# Dubbo系列

## Dubbo面试题

<https://blog.csdn.net/Y0Q2T57s/article/details/83005376>

## rpc和http restful的区别

rpc是远程过程调用,只能用服务本身提供的方法和接口与服务器无关，整个过程，请求，响应和方法客户端都是不可见的，异构的客户端很难与服务器通信，也很难解析服务器的响应请求。

**REST 是面向资源的，这个概念非常重要，而资源是通过 URI 进行暴露**

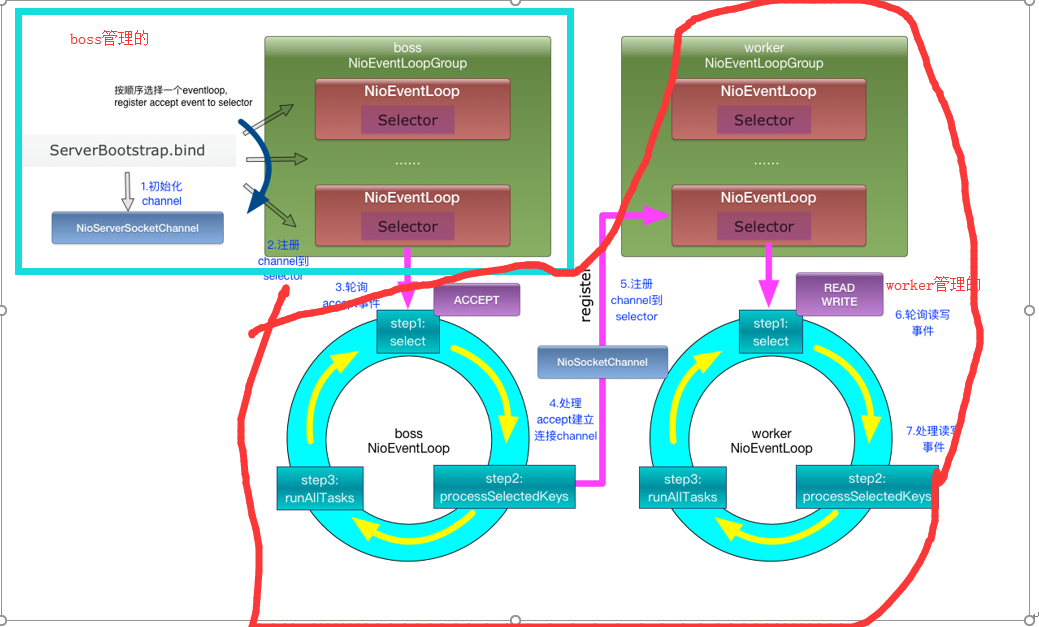
## RPC原理



|  |
| --- |
| 一次完整的RPC调用流程（同步调用，异步另说）如下：  **1）服务消费方（client）调用以本地调用方式调用服务；**  2）client stub接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；  3）client stub找到服务地址，并将消息发送到服务端；  4）server stub收到消息后进行解码；  5）server stub根据解码结果调用本地的服务；  6）本地服务执行并将结果返回给server stub；  7）server stub将返回结果打包成消息并发送至消费方；  8）client stub接收到消息，并进行解码；  **9）服务消费方得到最终结果。**  RPC框架的目标就是要2~8这些步骤都封装起来，这些细节对用户来说是透明的，不可见的。 |

rpc的底层就是netty

netty基本原理

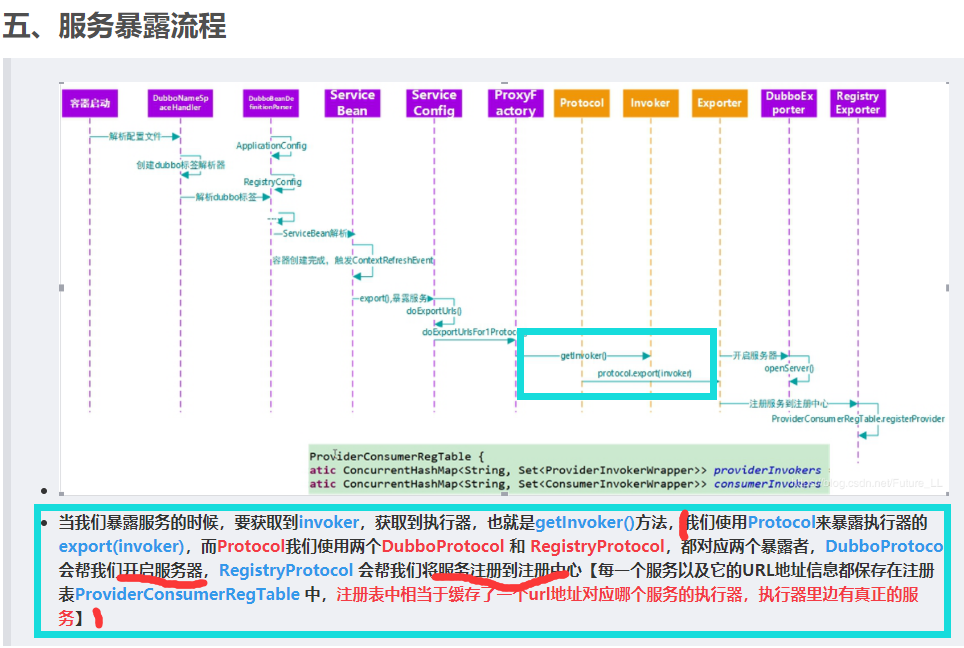


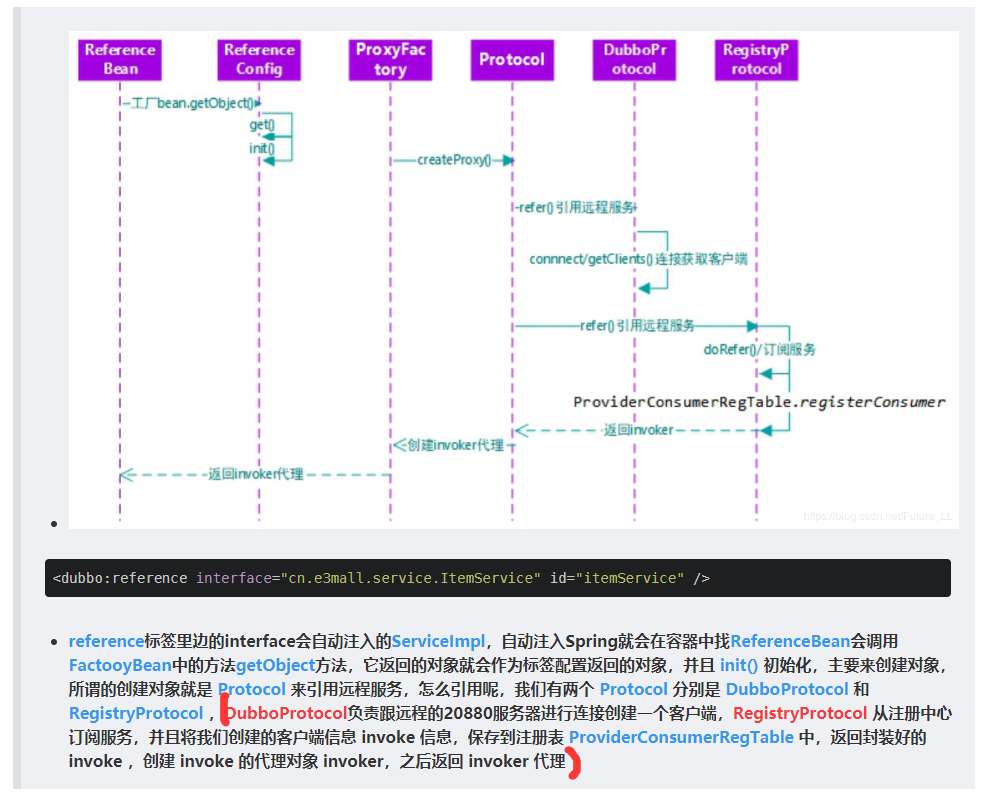
netty服务器启动，绑定监听端口 208080 所有给这个端口发的数据，netty服务器就能收到，netty服务器启动后会初始化通道，通道初始化好后会注册到selector中（selector是一个多路复用器），selector负责监听accept事件，accept事件就是当通道准备就绪，准备处理通道中的数据，怎么处理呢，netty与客户端建立连接，生成niosocketchannel(niosocketchannel是netty与客户端连接的通道)，把这个通道注册到另一个selector中，只不过这个selector监听read和write事件，相当于niosocketchannel中数据读准备就绪，数据写准备就绪，读准备就绪或者写准备就绪就会抛给任务队列来执行

