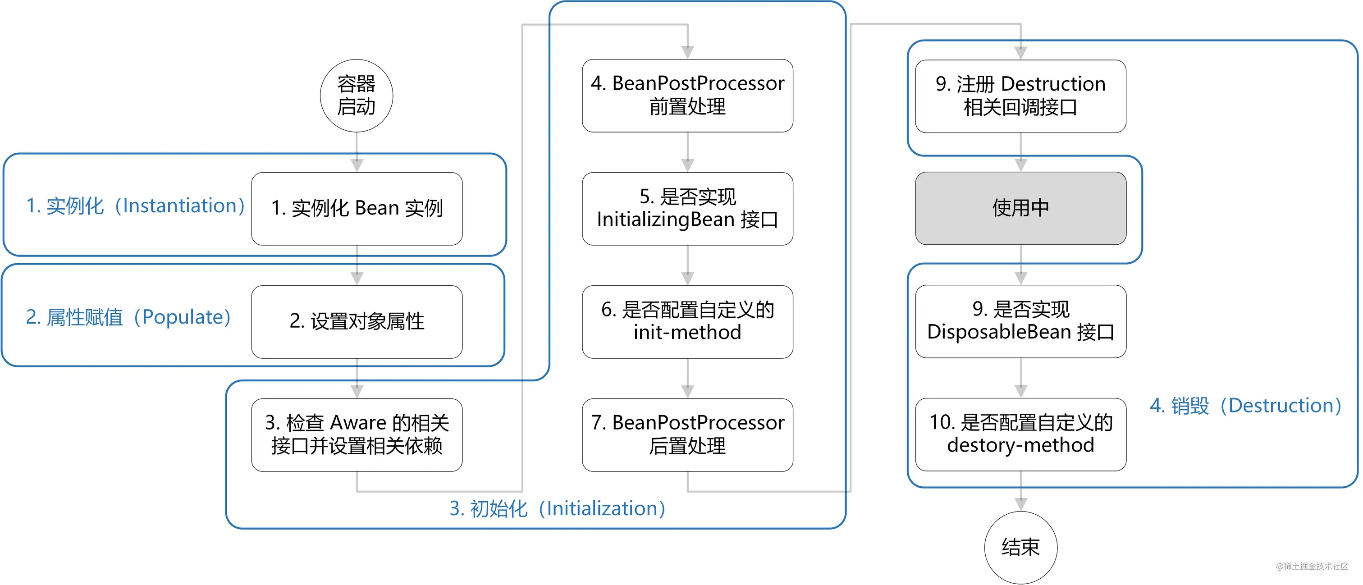
1. 看你以前还去实习过，实习期间做了什么
2. 栈帧有哪几个部分
3. 什么可以作为GC Roots
4. 事务隔离级别
5. 事务特性以及如何保证ACID
6. 为什么用B+树做索引
7. 非聚类索引的叶子结点和根结点分别是什么
8. TCP泛洪攻击听说过吗？
9. 为什么三次握手四次挥手
10. 看你简历上还说自己对JVM有过了解，那JVM有哪几个部分？说你最了解的地方
11. JVM 内存模型，内存回收策略，回收不了的对象
12. 你还研究过红黑树？大致说说？
13. 什么时候要打破双亲委派机制？
14. HashMap的Put过程
15. BeanFactory和FactoryBean有什么区别
16. 数据不一致可能是由于哪些引起的？我只能想到缓存。。。
17. 项目相关，说说分布式？
18. springIOC的设计思想？
19. bean的生命周期你知道吗？

实例化（Instantiation）——JVM类加载

属性赋值（Populate）——属性注入依赖

初始化（Initialization）——相关接口的set方法注入依赖，@postconstruction注解方法执行。。。

销毁（Destruction）



* Spring实例化vs初始化：

实例化：是对象创建的过程。 比如使用构造方法new对象，为对象在内存中分配空间。

初始化：是为对象中的属性赋值的过程。

<https://juejin.cn/post/6844904065457979405>

1. Java线程安全集合有哪些，concurrenthashmap如何做到线程安全
2. hashmap在什么情况下出现线程不安全（回答插入节点时，会短暂的形成环形）
3. hashmap扩容时为什么线程不安全（
4. Stiring是不是线程安全的（我猜的不是）
5. sync的底层原理
6. AQS怎么保证锁的可重入性
7. 内存管理的基本方式有哪些
8. Mysql用的什么引擎，Innodb的索引大概讲一下
9. 接上一问讲到的B+树，为什么使用B+树而不使用B树
10. 链表可以像B+树一样优化查询吗，怎么做，我猜是转化为跳表

写一个死锁

<https://cloud.tencent.com/developer/article/1671071>

联想：synchronised使用？注意：只锁定对象。一个对象一把锁。(除非用static修饰)

<https://blog.csdn.net/luoweifu/article/details/46613015>

输入一个url发生的过程

详细说说dns查询过程

#### Java插桩

讲一下对Spring，SpringMVC,SpringBoot的理解

如何判断对象已经死亡

## 中兴开发一面 Java

自我介绍；

项目介绍，项目为什么没用分布式，用户多大量级；

kafka不同的生产模式及消费模式；

kafka分区，副本；

常用的框架注解；

spring bean相关的注解；

1、Spring的@Bean注解用于告诉方法，产生一个Bean对象，然后这个Bean对象交给Spring管理。 产生这个Bean对象的方法Spring只会调用一次，随后这个Spring将会将这个Bean对象放在自己的IOC容器中。

2、@Component , @Repository , @ Controller , @Service 这些注解只局限于自己编写的类，而@Bean注解能把第三方库中的类实例加入IOC容器中并交给spring管理。

3、@Bean注解的另一个好处就是能够动态获取一个Bean对象，能够根据环境不同得到不同的Bean对象。

@autowired和@Resource区别；

**共同点**

@Resource和@Autowired都可以作为注入属性的修饰，在接口仅有单一实现类时，两个注解的修饰效果相同，可以互相替换，不影响使用。

**不同点**

1、@Resource是JDK原生的注解，@Autowired是Spring2.5 引入的注解

2、@Resource有两个属性name和type。Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。所以如果使用name属性，则使用byName的自动注入策略，而使用type属性时则使用byType自动注入策略。如果既不指定name也不指定type属性，这时将通过反射机制使用byName自动注入策略。

@Autowired只根据type进行注入，不会去匹配name。如果涉及到type无法辨别注入对象时，那需要依赖@Qualifier或@Primary注解一起来修饰。

线程创建方式；

线程池参数；

java集合；

hashmap[数据](file:///C:\jump\super-jump\word%3fword=%25E6%2595%25B0%25E6%258D%25AE)结构；

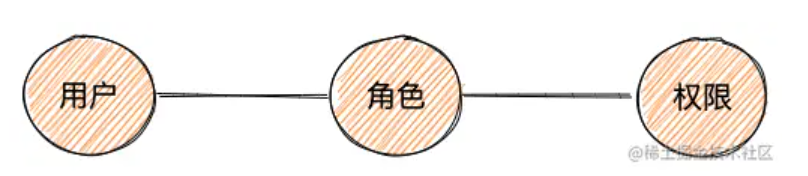
哈希碰撞；

## 云和恩墨 Java

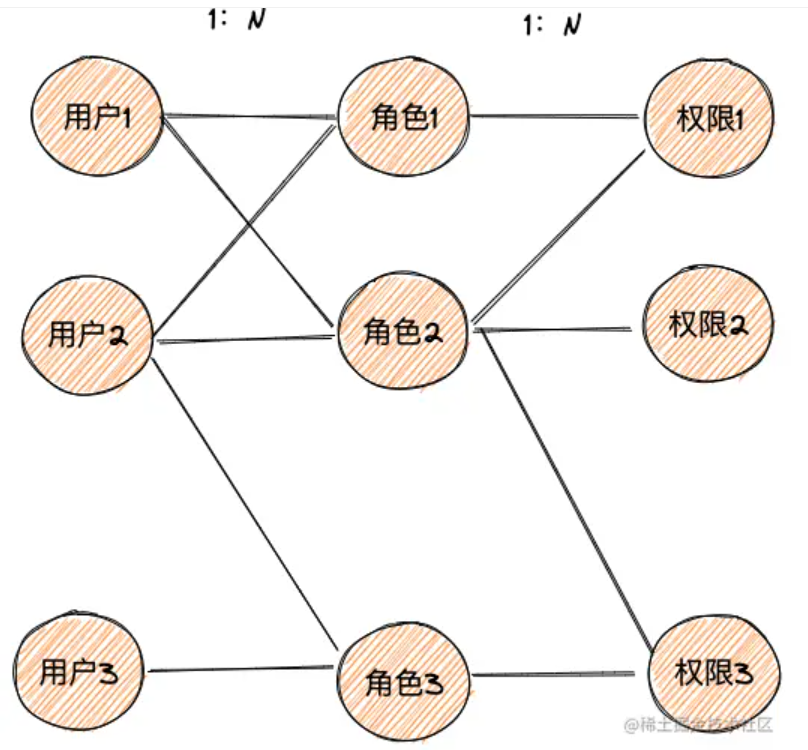
权限控制的设计

RBAC模型,基于角色的访问控制（Role-Based Access Control)

