# **一、Java 基础**

1.JDK 和 JRE 有什么区别？

2.== 和 equals 的区别是什么？

3.两个对象的 hashCode()相同，则 equals()也一定为 true，对吗？

4.final 在 java 中有什么作用？

5.java 中的 Math.round(-1.5) 等于多少？

6.String 属于基础的数据类型吗？

7.java 中操作字符串都有哪些类？它们之间有什么区别？

## 1. Java中的泛型是什么 ? 使用泛型的好处是什么?

泛型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 ?

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 ?

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>。

## 7. 编写一段泛型程序来实现LRU缓存?

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.String a="i"与 String b=new String(“i”)一样吗？

内存的分配方式不一样。String a="i"的方式，java 虚拟机会将其分配到常量池中；而 String b=new String(“i”) 则会被分到堆内存中。

## 9.String 类的常用方法都有那些？

* indexOf()：返回指定字符的索引。
* charAt()：返回指定索引处的字符。
* replace()：字符串替换。
* trim()：去除字符串两端空白。
* split()：分割字符串，返回一个分割后的字符串数组。
* getBytes()：返回字符串的 byte 类型数组。
* length()：返回字符串长度。
* toLowerCase()：将字符串转成小写字母。
* toUpperCase()：将字符串转成大写字符。
* substring()：截取字符串。
* equals()：字符串比较。

## 10.如何将字符串反转？

使用 StringBuilder 或者 stringBuffer 的 reverse() 方法

11.抽象类必须要有抽象方法吗？

不需要

12.普通类和抽象类有哪些区别？

抽象方法、实例化

13.抽象类能使用 final 修饰吗？

不能，定义抽象类就是让其他类继承的，如果定义为 final 该类就不能被继承

## 14.接口和抽象类有什么区别？

* 实现：抽象类的子类使用 extends 来继承；接口必须使用 implements 来实现接口。
* 构造函数：抽象类可以有构造函数；接口不能有。
* main 方法：抽象类可以有 main 方法，并且我们能运行它；接口不能有 main 方法。
* 实现数量：类可以实现很多个接口；但是只能继承一个抽象类。
* 访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的方法可以是任意访问修饰符

## 15.java 中 IO 流分为几种？

按功能来分：输入流（input）、输出流（output）。

按类型来分：字节流和字符流。

字节流和字符流的区别是：字节流按 8 位传输，以字节为单位输入输出数据；字符流按 16 位传输以字符为单位输入输出数据

## 16.BIO、NIO、AIO 有什么区别？

* BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。
* NIO：New IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。
* AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

17.Files的常用方法都有哪些？

## 18. HashMap和HashTable的区别

相同点：都是存储key-value键值对的

不同点：

* HashMap允许Key-value为null，hashTable不允许；
* hashMap没有考虑同步，是线程不安全的。hashTable是线程安全的，给api套上了一层synchronized修饰;
* HashMap继承于AbstractMap类，hashTable继承与Dictionary类。
* 迭代器(Iterator)。HashMap的迭代器(Iterator)是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。所以当有其它线程改变了HashMap的结构（增加或者移除元素），将会抛出ConcurrentModificationException。
* 容量的初始值和增加方式都不一样：HashMap默认的容量大小是16；增加容量时，每次将容量变为"原始容量x2"。Hashtable默认的容量大小是11；增加容量时,每次将容量变为"原始容量x2 + 1"；
* 添加key-value时的hash值算法不同：HashMap添加元素时，是使用自定义的哈希算法。Hashtable没有自定义哈希算法，而直接采用的key的hashCode()。

## **19. HashTable**

* 底层数组+链表实现，无论key还是value都**不能为null**，线程**安全**，实现线程安全的方式是在修改数据时锁住整个HashTable，效率低，ConcurrentHashMap做了相关优化
* 初始size为**11**，扩容：newsize = olesize\*2+1
* 计算index的方法：index = (hash & 0x7FFFFFFF) % tab.length

## **20. HashMap**

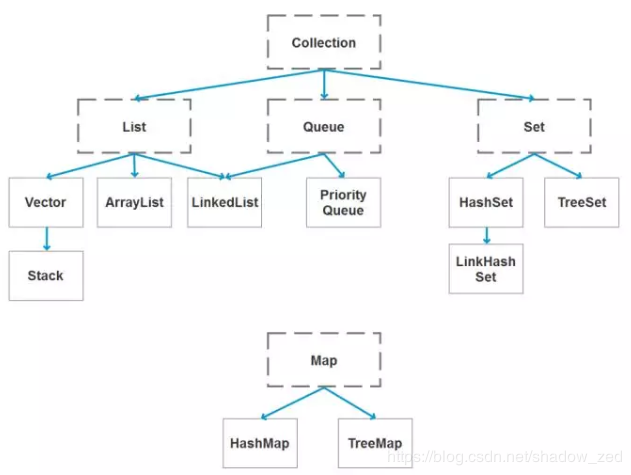
* 底层数组+链表实现，可**以存储null键和null值**，线程**不安全**
* 初始size为**16**，扩容：newsize = oldsize\*2，size一定为2的n次幂
* 扩容针对整个Map，每次扩容时，原来数组中的元素依次重新计算存放位置，并重新插入
* 插入元素后才判断该不该扩容，有可能无效扩容（插入后如果扩容，如果没有再次插入，就会产生无效扩容）
* 当Map中元素总数超过Entry数组的75%，触发扩容操作，为了减少链表长度，元素分配更均匀
* 计算index方法：index = hash & (tab.length – 1)

## 21. ConcurrentHashMap

* 底层采用分段的数组+链表实现，线程**安全**
* 通过把整个Map分为N个Segment，可以提供相同的线程安全，但是效率提升N倍，默认提升16倍。(读操作不加锁，由于HashEntry的value变量是 volatile的，也能保证读取到最新的值。)
* Hashtable的synchronized是针对整张Hash表的，即每次锁住整张表让线程独占，ConcurrentHashMap允许多个修改操作并发进行，其关键在于使用了锁分离技术
* 有些方法需要跨段，比如size()和containsValue()，它们可能需要锁定整个表而而不仅仅是某个段，这需要按顺序锁定所有段，操作完毕后，又按顺序释放所有段的锁
* 扩容：段内扩容（段内元素超过该段对应Entry数组长度的75%触发扩容，不会对整个Map进行扩容），插入前检测需不需要扩容，有效避免无效扩容

# **二、容器**

## 18. Java 容器都有哪些？



## 18.1 ArrayList、LinkedList、Vector的特性及底层实现？

**ArrayList的优点**

1、ArrayList底层以数组实现，是一种随机访问模式，实现了RandomAccess接口，查找元素的时候非常快。

2、ArrayList在顺序添加一个元素的时候非常方便，只是往数组里面添加了一个元素而已。

3、根据下标遍历、访问元素。

4、可以自动扩容，默认为每次扩容为原来的1.5倍。

**ArrayList的缺点**

1、插入和删除元素的效率不高。

2、根据元素下标查找元素需要遍历整个元素数组，效率不高。

3、线程不安全。

**LinkedList**

LinkedList是双向链表的数据结构，底层是用链表实现，是由相互引用的节点组成的双向链表。

**LinkedList的优点**

删除和添加数据所消耗的资源较少，且比ArrayList效率高。

**LinkedList缺点：**

线程不安全，查找消耗的资源大，效率低，不能随机访问。

Vector

1. Vector 是**矢量队列**，继承于AbstractList，实现了List, RandomAccess, Cloneable这些接口。  
   2、Vector 继承了AbstractList，实现了List；所以，**它是一个队列，支持相关的添加、删除、修改、遍历等功能**。  
   3、Vector 实现了RandmoAccess接口，即**提供了随机访问功能**。我们即可以通过元素的序号快速获取元素对象；这就是快速随机访问。  
   4、Vector 实现了Cloneable接口，即实现clone()函数。它能被克隆。

5、**Vector中的操作是线程安全的.**

## 18.2 ArrayList fail—fast的原理？如何避免？

在用迭代器遍历一个集合对象时，如果遍历过程中对集合对象的内容进行了修改（增加、删除、修改），则会抛出ConcurrentModificationException。

          原理：迭代器在遍历时直接访问集合中的内容，并且在遍历过程中使用一个 modCount 变量。集合在被遍历期间如果内容发生变化，就会改变modCount的值。每当迭代器使用hashNext()/next()遍历下一个元素之前，都会检测modCount变量是否为expectedmodCount值，是的话就返回遍历；否则抛出异常，终止遍历。

      注意：这里异常的抛出条件是检测到 modCount！=expectedmodCount 这个条件。如果集合发生变化时修改modCount值刚好又设置为了expectedmodCount值，则异常不会抛出。因此，不能依赖于这个异常是否抛出而进行并发操作的编程，这个异常只建议用于检测并发修改的bug.

使用并发集合类来避免ConcurrentModificationException，比如使用CopyOnArrayList，Collections.synchronizedlist。

19.Collection 和 Collections 有什么区别？

20.List、Set、Map 之间的区别是什么？

21.HashMap 和 Hashtable 有什么区别？

## 22.如何决定使用 HashMap 还是 TreeMap？

对于在Map中插入、删除和定位元素这类操作，HashMap是最好的选择。然而，假如你需要对一个有序的key集合进行遍历，TreeMap是更好的选择。

24.说一下 HashSet 的实现原理？

## 26.如何实现数组和 List 之间的转换？

* List转换成为数组：调用ArrayList的toArray方法。
* 数组转换成为List：调用Arrays的asList方法。

28.Array 和 ArrayList 有何区别？

## 29. 在 Queue 中 poll()和 remove()有什么区别？

poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

## 30. 哪些集合类是线程安全的？

* vector：就比arraylist多了个同步化机制（线程安全）
* statck：堆栈类，先进后出。
* hashtable：就比hashmap多了个线程安全。
* enumeration：枚举，相当于迭代器。

31.迭代器 Iterator 是什么？

32.Iterator 怎么使用？有什么特点？

33.Iterator 和 ListIterator 有什么区别？

## 34. 怎么确保一个集合不能被修改？

Collections.unmodifiableMap(Map)  
Collections.unmodifiableList(List)  
Collections.unmodifiableSet(Set)

java.lang.UnsupportedOperationException

# **二(1)、HashMap**

## 1. HashMap 的实现原理？

HashMap基于hashing原理，它的数据结构为散列表.jdk8后采用数组+链表+红黑树的数据结构。我们通过put和get存储和获取对象。当我们给put()方法传递键和值时，先对键做一个hashCode()的计算来得到它在bucket数组中的位置来存储Entry对象。当获取对象时，通过get获取到bucket的位置，再通过键对象的equals()方法找到正确的键值对，然后在返回值对象。

链表的阈值是8，超过8就要转换成红黑树。Index计算：计算出hash code, 然后tab[i = (table长度 - 1) & hash]

## 2. 如何解决Hash冲突？

当两个对象的hashcode相等，就会发生hash冲突，也就是哈希碰撞。

hash函数的返回值是一个int类型的数据，int的取值是有范围的，而散列表的key是没有范围的，可以是任何值。将多数key映射到少数hashCode，必然会有多个key对应同一个hashCode的情况。

HashMap使用链表法（chaining）解决冲突。在jdk1.8以前HashMap的实现是**散列表 = 数组 + 链表，**如果发现下标冲突，则**后插入的节点以链表的形式追加到前一个节点的后面**。

由于冲突的节点会不停的在链表上追加，大量的冲突会导致单个链表过长，使查询性能降低。所以一个好的散列表的实现应该从源头上减少冲突发生的可能性，冲突发生的概率和哈希函数返回值的均匀程度有直接关系，得到的哈希值越均匀，冲突发生的可能性越小。为了使哈希值更均匀，HashMap内部的hash()函数对key的hashCode做hash操作，与高16位做异或运算。

**下标冲突**：

1、散列表的做法是对key进行hash计算，获得一个int类型的hash值.