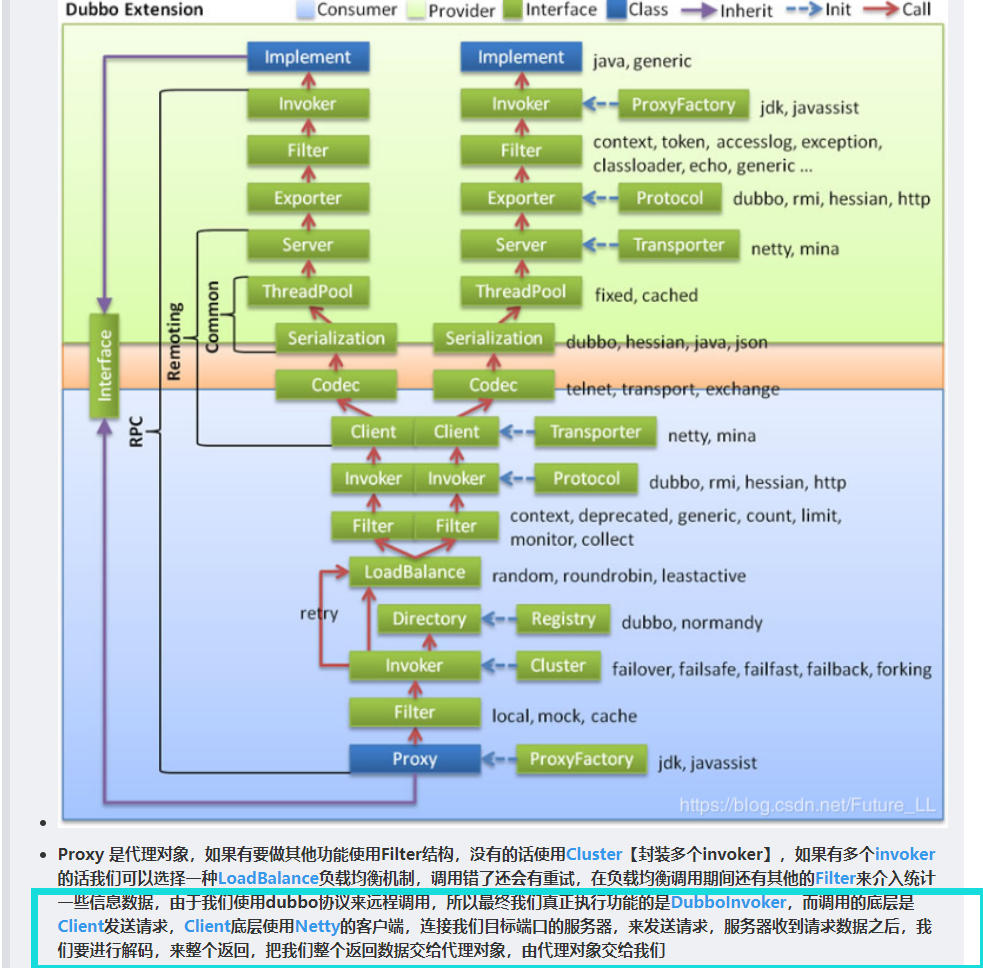
注意：boss线程组，用来监听来自20880所有连接准备就绪事件，worker线程组，就是准备就绪后要做什么工作

## Dubbo服务暴露，服务引用，服务调用

<https://blog.csdn.net/Future_LL/article/details/86721318>



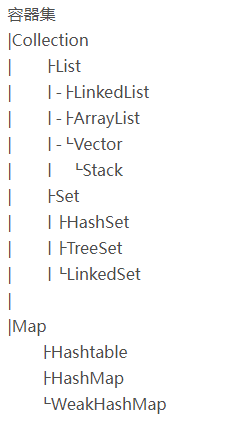




# 面试

## 蚂蚁花呗一面

### 1.Java容器有那些？那些是同步容器，那些是并发容器（容器代码）



同步容器（通过synchronized来实现同步的容器，如果有多个线程调用同步容器的方法，它们将会串行执行）

Vector

Stack

HashTable

Collections.synchronized方法生成

并发容器

ConcurrentHashMap：线程安全的HashMap的实现

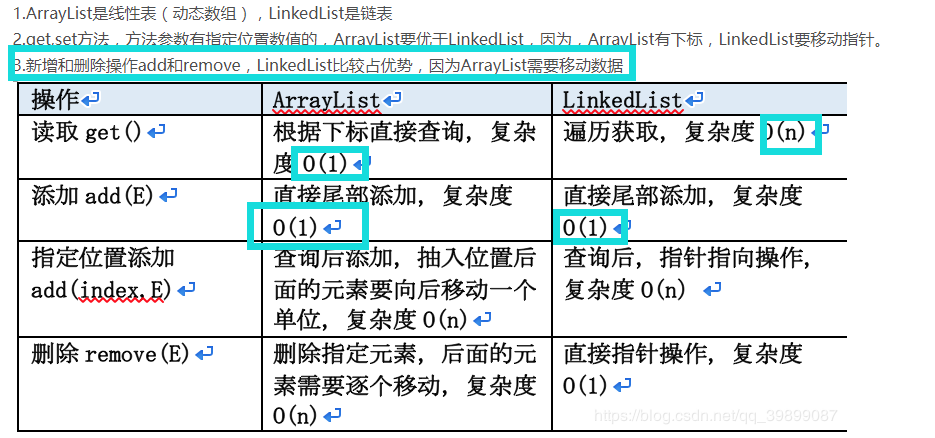
CopyOnWriteArrayList：线程安全且在读操作时无锁的ArrayList

CopyOnWriteArraySet：基于CopyOnWriteArrayList，不添加重复元素

ArrayBlockingQueue：基于数组、先进先出、线程安全，可实现指定时间的阻塞读写，并且容量可以限制

LinkedBlockingQueue：基于链表实现，读写各用一把锁，在高并发读写操作都多的情况下，性能优于ArrayBlockingQueue

### 2ArrayList和linkedList的插入和访问的事件复杂度



### 3.java反射原理，注解原理

#### 反射原理

1.将Java文件保存到本地硬盘

2.编译Java文件，生成.class文件

3.使用Java虚拟机（JVM）将字节码文件加载到内存

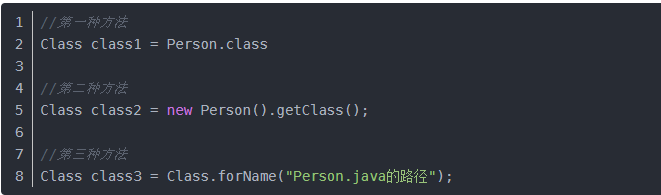
4.字节码文件在内存中使用Class类表示

5.使用反射的时候，首先获取到Class类，就可以得到class文件里的所有内容，包含属性、构造方法、普通方法

6.属性通过Filed类表示

7.构造方法通过Constructor表示

8.普通方法通过Method表示



#### 注解原理

注解是在编译时由编译器进行处理，编译器对注解符号处理并附加到class结构中（即保存到class结构的attributes属性中）

比如在类A上添加一个注解B，当类A被编译的时候，由于注解是作用在类上，所以此属性被添加到类的属性集上，即B注解的键值对会被记录起来。。

其实注解被编译后的本质就是一个继承Annotation接口的接口，

### 4新生代分为几个区？使用什么算法进行垃圾回收？为什么使用这个算法

一个eden，两个survvivor 8:1:1

在新生代中，每次垃圾收集时都发现有大批对象死去，只有少量存活，那就选用复制算法，只需要付出少量存活对象的复制成本就可以完成收集。

### 5HashMap在什么情况下会扩容，或者那些操作会导致扩容

Java中HashMap<K,V>的数据结构,虽然数据结构上稍微有点差异,不过思想都是一样的.我们还是以HashMap<K,V>进行讲解,我们知道HashMap<K,V>有一个叫装载因子的东西,默认情况下HashMap<K,V>的装载因子是75%这是在时间和空间上寻求的一个折衷.那么什么是所谓的装载因子,装载因子其实是用来判断当前的HashMap<K,V>中存放的数据量,如果我们存放的数据量大于了75%,那么HashMap<K,V>就需要进行扩容操作,扩容的空间大小就是原来空间的1.5倍，已有元素个数的2倍。0.75 \*ｌｅｎｇｔｈ。但是扩容的时候需要reshash操作,其实就是讲所有的数据重新计算HashCode,然后赋给新的HashMap<K,V>,rehash的过程是非常耗费时间和空间的,因此在我们对HashMap的大小进行控制的时候,应该要进行相当的考虑.还有一个误区(HashMap<K,V>可不是无限大的.)