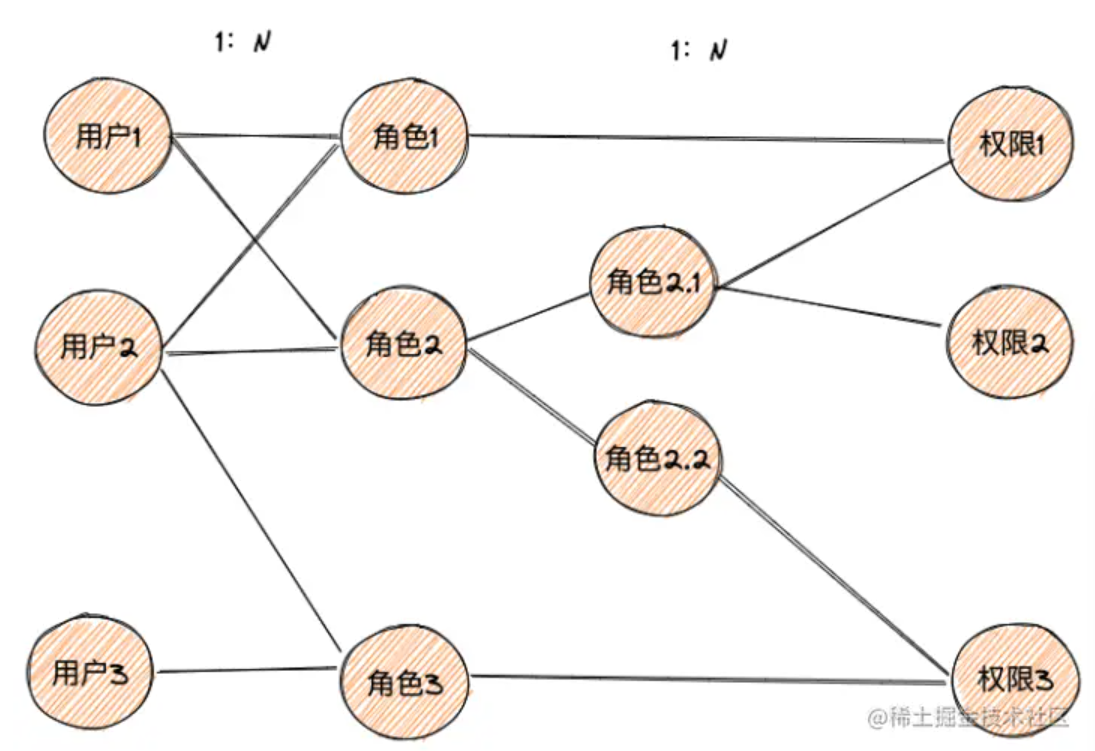
1. 用户是发起操作(增删改查)的主体。角色是桥梁，连接用户和权限。权限是用户可以访问的资源(页面、连接、接口、数据)。
2. RBAC模型：
   1. 理论RBAC



* 1. RBAC-0: 每个映射都是1-N



* 1. RBAC-1 限制了角色，只可以继承关系



* 1. RBAC2：基于RBAC0,增加了对角色和权限的限制



* 1. RBAC3=1+2
  2. 进一步，还可以加入，职责/组织/部门等

Java数据类型

Java基本类型共有八种，基本类型可以分为三类，字符类型char8，布尔类型boolean以及数值类型byte8、short16、int32、long64、float32、double64。(整型/浮点型)

基本数据类型的存储原理是：基本数据类型都是直接存储在内存中的内存栈上的，数据本身的值就是存储在栈空间里面。

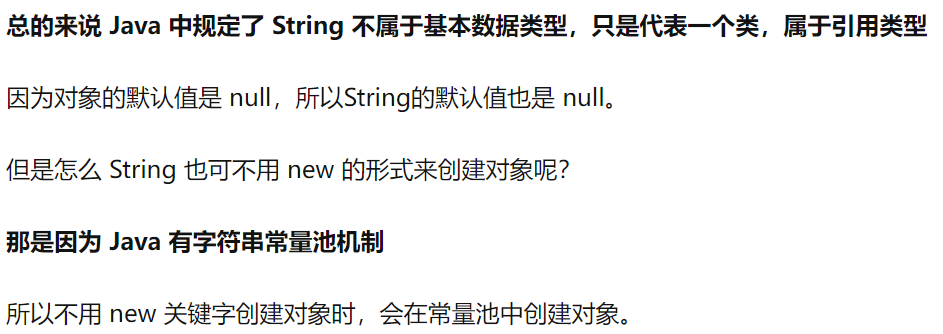
String属于什么类型？

字符串。是一个引用类型。

**Ps.Java引用数据类型**

**Java 的引用数据类型有五种**；其分别为：类、接口、枚举、注解、数组。

引用类型的数据创建对象时需要通过关键字 new 创建对象，创建的对象将会保存在堆中，然后将该对象赋值给一个引用，也就是这个引用将会指向该对象堆中的地址，而这个引用则存储在栈中。