2、对上面得到的hash值进行计算，获得一个不超过数组大小的下标（**tab[i = (table长度 - 1) & hash]**）

与哈希的碰撞类似，在将hashCode映射到数组下标时也是有可能重复的。在往数组的某个下标插入节点的时候发现该下标已经有其他节点，即为下标冲突。

**举个例子**，现在假设有节点B要添加到表中，那么遇到冲突后的整个流程是这样的：  
1、计算得到B.key对应的下标为 n  
2、判断 n 下标下是否有值  
3、发现有值，下标冲突，获取到该下标已有的节点A  
4、判断(B.key).equals(A.key)  
5、如果key相等，根据key唯一的特性，B节点覆盖A节点  
6、如果key不等，判断A.next是否有值  
7、如果A.next也有值，回到步骤4对A.next节点进行对比  
8、重复步骤4-7，直到发现key相等的节点，或遍历到链表的最后一个节点X，执行操作X.next = B  
Ps：读取节点的流程与添加类似，先计算下标，然后遍历链表，直到找到对应的key

**hashCode发生碰撞，则下标一定冲突；而下标冲突，hashCode并不一定碰撞。**

## 3. 传统HashMap的缺点(为什么引入红黑树？)

JDK 1.8 以前 HashMap 的实现是 数组+链表，即使哈希函数取得再好，也很难达到元素百分百均匀分布。当 HashMap 中有大量的元素都存放到同一个桶中时，这个桶下有一条长长的链表，这个时候 HashMap 就相当于一个单链表，假如单链表有 n 个元素，遍历的时间复杂度就是 O(n)，完全失去了它的优势。针对这种情况，JDK 1.8 中引入了 红黑树（查找时间复杂度为 O(logn),当链表长度超过8就自动转为红黑树）来优化这个问题。

## 3. HashMap中put是如何实现的？

1.计算关于key的hashcode值（与Key.hashCode的高16位做异或运算）

2.如果散列表为空时，调用resize()初始化散列表

3.如果没有发生碰撞，直接添加元素到散列表中去

4.如果发生了碰撞(hashCode值相同)，进行三种判断

    4.1:若key地址相同或者equals后内容相同，则替换旧值

    4.2:如果是红黑树结构，就调用树的插入方法

    4.3：链表结构，循环遍历直到链表中某个节点为空，尾插法进行插入，插入之后判断链表个数是否到达变成红黑树的阙值8；也可以遍历到有节点与插入元素的哈希值和内容相同，进行覆盖。

5.如果桶满了大于阀值，则resize进行扩容

## 4. HashMap中什么时候需要进行扩容? 扩容resize()又是如何实现的？

**调用场景：**

**1.初始化数组table**

**2.当数组table的size达到阙值时即++size > load factor \* capacity 时，也是在putVal函数中**

**实现过程：(细讲)**

**1.通过判断旧数组的容量是否大于0来判断数组是否初始化过**

**否：进行初始化**

* **判断是否调用无参构造器，**
  + **是:使用默认的大小和阙值**
  + **否:使用构造函数中初始化的容量，当然这个容量是经过tableSizefor计算后的2的次幂数**

**是，进行扩容，扩容成两倍(小于最大值的情况下)，之后在进行将元素重新进行与运算复制到新的散列表中**.

**概括的讲：扩容需要重新分配一个新数组，新数组是老数组的2倍长，然后遍历整个老结构，把所有的元素挨个重新hash分配到新结构中去。**

**PS：可见底层数据结构用到了数组，到最后会因为容量问题都需要进行扩容操作**

## 5. HashMap中get是如何实现的？

**对key的hashCode进行hashing，与运算计算下标获取bucket位置，如果在桶的首位上就可以找到就直接返回，否则在树中找或者链表中遍历找，如果有hash冲突，则利用equals方法去遍历链表查找节点。**

## 6. 为什么不直接将key作为哈希值而是与高16位做异或运算？

**因为数组位置的确定用的是与运算，仅仅最后四位有效，设计者将key的哈希值与高16为做异或运算使得在做&运算确定数组的插入位置时，此时的低位实际是高位与低位的结合，增加了随机性，减少了哈希碰撞的次数。**

HashMap默认初始化长度为16，并且每次自动扩展或者是手动初始化容量时，必须是2的幂。

## 7. 为什么是16？为什么必须是2的幂？如果输入值不是2的幂比如10会怎么样？

<https://blog.csdn.net/sidihuo/article/details/78489820>

<https://blog.csdn.net/eaphyy/article/details/84386313>

**1.为了数据的均匀分布，减少哈希碰撞。因为确定数组位置是用的位运算，若数据不是2的次幂则会增加哈希碰撞的次数和浪费数组空间。(PS:其实若不考虑效率，求余也可以就不用位运算了也不用长度必需为2的幂次)**

**2.输入数据若不是2的幂，HashMap通过一通位移运算和或运算得到的肯定是2的幂次数，并且是离那个数最近的数字**。

## 8. 如果两个键的hashcode相同，你如何获取值对象？

**HashCode相同，通过equals比较内容获取值对象.**

## 9. HashMap的参数loadFactor，它的作用是什么？

loadFactor表示HashMap的拥挤程度，影响hash操作到同一个数组位置的概率。默认loadFactor等于0.75，当HashMap里面容纳的元素已经达到HashMap数组长度的75%时，表示HashMap太挤了，需要扩容，在HashMap的构造器中可以定制loadFactor。

## 10. 如果HashMap的大小超过了负载因子(load factor)定义的容量，怎么办？

**超过阙值会进行扩容操作，概括的讲就是扩容后的数组大小是原数组的2倍，将原来的元素重新hashing放入到新的散列表中去。**

## 11. 平时在使用HashMap时一般使用什么类型的元素作为Key？

选择Integer，String这种不可变的类型，像对String的一切操作都是新建一个String对象，对新的对象进行拼接分割等，这些类已经很规范的覆写了hashCode()以及equals()方法。作为不可变类天生是线程安全的，

# **三、多线程**

35.并行和并发有什么区别？

36.线程和进程的区别？

## 37.守护线程是什么？

守护线程（即daemon thread），是个服务线程，准确地来说就是服务其他的线程

## 38.创建线程有哪几种方式？

①. 继承Thread类创建线程类

* 定义Thread类的子类，并重写该类的run方法，该run方法的方法体就代表了线程要完成的任务。因此把run()方法称为执行体。
* 创建Thread子类的实例，即创建了线程对象。
* 调用线程对象的start()方法来启动该线程。

②. 通过Runnable接口创建线程类

* 定义runnable接口的实现类，并重写该接口的run()方法，该run()方法的方法体同样是该线程的线程执行体。
* 创建 Runnable实现类的实例，并依此实例作为Thread的target来创建Thread对象，该Thread对象才是真正的线程对象。
* 调用线程对象的start()方法来启动该线程。

③. 通过Callable和Future创建线程

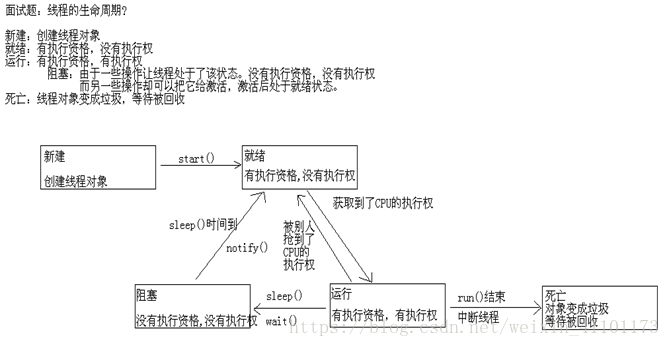
* 创建Callable接口的实现类，并实现call()方法，该call()方法将作为线程执行体，并且有返回值。
* 创建Callable实现类的实例，使用FutureTask类来包装Callable对象，该FutureTask对象封装了该Callable对象的call()方法的返回值。
* 使用FutureTask对象作为Thread对象的target创建并启动新线程。
* 调用FutureTask对象的get()方法来获得子线程执行结束后的返回值。

## 39.说一下 runnable 和 callable 有什么区别？

* 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。

## 40.线程有哪些状态？

线程通常都有五种状态，创建、就绪、运行、阻塞和死亡。



* 创建状态。在生成线程对象，并没有调用该对象的start方法，这是线程处于创建状态。
* 就绪状态。当调用了线程对象的start方法之后，该线程就进入了就绪状态，但是此时线程调度程序还没有把该线程设置为当前线程，此时处于就绪状态。在线程运行之后，从等待或者睡眠中回来之后，也会处于就绪状态。
* 运行状态。线程调度程序将处于就绪状态的线程设置为当前线程，此时线程就进入了运行状态，开始运行run函数当中的代码。
* 阻塞状态。线程正在运行的时候，被暂停，通常是为了等待某个时间的发生(比如说某项资源就绪)之后再继续运行。sleep,suspend，wait等方法都可以导致线程阻塞。
* 死亡状态。如果一个线程的run方法执行结束或者调用stop方法后，该线程就会死亡。对于已经死亡的线程，无法再使用start方法令其进入就绪

## 41.sleep() 和 wait() 有什么区别？

①**sleep()释放CPU执行权，但不释放同步锁**；sleep() 是static静态的方法，他不能改变对象的机锁，当一个synchronized块中调用了sleep() 方法，线程虽然进入休眠，但是对象的机锁没有被释放，其他线程依然无法访问这个对象。

②wait()是Object类的方法,**wait()释放CPU执行权，也释放同步锁，使得其他线程可以使用同步控制块或者方法**。如果wait(param),则等同sleep(param), 如果无参只用通过notify()，notifyAll()方法来“唤醒”.

## 42.notify()和 notifyAll()有什么区别？

* 如果线程调用了对象的 wait()方法，那么线程便会处于该对象的等待池中，等待池中的线程不会去竞争该对象的锁。
* 当有线程调用了对象的 notifyAll()方法（唤醒所有 wait 线程）或 notify()方法（只随机唤醒一个 wait 线程），被唤醒的的线程便会进入该对象的锁池中，锁池中的线程会去竞争该对象锁。也就是说，调用了notify后只要一个线程会由等待池进入锁池，而notifyAll会将该对象等待池内的所有线程移动到锁池中，等待锁竞争。
* 优先级高的线程竞争到对象锁的概率大，假若某线程没有竞争到该对象锁，它还会留在锁池中，唯有线程再次调用 wait()方法，它才会重新回到等待池中。而竞争到对象锁的线程则继续往下执行，直到执行完了 synchronized 代码块，它会释放掉该对象锁，这时锁池中的线程会继续竞争该对象锁。

43.线程的 run()和 start()有什么区别？

## 44.创建线程池有哪几种方式？

①. newFixedThreadPool(int nThreads)

创建一个固定长度的线程池，每当提交一个任务就创建一个线程，直到达到线程池的最大数量，这时线程规模将不再变化，当线程发生未预期的错误而结束时，线程池会补充一个新的线程。

②. newCachedThreadPool()

创建一个可缓存的线程池，如果线程池的规模超过了处理需求，将自动回收空闲线程，而当需求增加时，则可以自动添加新线程，线程池的规模不存在任何限制。

③. newSingleThreadExecutor()

这是一个单线程的Executor，它创建单个工作线程来执行任务，如果这个线程异常结束，会创建一个新的来替代它；它的特点是能确保依照任务在队列中的顺序来串行执行。

④. newScheduledThreadPool(int corePoolSize)

创建了一个固定长度的线程池，而且以延迟或定时的方式来执行任务，类似于Timer。

45.线程池都有哪些状态？

## 46.线程池中 submit()和 execute()方法有什么区别？

* submit有返回值，而execute没有
* submit方便Exception处理

## 47.在 java 程序中怎么保证多线程的运行安全？

线程安全在三个方面体现：

* 原子性：提供互斥访问，同一时刻只能有一个线程对数据进行操作，（atomic,synchronized）；
* 可见性：一个线程对主内存的修改可以及时地被其他线程看到，（synchronized,volatile）；
* 有序性：一个线程观察其他线程中的指令执行顺序，由于指令重排序，该观察结果一般杂乱无序，（happens-before原则）。