多线程下HashMap为啥会出现死循环

Resize步骤：

1. 扩容：创建一个新的Entity空数组，长度是原数组的2倍。
2. ReHash:遍历Entry数组，把所有的Entry重新hash到新数组，为什么要重新hash呢？因为长度扩大以后，hash的规则也随之改变

多个线程rehash的时候会出现死循环

### 6HashMap push方法的执行过程

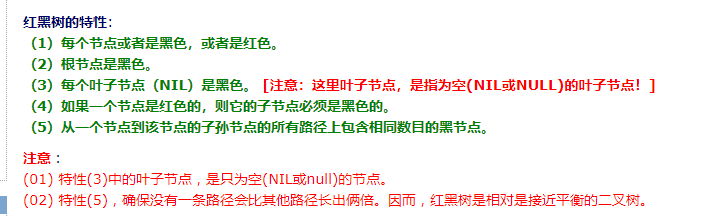
<https://blog.csdn.net/AJ1101/article/details/79413939>

### **7hashmap检测到hash冲突后，将元素插入在链表的末尾还是开头**

在jdk1.8之前是插入头部的，在jdk1.8中是插入尾部的。

<https://blog.csdn.net/qq_33256688/article/details/79938886>

### 8jdk1.8还采用了红黑树，讲讲红黑树的特性，为什么人家一定要用红黑树而不是avl，b数之类的？



#### 红黑树的应用

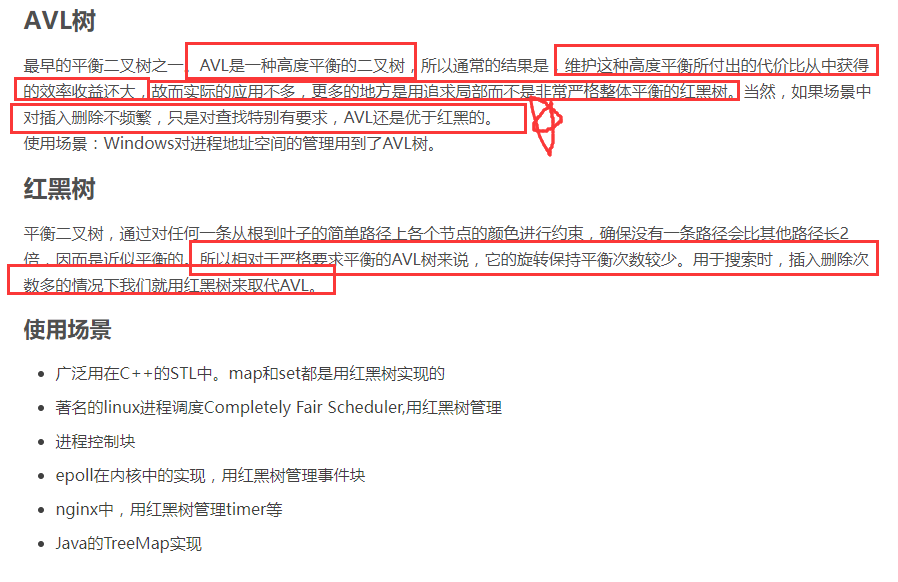
红黑树的应用比较广泛，主要是用它来存储有序的数据，它的时间复杂度是O(lgn)，效率非常之高。  
例如，Java集合中的[TreeSet](http://www.cnblogs.com/skywang12345/p/3311268.html)和[TreeMap](http://www.cnblogs.com/skywang12345/p/3310928.html)，C++ STL中的set、map，以及Linux虚拟内存的管理，都是通过红黑树去实现的。

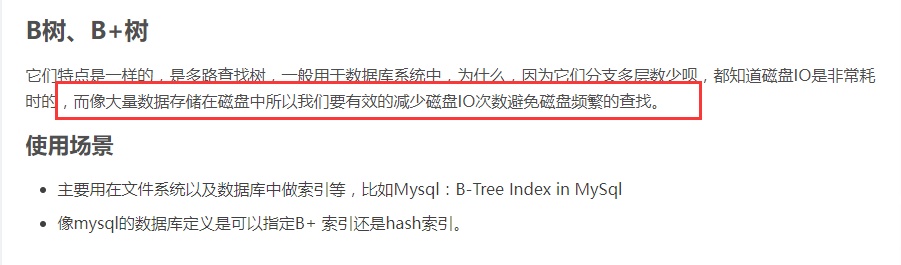
#### 为什么人家一定要用红黑树而不是avl，b数之类的？

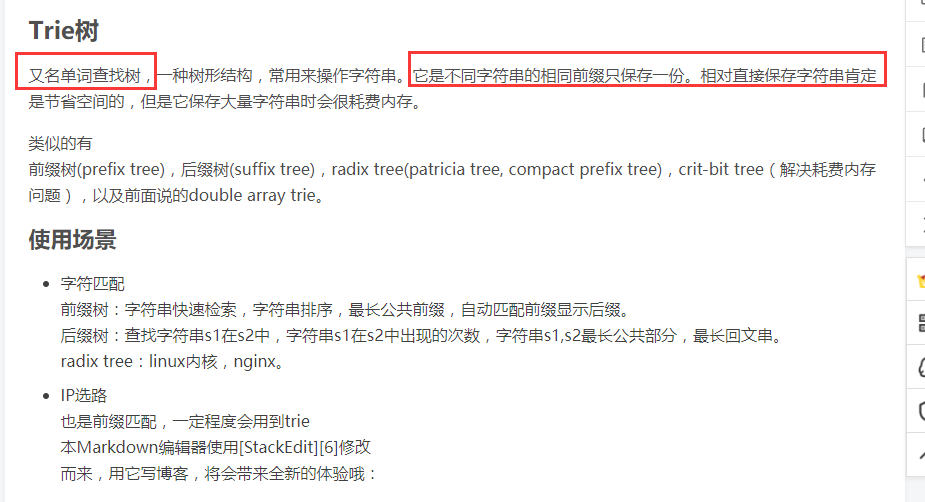
1. 红黑树并不追求“完全平衡”——它只要求部分地达到平衡要求，降低了对旋转的要求，从而提高了性能。红黑树能够以O(log2 n) 的时间复杂度进行搜索、插入、删除操作。此外，由于它的设计，任何不平衡都会在三次旋转之内解决。AVL树是最先发明的自平衡二叉查找树。在AVL树中任何节点的两个儿子子树的高度最大差别为一，所以它也被称为高度平衡树。查找、插入和删除在平均和最坏情况下都是O(log n)。增加和删除可能需要通过一次或多次树旋转来重新平衡这个树。

引入二叉树的目的是为了提高二叉树的搜索的效率,减少树的平均搜索长度.为此,就必须每向二叉树插入一个结点时调整树的结构,使得二叉树搜索保持平衡,从而可能降低树的高度,减少的平均树的搜索长度.

#### AVL树、红黑树、B/B+树和Trie树的比较







#### 比较符合实际的来说

红黑树就是一种平衡的二叉查找树，说他平衡的意思是他不会变成瘸子，左腿特别长或者右腿特别长。最长路径不超过最短路径的2倍。

### 9https和http区别，有没有用过其他安全传输手段

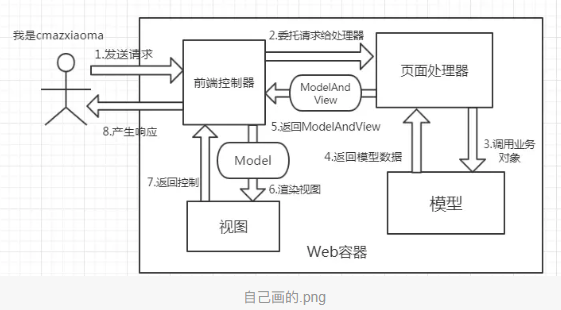
HTTP+Security s可以是TLS(安全传输层协议)，SSL（安全套接字）

* OpenVPN——这个协议可以实现任何出发点到目的地之间的数据安全连接，OpenVPN通过建立一个加密的UDP实现这项功能。其他的一些协议只与特定的操作系统兼容,OpenVPN则可以兼容任何的系统。

10线程池的工作原理，几个重要的参数，然后给了几个参数分析线程池会怎么做，最后问阻塞队列的作用是什么。

11linux怎么查看系统负载情况

12请详细描述springmvc处理请求全流程



1：首先客户端发送一个http请求，web服务器接受这个请求，然后web容器交给DispatcherSevlet处理。

2：DispatcherSevlet将负责将请求分发。DispatcherSevlet可以认为是spring提供的前端控制器，所有的请求都要经过他来进行统一的分发。

3:在DispatcherSevlet将请求分发给spring controller之前，需要借助于spring提供的HandleMapping定位到具体的controller。（HandleMapping是这样的一种对象，他能够完成客户请求和controller之间的映射。在structs中，这种映射是通过struts-config.xml文件完成的。spring为controller提供了若干实现，例如：BeanNameUrlHandlerMapping有，SimpleUrlHandlerMapping，CommonsPathMapHandlerMapping。）