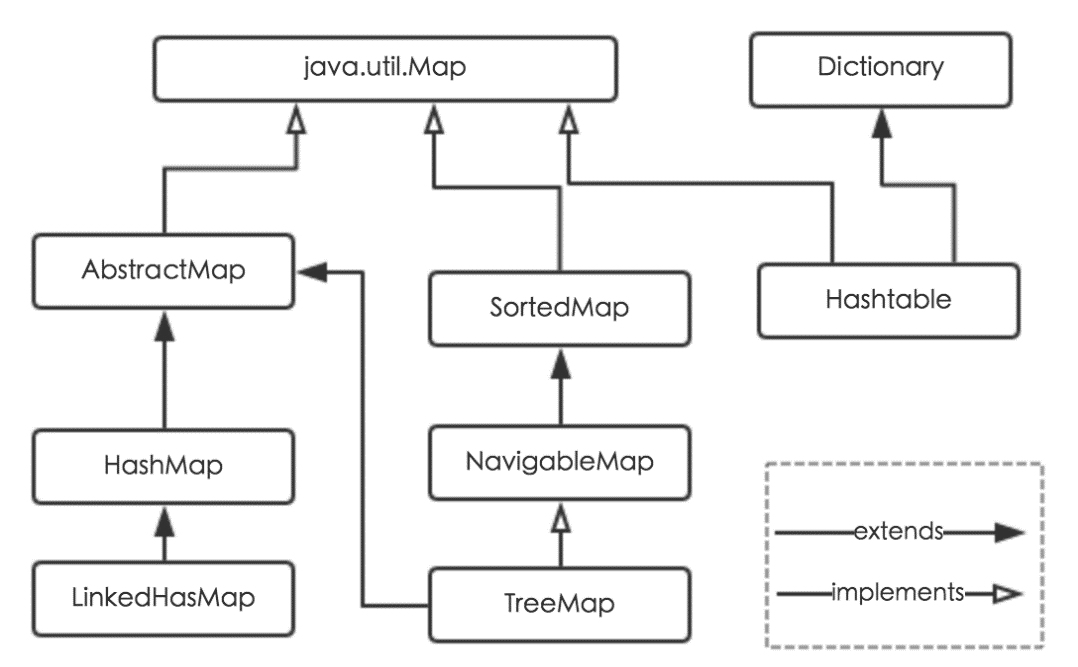


接口和抽象类区别？

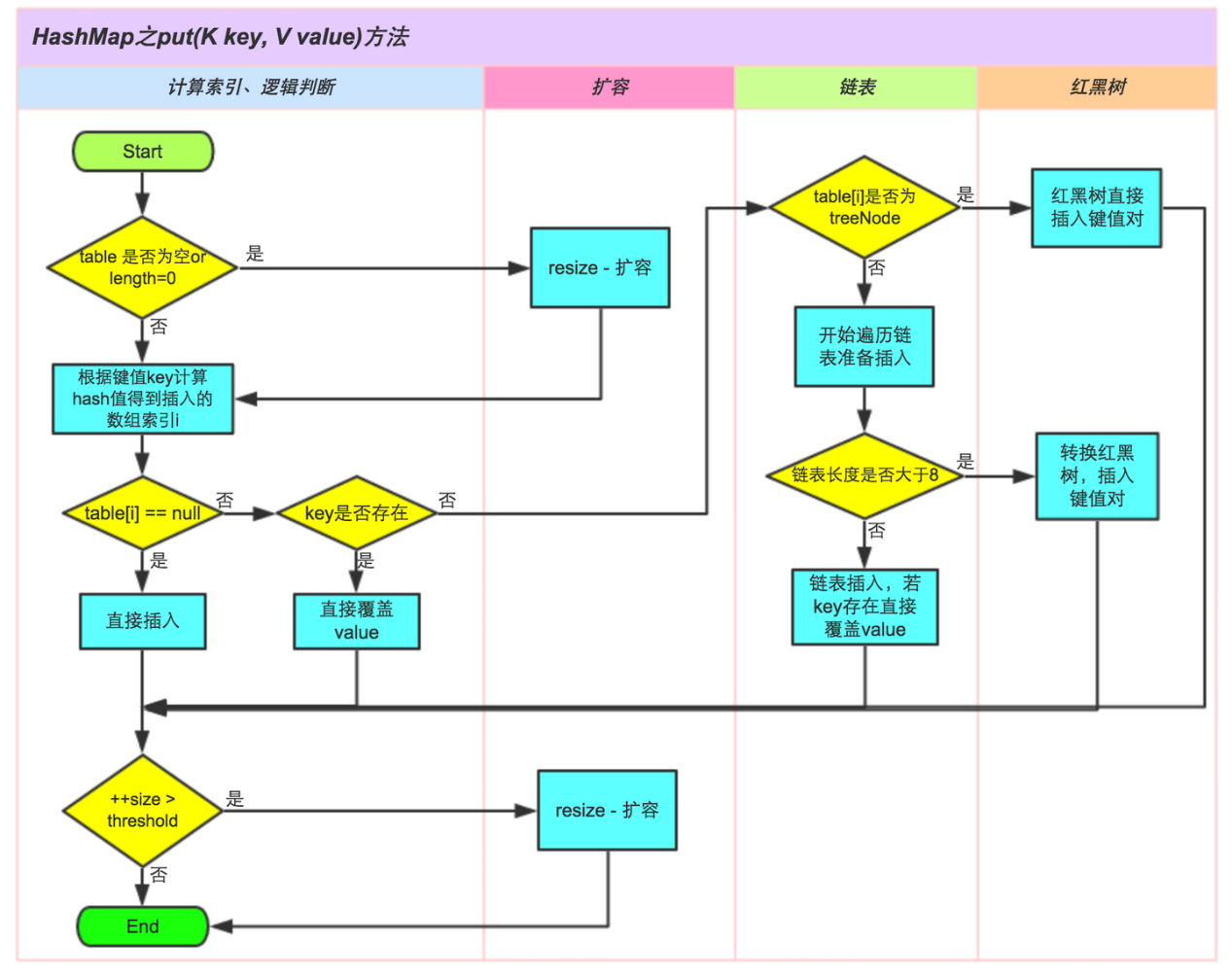


Map，Set，List？

HashMap底层原理？



1. 各个类简单说明：
   1. HashMap：根据键的hashCode存储数据。访问快。遍历顺序不确定。非线程安全，需要用ConcurrentHashMap
   2. LinkedHashMap：继承HashMap，按插入的顺序遍历。
   3. TreeMap：实现了SortedMap接口，默认升序。使用时，Key必须实现Comparable或者在构造TreeMap时传入自定义的Comparator。
   4. HashTable：遗留类。因为继承自dictionary，所以线程安全。
2. 内部实现之结构：数组+链表+红黑树
   1. Node[ ] table 哈希桶数组，是HashMap的一个内部类
   2. 哈希表解决冲突：链地址法和开放地址法。选用链地址法。
   3. Hash和扩容：
      1. 构造时，初始化table的length是16，threshold=loadFactor(0.75)\*length(16)
         1. 0.75是对时间和空间效率的平衡，泊松分布0.5
      2. 若size(k-v对数)超过threshold，则扩容。
      3. 每次扩容的新threshold，都是2的n次方。
         1. 其实素数更能避免hash冲突。Hashmap的设计是为了在取模和扩容时做优化
3. 内部实现之方法流程：
   1. 根据key定位哈希桶数组索引位置
      1. 源码本质三步：取key的hashCode值、高位运算、取模运算
         1. 取模开销大？见2.c.III -> 用hashCode&(length-1)来代替%运算，更高效。
   2. Put方法流程



* 1. 扩容过程
     1. 本质是使用一个新的数组代替旧的小容量数组。将数据转移。
     2. JDK1.8：2次幂的扩展(两倍) -> 使：重新hash后，元素的位置要么原地不动，要么在原地向二次幂的位置移动(如，原16新32，2^5的位置由0变1) -> 使：不用重新计算hash，只需要看看原来的hash值新增的那个bit是1还是0就好了，是0的话索引没变，是1的话索引变成“原索引+oldCap”
     3. 除了不用重新hash外，另一个优点是进一步将冲突的节点分散到新的bucket中。

1. 线程安全性
   1. 为什么说HashMap是线程不安全的？

并发场景下，put与resize一起造成了node指针的错乱。

* 1. ConcurrentHashMap为什么是线程安全的？

并发控制使用Synchronized关键字和CAS操作

<https://tech.meituan.com/2016/06/24/java-hashmap.html>

红黑树

<https://blog.csdn.net/v_july_v/article/details/6105630>

<https://tech.meituan.com/2016/12/02/redblack-tree.html>

==和equals()

== 对于基本类型来说是值比较，对于引用类型来说是比较的是引用；而 equals 默认情况下是引用比较，只是很多类重新了 equals 方法，比如 String、Integer 等把它变成了值比较，所以一般情况下 equals 比较的是值是否相等。

线程安全的集合？底层原理？

创建线程的方式？

线程池工作原理？

锁机制？

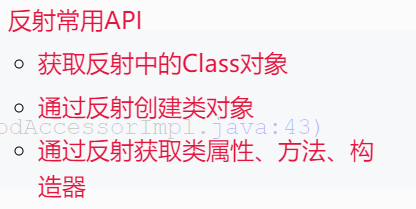
可重入锁原理？

反射？

**反射就是在运行时才知道要操作的类是什么，并且可以在运行时获取类的完整构造，并调用对应的方法。**

个人理解，反射机制实际上就是上帝模式，如果说方法的调用是 Java 正确的打开方式，那反射机制就是上帝偷偷开的后门，只要存在对应的class，一切都能够被调用。

* 优势
  + 运行期类型的判断，动态类加载：提高代码灵活度
* 劣势
  + 性能瓶颈：反射相当于一系列解释操作，通知 JVM 要做的事情，性能比直接的java代码要慢很多



<https://www.cnblogs.com/chanshuyi/p/head_first_of_reflection.html>

<https://segmentfault.com/a/1190000010162647>

反射原理？

java虚拟机在加载完类之后，在堆内存的方法区中就产生了一个Class类型的对象(一个类只有一个Class对象)，这个对象就包含了完整的类的结构信息。我们可以通过这个对象看到类的结构。这个对象就像一面镜子， 透过这个镜子看到类的结构，所以，我们形象的称之为：反射

反射应用场景？

* JDBC 的数据库的连接
* Spring 框架的使用（装配Bean）

MySQL表连接方式？

数据库三大范式？

索引？

索引设计原则？

MySQL分页查询如何实现？

having与where区别？

数据库乐观锁与悲观锁？

事务？

事务ACID？

深复制与浅复制原理？

TCP三次握手？

B+与红黑树区别

## 帆软 Java

自我介绍

蓝桥杯是干什么（自我介绍里面提到了，我参加的是算法，Java大学A组）

简历里面的项目（好好准备，回答的时候要有条理）

Spring 的 IOC 和 AOP

如何通过一个字符串获得对象的（反射，我当时答 Spring IOC 可以以 byName 或者 byType 的方式从容器中获取，class.forName）

Java 的 JDK 动态代理和 CGLib 动态代理的差异（回答得不太好，一直都没太去关心 CGLib 动态代理）

类被 final 修饰还可以使用动态代理吗？

数据库的索引

B+ 树和 B 树有什么差异，MySQL 为什么要选择 B+ 树？

前面项目里面，有用到数据库连接池吗？讲一讲数据库连接池。

讲一讲 HashMap

如何解决 Hash 冲突的

HashMap 可能出现的环形链表问题知道吗？（这个是我的 lucky problem，jdk1.7 头插，resize 时可能会出现环形链表问题）

红黑树有什么优点

为什么要用红黑树，不用普通的二叉查找树？（讲了普通的二叉查找树 和 AVL 树可能有的问题）

红黑树怎么保证自平衡？

在一个多线程的场景下，假设非要使用 map，有什么解决方案？

乐观锁与悲观锁

CAS了解吗？

CAS假设发现不一致会发生什么操作？

Java 设计模式知道哪些？

单例模式怎么实现的？【这里优点嘴瓢，饿汉和懒汉讲反了haha】、

懒汉式为什么要加双重锁？

为什么要加 volatile 关键字？

工厂模式

Java 的垃圾回收

Minor GC 和 Full GC 的触发机制有什么区别？

一个场景：线上的 Tomcat 单体应用，发现它一天要 Full GC 几十次，可能有哪些排查思路？

动态规划和贪心算法的区别

1，5，11 ，数字可以重复，想得到15，且使用的数字最少，用贪心的思路和用动态规划的思路是什么？

二叉树的深度优先和广度优先遍历

为什么有这三种遍历方式，这三种有什么差别？

中序是左 中 右，为什么不是 右 中 左？

假如内存空间比较小，使用广度还是深度遍历？

问算法，先问了我做没做过上梯子的问题，我回答做过，又改了一个问题（建议诚实回答）

1-99，不重复，随机拿掉一个数，找出拿掉的那个数

改成大数，比如1-1亿，怎么解决【这里应该还有几个类似的问题】。（提到了一些解决方案，出来查查，应该叫位图）

1-99，找出一个出现了一半以上的数

反问（深度方面还需要精进）

## 携程一面 java后端开发

<https://www.nowcoder.com/discuss/1045309?type=post&order=create&pos=&page=1&ncTraceId=&channel=-1&source_id=search_post_nctrack>

自我介绍

2. 笔试复盘，第四题：平滑值；问：当时没有全部做出来，超时了，笔试完之后有自己复盘再做过吗？答：没有，我当时用最暴力的思路写的；问：现在要你优化一下你会怎么做？然后我就问了一下题目之后，大概说出了优化思路，在面试官的引导下一步一步答出来了； 差点以为要挂

3. 问项目，我的项目就是一个商城项目；问：查询订单怎么实现，怎么进行sql优化？具体：用户要从订单表中按时间顺序查询他最近的10个订单，说一下sql语句应该怎么写，怎么优化？答：select 要查询的具体数据（不能是\*，不需要的数据不要查）from 订单表 where uid=用户id order by createTime limit 1, 10; 优化的话可以创建索引，开始说了一个根据uid创建，然后面试官引导说只是根据uid创建索引，我立马说创建 uid + createTime 的组合索引，createTime用时间戳存；

4. InnoDB引擎是怎么存储数据的？答：问了一下面试官是想问索引的存储还是什么，然后说了InnoDB是聚簇索引，主索引和数据页是存储在一个文件中的；后来查了一下InnoDB是通过页来存储数据的；这个地方没答好

5. 项目中遇到的难点，怎么解决的；

6. 用到了Redis，你了解哪些Redis数据类型，用到了哪些，你项目中Redis查询，更新，删除数据的过程；答：了解zset，set，string，list，hash，bitmap这些，项目中只用到了string，Redis查询，更新，删除就是在service层加的redis缓冲层，后面答案基本网上都有，这里不细说了；

7. zset底层数据结构你了解吗？答：底层是用字典+跳跃表实现的，Redis是用C语言写的，里边实现了一个zset数据结构，包括字典和跳跃表两个指针...