48.多线程锁的升级原理是什么？

## 49.什么是死锁？

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。

## 50.怎么防止死锁？

死锁的四个必要条件：

* 互斥条件：进程对所分配到的资源不允许其他进程进行访问，若其他进程访问该资源，只能等待，直至占有该资源的进程使用完成后释放该资源
* 请求和保持条件：进程获得一定的资源之后，又对其他资源发出请求，但是该资源可能被其他进程占有，此事请求阻塞，但又对自己获得的资源保持不放
* 不可剥夺条件：是指进程已获得的资源，在未完成使用之前，不可被剥夺，只能在使用完后自己释放
* 环路等待条件：是指进程发生死锁后，若干进程之间形成一种头尾相接的循环等待资源关系

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之 一不满足，就不会发生死锁。

理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和 解除死锁。

所以，在系统设计、进程调度等方面注意如何不让这四个必要条件成立，如何确 定资源的合理分配算法，避免进程永久占据系统资源。

此外，也要防止进程在处于等待状态的情况下占用资源。因此，对资源的分配要给予合理的规划。

## 51.ThreadLocal 是什么？有哪些使用场景？

线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。Java提供ThreadLocal类来支持线程局部变量，是一种实现线程安全的方式。但是在管理环境下（如 web 服务器）使用线程局部变量的时候要特别小心，在这种情况下，工作线程的生命周期比任何应用变量的生命周期都要长。任何线程局部变量一旦在工作完成后没有释放，Java 应用就存在内存泄露的风险。

## 52.说一下 synchronized 底层实现原理？

synchronized可以保证方法或者代码块在运行时，同一时刻只有一个方法可以进入到临界区，同时它还可以保证共享变量的内存可见性。

Java中每一个对象都可以作为锁，这是synchronized实现同步的基础：

* 普通同步方法，锁是当前实例对象
* 静态同步方法，锁是当前类的class对象
* 同步方法块，锁是括号里面的对象

## 53.synchronized 和 volatile 的区别是什么？

* volatile本质是在告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取； synchronized则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。
* volatile仅能使用在变量级别；synchronized则可以使用在变量、方法、和类级别的。
* volatile仅能实现变量的修改可见性，不能保证原子性；而synchronized则可以保证变量的修改可见性和原子性。
* volatile不会造成线程的阻塞；synchronized可能会造成线程的阻塞。
* volatile标记的变量不会被编译器优化；synchronized标记的变量可以被编译器优化。

## 54.synchronized 和 Lock 有什么区别？

* 首先synchronized是java内置关键字，在jvm层面，Lock是个java类；
* synchronized无法判断是否获取锁的状态，Lock可以判断是否获取到锁；
* synchronized会自动释放锁(a 线程执行完同步代码会释放锁 ；b 线程执行过程中发生异常会释放锁)，Lock需在finally中手工释放锁（unlock()方法释放锁），否则容易造成线程死锁；
* 用synchronized关键字的两个线程1和线程2，如果当前线程1获得锁，线程2线程等待。如果线程1阻塞，线程2则会一直等待下去，而Lock锁就不一定会等待下去，如果尝试获取不到锁，线程可以不用一直等待就结束了；
* synchronized的锁可重入、不可中断、非公平，而Lock锁可重入、可判断、可公平（两者皆可）；
* Lock锁适合大量同步的代码的同步问题，synchronized锁适合代码少量的同步问题。

## 55.synchronized 和 ReentrantLock 区别是什么？

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：

* ReentrantLock可以对获取锁的等待时间进行设置，这样就避免了死锁
* ReentrantLock可以获取各种锁的信息
* 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。

## 56.说一下 atomic 的原理？

Atomic包中的类基本的特性就是在多线程环境下，当有多个线程同时对单个（包括基本类型及引用类型）变量进行操作时，具有排他性，即当多个线程同时对该变量的值进行更新时，仅有一个线程能成功，而未成功的线程可以向自旋锁一样，继续尝试，一直等到执行成功。

Atomic系列的类中的核心方法都会调用unsafe类中的几个本地方法。我们需要先知道一个东西就是Unsafe类，全名为：sun.misc.Unsafe，这个类包含了大量的对C代码的操作，包括很多直接内存分配以及原子操作的调用，而它之所以标记为非安全的，是告诉你这个里面大量的方法调用都会存在安全隐患，需要小心使用，否则会导致严重的后果，例如在通过unsafe分配内存的时候，如果自己指定某些区域可能会导致一些类似C++一样的指针越界到其他进程的问题

## 57.如果你提交任务时，线程池队列已满。会时发会生什么？

这个问题问得很狡猾，许多程序员会认为该任务会阻塞直到线程池队列有空位。事实上如果一个任务不能被调度执行，那么ThreadPoolExecutor’s submit()方法将会抛出一个RejectedExecutionException异常。

## 58. Java如何实现一个线程,等待另外N个线程完成某个事情之后才能执行？(CountdownLatch ，CyclicBarrier)

CyclicBarrier可以重复使用已经通过的障碍，而CountdownLatch不能重复使用。

**CountdownLatch**一个线程(或者多个)，等待另外N个线程完成某个事情之后才能执行。是并发包中提供的一个可用于控制多个线程同时开始某个动作的类，其采用的方法为减少计数的方式，当计数减至零时位于latch.Await()后的代码才会被执行，CountDownLatch是减计数方式，计数==0时释放所有等待的线程；CountDownLatch当计数到0时，计数无法被重置；

**CyclicBarrier**字面意思回环栅栏，通过它可以实现让一组线程等待至某个状态之后再全部同时执行。叫做回环是因为当所有等待线程都被释放以后，CyclicBarrier可以被重用。 即:N个线程相互等待，任何一个线程完成之前，所有的线程都必须等待。CyclicBarrier是当await的数量到达了设置的数量的时候，才会继续往下面执行，CyclicBarrier计数达到指定值时，计数置为0重新开始。

# **四、反射**

## 57.什么是反射？

反射主要是指程序可以访问、检测和修改它本身状态或行为的一种能力。

Java反射机制主要提供了以下功能：

* 在运行时判断任意一个对象所属的类。
* 在运行时构造任意一个类的对象。
* 在运行时判断任意一个类所具有的成员变量和方法。
* 在运行时调用任意一个对象的方法。

## 58.什么是 java 序列化？

简单说就是为了保存在内存中的各种对象的状态（也就是实例变量，不是方法），并且可以把保存的对象状态再读出来。

## 58.1 Java如何实现序列化？

只有实现了Serializable或Externalizable接口的类的对象才能被序列化。Externalizable接口继承自Serializable接口，实现Externalizable接口的类完全由自身来控制序列化的行为，而仅实现Serializable接口的类可以采用默认的序列化方式。

## 58.2 什么情况下需要序列化？

a）当你想把的内存中的对象状态保存到一个文件中或者数据库中时候；  
b）当你想用套接字在网络上传送对象的时候；  
c）当你想通过RMI传输对象的时候；

## 58.3 有些字段不需要序列化，怎么实现？

transient

## 59.动态代理是什么？

当想要给实现了某个接口的类中的方法，加一些额外的处理。比如说加日志，加事务等。可以给这个类创建一个代理，故名思议就是创建一个新的类，这个类不仅包含原来类方法的功能，而且还在原来的基础上添加了额外处理的新类。这个代理类并不是定义好的，是动态生成的。具有解耦意义，灵活，扩展性强。

有哪些应用？

* Spring的AOP
* 加事务
* 加权限
* 加日志

## 60.怎么实现动态代理？

常用的动态代理实现方式有两种，一种是利用JDK反射机制生成代理，另外一种是使用CGLIB代理。JDK代理必须要提供接口，而CGLIB则不需要，可以直接代理类

implements InvocationHandler

\* Proxy.newProxyInstance(ClassLoader loader, Class<?>[] interfaces,InvocationHandler h) 可以创建动态代理对象.

三个参数： 1.ClassLoader 类加载器， 2.该代理类实现的接口列表(不是类或抽象类) 3.InvocationHandler接口的一个实现

# **五、对象拷贝**

## 61.为什么要使用克隆？

想对一个对象进行处理，又想保留原有的数据进行接下来的操作，就需要克隆了，Java语言中克隆针对的是类的实例。

## 62.如何实现对象克隆？

有两种方式：

* 实现Cloneable接口并重写Object类中的clone()方法；
* 实现Serializable接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆.

注意：基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化，这项检查是编译器完成的，不是在运行时抛出异常，这种是方案明显优于使用Object类的clone方法克隆对象。让问题在编译的时候暴露出来总是好过把问题留到运行时。

## 63.深拷贝和浅拷贝区别是什么？

* 浅拷贝只是复制了对象的引用地址，两个对象指向同一个内存地址，所以修改其中任意的值，另一个值都会随之变化，这就是浅拷贝（例：assign()）
* 深拷贝是将对象及值复制过来，两个对象修改其中任意的值另一个值不会改变，这就是深拷贝（例：JSON.parse()和JSON.stringify()，但是此方法无法复制函数类型）

# **六、Java Web**

## 64.jsp 和 servlet 有什么区别？

1. jsp经编译后就变成了Servlet.（JSP的本质就是Servlet，JVM只能识别java的类，不能识别JSP的代码，Web容器将JSP的代码编译成JVM能够识别的java类）
2. jsp更擅长表现于页面显示，servlet更擅长于逻辑控制。
3. Servlet中没有内置对象，Jsp中的内置对象都是必须通过HttpServletRequest对象，HttpServletResponse对象以及HttpServlet对象得到。
4. Jsp是Servlet的一种简化，使用Jsp只需要完成程序员需要输出到客户端的内容，Jsp中的Java脚本如何镶嵌到一个类中，由Jsp容器完成。而Servlet则是个完整的Java类，这个类的Service方法用于生成对客户端的响应。

## 65.jsp 有哪些内置对象？作用分别是什么？

JSP有9个内置对象：

* request：封装客户端的请求，其中包含来自GET或POST请求的参数；
* response：封装服务器对客户端的响应；
* pageContext：通过该对象可以获取其他对象；
* session：封装用户会话的对象；
* application：封装服务器运行环境的对象；
* out：输出服务器响应的输出流对象；
* config：Web应用的配置对象；
* page：JSP页面本身（相当于Java程序中的this）；
* exception：封装页面抛出异常的对象。

## 66.说一下 jsp 的 4 种作用域？

JSP中的四种作用域包括page、request、session和application，具体来说：

* **page**代表与一个页面相关的对象和属性。
* **request**代表与Web客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个Web组件；需要在页面显示的临时数据可以置于此作用域。
* **session**代表与某个用户与服务器建立的一次会话相关的对象和属性。跟某个用户相关的数据应该放在用户自己的session中。
* **application**代表与整个Web应用程序相关的对象和属性，它实质上是跨越整个Web应用程序，包括多个页面、请求和会话的一个全局作用域。

## 67.session 和 cookie 有什么区别？

* 由于HTTP协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是Session. Session是保存在服务端的，有一个唯一标识。在服务端保存Session的方法很多，内存、数据库、文件都有。集群的时候也要考虑Session的转移，在大型的网站，一般会有专门的Session服务器集群，用来保存用户会话，这个时候 Session 信息都是放在内存的，使用一些缓存服务比如Memcached之类的来放 Session。
* HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用 Cookie 来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在 Cookie 里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了。有人问，如果客户端的浏览器禁用了 Cookie 怎么办？一般这种情况下，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户。
* Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式。

## 68.说一下 session 的工作原理？

其实session是一个存在服务器上的类似于一个散列表格的文件。里面存有我们需要的信息，在我们需要用的时候可以从里面取出来。类似于一个大号的map吧，里面的键存储的是用户的sessionid，用户向服务器发送请求的时候会带上这个sessionid。这时就可以从中取出对应的值了