4:spring controller将处理来自DispatcherSevlet的请求。spring的controller类似于structs的action。能够接受HttpServletRequest和HttpServletResponse。spring为controller提供了若干实现类，由于controller要为并发用户处理上述请求，因此实现controller接口时，必须保证线程安全并且可重用。controller将处理用户请求，这和structs中的action是一致的。

5:一旦controller处理完用户请求，则返回ModelAndView对象给DispatcherSevlet前端控制。ModelAndView包含了模型（model）和视图（view）。从宏观角度来看，DispatcherSevlet是整个web应用的控制器。从微观来看，controller是单个http请求的控制器。而ModelAndView是http请求过程中返回的模型和视图。

6:ModelAndView中包含了视图逻辑名与模型数据信息。DispatcherSevlet通过ViewResolver完成逻辑视图名到真实视图对象的解析工作。

7:当得到真实对象的view，DispatcherServlet就使用这个View对象对ModelAndView中的模型数据进行视图渲染。然后将渲染后的结果返回给客户。

13讲一讲AtomicInteger,为什么要用CAS而不是synchronized

## 蚂蚁金服电话二面1

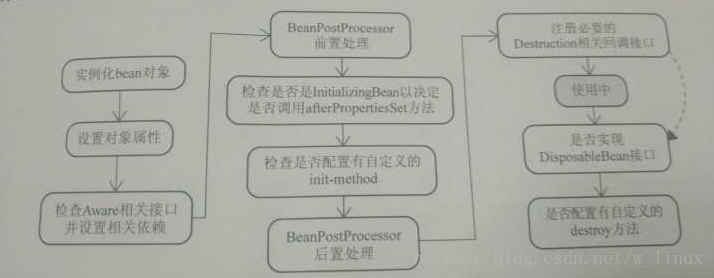
<https://blog.csdn.net/baidu_26780083/article/details/87902877>

1. 自我介绍，工作经历，技术栈
2. 项目中你学到了什么技术（把3个项目具体描述了很久）
3. 微服务划分的粒度
4. 微服务的高可用是怎么保证的
5. 常用的负载均衡，该怎么用

### 6网关能够为后端服务带来那些好处

后端服务器可以专心处理业务请求，节省了大量连接管理的开销

### 7Spring bean的生命周期



1. 实例化bean对象(通过构造方法或者工厂方法)
2. 设置对象属性(setter等)（依赖注入）
3. 如果Bean实现了BeanNameAware接口，工厂调用Bean的setBeanName()方法传递Bean的ID。（和下面的一条均属于检查Aware接口）
4. 如果Bean实现了BeanFactoryAware接口，工厂调用setBeanFactory()方法传入工厂自身
5. 将Bean实例传递给Bean的前置处理器的postProcessBeforeInitialization(Object bean, String beanname)方法
6. 调用Bean的初始化方法
7. 将Bean实例传递给Bean的后置处理器的postProcessAfterInitialization(Object bean, String beanname)方法
8. 使用Bean
9. 容器关闭之前，调用Bean的销毁方法

### 8HashSet是不是线程安全的？为什么不是线程安全的

什么是线程安全，就是对于数据的读写要线程隔离，不能导致数据的丢失和不一致，每次修改数据都不应该被覆盖掉。

查看源代码发现HashSet内部维护数据的采用的是HashMap，根本原因是HashMap不是线程安全的类。导致了HashSet的非线程安全。

### 9HashMap的底层实现

#### 简单说下HashMap的实现原理：

首先有一个每个元素都是链表（可能表述不准确）的数组，当添加一个元素（key-value）时，就首先计算元素key的hash值，以此确定插入数组中的位置，但是可能存在同一hash值的元素已经被放在数组同一位置了，这时就添加到同一hash值的元素的后面，他们在数组的同一位置，但是形成了链表，同一各链表上的Hash值是相同的，所以说数组存放的是链表。而当链表长度太长时，链表就转换为红黑树，这样大大提高了查找的效率。

#### 加载因子

加载因子（默认0.75）：为什么需要使用加载因子，为什么需要扩容呢？因为如果填充比很大，说明利用的空间很多，如果一直不进行扩容的话，链表就会越来越长，这样查找的效率很低，因为链表的长度很大（当然最新版本使用了红黑树后会改进很多），扩容之后，将原来链表数组的每一个链表分成奇偶两个子链表分别挂在新链表数组的散列位置，这样就减少了每个链表的长度，增加查找效率

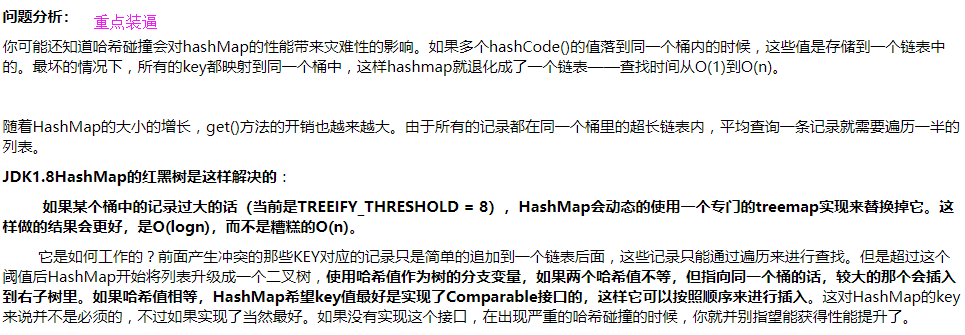
#### put（key,value）的流程

1，判断键值对数组tab[]是否为空或为null，否则以默认大小resize()；

2，根据键值key计算hash值得到插入的数组索引i，如果tab[i]==null，直接新建节点添加，否则转入3

3，判断当前数组中处理hash冲突的方式为链表还是红黑树(check第一个节点类型即可),分别处理

#### 重点装逼（哈希碰撞的优化）



## 蚂蚁金服电话二面2

### 1查询中哪些情况不会使用索引？

* 查询谓词没有使用索引的主要边界,换句话说就是select \*，可能会导致不走索引。

比如，你查询的是SELECT \* FROM T WHERE Y=XXX;假如你的T表上有一个包含Y值的组合索引，但是优化器会认为需要一行行的扫描会更有效，这个时候，优化器可能会选择TABLE ACCESS FULL，但是如果换成了SELECT Y FROM T WHERE Y = XXX，优化器会直接去索引中找到Y的值，因为从B树中就可以找到相应的值。

* 单键值的b树索引列上存在null值，导致COUNT(\*)不能走索引。

如果在B树索引中有一个空值，那么查询诸如SELECT COUNT(\*) FROM T 的时候，因为HASHSET中不能存储空值的，所以优化器不会走索引，有两种方式可以让索引有效，一种是SELECT COUNT(\*) FROM T WHERE XXX IS NOT NULL或者把这个列的属性改为not null (不能为空)。

* 索引列上有函数运算，导致不走索引

如果在T表上有一个索引Y，但是你的查询语句是这样子SELECT \* FROM T WHERE FUN(Y) = XXX。这个时候索引也不会被用到，因为你要查询的列中所有的行都需要被计算一遍，因此，如果要让这种sql语句的效率提高的话，在这个表上建立一个基于函数的索引，比如CREATE INDEX IDX FUNT ON T(FUN(Y));这种方式，等于Oracle会建立一个存储所有函数计算结果的值，再进行查询的时候就不需要进行计算了，因为很多函数存在不同返回值，因此必须标明这个函数是有固定返回值的。

* 隐式转换导致不走索引。

索引不适用于隐式转换的情况，比如你的SELECT \* FROM T WHERE Y = 5 在Y上面有一个索引，但是Y列是VARCHAR2的，那么Oracle会将上面的5进行一个隐式的转换，SELECT \* FROM T WHERE TO\_NUMBER(Y) = 5,这个时候也是有可能用不到索引的。

* 表的数据库小或者需要选择大部分数据，不走索引
* ！=或者<>(不等于），可能导致不走索引，也可能走 INDEX FAST FULL SCAN

例如select id  from test where id<>100

* 建立组合索引，但查询谓词并未使用组合索引的第一列，此处有一个INDEX SKIP SCAN概念,
* like '%liu' 百分号在前
* not in ,not exist

### 2数据库索引，底层是怎样实现的，为什么要用乙树索引？

MySQL数据库支持多种索引类型，如BTree索引（下面是对BTree索引的介绍），哈希索引，全文索引等等.