8. java垃圾回收机制你了解吗，有哪些垃圾回收算法？从分代回收假说扯到了三种回收算法，再说了一下垃圾回收算法各自的优缺点，适用场景；

9. java垃圾回收时是把整个内存都扫描一遍吗还是？首先说了一下怎么定义垃圾，一种是引用计数法，另一种是可达性分析法，然后引用计数法不能解决循环引用的问题，可达性分析法是从GC root集合往下开始扫描；

10. 具体是怎么扫描的/标记的？答：根据引用链来对对象进行标记；

11. GC Root对象有哪些？答：常量指向的对象，虚拟机栈中局部变量指向的对象，本地方法栈中局部变量指向的对象...，没说全；

12. 平时是怎么学习的；

反问：

1. 被分配到的事业部？

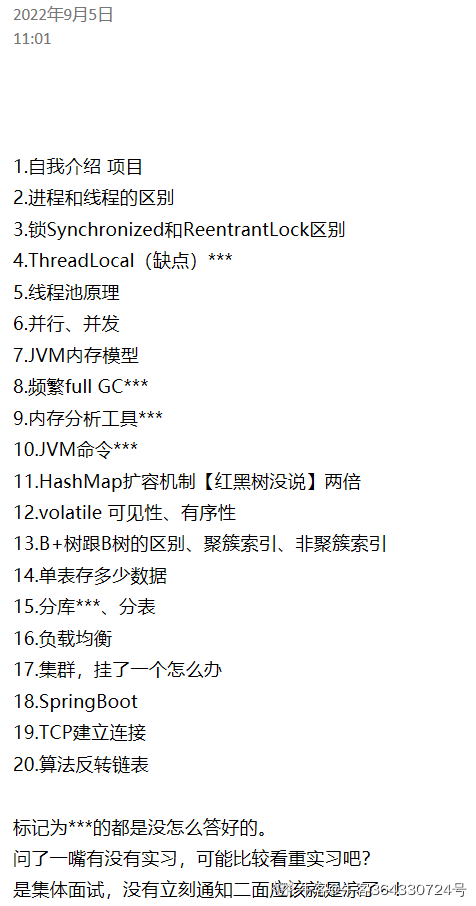
校招先招进来，暂时不能确定是哪个事业部。

2. 那业务应该也不确定了吧？

很有可能是分配到面试官对应的部门，主要是做携程内部使用的工具开发，（那感情好）。

3. 技术栈？

和你们现在使用的差不多，除了一些公司内部的技术，原理都差不多。



## 笔试延伸

Annotation注解？

<https://www.liaoxuefeng.com/wiki/1252599548343744/1265102026065728>

* **编写文档：** 通过代码里标识的元数据生成文档【生成文档doc文档】
* **代码分析：** 通过代码里标识的元数据对代码进行分析【使用反射】
* **编译检查：** 通过代码里标识的元数据让编译器能够实现基本的编译检查【Override等】

#### Redis数据类型，各自应用？

常用的高并发框架？

#### Optional类？

Optional 类是一个可以为null的容器对象。如果值存在则isPresent()方法会返回true，调用get()方法会返回该对象。

Optional 是个容器：它可以保存类型T的值，或者仅仅保存null。Optional提供很多有用的方法，这样我们就不用显式进行空值检测。

Optional 类的引入很好的解决空指针异常。



面向报文传输(UDP)和面向字节流传输(TCP)

<https://blog.csdn.net/ce123_zhouwei/article/details/8976006>

HTTP请求方法



观察者模式

观察者模式，又称发布-订阅模式，是一种一对多的通知机制，使得双方无需关心对方，只关心通知本身。

实质：加一层，把所有关心变动的成员归为Obserber，作为Productor的一个私有成员。频道无序变动，若增加观察者，改变Oberver类即可。

使用场景：

* 邮件订阅、RSS Feeds
* 消息队列
* 浏览器DOM事件监听与交互

<https://www.liaoxuefeng.com/wiki/1252599548343744/1281319577321505>

锁与并发与多线程9.15

1. 锁的出现：资源共享发生争抢冲突
2. 一个类比。进程是任务，线程是资源。Win创建进程开销大(时间上)，Linux创建进程开销小->因此建议win多线程，面对资源争抢与同步问题。建议linux多进程，面对进程间通讯问题。 ->尤其 服务端应用多核CPU，由于进程之间的资源独立，线程间资源共享需要在多核间复制。多线程在性能上不如多进程。
3. 多线程应用场景：客户端和服务端。服务端多一点，即多个客户端并发访问服务器的问题。
4. 并发：多个时间在同一时间间隔内发生。、区分并行：同时开始。
5. 并发处理机制：加锁；不可变消息；单线程

## 莱斯

Java8有哪些新特性，怎么用？

1. Lambda 表达式，也可称为闭包
   1. Lambda 允许**把函数作为一个方法的参数**（函数作为参数传递进方法中）。
   2. 使用 Lambda 表达式可以使代码变的更加简洁紧凑。
2. 方法引用
   1. 方法引用通过方法的名字来指向一个方法。
   2. 方法引用使用一对冒号 **::**
3. 函数式接口
   1. 一个接口，有且仅有一个抽象方法，但是可以有多个非抽象方法
   2. 函数式接口可以被隐式转换为 lambda 表达式
   3. 使用举例：runnable、Callable、Comparator、FileFilter
4. Stream
   1. 是什么？：Stream（流）是一个来自数据源的元素队列并支持聚合操作
   2. 类似用 SQL 语句从数据库查询数据的直观方式来提供一种对 Java 集合运算和表达的高阶抽象
   3. Spring配置类中常用
5. Optional类
   1. 是什么？Optional 是个容器：它可以保存类型T的值，或者仅仅保存null。
   2. 有什么用？：Optional 类的引入很好的解决空指针异常。这样我们就不用显式进行空值检测。
6. Supplier
   1. 是什么？：Supplier表示结果的提供者，该结果返回一个对象且不接受任何参数
   2. 有什么用？：最适合用于表示工厂。简而言之，Supplier就是用来创建对象的，相当于new。
   3. 使用举例：git，其中加一个volatile的stage列表，不必输入git指令时每次new。

有哪些集合类

list都有哪些实现方式，底层结构是什么

哪种集合没有重复元素，实现类有哪些，底层结构是什么

map用过哪些实现方式

hashmap线程安全吗

哪种map比较安全，为什么

多个线程怎么共用变量的

问题本质就是：怎么保证可见性的？

Java中可以通过synchronized、volatile、java concurrent类来实现共享变量的可见性

Java垃圾处理机制

* **回收的对象？**

堆。

* **那些内存需要回收？(对象是否可以被回收的两种经典算法: 引用计数法 和 可达性分析算法)**
  1. 引用计数算法：判断对象的引用数量
     + 难以解决：对象之间循环引用
  2. 可达性分析算法：判断对象的引用链是否可达
     + 可作为 GC Root 的对象包括以下几种：
       - 虚拟机栈(栈帧中的局部变量表)中引用的对象；
       - 方法区中类静态属性引用的对象；
       - 方法区中常量引用的对象；
       - 本地方法栈中Native方法引用的对象；
* **什么时候回收？ （堆的新生代、老年代、永久代的垃圾回收时机，MinorGC 和 FullGC）**
* **如何回收？(三种经典垃圾回收算法(标记清除算法、复制算法、标记整理算法)及分代收集算法 和 七种垃圾收集器)**

引用计数法有什么问题吗

可达性分析法在一次标记之后就回收吗

即使被可达性分析算法标记为垃圾的对象，并一定就会被GC回收，也就是不一定 “非死不可”。要真正宣告一个对象死亡（成为一定被回收的垃圾对象），至少要经历两次标记过程：

引用链标记（第一次标记）

F-Queue标记（第二次标记）

如果对象在进行可达性分析后发现没有与任何的 “GC Roots” 引用链相连接，则会被第一次标记，随后会进行一次筛选过程，筛选的条件是 “该对象是否有必要执行 finalize() 方法”。只有在该对象\*\*\*没有覆盖 finalize() 方法\*\*\*，或者 finalize() 方法已经被虚拟机调用过 时，才认为是 “没有必要执行”。

Java内存模型

* JVM 内存模型一共包括三个部分：
  + 堆 ( Java代码可及的 Java堆 和 JVM自身使用的方法区)、
  + 栈 ( 服务Java方法的虚拟机栈 和 服务Native方法的本地方法栈 )
  + 保证程序在多线程环境下能够连续执行的程序计数器

进程间有哪些通信方式

管道、信号量、信号、socket套接字

socket通信都需要哪些信息

进程和线程的定义、区别

OSI四层模型都包括哪几层

tcp和udp

tcp: 传输控制协议,全拼:Transmission Control Protocol 它是一个面向连接，可靠的传输协议

udp: 用户数据报协议,全拼:User Datagram Protocol 它不是面向连接，不是可靠的传输协议, udp协议传输速度快

tcp和udp都是传输层的两个传输协议

tcp的特点:

4.1 面向连接，间接验证对方ip的有效性

4.2 可靠的传输

4.2.1 应答机制:发送数据包完成以后，对方收到数据底层会回复

4.2.2 超时重传:发送数据以后对方没有进行回复，会隔一段时间再次给对方发送数据，如果对方一直没有回复，那么会认为对方已经掉线了

4.2.3 错误校验:如果收到的数据和之前发送数据包的序号不一致，会自动根据需要进行排序，如果收到重复的数据包，会把重复的数据包删除

4.2.4 流量控制:使用tcp能保证接收数据的时候电脑不会卡死

5. tcp和udp的不同点对比

5.1 tcp 面向连接， udp不面向连接

5.2 tcp 能保证数据有效和有序的传输，udp保证不了

5.3 tcp 有超时重传，udp没有

5.4 tcp 有错误校验，如果出现数据包顺序不一致会自动排序，还有如果收到数据包重复会自动删除重复的数据包，udp没有

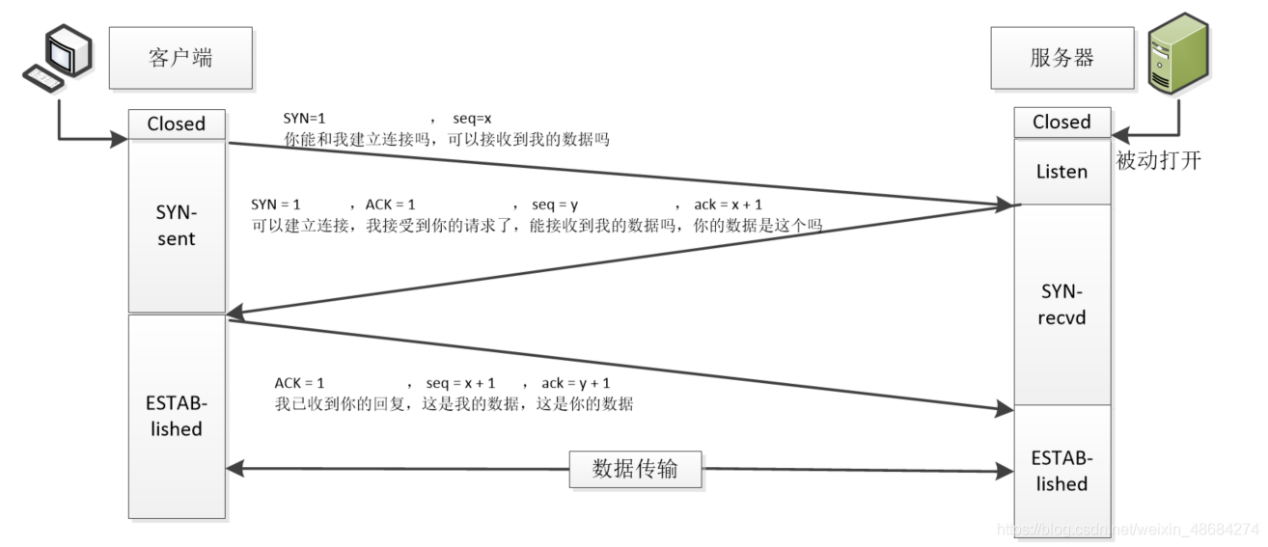
5.5 tcp 有流量控制 udp没有

5.6 tcp 需要建立连接然后需要资源开销要大， udp不需要建立连接资源开销小

tcp属于哪一层

tcp是怎么保证可靠连接的

三次握手中ack和ACK分别表示什么



广播适合用哪种传输协议

广播和多播仅应用于UDP

广播用于一个主机对整个局域网上所有主机上的数据通信

因为不用建立连接？效率高？

知道哪些设计模式，怎么分类

（1）创建型模式：单例模式、抽象工厂模式、建造者模式、工厂模式、原型模式。 ——他抽象了对象的实例化过程，将系统与实例的创建解耦

（2）结构型模式：适配器模式、桥接模式、装饰模式、组合模式、外观模式、享元模式、代理模式。——用于处理类和对象的组合，以获得更大的结构

（3）行为型模式：模版方法模式(AQS和reentrantlock)、命令模式、迭代器模式、观察者模式、中介者模式、备忘录模式、解释器模式、状态模式、策略模式、职责链模式、访问者模式。——涉及到算法和对象间职责的分配，它还处理着对象或类之间的通信模式

单例模式是什么

<https://www.runoob.com/design-pattern/singleton-pattern.html>

懒汉式是怎么处理的

工厂模式属于哪种类型的设计模式

用过哪些设计模式

闲聊

对28所的了解

怎么看待加班

团队中有人甩锅怎么办

其他公司今年的招聘情况

## 莱斯

gitlet项目是怎么开始的？有阅读论文学习吗？

讲讲面向对象

1. 面向对象是什么：

通过对象的方式，把现实世界映射到计算机模型。

现实世界中，我们定义了“人”这种抽象概念，而具体的人则是“小明”、“小红”、“小军”等一个个具体的人。所以，“人”可以定义为一个类（class），而具体的人则是实例（instance）

2. 三大特征：

封装： 把一个对象的状态信息(即属性)隐藏在对象内部，不允许外部对象直接访问对象的内部信息，但是可以提供一些可以被外界访问的方法来操作属性。

继承：表示is-a关系。

1. 子类拥有父类对象所有的属性和方法（包括私有属性和私有方法），但是父类中的私有属性和方法子类是无法访问，**只是拥有**。
2. 子类可以拥有自己属性和方法，即子类可以对父类进行扩展。
3. 子类可以用自己的方式实现父类的方法。（以后介绍）

多态：

多态是同一个行为具有多个不同表现形式或形态的能力。比如，飞机、鸟都能执行“飞”

具体表现为父类的引用指向子类的实例。

* 对象类型和引用类型之间具有继承（类）/实现（接口）的关系；
* 引用类型变量发出的方法调用的到底是哪个类中的方法，必须在程序运行期间才能确定；
  + 多态不能调用“只在子类存在但在父类不存在”的方法；
  + 如果子类重写了父类的方法，真正执行的是子类覆盖的方法，如果子类没有覆盖父类的方法，执行的是父类的方法。

Spring依赖注入和控制反转

1. 控制反转IOC

反转之前：对象之间相互依赖，耦合度过高，牵一发而动全身。

控制反转的基本思想：借助于“第三方”实现具有依赖关系的对象之间的解耦。

第三方指：IOC容器。

第三方作为“粘合剂”，把系统中的对象粘合在一起

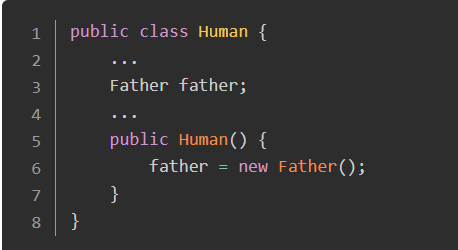
为什么“反转”？

* 1. 对象A依赖于对象B，A在初始化或者运行过程中，必须主动创建B或者使用已经创建的B。——控制权在对象A的手上
  2. 系统引入IOC容器后，对象A与对象B失去了直接的联系。当A运行到了需要B的地方，IOC容器会主动创建一个对象B注入到对象A需要的地方。——控制权到了IOC容器的手上，A与B同样的被控制。

2. 依赖注入DI

什么是“依赖”？

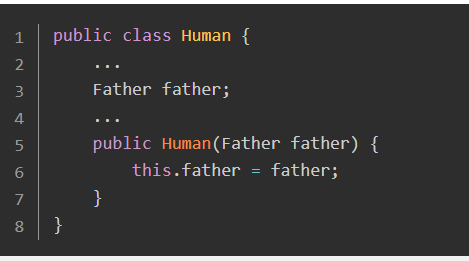
1. 如果在 Class A 中，有 Class B 的实例，则称 Class A 对 Class B 有一个依赖



缺点：

1. 如果现在要改变 father 生成方式，如需要用new Father(String name)初始化 father，需要修改 Human 代码；
2. 如果想测试不同 Father 对象对 Human 的影响很困难，因为 father 的初始化被写死在了 Human 的构造函数中；
3. 如果new Father()过程非常缓慢，单测时我们希望用已经初始化好的 father 对象 Mock 掉这个过程也很困难。
4. 依赖注入

我们将 father 对象(实例)作为构造函数的一个参数传入。在调用 Human 的构造方法之前外部就已经初始化好了 Father 对象。像这种非自己主动初始化依赖，而通过外部来传入依赖的方式，我们就称为依赖注入。

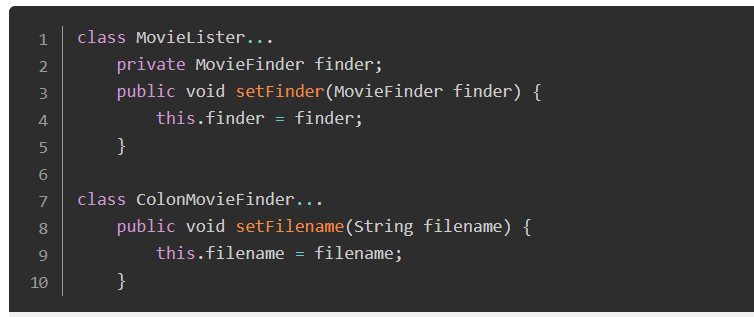


优点：

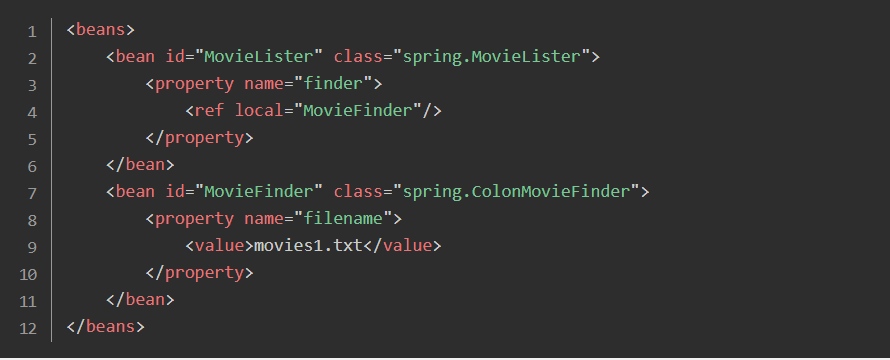
1. 解耦，将依赖之间解耦。
2. 因为已经解耦，所以方便做单元测试，尤其是 Mock 测试。