69.如果客户端禁止 cookie 能实现 session 还能用吗？

70.spring mvc 和 struts 的区别是什么？

## 71.如何避免 sql 注入？

1. PreparedStatement（简单又有效的方法）
2. 使用正则表达式过滤传入的参数
3. 字符串过滤
4. JSP中调用该函数检查是否包函非法字符
5. JSP页面判断代码

## 72.什么是 XSS 攻击，如何避免？

XSS攻击又称CSS,全称Cross Site Script  （跨站脚本攻击），其原理是攻击者向有XSS漏洞的网站中输入恶意的 HTML 代码，当用户浏览该网站时，这段 HTML 代码会自动执行，从而达到攻击的目的。XSS 攻击类似于 SQL 注入攻击，SQL注入攻击中以SQL语句作为用户输入，从而达到查询/修改/删除数据的目的，而在xss攻击中，通过插入恶意脚本，实现对用户游览器的控制，获取用户的一些信息。 XSS是 Web 程序中常见的漏洞，XSS 属于被动式且用于客户端的攻击方式。

XSS防范的总体思路是：对输入(和URL参数)进行过滤，对输出进行编码。

## 73.什么是 CSRF 攻击，如何避免？

CSRF（Cross-site request forgery）也被称为 one-click attack或者 session riding，中文全称是叫**跨站请求伪造**。一般来说，攻击者通过伪造用户的浏览器的请求，向访问一个用户自己曾经认证访问过的网站发送出去，使目标网站接收并误以为是用户的真实操作而去执行命令。常用于盗取账号、转账、发送虚假消息等。攻击者利用网站对请求的验证漏洞而实现这样的攻击行为，网站能够确认请求来源于用户的浏览器，却不能验证请求是否源于用户的真实意愿下的操作行为。

# **七、异常**

74.throw 和 throws 的区别？

## 75.final、finally、finalize 有什么区别？

* final可以修饰类、变量、方法，修饰类表示该类不能被继承、修饰方法表示该方法不能被重写、修饰变量表示该变量是一个常量不能被重新赋值。
* finally一般作用在try-catch代码块中，在处理异常的时候，通常我们将一定要执行的代码方法finally代码块中，表示不管是否出现异常，该代码块都会执行，一般用来存放一些关闭资源的代码。
* finalize是一个方法，属于Object类的一个方法，而Object类是所有类的父类，该方法一般由垃圾回收器来调用，当我们调用System的gc()方法的时候，由垃圾回收器调用finalize(),回收垃圾。

## 76.try-catch-finally 中哪个部分可以省略？

catch-finally可以省略其中一个

## 77.try-catch-finally 中，如果 catch 中 return 了，finally 还会执行吗？

会执行，在 return 前执行。

## 78.常见的异常类有哪些？

* NullPointerException：当应用程序试图访问空对象时，则抛出该异常。
* SQLException：提供关于数据库访问错误或其他错误信息的异常。
* IndexOutOfBoundsException：指示某排序索引（例如对数组、字符串或向量的排序）超出范围时抛出。
* NumberFormatException：当应用程序试图将字符串转换成一种数值类型，但该字符串不能转换为适当格式时，抛出该异常。
* FileNotFoundException：当试图打开指定路径名表示的文件失败时，抛出此异常。
* IOException：当发生某种I/O异常时，抛出此异常。此类是失败或中断的I/O操作生成的异常的通用类。
* ClassCastException：当试图将对象强制转换为不是实例的子类时，抛出该异常。
* ArrayStoreException：试图将错误类型的对象存储到一个对象数组时抛出的异常。
* IllegalArgumentException：抛出的异常表明向方法传递了一个不合法或不正确的参数。
* ArithmeticException：当出现异常的运算条件时，抛出此异常。例如，一个整数“除以零”时，抛出此类的一个实例。
* NegativeArraySizeException：如果应用程序试图创建大小为负的数组，则抛出该异常。
* NoSuchMethodException：无法找到某一特定方法时，抛出该异常。
* SecurityException：由安全管理器抛出的异常，指示存在安全侵犯。
* UnsupportedOperationException：当不支持请求的操作时，抛出该异常。
* RuntimeException：是那些可能在Java虚拟机正常运行期间抛出的异常的超类。

## 79 什么是“异常链”?

　　“异常链”是Java中非常流行的异常处理概念，是指在进行一个异常处理时抛出了另外一个异常，由此产生

了一个异常链条。该技术大多用于将“ 受检查异常” （ checked exception）封装成为“非受检查异常”

（unchecked exception)或者RuntimeException。顺便说一下，如果因为因为异常你决定抛出一个新的异常，

你一定要包含原有的异常，这样，处理程序才可以通过getCause()和initCause()方法来访问异常最终的根源。

## 80 你遇到过 OutOfMemoryError 错误嘛？你是怎么搞定的？

jvm管理的内存大致包括三种不同类型的内存区域：Permanent Generation space（永久保存区域）、Heap space(堆区域)、Java Stacks(Java栈）。其中永久保存区域主要存放Class（类）和Meta的信息，Class第一次被Load的时候被放入PermGen space区域，Class需要存储的内容主要包括方法和静态属性。堆区域用来存放Class的实例（即对象），对象需要存储的内容主要是非静态属性。每次用new创建一个对象实例后，对象实例存储在堆区域中，这部分空间也被jvm的垃圾回收机制管理。而Java栈跟大多数编程语言包括汇编语言的栈功能相似，主要基本类型变量以及方法的输入输出参数。Java程序的每个线程中都有一个独立的堆栈。容易发生内存溢出问题的内存空间包括：Permanent Generation space和Heap space。

第一种OutOfMemoryError： PermGen space

发生这种问题的原意是程序中使用了大量的jar或class，使java虚拟机装载类的空间不够，与Permanent Generation space有关。解决这类问题有以下两种办法：

1. 增加java虚拟机中的XX:PermSize和XX:MaxPermSize参数的大小，其中XX:PermSize是初始永久保存区域大小，XX:MaxPermSize是最大永久保存区域大小。如针对tomcat6.0，在catalina.sh 或catalina.bat文件中一系列环境变量名说明结束处（大约在70行左右） 增加一行：

JAVA\_OPTS=" -XX:PermSize=64M -XX:MaxPermSize=128m"

如果是windows服务器还可以在系统环境变量中设置。感觉用tomcat发布sprint+struts+hibernate架构的程序时很容易发生这种内存溢出错误。使用上述方法，我成功解决了部署ssh项目的tomcat服务器经常宕机的问题。

2. 清理应用程序中web-inf/lib下的jar，如果tomcat部署了多个应用，很多应用都使用了相同的jar，可以将共同的jar移到tomcat共同的lib下，减少类的重复加载。这种方法是网上部分人推荐的，我没试过，但感觉减少不了太大的空间，最靠谱的还是第一种方法。

第二种OutOfMemoryError：  Java heap space

发生这种问题的原因是java虚拟机创建的对象太多，在进行垃圾回收之间，虚拟机分配的到堆内存空间已经用满了，与Heap space有关。解决这类问题有两种思路：

1. 检查程序，看是否有死循环或不必要地重复创建大量对象。找到原因后，修改程序和算法。

我以前写一个使用K-Means文本聚类算法对几万条文本记录（每条记录的特征向量大约10来个）进行文本聚类时，由于程序细节上有问题，就导致了Java heap space的内存溢出问题，后来通过修改程序得到了解决。

2. 增加Java虚拟机中Xms（初始堆大小）和Xmx（最大堆大小）参数的大小。如：set JAVA\_OPTS= -Xms256m -Xmx1024m

第三种OutOfMemoryError：unable to create new native thread

## 81在Java异常处理的过程中，你遵循的那些最好的实践是什么？

这个问题在面试技术经理是非常常见的一个问题。因为异常处理在项目设计中是非常关键的，所以精通异常处

理是十分必要的。异常处理有很多最佳实践，下面列举集中，它们提高你代码的健壮性和灵活性：

1) 调用方法的时候返回布尔值来代替返回null，这样可以 NullPointerException。由于空指针是java异常里最恶

心的异常。

2) catch块里别不写代码。空catch块是异常处理里的错误事件，因为它只是捕获了异常，却没有任何处理或者

提示。通常你起码要打印出异常信息，当然你最好根据需求对异常信息进行处理。

3)能抛受控异常（checked Exception）就尽量不抛受非控异常(checked Exception)。通过去掉重复的异常处

理代码，可以提高代码的可读性。

4) 绝对不要让你的数据库相关异常显示到客户端。由于绝大多数数据库和SQLException异常都是受控异常，在Java中，

你应该在DAO层把异常信息处理，然后返回处理过的能让用户看懂并根据异常提示信息改正操作的异常信息。

**八、网络**

79.http 响应码 301 和 302 代表的是什么？有什么区别？

80.forward 和 redirect 的区别？

81.简述 tcp 和 udp的区别？

82.tcp 为什么要三次握手，两次不行吗？为什么？

83.说一下 tcp 粘包是怎么产生的？

84.OSI 的七层模型都有哪些？

85.get 和 post 请求有哪些区别？

86.如何实现跨域？

87.说一下 JSONP 实现原理？

# **九、设计模式**

## 88.说一下你熟悉的设计模式？

89.简单工厂和抽象工厂有什么区别？

# **十、Spring/Spring MVC**

90.为什么要使用 spring？

91.解释一下什么是 aop？

92.解释一下什么是 ioc？

93.spring 有哪些主要模块？

94.spring 常用的注入方式有哪些？

## 93. Spring MVC生命周期？

**一、**浏览器会发送请求，通过 (前端控制器)DispatcherServlet这个servlet来接收请求.

DispatcherServlet接收请求后,此时会做一些初始化操作。

           1. DispatcherServlet会初始化HandlerMapping（注：通过它来处理客户端请求到各个Controller处理器的映射）

           2. DispatcherServlet会初始化HandlerAdapter（注：HandlerMapping会根据它来调用Controller里需要被执行的方法）

           3. DispatcherServlet会初始化handlerExceptionResolver（注：SpringMVC处理流程中，如果有异常抛出，会交给它来进行异常处理）

           4. DispatcherServlet会初始化ViewResolver （注：HandlerAdapter会把Controller中调用返回值最终包装成ModelAndView。ViewResolver会检查其中的view）

上述这些都初始化完毕后  然后DispatcherServlet会开始查询一个或者多个HandlerMaprring ,将请求映射到控制器对象。

**二、** 如果此时没有找到一个控制器对象 则会抛出异常.

         如果找到了一个控制器对象 那么这个请求会经过一个拦截器链的preHandle处理 通过HandlerAdapter来匹配到控制器对象的具体方法，然后这个方法中会根据我们的业务逻辑来处理请求。

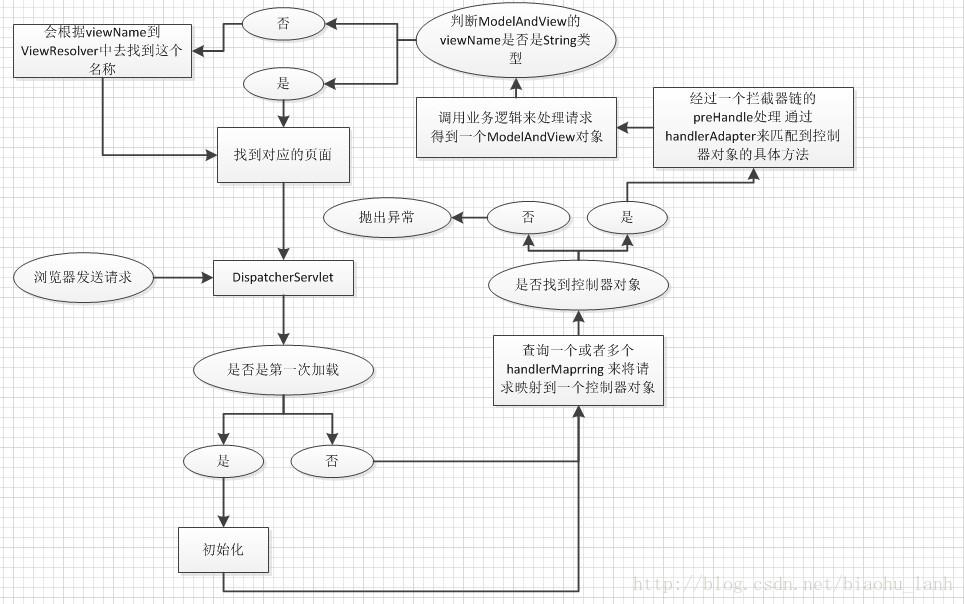
完成业务逻辑的处理后 控制器会得到一个ModelAndView对象（注：这个类中的view这个属性是 Object 类型的，它可以是一个视图名也可以是一个实际的View）

**三、** 得到这个ModelAndView后, Spring会判断ModelAndView的viewName是否是String类型

         如果是String类型 ,则直接调用ModelAndView的getView方法   然后找到对应的页面 然后将信息返回给DispatcherServlet.