MySQL官方对索引的定义为：索引（Index）是帮助MySQL高效获取数据的数据结构。提取句子主干，就可以得到索引的本质：索引是数据结构。

* MyISAM索引实现：

MyISAM使用B+Tree作为索引结构，叶子结点的data域存放的是数据记录的地址。MyISAM表的主索引和辅助索引在结构上没有任何区别，只是主索引要求key是唯一的，而辅助索引的key可以重复

* InnoDB索引实现

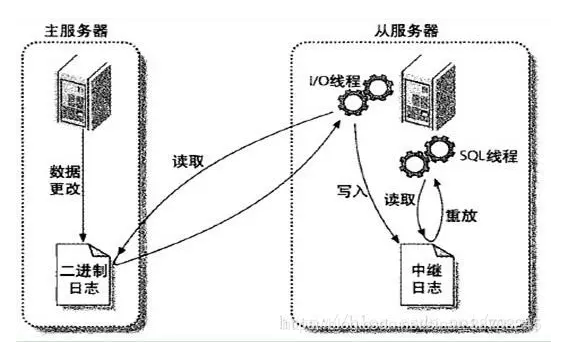
InnoDB的数据文件本身就是索引文件，树的叶子结点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。MyISAM索引的不同是InnoDB的辅助索引data域存储的是相应的主键的值而不是地址。

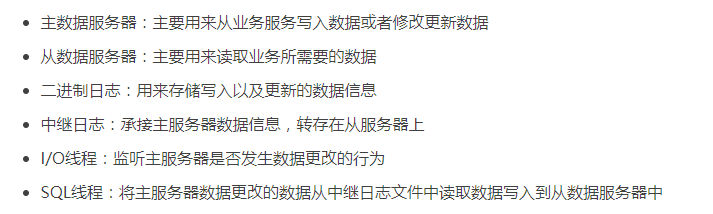
### 3mysql的主从同步的实现原理？

#### 博客上抄下来的

<https://blog.csdn.net/sinat_38798245/article/details/88351160>

#### mysql主从同步的原理





当主数据服务器master进行写入数据或者更新数据操作的时候，数据更改会记录在二进制日志（binary log file）中，主服务器master与从服务器slave进行通讯的是I/O线程，它将修改的数据异步复制写入到slave服务器的中继日志（relay log file）中,从服务器slave与中继日志之间通信使用SQL线程，SQL线程可以异步从中继日志中读取数据后再写入到自己的数据库中，就完成了数据的主从同步功能。

#### 从服务器slave为什么不能直接存储二进制日志文件里面的数据

个人理解的原因是本来做数据的主从同步就是为了让计算机快速的进行读写操作，而且是大批量的数据，一旦大量数据进行写入或者更新数据，从数据库slave如果直接从二进制日志来接收，数据是以队列形式进行传输的，若队列的数据没有快速处理，堆积起来，从服务器可能也会崩溃宕机，所以从性能上考虑，从服务器slave创建了I/O线程对象将数据转到中继日志，起个缓存功能。

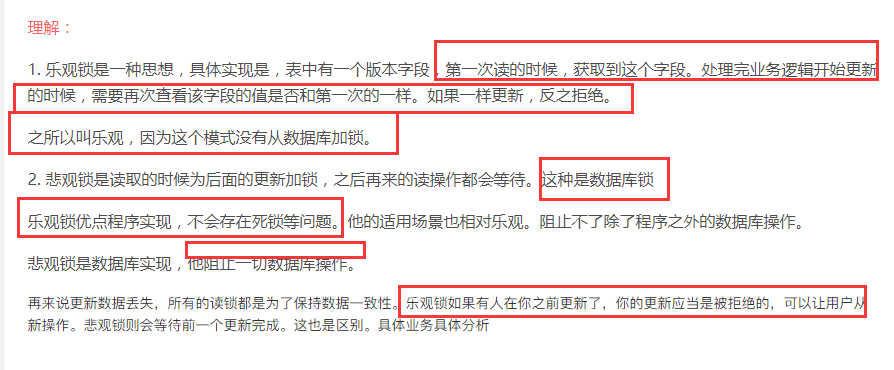
### 4MySQL的是怎么用B +树？

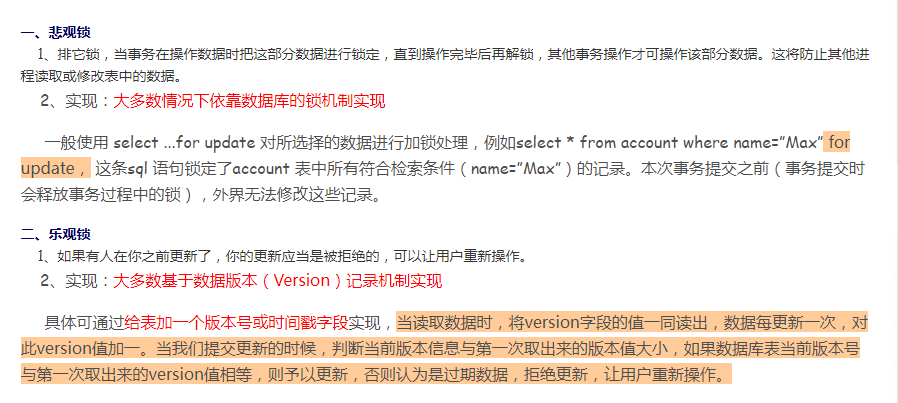
### 5谈谈数据库乐观锁与悲观锁？

数据的锁定分为两种，第一种叫作悲观锁，第二种叫作乐观锁。

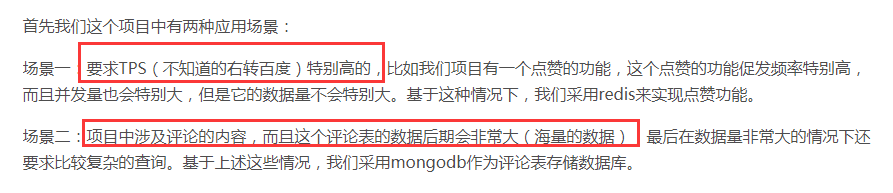
1、悲观锁，就是对数据的冲突采取一种悲观的态度，也就是说假设数据肯定会冲突，所以在数据开始读取的时候就把数据锁定住。【数据锁定：数据将暂时不会得到修改】

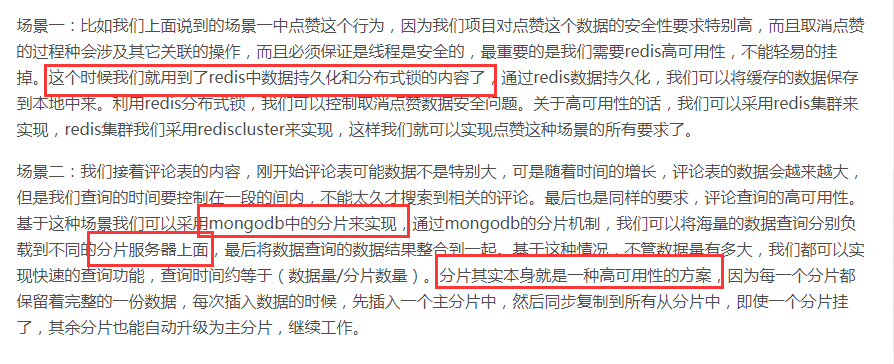
2、乐观锁，认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让用户返回错误的信息。让用户决定如何去做。





6有使用过哪些的NoSQL数据库？MongoDB的和Redis的适用哪些场景？





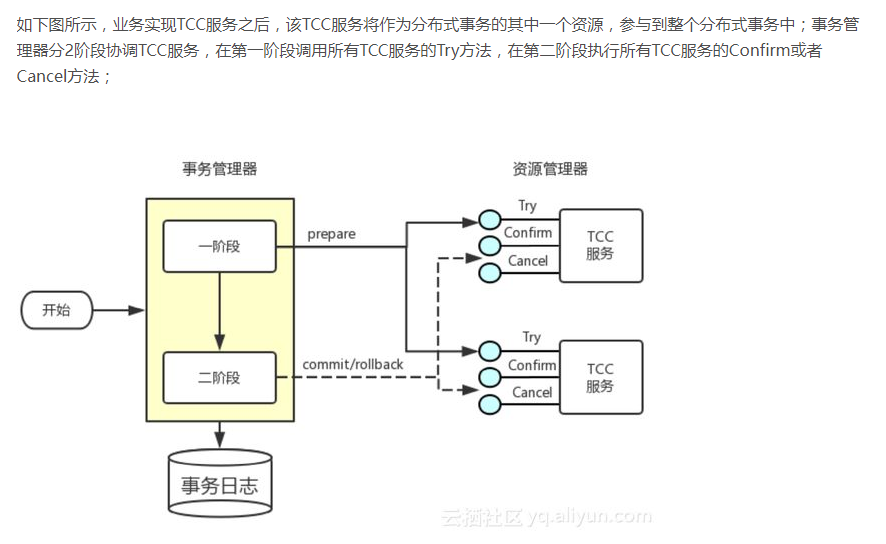
### 7描述分布式事务之TCC服务设计？

<https://blog.csdn.net/eo63y6pKI42Ilxr/article/details/81040048>(没摘抄全)

TCC是一种比较成熟的分布式事务解决方案，可用于解决跨库操作的数据一致性问题.