测试举例：

* + 1. 定义两个类，使用依赖注入

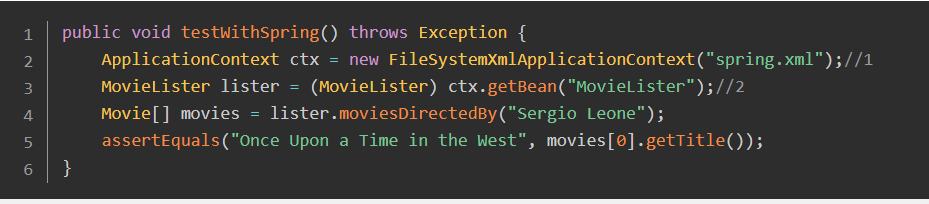


* + 1. 使用IOC容器配置:xml配置/注解配置



在Spring中，每个bean代表一个对象的实例，默认是单例模式，即在程序的生命周期内，所有的对象都只有一个实例，进行重复使用。通过配置bean，IoC容器在启动的时候会根据配置生成bean实例。具体的配置语法参考Spring文档。这里只要知道IoC容器会根据配置创建MovieFinder，在运行的时候把MovieFinder赋值给MovieLister的finder属性，完成依赖注入的过程。

* + 1. 测试



**1.** 根据配置生成ApplicationContext，即IoC容器。  
**2.** 从容器中获取MovieLister的实例。

3. 区别总结

a. 控制反转是软件工程中一种解耦合的思想。调用类只依赖接口，而不依赖具体的实现类。控制权交给了容器。在运行时，由容器决定将具体的实现提供给调用类的对象中。

b. 依赖注入是一种设计模式，可以看做是控制反转的一种实现方式。

JPA

1. 是什么？

Java Persistence API

基于OR映射的标准**规范**（**ORM**:对象关系映射）

1. JPA与hibernate的关系
   1. **JPA是一个规范**，不是框架
   2. hibernate是JPA的实现
2. JPA的供应商

Hibernate (JPA的始作俑者就是hibernate的作者)、spring data jpa

1. JPA和Mybatis区别

JPA：

* 1. 从根本上来说是开发模式的转变，新项目不用事先就设计好完整的数据库和表结构，开发过程中自动生成数据库表结构，可以快速开发和迭代，逐渐完善。
  2. 简单的CRUD不需要写SQL语句。持久层的代码量更少，维护起来更加简单，后续新增、修改字段时，大多数时候只需要维护 Java Entity 的注解映射关系就行了
  3. **根本上来说，完全的面向对象。**
  4. 不便于编写复杂动态SQL

Mybatis：

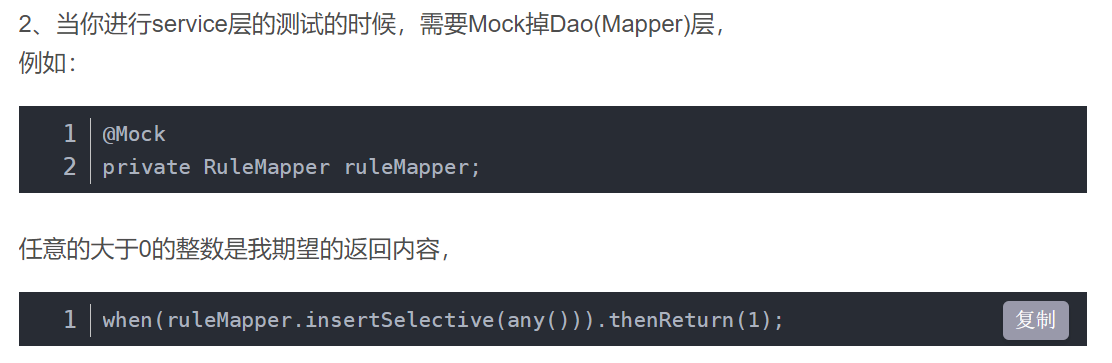
1. 对SQL的编写非常灵活，可以完全自由控制
2. **根本上来说，是面向关系型数据库的。**

junit。知不知道mock？

1. 是什么？：单元测试时，对于某些不容易构造的对象，创建一个虚拟的mock对象。
2. 为什么？：解耦，不关注北侧单元的依赖对象，只关注该被测单元。
3. 应用场景：
   1. 前后端开发，后端接口未实现
   2. service层的代码中，包含对Dao层的调用，但是，DAO层代码尚未实现
   3. 支付宝支付的异常条件有很多，直接呈现这些异常，而不是实现。

如果现在需要测试数据，但是还没有数据库，在不改变service层代码的前提下，怎么测试？

mock掉dao层，直接给service层提供想要的数据



[Mockito在spring中使用](https://blog.csdn.net/qq_25905629/article/details/123521322?spm=1001.2101.3001.6650.7&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-7-123521322-blog-53561435.pc_relevant_multi_platform_featuressortv2dupreplace&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-7-123521322-blog-53561435.pc_relevant_multi_platform_featuressortv2dupreplace&utm_relevant_index=8)

[**Mockito编写Service层单元测试**](https://blog.csdn.net/u012760435/article/details/90643872?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-90643872-blog-105039414.pc_relevant_multi_platform_whitelistv3&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-90643872-blog-105039414.pc_relevant_multi_platform_whitelistv3&utm_relevant_index=1)

JMM有了解吗？

## 满帮

1.介绍一下springboot自动装配

2.实现自动转配的是那个类

3.是如何实现配置类的装入的

（问的挺深的，这里答的不太好）

4.业务题：数据库中有一张user表，有一个字段name，让你实现查询某一姓氏的用户的数量，你会怎么实现

5.你说用到了count(\*)，那你说说count(\*),count(1),count(列)的区别

6.事务的隔离级别

7.介绍一下MVCC

8.介绍一下常用的集合

9.介绍一下有序集合（说了TreeSet、PriorityQueue）

10.那TreeSet底层数据结构（笑死，我说完二叉搜索树，面试官说是红黑树还是二叉搜索树，当时脑子抽了，以为面试官给我挖坑呢，都提示了还是说二叉搜索树）

11.jdk中priorityQueue的应用场景（不太懂什么意思）

12.认识跳表吗（只说了在redis中zset底层使用了跳表，原理不太了解）

13.你刚才说到了可以使用工具类让集合有序，那sort方法底层使用了什么排序算法？（这里只说了快排，问，除了快排还有吗，又答了归并，应该是快排、插入、归并；数据量小的时候使用插入，数据量大，数据没有结构时使用快排，有一定结构使用归并）

14.说说你了解的排序算法，快排一定是nlogn吗

15.说说你对线程安全的理解

16.举个线程不安全的例子（举了两个线程同时自增10000次）

17.那你会如何解决（CAS、加锁）

18.CAS原理

19.你会自己实现CAS吗（这里不太理解面试官的意思，我说CAS中compare和swap两个操作必须是原子操作，只能借助操作系统底层指令完成，我们直接调用unsafe对应的native api就好了）

20.那你会调用unsafe中的api吗（也不太理解面试官意思，答了我们不需要直接调用，只需要调用JUC封装好的原子类就好了）

21.说说线程池参数（漏了workQueue和拒绝策略）

22.线程池执行任务的过程

23.常见的拒绝策略

24.救急线程被回收的过程

25.业务题：你的部门要完成一个“十月大促的任务”，ddl已经很近了，你作为负责人会如何完成这项任务；（清晰规划+及时反馈）

## 锐捷网络

-java 集合框架，concurrentHashmap哪里用会有线程安全问题，treemap

我们都知道ConcurrentHashMap是个线程安全的哈希表容器，但它仅保证提供的原子性读写操作线程安全。



解决方案：整段逻辑加锁

java 锁 lock了解吗，原理

看jvm内存怎么看，线程cpu占用呢

* + 1. jps:查看本地正在运行的java进程和进程ID（pid）
    2. jinfo pid，查看指定pid的所有[JVM](https://so.csdn.net/so/search?q=JVM&spm=1001.2101.3001.7020)信息
    3. jmap，堆内存设置和占用情况
    4. jstat：Java虚拟机统计工具，可以用于监视JVM各种堆和非堆内存大小和使用量
    5. jconsole，Java的GUI监视工具