         如果不是String类型,然后会根据viewName到ViewResolver中去找到这个名称  然后确定其对应的页面，然后将页面返回给DispatcherSerlvet.

**四、**最后DispatcherSerlvet会将得到页面返回给浏览器。



## 94. Spring Bean生命周期

1.Spring启动，查找并加载需要被Spring管理的bean，进行Bean的实例化

2.Bean实例化后对将Bean的引入和值注入到Bean的属性中

3.如果Bean实现了BeanNameAware接口的话，Spring将Bean的Id传递给setBeanName()方法

4.如果Bean实现了BeanFactoryAware接口的话，Spring将调用setBeanFactory()方法，将BeanFactory容器实例传入

5.如果Bean实现了ApplicationContextAware接口的话，Spring将调用Bean的setApplicationContext()方法，将bean所在应用上下文引用传入进来。

6.如果Bean实现了BeanPostProcessor接口，Spring就将调用他们的postProcessBeforeInitialization()方法。

7.如果Bean 实现了InitializingBean接口，Spring将调用他们的afterPropertiesSet()方法。类似的，如果bean使用init-method声明了初始化方法，该方法也会被调用

8.如果Bean 实现了BeanPostProcessor接口，Spring就将调用他们的postProcessAfterInitialization()方法。

9.此时，Bean已经准备就绪，可以被应用程序使用了。他们将一直驻留在应用上下文中，直到应用上下文被销毁。

10.如果bean实现了DisposableBean接口，Spring将调用它的destory()接口方法，同样，如果bean使用了destory-method 声明销毁方法，该方法也会被调用。

## 95.spring 中的 bean 是线程安全的吗？

Spring容器中的Bean是否线程安全，容器本身并没有提供Bean的线程安全策略，因此可以说spring容器中的Bean本身不具备线程安全的特性，但是具体还是要结合具体scope的Bean去研究。

## 96.spring 支持几种 bean 的作用域？

当通过spring容器创建一个Bean实例时，不仅可以完成Bean实例的实例化，还可以为Bean指定特定的作用域。Spring支持如下5种作用域：

* singleton：单例模式，在整个Spring IoC容器中，使用singleton定义的Bean将只有一个实例
* prototype：原型模式，每次通过容器的getBean方法获取prototype定义的Bean时，都将产生一个新的Bean实例
* equest：对于每次HTTP请求，使用request定义的Bean都将产生一个新实例，即每次HTTP请求将会产生不同的Bean实例。只有在Web应用中使用Spring时，该作用域才有效
* session：对于每次HTTP Session，使用session定义的Bean豆浆产生一个新实例。同样只有在Web应用中使用Spring时，该作用域才有效
* globalsession：每个全局的HTTP Session，使用session定义的Bean都将产生一个新实例。典型情况下，仅在使用portlet context的时候有效。同样只有在Web应用中使用Spring时，该作用域才有效

97.spring 自动装配 bean 有哪些方式？

## 98.spring 事务实现方式有哪些？

* 编程式事务管理对基于 POJO 的应用来说是唯一选择。我们需要在代码中调用beginTransaction()、commit()、rollback()等事务管理相关的方法，这就是编程式事务管理。
* 基于 TransactionProxyFactoryBean 的声明式事务管理
* 基于 @Transactional 的声明式事务管理
* 基于 Aspectj AOP 配置事务.

## 99.说一下 spring 的事务隔离？

事务隔离级别指的是一个事务对数据的修改与另一个并行的事务的隔离程度，当多个事务同时访问相同数据时，如果没有采取必要的隔离机制，就可能发生以下问题：

* 脏读：一个事务读到另一个事务未提交的更新数据。
* 幻读：例如第一个事务对一个表中的数据进行了修改，比如这种修改涉及到表中的“全部数据行”。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入“一行新数据”。那么，以后就会发生操作第一个事务的用户发现表中还存在没有修改的数据行，就好象发生了幻觉一样。
* 不可重复读：比方说在同一个事务中先后执行两条一模一样的select语句，期间在此次事务中没有执行过任何DDL语句，但先后得到的结果不一致，这就是不可重复读。

100.说一下 spring mvc 运行流程？

101.spring mvc 有哪些组件？

## 102.@RequestMapping 的作用是什么？

RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。RequestMapping注解有六个属性:

**value， method：**

* value：指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；
* method：指定请求的method类型， GET、POST、PUT、DELETE等；

**consumes，produces**

* consumes：指定处理请求的提交内容类型（Content-Type），例如application/json, text/html；
* produces：指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回；

**params，headers**

* params： 指定request中必须包含某些参数值是，才让该方法处理。
* headers：指定request中必须包含某些指定的header值，才能让该方法处理请求。

103.@Autowired 的作用是什么？

# **十一、Spring Boot**

## 104.什么是 spring boot？

springboot就是Spring开源框架下的子项目，是Spring的一站式解决方案，主要是简化了spring的使用难度，降低了对配置文件的要求，使得开发人员能够更容易得上手。

## 105.为什么要用 spring boot？

使用JavaConfig有助于避免使用XML。

避免大量maven导入和各种版本冲突。

提供可选的开发方法。

通过提供默认开发方式进行快速开发。

不需要单独的Web服务器。这意味着您不再需要启动Tomcat、Glassfish或其他任何东西。

由于没有web.xml文件，所以需要更少的配置。只需添加带@ configuration注释的类，然后可以添加带@ bean注释的方法，Spring将自动加载对象并像往常一样管理它。您甚至可以将@Autowired添加到bean方法中，使Spring autowire成为bean所需的依赖项。

基于环境的配置——使用这些属性，您可以将其传递到您正在使用的应用程序环境中spring.profile.active={enviorement}。在加载主应用程序属性文件之后，Spring将在(application-{environment}.properties)处加载后续的应用程序属性文件。

## 106. 什么是JavaConfig

Spring JavaConfig是Spring社区的一个产品，它提供了一种纯java方法来配置Spring IoC容器。因此，它有助于避免使用XML配置。使用JavaConfig的优点是：

面向对象的配置。因为配置在JavaConfig中定义为类，所以用户可以充分利用Java中的面向对象特性。一个配置类可以子类化另一个配置类，覆盖它的@Bean方法，等等。

减少或消除XML配置。已经证明了基于依赖注入原则的外部化配置的好处。然而，许多开发人员不愿意在XML和Java之间来回切换。JavaConfig为开发人员提供了一种纯java方法来配置Spring容器，这种方法在概念上类似于XML配置。从技术上讲，仅使用　　 JavaConfig配置类来配置容器是可行的，但是在实践中，许多人发现将JavaConfig与XML混合并匹配是理想的。

类型安全的重构能力。JavaConfig提供了一种类型安全的配置Spring容器的方法。由于Java 5.0对泛型的支持，现在可以通过类型而不是名称检索bean，不需要进行任何基于类型转换或字符串的查找。

## 107. 什么是Spring boot actuator?

Spring boot actuator是Spring boot framework的重要特性之一。Spring boot actuator帮助您访问生产环境中正在运行的应用程序的当前状态，在生产环境中必须检查和监视几个指标。甚至一些外部应用程序也可能使用这些服务来触发对相关人员的警报消息。actuator模块公开一组REST端点，这些端点可以作为HTTP URL直接访问，以检查状态。

## 108. 什么是Spring Profiles? 如何使用Spring Boot实现它?

Spring Profiles允许用户根据配置文件(dev, test, prod等)注册bean。因此，当应用程序在开发中运行时，只能加载某些bean，当应用程序在生产中运行时，只能加载某些其他bean。假设我们的需求是Swagger文档只对QA环境启用，对所有其他环境禁用。这可以使用配置文件来完成。Spring Boot使得使用配置文件非常容易。

## 109. 如何使用Spring Boot实现异常处理?

Spring提供了一种非常有用的方法，可以使用ControllerAdvice处理异常。我们将实现一个ControlerAdvice类，它将处理控制器类抛出的所有异常。

## 106. spring boot 配置文件有哪几种类型？它们有什么区别?

Spring Boot提供了两种常用的配置文件：

* properties文件
* yml文件, yml通过空格来确定层级关系，使配置文件结构跟清晰，但也会因为微不足道的空格而破坏了层级关系

## 108.spring boot 有哪些方式可以实现热部署？

SpringBoot热部署实现有两种方式：1使用spring loaded. 2使用spring-boot-devtools

## 109.jpa 和 hibernate 有什么区别？

* JPA Java Persistence API，是Java EE 5的标准ORM接口，也是ejb3规范的一部分。
* Hibernate，当今很流行的ORM框架，是JPA的一个实现，但是其功能是JPA的超集。
* JPA和Hibernate之间的关系，可以简单的理解为JPA是标准接口，Hibernate是实现。那么Hibernate是如何实现与JPA的这种关系的呢。Hibernate主要是通过三个组件来实现的，及hibernate-annotation、hibernate-entitymanager和hibernate-core。
* ibernate-annotation是Hibernate支持annotation方式配置的基础，它包括了标准的JPA annotation以及Hibernate自身特殊功能的annotation。
* hibernate-core是Hibernate的核心实现，提供了Hibernate所有的核心功能。
* hibernate-entitymanager实现了标准的JPA，可以把它看成hibernate-core和JPA之间的适配器，它并不直接提供ORM的功能，而是对hibernate-core进行封装，使得Hibernate符合JPA的规范。

# 十三、Spring Cloud

## 110.什么是 spring cloud？

Spring Cloud是一个微服务框架的规范，注意，只是规范，他不是任何具体的框架。我们知道java大佬最喜欢的做法就是自己制定规范，然后别人基于我这个规范来做实现。那么这个规范里面有什么呢，它规定大概要有以下几种功能:

服务的注册与发现, 负载均衡, 服务熔断和限流, 智能路由 ,控制总线,链路监控…

刚好，这个时候有一个框架集合几乎能满足上面所有的需求，他就是Spring Cloud Netflix。当然，Spring Cloud的实现产品不止这一个，还有最近由阿里新起的Spring Cloud Alibaba等。目前国内主流的是Spring Cloud Netflix。

## 2. 使用Spring Cloud有什么优势？

使用Spring Boot开发分布式微服务时，我们面临以下问题

1. 与分布式系统相关的复杂性-这种开销包括网络问题，延迟开销，带宽问题，安全问题。
2. 服务发现-服务发现工具管理群集中的流程和服务如何查找和互相交谈。它涉及一个服务目录，在该目录中注册服务，然后能够查找并连接到该目录中的服务。
3. 冗余-分布式系统中的冗余问题。
4. 负载平衡 --负载平衡改善跨多个计算资源的工作负荷，诸如计算机，计算机集群，网络链路，中央处理单元，或磁盘驱动器的分布。
5. 性能-问题 由于各种运营开销导致的性能问题。
6. 部署复杂性-Devops技能的要求。

## 3. 服务注册和发现是什么意思？Spring Cloud如何实现？

1、当我们开始一个项目时，我们通常在属性文件中进行所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降，而某些位置可能会发生变化。手动更改属性可能会产生问题。 Eureka服务注册和发现可以在这种情况下提供帮助。由于所有服务都在Eureka服务器上注册并通过调用Eureka服务器完成查找，因此无需处理服务地点的任何更改和处理。

Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块来实现服务注册和发现(请对比Zookeeper)。

2、Eureka 采用了 C-S 的设计架构。Eureka Server 作为服务注册功能的服务器，它是服务注册中心。

而系统中的其他微服务，使用 Eureka 的客户端连接到 Eureka Server并维持心跳连接。这样系统的维护人员就可以通过 Eureka Server 来监控系统中各个微服务是否正常运行。SpringCloud 的一些其他模块（比如Zuul）就可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑

## 4. 负载平衡的意义什么？

在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

## 5. 什么是Hystrix？它如何实现容错？

Hystrix是一个延迟和容错库，旨在隔离远程系统，服务和第三方库的访问点，当出现故障是不可避免的故障时，停止级联故障并在复杂的分布式系统中实现弹性。

通常对于使用微服务架构开发的系统，涉及到许多微服务。这些微服务彼此协作。

## 6.什么是Hystrix断路器？我们需要它吗？

## 7. 什么是Netflix Feign？它的优点是什么？

**Ribbon**

Ribbon 是一个基于 HTTP 和 TCP 客户端的负载均衡器

它可以在客户端配置 ribbonServerList（服务端列表），然后轮询请求以实现均衡负载，它在联合 Eureka 使用时，ribbonServerList 会被 DiscoveryEnabledNIWSServerList 重写，扩展成从 Eureka 注册中心获取服务端列表，同时它也会用 NIWSDiscoveryPing 来取代 IPing，它将职责委托给 Eureka 来确定服务端是否已经启动

**Feign**

Spring Cloud Netflix 的微服务都是以 HTTP 接口的形式暴露的，所以可以用 Apache 的 HttpClient 或 Spring 的 RestTemplate 去調用

而 Feign 是一個使用起來更加方便的 HTTP 客戶端，它用起來就好像調用本地方法一樣，完全感覺不到是調用的遠程方法

总结起来就是：发布到注册中心的服务方接口，是 HTTP 的，也可以不用 Ribbon 或者 Feign，直接浏览器一样能够访问

只不过 Ribbon 或者 Feign 调用起来要方便一些，最重要的是：它俩都支持软负载均衡

注意：spring-cloud-starter-feign 里面已经包含了 spring-cloud-starter-ribbon（Feign 中也使用了 Ribbon）

111.spring cloud 断路器的作用是什么？

112.spring cloud 的核心组件有哪些？

**十二、Hibernate**

113.为什么要使用 hibernate？

114.什么是 ORM 框架？

115.hibernate 中如何在控制台查看打印的 sql 语句？

116.hibernate 有几种查询方式？

117.hibernate 实体类可以被定义为 final 吗？

118.在 hibernate 中使用 Integer 和 int 做映射有什么区别？

119.hibernate 是如何工作的？

120.get()和 load()的区别？

121.说一下 hibernate 的缓存机制？

122.hibernate 对象有哪些状态？

123.在 hibernate 中 getCurrentSession 和 openSession 的区别是什么？

124.hibernate 实体类必须要有无参构造函数吗？为什么？

**十三、Mybatis**

125.mybatis 中 #{}和 ${}的区别是什么？

126.mybatis 有几种分页方式？

127.RowBounds 是一次性查询全部结果吗？为什么？

128.mybatis 逻辑分页和物理分页的区别是什么？

129.mybatis 是否支持延迟加载？延迟加载的原理是什么？

130.说一下 mybatis 的一级缓存和二级缓存？

131.mybatis 和 hibernate 的区别有哪些？

132.mybatis 有哪些执行器（Executor）？

133.mybatis 分页插件的实现原理是什么？

134.mybatis 如何编写一个自定义插件？

# **十四、RabbitMQ**

## 135.rabbitmq 的使用场景有哪些？

①. 跨系统的异步通信，所有需要异步交互的地方都可以使用消息队列。就像我们除了打电话（同步）以外，还需要发短信，发电子邮件（异步）的通讯方式。

②. 多个应用之间的耦合，由于消息是平台无关和语言无关的，而且语义上也不再是函数调用，因此更适合作为多个应用之间的松耦合的接口。基于消息队列的耦合，不需要发送方和接收方同时在线。在企业应用集成（EAI）中，文件传输，共享数据库，消息队列，远程过程调用都可以作为集成的方法。

③. 应用内的同步变异步，比如订单处理，就可以由前端应用将订单信息放到队列，后端应用从队列里依次获得消息处理，高峰时的大量订单可以积压在队列里慢慢处理掉。由于同步通常意味着阻塞，而大量线程的阻塞会降低计算机的性能。

④. 消息驱动的架构（EDA），系统分解为消息队列，和消息制造者和消息消费者，一个处理流程可以根据需要拆成多个阶段（Stage），阶段之间用队列连接起来，前一个阶段处理的结果放入队列，后一个阶段从队列中获取消息继续处理。

⑤. 应用需要更灵活的耦合方式，如发布订阅，比如可以指定路由规则。

⑥. 跨局域网，甚至跨城市的通讯（CDN行业），比如北京机房与广州机房的应用程序的通信。

## 136.rabbitmq 有哪些重要的角色？

RabbitMQ 中重要的角色有：生产者、消费者和代理：

生产者：消息的创建者，负责创建和推送数据到消息服务器；  
消费者：消息的接收方，用于处理数据和确认消息；  
代理：就是 RabbitMQ 本身，用于扮演“快递”的角色，本身不生产消息，只是扮演“快递”的角色。

## 137.rabbitmq 有哪些重要的组件？

Broker：简单来说就是队列服务器实体。

Exchange：消息交换机，它指定消息按什么规则，路由到哪个队列。

Queue：消息队列载体，每个消息都会被投入到一个或多个队列。

Binding：绑定，它的作用就是把exchange和queue按照路由规则绑定起来。

Routing Key：路由关键字， exchange根据这个关键字进行消息投递。

vhost：虚拟主机，一个broker里可以开设多个vhost，用作不同用户的权限分离。

producer：消息生产者，就是投递消息的程序。

consumer：消息消费者，就是接受消息的程序。

channel：消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务.

## 138.rabbitmq Exchange？

Exchange，用来接收生产者发送的消息并将这些消息路由给服务器中的队列。包含4种类型：Direct, Topic, Fanout, Headers。

**Direct**类型: 允许一个队列通过一个固定的Routing-key(通常是队列的名字)进行绑定。 Direct交换器将消息根据其routing-key属性投递到包含对应key属性的绑定器上。

**Topic**类型: 支持消息的Routing-key用\*或#的模式，进行绑定。\*匹配一个单词，#匹配0个或者多个单词。例如，binding key \*.user.# 匹配routing key为 usd.user和eur.user.db，但是不匹配user.hello。

**Fanout**类型：它只是将消息广播到所有绑定到它的队列中,而不考虑routing key的值。

**Header**类型： 它根据应用程序消息的特定属性进行匹配，这些消息可能在binding key中标记为可选或者必选。

## 139.rabbitmq RPC

MQ本身是基于异步的消息处理，前面的示例中所有的生产者（P）将消息发送到RabbitMQ后不会知道消费者（C）处理成功或者失败（甚至连有没有消费者来处理这条消息都不知道）。  
但实际的应用场景中，我们很可能需要一些同步处理，需要同步等待服务端将我的消息处理完成后再进行下一步处理。这相当于RPC（Remote Procedure Call，远程过程调用）。在RabbitMQ中也支持RPC。

RabbitMQ中实现RPC的机制是：

* 客户端发送请求（消息）时，在消息的属性（MessageProperties，在AMQP协议中定义了14中properties，这些属性会随着消息一起发送）中设置两个值replyTo（一个Queue名称，用于告诉服务器处理完成后将通知我的消息发送到这个Queue中）和correlationId（此次请求的标识号，服务器处理完成后需要将此属性返还，客户端将根据这个id了解哪条请求被成功执行了或执行失败）
* 服务器端收到消息并处理
* 服务器端处理完消息后，将生成一条应答消息到replyTo指定的Queue，同时带上correlationId属性
* 客户端之前已订阅replyTo指定的Queue，从中收到服务器的应答消息后，根据其中的correlationId属性分析哪条请求被执行了，根据执行结果进行后续业务处理

## 140.rabbitmq 中 vhost 的作用是什么？

vhost 可以理解为虚拟 broker ，即 mini-RabbitMQ server。其内部均含有独立的 queue、exchange 和 binding 等，但最最重要的是，其拥有独立的权限系统，可以做到 vhost 范围的用户控制。当然，从 RabbitMQ 的全局角度，vhost 可以作为不同权限隔离的手段（一个典型的例子就是不同的应用可以跑在不同的 vhost 中）。

## 141.rabbitmq 的消息是怎么发送的？

首先客户端必须连接到 RabbitMQ 服务器才能发布和消费消息，客户端和 rabbit server 之间会创建一个 tcp 连接，一旦 tcp 打开并通过了认证（认证就是你发送给 rabbit 服务器的用户名和密码），你的客户端和 RabbitMQ 就创建了一条 amqp 信道（channel），信道是创建在“真实” tcp 上的虚拟连接，amqp 命令都是通过信道发送出去的，每个信道都会有一个唯一的 id，不论是发布消息，订阅队列都是通过这个信道完成的。

## 142.rabbitmq 怎么保证消息的稳定性？

提供了事务的功能。  
通过将 channel 设置为 confirm（确认）模式。

## 143.rabbitmq 怎么避免消息丢失？

消息持久化  
ACK确认机制  
设置集群镜像模式  
消息补偿机制

## 144.要保证消息持久化成功的条件有哪些？

声明队列必须设置持久化 durable 设置为 true.  
消息推送投递模式必须设置持久化，deliveryMode 设置为 2（持久）。  
消息已经到达持久化交换器。  
消息已经到达持久化队列。

以上四个条件都满足才能保证消息持久化成功

145.rabbitmq 持久化有什么缺点？

## 146.rabbitmq 怎么实现延迟消息队列？

通过消息过期后进入死信交换器，再由交换器转发到延迟消费队列，实现延迟功能；  
使用 RabbitMQ-delayed-message-exchange 插件实现延迟功能。

147.rabbitmq 集群有什么用？

148.rabbitmq 节点的类型有哪些？

149.rabbitmq 集群搭建需要注意哪些问题？

## 150.rabbitmq 每个节点是其他节点的完整拷贝吗？为什么？

不是，原因有以下两个：

存储空间的考虑：如果每个节点都拥有所有队列的完全拷贝，这样新增节点不但没有新增存储空间，反而增加了更多的冗余数据；  
性能的考虑：如果每条消息都需要完整拷贝到每一个集群节点，那新增节点并没有提升处理消息的能力，最多是保持和单节点相同的性能甚至是更糟。

151.rabbitmq 集群中唯一一个磁盘节点崩溃了会发生什么情况？

## 152.rabbitmq 对集群节点停止顺序有要求吗？

RabbitMQ 对集群的停止的顺序是有要求的，应该先关闭内存节点，最后再关闭磁盘节点。如果顺序恰好相反的话，可能会造成消息的丢失。

# **十五、Kafka**

## 152.kafka 可以脱离 zookeeper 单独使用吗？为什么？

kafka 不能脱离 zookeeper 单独使用，因为 kafka 使用 zookeeper 管理和协调 kafka 的节点服务器。

## 153.kafka 有几种数据保留的策略？

kafka 有两种数据保存策略：按照过期时间保留和按照存储的消息大小保留。

## 154.kafka 同时设置了 7 天和 10G 清除数据，到第五天的时候消息达到了 10G，这个时候 kafka 将如何处理？

这个时候 kafka 会执行数据清除工作，时间和大小不论那个满足条件，都会清空数据。

## 155.什么情况会导致 kafka 运行变慢？

cpu 性能瓶颈  
磁盘读写瓶颈  
网络瓶颈

## 156.使用 kafka 集群需要注意什么？

集群的数量不是越多越好，最好不要超过 7 个，因为节点越多，消息复制需要的时间就越长，整个群组的吞吐量就越低。  
集群数量最好是单数，因为超过一半故障集群就不能用了，设置为单数容错率更高。