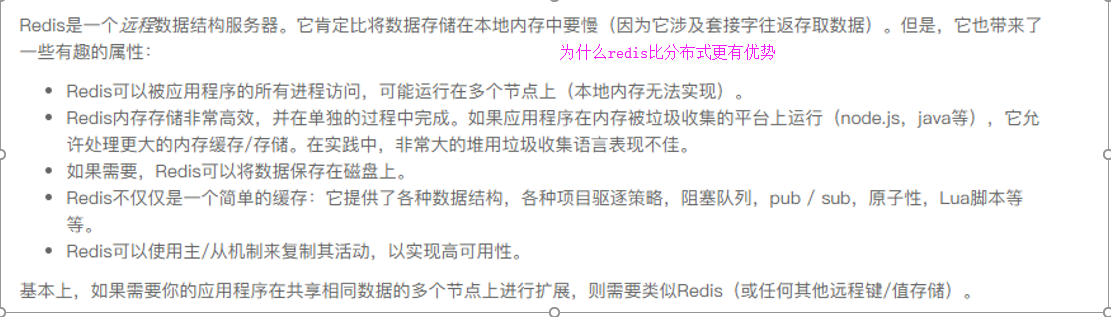
TCC是服务化的两阶段编程模型，其Try、Confirm、Cancel 3个方法均由业务编码实现.

其中Try操作作为一阶段，负责资源的检查和预留，Confirm操作作为二阶段提交操作，执行真正的业务，Cancel是预留资源的取消；



### 8Redis的缓存和内存缓存有什么区别？Redis的为什么比内存缓存有优势？





9考虑Redis的的时候，有没有考虑容量？大概数据量会有多少？

bloom filter

Bloom filter 是由 Howard Bloom 在 1970 年提出的二进制向量数据结构，它具有很好的空间和时间效率，被用来检测一个元素是不是集合中的一个成员。如果检测结果为是，该元素不一定在集合中；但如果检测结果为否，该元素一定不在集合中。因此Bloom filter具有100%的召回率。这样每个检测请求返回有“在集合内（可能错误）”和“不在集合内（绝对不在集合内）”两种情况，可见 Bloom filter 是牺牲了正确率和时间以节省空间。

## 美团一面经验

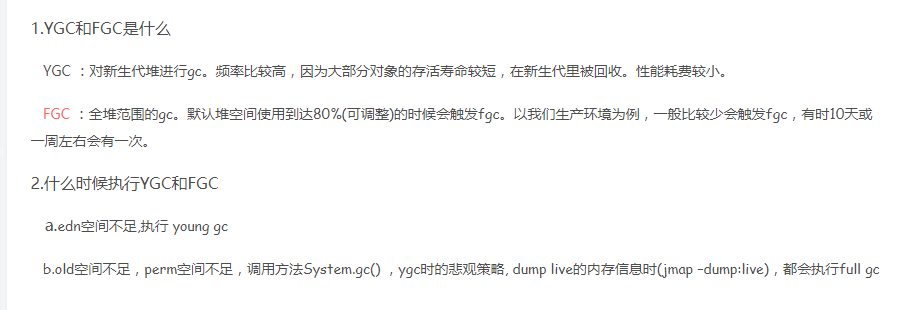
1.最近做的比较熟悉的项目是那个，画一下项目技术架构图。

### 2.JVM老年代和新生代的比例

eden:surivorfrom:surivorto=8:1:1

yonng:old=1:2

### 3YGC和FGC发生的具体场景



### 4jstack,jmap,jstat分别的意义？如何线上排查JVM的相关问题

jstack能得到运行java程序的java stack和native stack的信息。

jmap得到运行java程序的内存分配的详细情况。例如实例个数，大小等 。

jstat这是一个比较实用的一个命令，可以观察到classloader，compiler，gc相关信息。可以时时监控资源和性能

5线程池构造累的方法的5个参数的具体意义

### 6单机上一个线程池正在处理服务如果忽然断点怎么办（正在处理和阻塞队列里的请求怎么处理）

可以通过回溯日志的方式来撤销正在处理的已经执行成功的操作。然后重新执行整个阻塞队列。

阻塞队列持久化，正在处理事物控制。断电之后正在处理的回滚，日志恢复该次操作。服务器重启后阻塞队列中的数据再加载

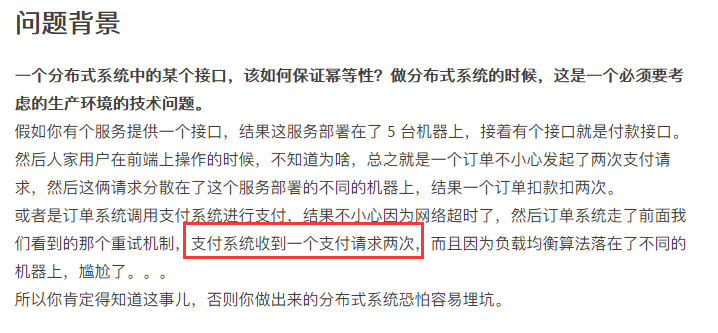
### 7使用无界阻塞队列会出现什么问题

LinkedBlockingQueue默认的最大任务数量是Integer.MAX\_VALUE,非常大，可以理解为无限大吧；但是存在这种情况，当每个线程获取到一个任务后，执行时间比较长，导致workQueue里积压的任务越来越多，机器的内存使用不停的飙升，最后也会导致OOM。

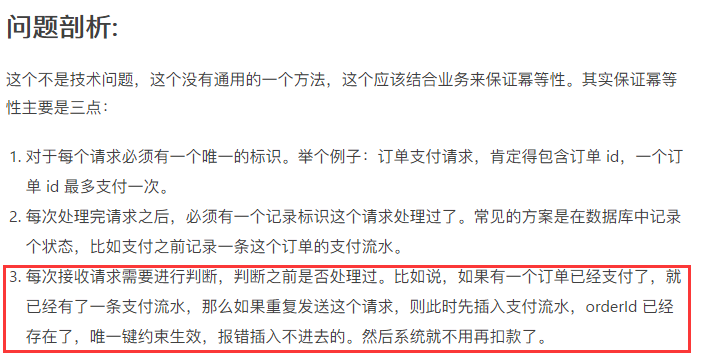
### 8接口如何处理重复的请求

所谓幂等性，就是说一个接口，多次发起同一个请求，你这个接口得保证结果是准确的，比如不能多扣款、不能多插入一条数据、不能将统计值多加了 1。这就是幂等性。分布式服务接口如何设计保证幂等性？