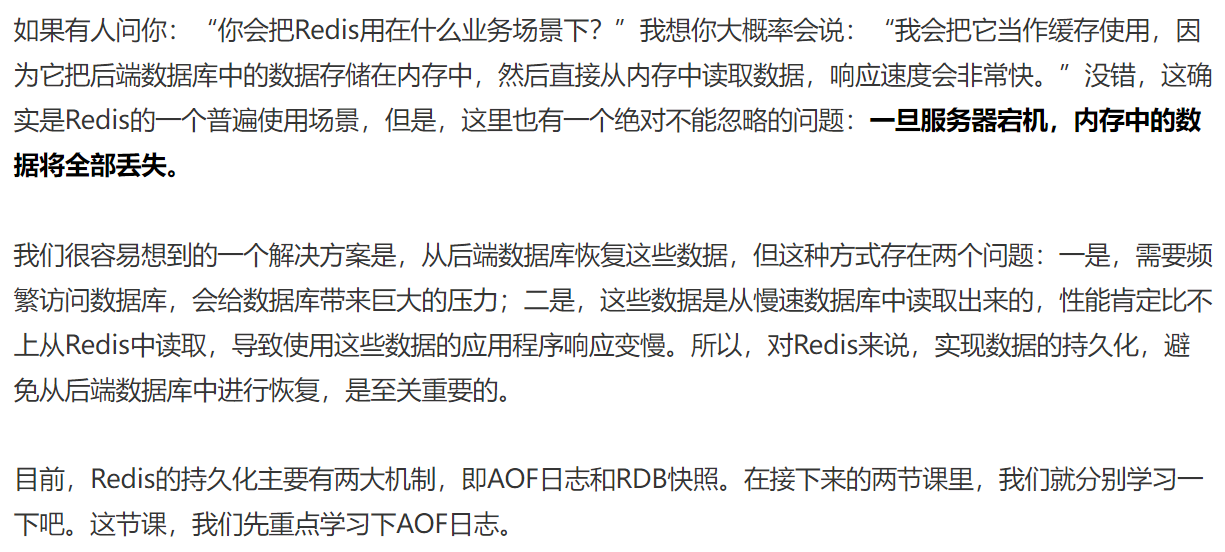
**ps 命令**

通过ps命令可以查看系统中相关进程的Cpu使用率的信息

**top命令**

 ps只是查看进程,而top还可以监视系统性能,如平均负载,cpu和内存的消耗

redis 宕机丢失数据？持久化？如果集群解决不了呢？



为避免服务器宕机，redis作为内存服务器，数据全部消失->数据持久化与快速恢复

1. **AOF：AppendOnlyFile写后日志**
2. AOF是写后日志，通过记录操作命令持久化数据。
3. 由于AOF是在命令执行之后记录日志，如果在写入磁盘之前服务器宕机，则会丢失数据；如果写入磁盘的时候突然阻塞，则会阻塞主线程；为了解决以上问题，AOF机制提供了三种写回的策略，每种策略都有不同的优缺点。
4. AOF日志文件过大怎么办？AOF通过fork一个子线程重写一个新的日志文件（共享主线程的内存，记录最新数据的写入命令），同时子线程重写，避免阻塞主线程。。
5. **RDB：RedisDataBase内存快照**
6. RDB是内存快照，记录某一个时刻的内存数据，而不是操作命令。
7. Redis提供了两个命令，分别是save、bgsave来执行全量快照，这两个命令的区别则是save是在主线程执行，势必会阻塞主线程，bgsave是在fork一个子线程，共享内存。
8. RDB通过操作系统的「写时复制技术」，能够保证在执行快照的同时主线程能够修改快照。
9. 由于两次快照之间是存在间隔的，一旦服务器宕机，则会丢失两次间隔时刻的数据，Redis4.0开始使用AOF日志记录两次快照之间执行的命令（AOF和RDB混合使用）。

<http://118.25.23.115/redis/04-AOF%E6%97%A5%E5%BF%97%EF%BC%9A%E5%AE%95%E6%9C%BA%E4%BA%86%EF%BC%8CRedis%E5%A6%82%E4%BD%95%E9%81%BF%E5%85%8D%E6%95%B0%E6%8D%AE%E4%B8%A2%E5%A4%B1%EF%BC%9F.html>

<https://zhuanlan.zhihu.com/p/348347905>

ping www.baidu.com 发生了什么

递归找，dns本地域名服务器ip在哪，怎么找

①DNS服务器有很多级，更靠近用户的那级服务器就叫做本地DNS服务器，也叫**递归解析器**。本地DNS服务器一般是网络服务商提供的DNS，也可以手动修改为公共DNS。

- ARP协议理解

- tcp怎么理解，一些握手挥手的常规问题

- redis 事务，缓存击穿

- mysql 索引 日志原理

## 携程一面1

[算法题](https://www.nowcoder.com/jump/super-jump/word?word=%E7%AE%97%E6%B3%95%E9%A2%98)：利用数组实现一个支持泛型的栈

八股：快排和归并[排序](https://www.nowcoder.com/jump/super-jump/word?word=%E6%8E%92%E5%BA%8F)区别和时间复杂度

以及如何证明快排的平均时间复杂度

tcp工作流程

tcp重传机制 他是如何避免杯攻击的（设置重传次数

又问如何设置重传次数

## 携程一面2

1. 自我介绍

2. 单例模式的实现步骤，饿汉式和懒汉式各自的优缺点

（static对象、public的外界获取方法、私有化的构造函数；影响初始化速度但内存占用少 VS 线程安全问题和创建大对象时占用CPU时间）

3. HashMap的底层实现，对应线程安全的HashTable和ConcurrentHashMap的区别

4. Redis 实现消息队列的话用什么[数据](/jump/super-jump/word?word=%E6%95%B0%E6%8D%AE)结构

5. Redis 的 key 到达过期时间后的删除策略

6. 若大 Key 读取速度时间长达 60s，与此同时 Redis 还会做其他事情吗，通过什么实现这样并行的

（Redis本身是单线程，Linux内核帮助处理 I/O，多路复用）

7. 场景题：

对于抢购火车票，用户可以有多订单（多个行程）、多出行时间、多始发地与目的地的请求；而对 12306 爬取到的[数据](/jump/super-jump/word?word=%E6%95%B0%E6%8D%AE)量巨大，上报给服务端后需完成大量的订单信息匹配工作，QPS达到50-100万。有哪些优化的点？

（分布式业务扩展、Redis缓存（将订单按始发地等不同维度合并到一起方便查询）、消息队\*\*\*定抢到票后的后续操作异步执行）、降级（抢购不激烈的票延后几秒执行也不影响抢到的概率）和削峰（过滤票已经售罄的无效请求））

8. 对于爬虫请求如何充分利用服务器资源实现一个客户端同时发送多个请求

（多线程、I/O多路复用）

9. Docker的作用、底层如何实现独立的文件系统和IP地址与资源隔离的

（Linux namespace、 cgroups 、软虚拟化）

## 比亚迪二面

1、Java多态的实现方式

父类or接口定义的引用变量可以指向子类or实现类的实例化对象。

程序调用的方法在运行期才动态绑定

2、JVM对象的创建，方法、属性、静态变量存放的位置

方法-方法区

属性-方法区

静态变量-GC堆

[对象的创建](https://javaguide.cn/java/jvm/memory-area.html#%E5%AF%B9%E8%B1%A1%E7%9A%84%E5%88%9B%E5%BB%BA)