# **十六、Zookeeper**

## 157.zookeeper 是什么？

zookeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 google chubby 的开源实现，是 hadoop 和 hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

## 158.zookeeper 都有哪些功能？

集群管理：监控节点存活状态、运行请求等。  
主节点选举：主节点挂掉了之后可以从备用的节点开始新一轮选主，主节点选举说的就是这个选举的过程，使用 zookeeper 可以协助完成这个过程。  
分布式锁：zookeeper 提供两种锁：独占锁、共享锁。独占锁即一次只能有一个线程使用资源，共享锁是读锁共享，读写互斥，即可以有多线线程同时读同一个资源，如果要使用写锁也只能有一个线程使用。zookeeper可以对分布式锁进行控制。  
命名服务：在分布式系统中，通过使用命名服务，客户端应用能够根据指定名字来获取资源或服务的地址，提供者等信息。

## 159.zookeeper 有几种部署模式？

zookeeper 有三种部署模式：

单机部署：一台集群上运行；  
集群部署：多台集群运行；  
伪集群部署：一台集群启动多个 zookeeper 实例运行。

## 160.zookeeper 怎么保证主从节点的状态同步？

zookeeper 的核心是原子广播，这个机制保证了各个 server 之间的同步。实现这个机制的协议叫做 zab 协议。 zab 协议有两种模式，分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，zab 就进入了恢复模式，当领导者被选举出来，且大多数 server 完成了和 leader 的状态同步以后，恢复模式就结束了。状态同步保证了 leader 和 server 具有相同的系统状态。

## 161.集群中为什么要有主节点？

在分布式环境中，有些业务逻辑只需要集群中的某一台机器进行执行，其他的机器可以共享这个结果，这样可以大大减少重复计算，提高性能，所以就需要主节点。

## 162.集群中有 3 台服务器，其中一个节点宕机，这个时候 zookeeper 还可以使用吗？

可以继续使用，单数服务器只要没超过一半的服务器宕机就可以继续使用。

## 163.说一下 zookeeper 的通知机制？

客户端端会对某个 znode 建立一个 watcher 事件，当该 znode 发生变化时，这些客户端会收到 zookeeper 的通知，然后客户端可以根据 znode 变化来做出业务上的改变。

# **十七、SQL**

## 1. union和union all区别

union：对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序；

union All：对两个结果集进行并集操作，包括重复行，不进行排序；

## 2. having

having是分组（group by）后的筛选条件，分组后的数据组内再筛选

where则是在分组前筛选

## 3、sql的left join 、right join 、inner join之间的区别

left join(左联接) 返回包括左表中的所有记录和右表中联结字段相等的记录

right join(右联接) 返回包括右表中的所有记录和左表中联结字段相等的记录

inner join(等值连接) 只返回两个表中联结字段相等的行。

## 4. NVL()

NVL（表达式1，表达式2）  
  
如果表达式1为空值，NVL返回值为表达式2的值，否则返回表达式1的值。 该函数的目的是把一个空值（null）转换成一个实际的值。其表达式的值可以是数字型、字符型和日期型。但是表达式1和表达式2的数据类型必须为同一个类型。

## 5. decode()

decode（columnname，值1,翻译值1,值2,翻译值2,...值n,翻译值n,缺省值）

主要作用：相当于IF语句， 将查询结果翻译成其他值

columnname为要选择的table中所定义的column；缺省值可以是你要选择的column name本身，也可以是你想定义的其他值，比如Other等；

## 164.数据库的三范式是什么？

第一范式：强调的是列的原子性，即数据库表的每一列都是不可分割的原子数据项。  
第二范式：要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性。  
第三范式：任何非主属性不依赖于其它非主属性。

## 165.一张自增表里面总共有 7 条数据，删除了最后 2 条数据，重启 mysql 数据库，又插入了一条数据，此时 id 是几？

表类型如果是 MyISAM ，那 id 就是 8。  
表类型如果是 InnoDB，那 id 就是 6。

InnoDB 表只会把自增主键的最大 id 记录在内存中，所以重启之后会导致最大 id 丢失。

## 166.如何获取当前数据库版本？

使用 select version() 获取当前 MySQL 数据库版本。

## 167.说一下 ACID 是什么？

**Atomicity**（原子性）：一个事务（transaction）中的所有操作，或者全部完成，或者全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被恢复（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。即，事务不可分割、不可约简。  
**Consistency**（一致性）：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设约束、触发器、级联回滚等。  
**Isolation**（隔离性）：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。  
**Durability**（持久性）：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失.

## 168.char 和 varchar 的区别是什么？

char(n) ：固定长度类型，比如订阅 char(10)，当你输入"abc"三个字符的时候，它们占的空间还是 10 个字节，其他 7 个是空字节。  
chat 优点：效率高；缺点：占用空间；适用场景：存储密码的 md5 值，固定长度的，使用 char 非常合适。  
varchar(n) ：可变长度，存储的值是每个值占用的字节再加上一个用来记录其长度的字节的长度。  
所以，从空间上考虑 varcahr 比较合适；从效率上考虑 char 比较合适，二者使用需要权衡。

169.float 和 double 的区别是什么？

170.mysql 的内连接、左连接、右连接有什么区别？

## 171.mysql 索引是怎么实现的？

索引是满足某种特定查找算法的数据结构，而这些数据结构会以某种方式指向数据，从而实现高效查找数据。

具体来说 MySQL 中的索引，不同的数据引擎实现有所不同，但目前主流的数据库引擎的索引都是 B+ 树实现的，B+ 树的搜索效率，可以到达二分法的性能，找到数据区域之后就找到了完整的数据结构了，所有索引的性能也是更好的。

## 172.怎么验证 mysql 的索引是否满足需求？

使用 explain 查看 SQL 是如何执行查询语句的，从而分析你的索引是否满足需求。

explain 语法：explain select \* from table where type=1。

## 173.说一下数据库的事务隔离？

可用的配置值：READ-UNCOMMITTED、READ-COMMITTED、REPEATABLE-READ、SERIALIZABLE。

READ-UNCOMMITTED：未提交读，最低隔离级别、事务未提交前，就可被其他事务读取（会出现幻读、脏读、不可重复读）。  
READ-COMMITTED：提交读，一个事务提交后才能被其他事务读取到（会造成幻读、不可重复读）。  
REPEATABLE-READ：可重复读，默认级别，保证多次读取同一个数据时，其值都和事务开始时候的内容是一致，禁止读取到别的事务未提交的数据（会造成幻读）。  
SERIALIZABLE：序列化，代价最高最可靠的隔离级别，该隔离级别能防止脏读、不可重复读、幻读。

脏读 ：表示一个事务能够读取另一个事务中还未提交的数据。比如，某个事务尝试插入记录 A，此时该事务还未提交，然后另一个事务尝试读取到了记录 A。

不可重复读 ：是指在一个事务内，多次读同一数据。

幻读 ：指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A 第一次查询时候有 n 条记录，但是第二次同等条件下查询却有 n+1 条记录，这就好像产生了幻觉。发生幻读的原因也是另外一个事务新增或者删除或者修改了第一个事务结果集里面的数据，同一个记录的数据内容被修改了，所有数据行的记录就变多或者变少了。

## 174.说一下 mysql 常用的引擎？

InnoDB 引擎：InnoDB 引擎提供了对数据库 acid 事务的支持，并且还提供了行级锁和外键的约束，它的设计的目标就是处理大数据容量的数据库系统。MySQL 运行的时候，InnoDB 会在内存中建立缓冲池，用于缓冲数据和索引。但是该引擎是不支持全文搜索，同时启动也比较的慢，它是不会保存表的行数的，所以当进行 select count(\*) from table 指令的时候，需要进行扫描全表。由于锁的粒度小，写操作是不会锁定全表的,所以在并发度较高的场景下使用会提升效率的。

MyIASM 引擎：MySQL 的默认引擎，但不提供事务的支持，也不支持行级锁和外键。因此当执行插入和更新语句时，即执行写操作的时候需要锁定这个表，所以会导致效率会降低。不过和 InnoDB 不同的是，MyIASM 引擎是保存了表的行数，于是当进行 select count(\*) from table 语句时，可以直接的读取已经保存的值而不需要进行扫描全表。所以，如果表的读操作远远多于写操作时，并且不需要事务的支持的，可以将 MyIASM 作为数据库引擎的首选。

## 175.说一下 mysql 的行锁和表锁？

MyISAM 只支持表锁，InnoDB 支持表锁和行锁，默认为行锁。

表级锁：开销小，加锁快，不会出现死锁。锁定粒度大，发生锁冲突的概率最高，并发量最低。  
行级锁：开销大，加锁慢，会出现死锁。锁力度小，发生锁冲突的概率小，并发度最高。

## 176.说一下乐观锁和悲观锁？

乐观锁：每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在提交更新的时候会判断一下在此期间别人有没有去更新这个数据。  
悲观锁：每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻止，直到这个锁被释放。

数据库的乐观锁需要自己实现，在表里面添加一个 version 字段，每次修改成功值加 1，这样每次修改的时候先对比一下，自己拥有的 version 和数据库现在的 version 是否一致，如果不一致就不修改，这样就实现了乐观锁。

## 177.mysql 问题排查都有哪些手段？

使用 show processlist 命令查看当前所有连接信息。  
使用 explain 命令查询 SQL 语句执行计划。  
开启慢查询日志，查看慢查询的 SQL。

## 178.如何做 mysql 的性能优化？

为搜索字段创建索引。  
避免使用 select \*，列出需要查询的字段。  
垂直分割分表。  
选择正确的存储引擎。

# **十八、Redis**

## 179.redis 是什么？都有哪些使用场景？

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

Redis 使用场景：  
数据高并发的读写  
海量数据的读写  
对扩展性要求高的数据

## 180.redis 有哪些功能？

数据缓存功能  
分布式锁的功能  
支持数据持久化  
支持事务  
支持消息队列

## 181.redis 和 memecache 有什么区别？

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  
redis的速度比memcached快很多  
redis可以持久化其数据

## 182.redis 为什么是单线程的？

因为 cpu 不是 Redis 的瓶颈，Redis 的瓶颈最有可能是机器内存或者网络带宽。既然单线程容易实现，而且 cpu 又不会成为瓶颈，那就顺理成章地采用单线程的方案了。

## 183.什么是缓存穿透？怎么解决？

缓存穿透：指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿透。

解决方案：最简单粗暴的方法如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们就把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

## 184.redis 支持的数据类型有哪些？

string、list、hash、set、zset。

185.redis 支持的 java 客户端都有哪些？

186.jedis 和 redisson 有哪些区别？

187.怎么保证缓存和数据库数据的一致性？

## 188.redis 持久化有几种方式？

Redis 的持久化有两种方式，或者说有两种策略：

RDB（Redis Database）：指定的时间间隔能对你的数据进行快照存储。  
AOF（Append Only File）：每一个收到的写命令都通过write函数追加到文件中

## 189.redis 怎么实现分布式锁？

Redis 分布式锁其实就是在系统里面占一个“坑”，其他程序也要占“坑”的时候，占用成功了就可以继续执行，失败了就只能放弃或稍后重试。

占坑一般使用 setnx(set if not exists)指令，只允许被一个程序占有，使用完调用 del 释放锁。

## 190.redis 分布式锁有什么缺陷？

Redis 分布式锁不能解决超时的问题，分布式锁有一个超时时间，程序的执行如果超出了锁的超时时间就会出现问题.