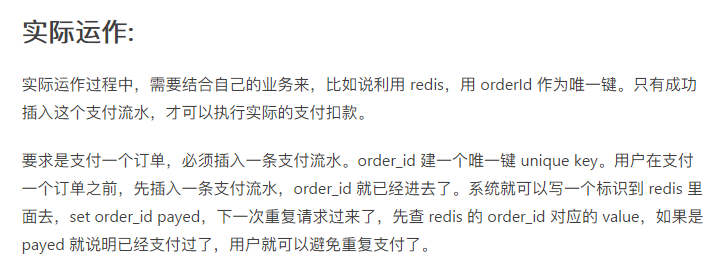
#### 问题背景



#### 问题剖析

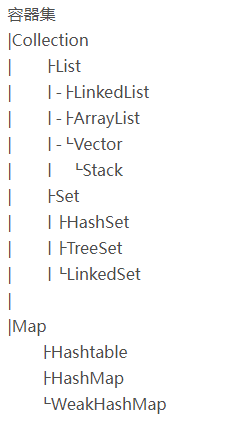


#### 实际操作



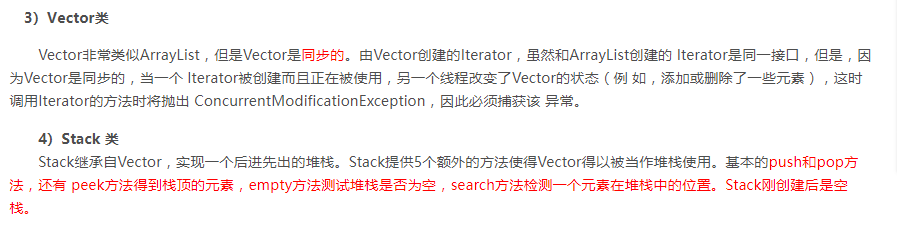
## 面试

### 1介绍下集合框架

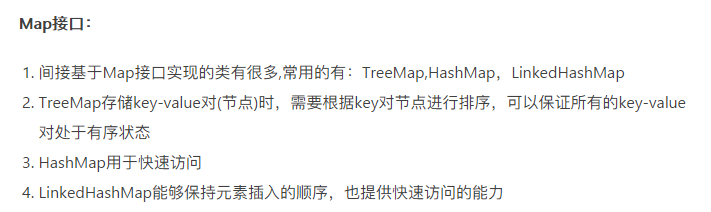


**LinkedList类** ：基于链表的数据结构，允许null元素，增加、删除、修改元素方面效率比ArrayList高。

**ArrayList类** ：基于数组的数据结构，不同步，线程不安全，查询（get set）效率高。



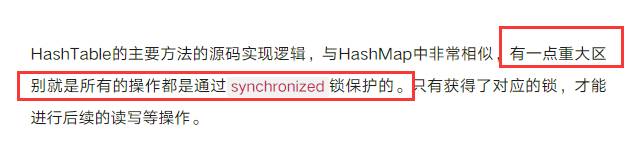




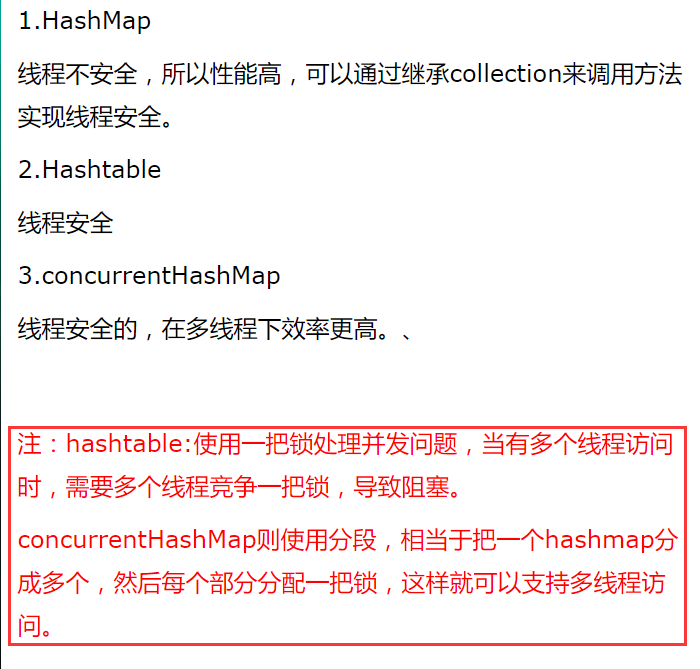
### 2hashmap,hashtable底层实现什么区别？hashtable和concurrenthashtable呢？



#### hashmap与hashtable的区别



#### hashtable与currenthashtabel的区别



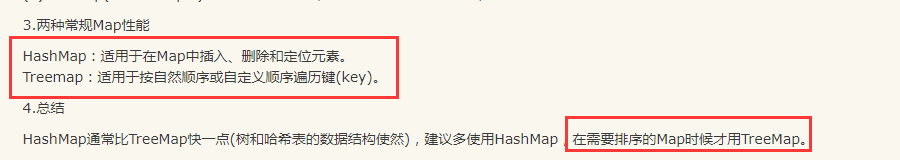
### 3hashmap和treemap什么区别？底层数据结构是什么。

hashmap底层是数组加链表结构，数组用来根据key的hashcode值来定位一个对象应该属于哪一个index，而链表结构可以解决当key的hashcode值相等而equals方法值不相等的问题。，而treemap底层是红黑树结构，红黑树的结构是为了兼顾增删改查的多维度性能而设计的。



HashMap：基于哈希表实现。使用HashMap要求添加的键类明确定义了hashCode()和equals()[可以重写hashCode()和equals()]，为了优化HashMap空间的使用，您可以调优初始容量和负载因子。

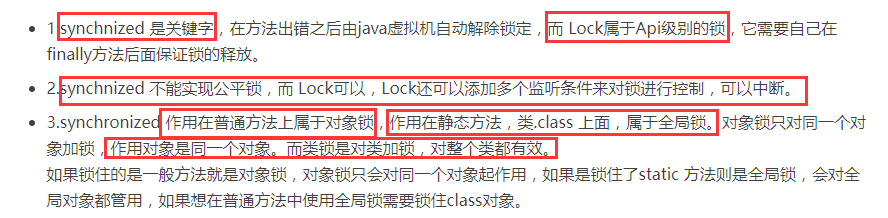
TreeMap：基于红黑树实现。TreeMap没有调优选项，因为该树总处于平衡状态。



### 4线程池用过吗都有什么参数？底层是如何实现的？

<https://blog.csdn.net/zhangliangzi/article/details/52389766>

### 5sychnized和Lock什么区别？sychnize什么情况是对象锁？什么时候是全局锁为什么？



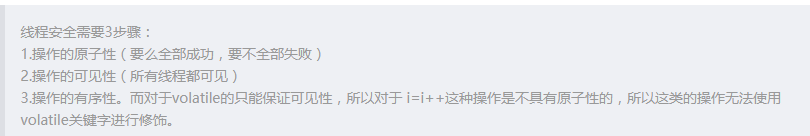
### 6ThreadLocal是什么底层如何实现？写一个例子呗？

ThreadLocal为解决多线程程序的并发问题提供了一种新的思路。使用这个工具类可以很简洁地编写出优美的多线程程序。

当使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。

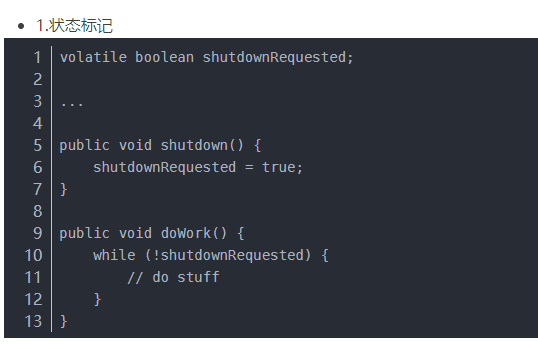
ThreadLocal是如何做到为每一个线程维护变量的副本的呢？其实实现的思路很简单：在ThreadLocal类中有一个Map，用于存储每一个线程的变量副本，Map中元素的键为线程对象，而值对应线程的变量副本。我们自己就可以提供一个简单的实现版本：