3、如何解决相互争夺资源的问题

串行->线程池管理一组同构的工作线程，底层是通过内置锁或者可重入锁

4、高并发场景中，如何解决余额为-1的问题

解决超买超卖：

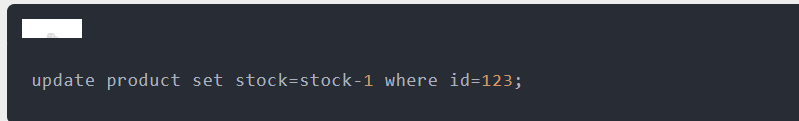
针对高并发 ：

1. 不能直接读写数据库，要用缓存
2. 多用户并发修改，需要加锁（悲观锁、乐观锁）
3. 不要在数据库层面加锁，通过服务端的内存锁

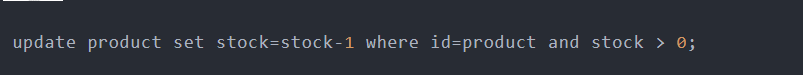
10/21更新：

①基于数据库的乐观锁

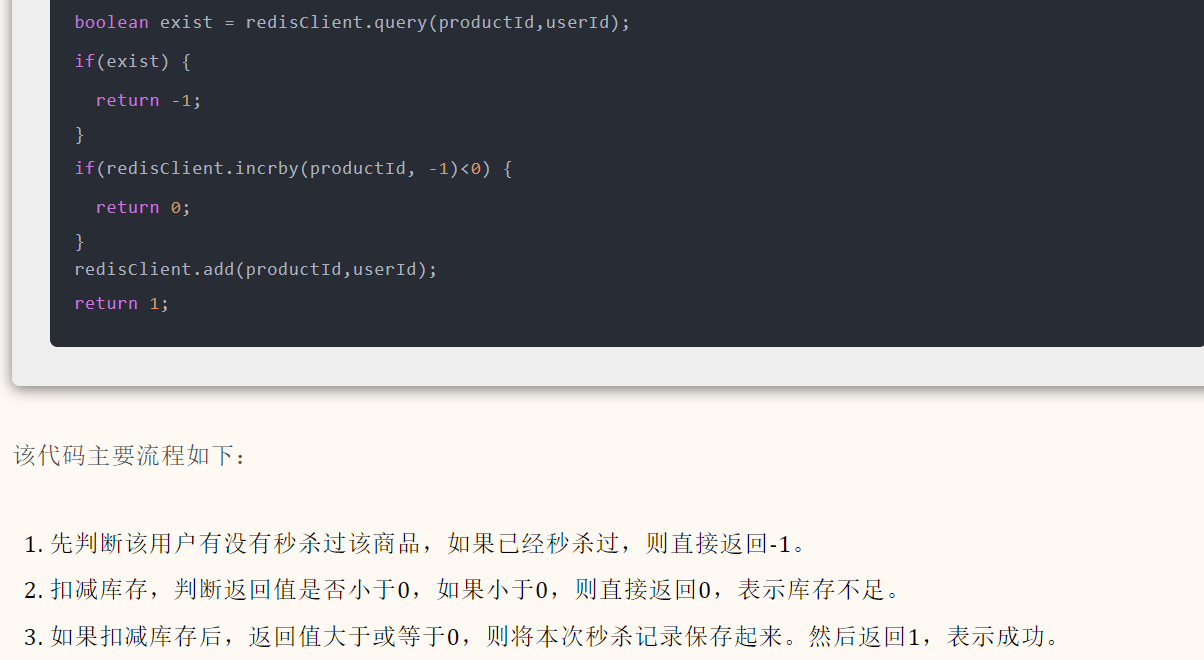
线程不安全的：



基于数据库的乐观锁，这样会少一次数据库查询，而且能够天然的保证数据操作的原子性。



②但是数据库连接还是昂贵的，竞争行锁也会互相等待死锁->redis扣除库存



\*一个redis服务层上的乐观锁

5、谈谈你对同步和异步的理解

是一种消息通知机制。多线程的、线程之间协作的，高层次、全局的视角

同步：按顺序一个一个来

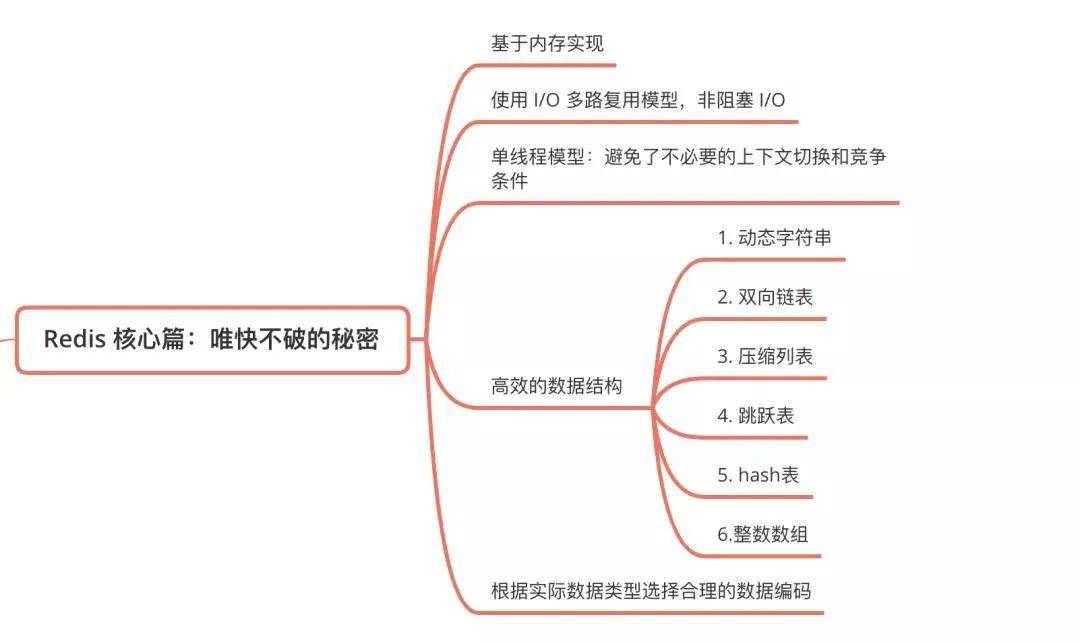
异步：交交接任务，后台等待，等待回调

6、单线程和多线程具体的使用场景

多线程 ：servlet分发来自客户端的请求、数据库连接池、数据库事务处理、JVM启动（后台线程+main主线程）

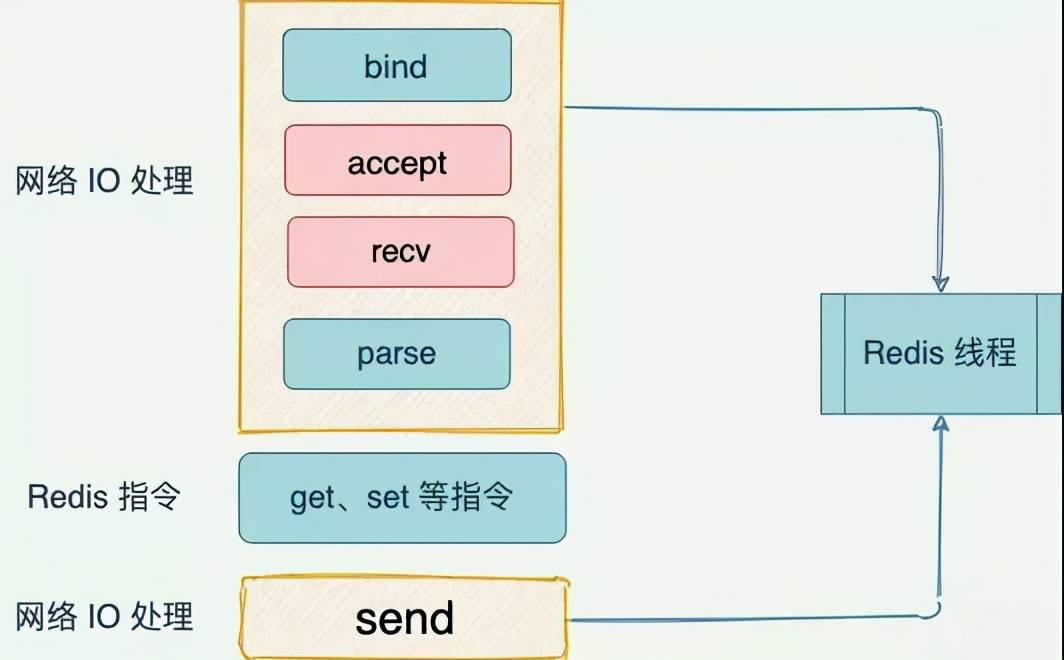
单线程：**Redis** 中只有网络请求模块和数据操作模块是单线程的**-** **"其网络 IO 和键值对读写是由一个线程完成的"。**其持久化和集群是多线程的。（因为Redis基于内存操作，内存直接与CPU对接，读写速度不像磁盘IO那么慢，没必要提高cpu利用率）

Ps.Redis为什么快？——十万QPS



<https://www.sohu.com/a/458584271_411876>

由IO多路复用说开来，



阻塞的原因由于使用传统阻塞 IO ，也就是在执行 read、accept 、recv 等网络操作会一直阻塞等待。

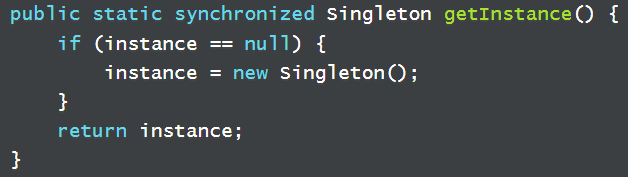
**IO 多路复用**

**多路**指的是多个 socket 连接，**复用**指的是复用一个线程。多路复用主要有三种技术：select，poll，epoll。epoll 是最新的也是目前最好的多路复用技术。

7、单例模式对象创建的安全性？

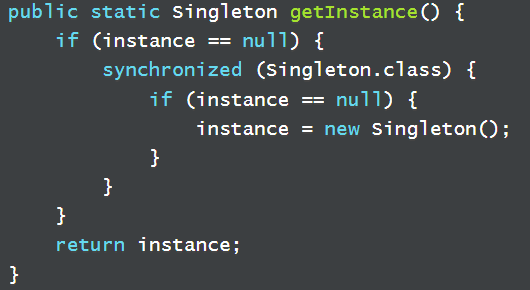
<https://www.sunzn.com/2020/06/14/%E5%8D%95%E4%BE%8B%E6%A8%A1%E5%BC%8F%E7%9A%84%E5%AE%89%E5%85%A8%E6%80%A7%E6%BC%94%E8%BF%9B/>

1. 饿汉模式：
   1. 饿汉模式是最简单的一种实现方式，静态变量在类加载的时候就对实例进行创建，实例在整个程序周期都存在。
   2. 安全。没有同步问题。But即使不用到、很晚用掉也在类加载就创建，浪费内存！
2. 懒汉模式：
   1. 第一次需要时创建
   2. 在多个线程并发调用它的 getInstance() 方法时，会导致创建多个实例，因此需要加锁解决线程同步问题
   3. 解决：getInstance加synchronised锁 。



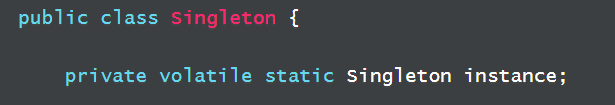
But整个方法加锁，存在性能问题

1. 双重校验锁
   1. 在方法内部，对Singleton.class加锁。仅当第一次创建时才加锁。类似读写锁？



But指令重排序，导致初始化 Singleton 和将对象地址赋给 instance 字段的顺序是不确定的。

1. 对成员变量instance加volatile关键字，禁止重排序



## 蚂蚁一、二面（很多场景应用题）

<https://www.nowcoder.com/discuss/1068792>

项目为什么用springboot？

在使用Spring框架进行开发的过程中，需要配置很多Spring框架包的依赖，如spring-core、spring-bean、spring-context等，而这些配置通常都是重复添加的，而且需要做很多框架使用及环境参数的重复配置，如开启注解、配置日志等。Spring Boot致力于弱化这些不必要的操作，提供默认配置，当然这些默认配置是可以按需修改的，快速搭建、开发和运行Spring应用。

## 秒杀系统是怎么设计的？

<https://www.teqng.com/2021/09/07/%E9%9D%A2%E9%9C%B8%EF%BC%9A%E7%A7%92%E6%9D%80%E7%B3%BB%E7%BB%9F%E5%A6%82%E4%BD%95%E8%AE%BE%E8%AE%A1%EF%BC%9F/>