## 191.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。

## 192.redis 淘汰策略有哪些？

volatile-lru：从已设置过期时间的数据集（server. db[i]. expires）中挑选最近最少使用的数据淘汰。  
volatile-ttl：从已设置过期时间的数据集（server. db[i]. expires）中挑选将要过期的数据淘汰。  
volatile-random：从已设置过期时间的数据集（server. db[i]. expires）中任意选择数据淘汰。  
allkeys-lru：从数据集（server. db[i]. dict）中挑选最近最少使用的数据淘汰。  
allkeys-random：从数据集（server. db[i]. dict）中任意选择数据淘汰。  
no-enviction（驱逐）：禁止驱逐数据。

193.redis 常见的性能问题有哪些？该如何解决？

# **十九、Memcached**

## 1. memcached是怎么工作的？

Memcached的神奇来自两阶段哈希（two-stage hash）。Memcached就像一个巨大的、存储了很多<key,value>对的哈希表。通过key，可以存储或查询任意的数据。   
  
客户端可以把数据存储在多台memcached上。当查询数据时，客户端首先参考节点列表计算出key的哈希值（阶段一哈希），进而选中一个节点；客户端将请求发送给选中的节点，然后memcached节点通过一个内部的哈希算法（阶段二哈希），查找真正的数据（item）。   
  
举个列子，假设有3个客户端1, 2, 3，3台memcached A, B, C：   
Client 1想把数据”barbaz”以key “foo”存储。Client 1首先参考节点列表（A, B, C），计算key “foo”的哈希值，假设memcached B被选中。接着，Client 1直接connect到memcached B，通过key “foo”把数据”barbaz”存储进去。　　Client 2使用与Client 1相同的客户端库（意味着阶段一的哈希算法相同），也拥有同样的memcached列表（A, B, C）。   
于是，经过相同的哈希计算（阶段一），Client 2计算出key “foo”在memcached B上，然后它直接请求memcached B，得到数据”barbaz”。   
  
各种客户端在memcached中数据的存储形式是不同的（perl Storable, php serialize, java hibernate, JSON等）。一些客户端实现的哈希算法也不一样。但是，memcached服务器端的行为总是一致的。   
  
最后，从实现的角度看，memcached是一个非阻塞的、基于事件的服务器程序。这种架构可以很好地解决C10K problem ，并具有极佳的可扩展性。

## 2. memcached最大的优势是什么？

请仔细阅读上面的问题（即memcached是如何工作的）。Memcached最大的好处就是它带来了极佳的水平可扩展性，特别是在一个巨大的系统中。由于客户端自己做了一次哈希，那么我们很容易增加大量memcached到集群中。memcached之间没有相互通信，因此不会增加 memcached的负载；没有多播协议，不会网络通信量爆炸（implode）。memcached的集群很好用。内存不够了？增加几台 memcached吧；CPU不够用了？再增加几台吧；有多余的内存？在增加几台吧，不要浪费了。

## 3. memcached的cache机制是怎样的？

Memcached主要的cache机制是LRU（最近最少用）算法+超时失效。当您存数据到memcached中，可以指定该数据在缓存中可以呆多久Which is forever, or some time in the future。如果memcached的内存不够用了，过期的slabs会优先被替换，接着就轮到最老的未被使用的slabs

## 4. memcached如何实现冗余机制？

不实现！我们对这个问题感到很惊讶。Memcached应该是应用的缓存层。它的设计本身就不带有任何冗余机制。如果一个memcached节点失去了所有数据，您应该可以从数据源（比如数据库）再次获取到数据。您应该特别注意，您的应用应该可以容忍节点的失效。不要写一些糟糕的查询代码，寄希望于 memcached来保证一切！如果您担心节点失效会大大加重数据库的负担，那么您可以采取一些办法。比如您可以增加更多的节点（来减少丢失一个节点的影响），热备节点（在其他节点down了的时候接管IP），等等。 

## 5. memcached如何处理容错的？

不处理！在memcached节点失效的情况下，集群没有必要做任何容错处理。如果发生了节点失效，应对的措施完全取决于用户。节点失效时，下面列出几种方案供您选择：   
  
\* 忽略它！ 在失效节点被恢复或替换之前，还有很多其他节点可以应对节点失效带来的影响。   
  
\* 把失效的节点从节点列表中移除。做这个操作千万要小心！在默认情况下（余数式哈希算法），客户端添加或移除节点，会导致所有的缓存数据不可用！因为哈希参照的节点列表变化了，大部分key会因为哈希值的改变而被映射到（与原来）不同的节点上。   
  
\* 启动热备节点，接管失效节点所占用的IP。这样可以防止哈希紊乱（hashing chaos）。   
  
\* 如果希望添加和移除节点，而不影响原先的哈希结果，可以使用一致性哈希算法（consistent hashing）。您可以百度一下一致性哈希算法。支持一致性哈希的客户端已经很成熟，而且被广泛使用。去尝试一下吧！   
  
\* 两次哈希（reshing）。当客户端存取数据时，如果发现一个节点down了，就再做一次哈希（哈希算法与前一次不同），重新选择另一个节点（需要注意的时，客户端并没有把down的节点从节点列表中移除，下次还是有可能先哈希到它）。如果某个节点时好时坏，两次哈希的方法就有风险了，好的节点和坏的节点上都可能存在脏数据（stale data）。

## 6. memcached的多线程是什么？如何使用它们？

线程就是定律（threads rule）！在Steven Grimm和Facebook的努力下，memcached 1.2及更高版本拥有了多线程模式。多线程模式允许memcached能够充分利用多个CPU，并在CPU之间共享所有的缓存数据。memcached使用一种简单的锁机制来保证数据更新操作的互斥。相比在同一个物理机器上运行多个memcached实例，这种方式能够更有效地处理multi gets。

## 7. memcached能接受的key的最大长度是多少？

key的最大长度是250个字符。需要注意的是，250是memcached服务器端内部的限制，如果您使用的客户端支持”key的前缀”或类似特性，那么key（前缀+原始key）的最大长度是可以超过250个字符的。我们推荐使用使用较短的key，因为可以节省内存和带宽。

## 8. memcached对item的过期时间有什么限制？

过期时间最大可以达到30天。memcached把传入的过期时间（时间段）解释成时间点后，一旦到了这个时间点，memcached就把item置为失效状态。这是一个简单但obscure的机制。

## 9. memcached最大能存储多大的单个item？

1MB。如果你的数据大于1MB，可以考虑在客户端压缩或拆分到多个key中。

## 10. 为什么单个item的大小被限制在1M byte之内？

简单的回答：因为内存分配器的算法就是这样的。

详细的回答：Memcached的内存存储引擎（引擎将来可插拔…），使用slabs来管理内存。内存被分成大小不等的slabs chunks（先分成大小相等的slabs，然后每个slab被分成大小相等chunks，不同slab的chunk大小是不相等的）。chunk的大小依次从一个最小数开始，按某个因子增长，直到达到最大的可能值。

如果最小值为400B，最大值是1MB，因子是1.20，各个slab的chunk的大小依次是：slab1 – 400B slab2 – 480B slab3 – 576B …

slab中chunk越大，它和前面的slab之间的间隙就越大。因此，最大值越大，内存利用率越低。Memcached必须为每个slab预先分配内存，因此如果设置了较小的因子和较大的最大值，会需要更多的内存。

还有其他原因使得您不要这样向memcached中存取很大的数据…不要尝试把巨大的网页放到mencached中。把这样大的数据结构load和unpack到内存中需要花费很长的时间，从而导致您的网站性能反而不好。

如果您确实需要存储大于1MB的数据，你可以修改slabs.c:POWER\_BLOCK的值，然后重新编译memcached；或者使用低效的malloc/free。其他的建议包括数据库、MogileFS等。

## 11.简述Memcached内存管理机制原理？ malloc的全称是memory allocation ,中文叫运态内存分配，当无法知道内存具体位置的时候，想要绑定真正的内存空间，就需要用到动态的分配内存。

早期的Memcached内存管理方式是通过malloc的分配的内存，使用完后通过free来回收内存，这种方式容易产生内存碎片，并降低操作系统对内存的管理效率。加重操作系统内存管理器的负担，最坏的情况下，会导致操作系统比memcached进程本身还慢，为了解决这个问题，Slab Allocation内存分配机制就延生了。  
　　现在Memcached利用Slab Allocation机制来分配和管理内存。  
　　Slab Allocation机制原理是按照预先规定的大小，将分配给memcached的内存分割成特定长度的内存块（chunk)，再把尺寸相同的内存块，分成组  
（chunks slab class),这些内存块不会释放，可以重复利用。

## 12.redis和memcached的区别

1、Redis和Memcache都是将数据存放在内存中，都是内存数据库。不过memcache还可用于缓存其他东西，例如图片、视频等等；

2、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储；

3、虚拟内存--Redis当物理内存用完时，可以将一些很久没用到的value 交换到磁盘；

4、过期策略--memcache在set时就指定，例如set key1 0 0 8,即永不过期。Redis可以通过例如expire 设定，例如expire name 10；

5、分布式--设定memcache集群，利用magent做一主多从;redis可以做一主多从。都可以一主一从；

6、存储数据安全--memcache挂掉后，数据没了；redis可以定期保存到磁盘（持久化）；

7、灾难恢复--memcache挂掉后，数据不可恢复; redis数据丢失后可以通过aof恢复；

8、Redis支持数据的备份，即master-slave模式的数据备份；

# 二十、JVM

194.说一下 jvm 的主要组成部分？及其作用？

195.说一下 jvm 运行时数据区？

196.说一下堆栈的区别？

197.队列和栈是什么？有什么区别？

198.什么是双亲委派模型？

## 199.说一下类加载的执行过程？

类加载分为以下 5 个步骤：

加载：根据查找路径找到相应的 class 文件然后导入；  
检查：检查加载的 class 文件的正确性；  
准备：给类中的静态变量分配内存空间；  
解析：虚拟机将常量池中的符号引用替换成直接引用的过程。符号引用就理解为一个标示，而在直接引用直接指向内存中的地址；  
初始化：对静态变量和静态代码块执行初始化工作。

## 200. JVM中的classloader机制：

jvm中默认定了三种classloader，分别为：bootstrap classloader， extension classloader, system classloader.

**启动类加载器**（Bootstrap ClassLoader），是虚拟机自身的一部分，用来加载Java\_HOME/lib/目录中的，或者被 -Xbootclasspath 参数所指定的路径中并且被虚拟机识别的类库；  
其他类加载器：  
**扩展类加载器**（Extension ClassLoader）：负责加载\lib\ext目录或Java. ext. dirs系统变量指定的路径中的所有类库；</java\_home>  
**应用程序类加载器**（Application ClassLoader）。负责加载用户类路径（classpath）上的指定类库，我们可以直接使用这个类加载器。一般情况，如果我们没有自定义类加载器默认就是用这个加载器。

双亲委派模型：除此之外，java的classloader采用委托机制，即classloader都有一个 parent classloader。当收到一个类加载请求时，会先请求parent classloader加载类，如果parent classloader加载不到，再由自身尝试加载（如果加载不到，throw ClassNotFountException). 这种机制主要是出于安全考虑，如果用户自定义一个java.lang.Object 不至于覆盖jdk中的Object。

200.怎么判断对象是否可以被回收？

201.java 中都有哪些引用类型？

202.说一下 jvm 有哪些垃圾回收算法？

203.说一下 jvm 有哪些垃圾回收器？

204.详细介绍一下 CMS 垃圾回收器？

205.新生代垃圾回收器和老生代垃圾回收器都有哪些？有什么区别？

206.简述分代垃圾回收器是怎么工作的？

207.说一下 jvm 调优的工具？

## 208.常用的 jvm 调优的参数都有哪些？

-Xms2g：初始化推大小为 2g；  
-Xmx2g：堆最大内存为 2g；  
-XX:NewRatio=4：设置年轻的和老年代的内存比例为 1:4；  
-XX:SurvivorRatio=8：设置新生代 Eden 和 Survivor 比例为 8:2；  
–XX:+UseParNewGC：指定使用 ParNew + Serial Old 垃圾回收器组合；  
-XX:+UseParallelOldGC：指定使用 ParNew + ParNew Old 垃圾回收器组合；  
-XX:+UseConcMarkSweepGC：指定使用 CMS + Serial Old 垃圾回收器组合；  
-XX:+PrintGC：开启打印 gc 信息；  
-XX:+PrintGCDetails：打印 gc 详细信息。