### 7volitile的工作原理

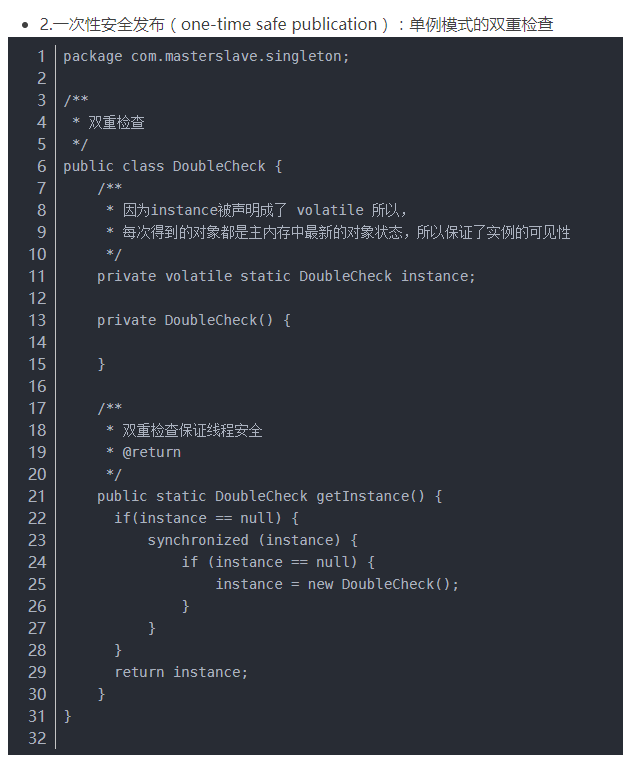


#### violate的使用场景

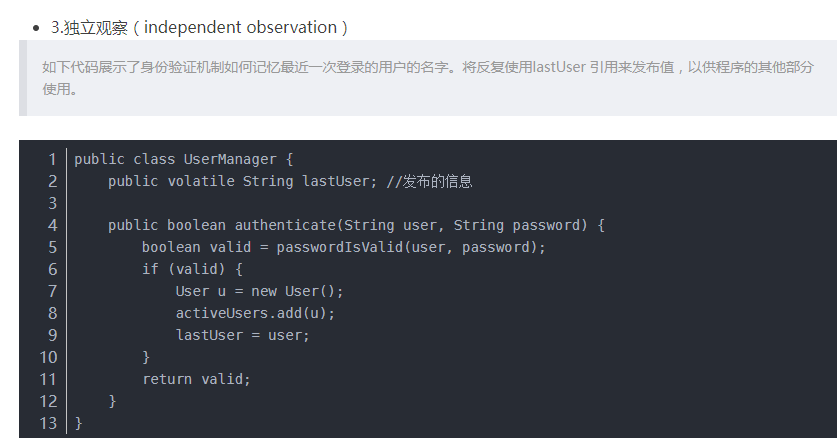
##### 1状态标记



##### 2一次性安全发布



##### 3独立观察



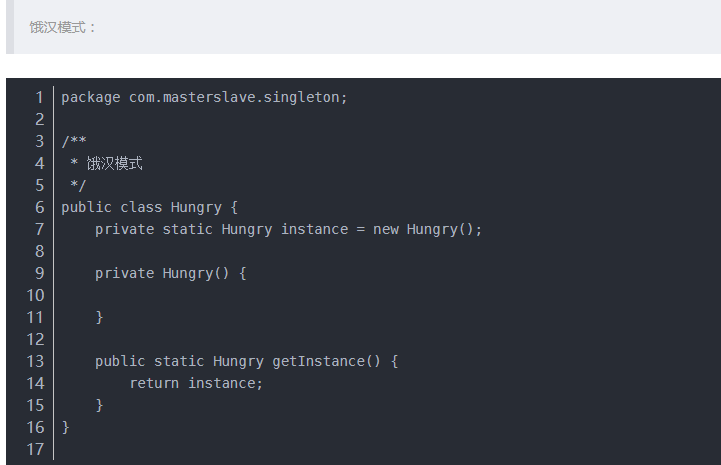
##### 4开销较低的读写锁策略

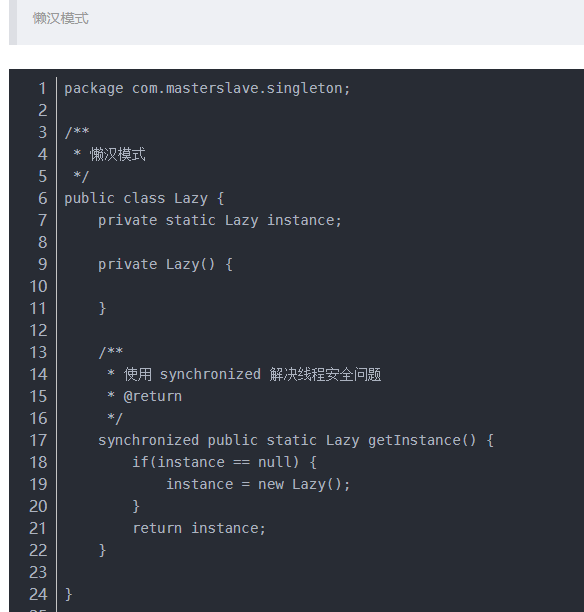


### 8cas知道吗？如何实现的？

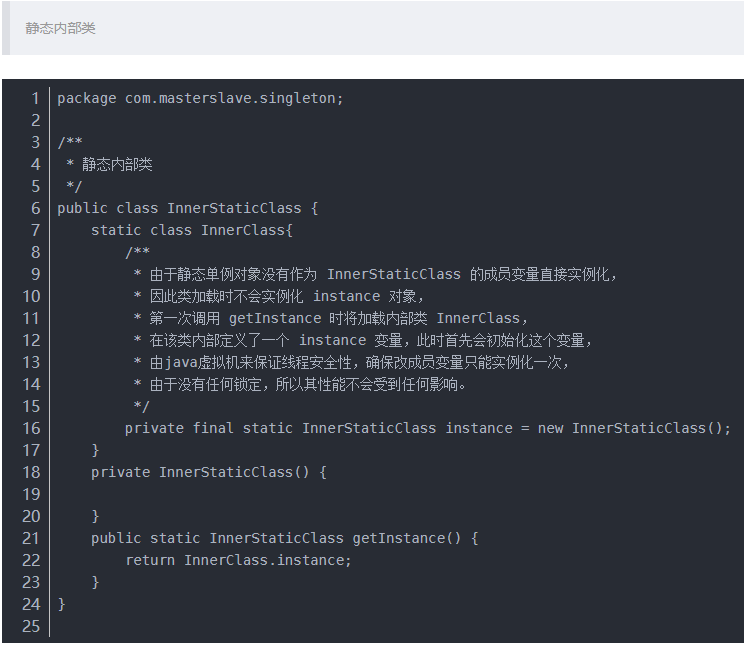
Compare and Swap 比较并刷新到缓存。通过比较线程旧值跟内存中的值是否相等来判断当前的值能否刷新到缓存中。

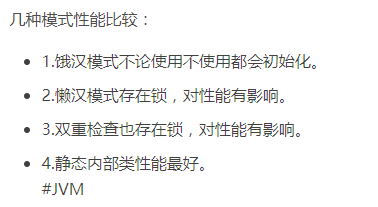
### 9请用至少四种写法写一个单例模式











### 10请介绍一下JVM内存模型？用过什么垃圾回收器都说说呗

jvm包括2方面：

1.1.线程私有区：

1.程序计数器，记录正在执行的虚拟机字节码的地址。

2.虚拟机栈：方法执行的内存区，每个方法执行时会在虚拟机栈中创建栈帧。

3.本地方法栈：虚拟机的Native方法执行的内存区。

1.2.线程共享区：

1.Java堆：对象分配内存的区域；

2.方法区：也称为持久代

3.常量池：存放编译器生成的各种字面量和符号引用，是方法区的一部分。

### 11线上频繁发送full gc如何处理，cpu使用率过高怎么办



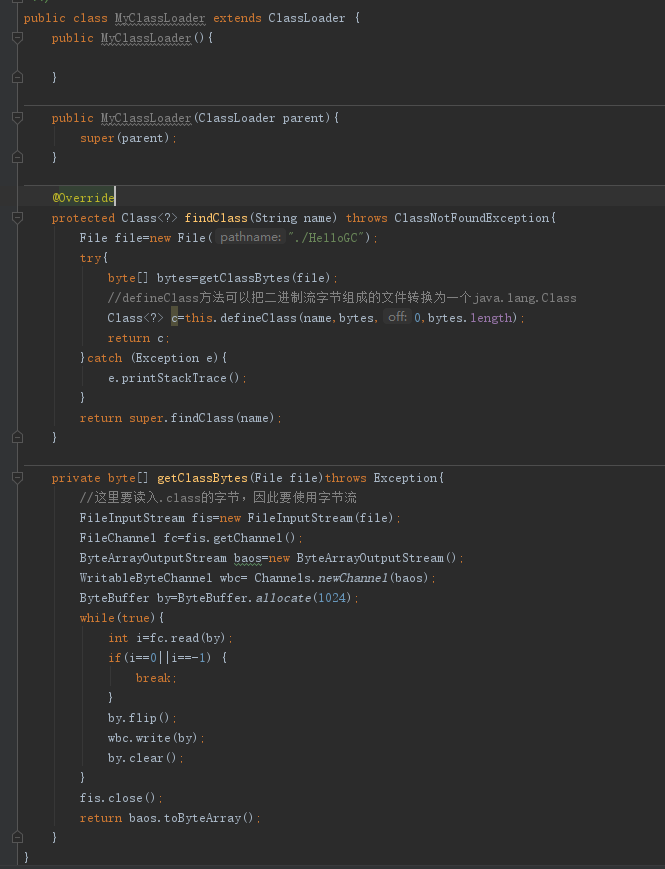
### 12如何定位问题？如何解决说一下解决思路和处理方法

13知道字节码吗？字节码都有那些？Integer x=5,int y=5,比较x=y都经过那些步骤

### 14讲讲类加载机制呗都有那些类加载器，这些类加载器都加载那些文件。



### 15手写一下类加载Demo



### 16知道osgi吗？他是如何实现的？？

OSGi（开放服务网关协议，Open Service Gateway Initiative）

你可以动态地安装、卸载、启动、停止不同的应用模块，而不需要重启容器。

17请问你做过那些JVM优化？使用什么方法达到什么效果

### 18classforName(“java.lang.String”)和String classgetClassLoader() LoadClass(“java.lang.String”)什么区别

Class.forName返回的Class对象可以决定是否初始化。而ClassLoader.loadClass返回的类型绝对不会初始化，最多只会做连接操作

Class.forName可以决定由哪个classLoader来请求这个类型。而ClassLoader.loadClass是用当前的classLoader去请求

# 待完成事件

容器的底层源码

设计模式