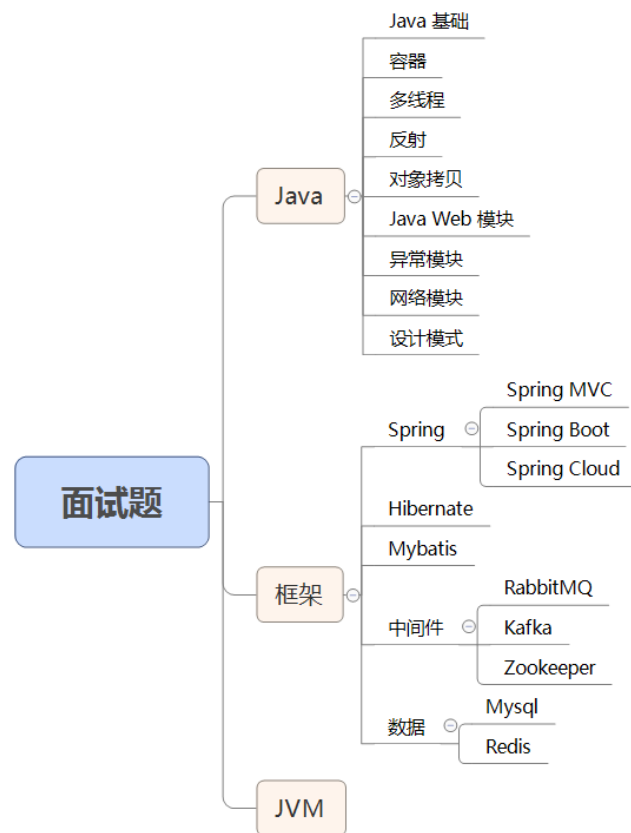


原 java初、中、高级面试题——基础、集合、线程

2019年04月22日 16:07:49 在IT中穿梭旅行 阅读数 95 更多

编辑

面试题，包含的内容分为十九个模块：Java 基础、集合、多线程、反射、对象拷贝、Java Web 模块、异常、网络、设计模式、Spring/Spring MVC、Spring Boot/Spring Cloud、Hibernate、Mybatis、RabbitMQ、Kafka、Zookeeper、MySQL、Redis、JVM。



Java 基础

1.JDK 和 JRE 有什么区别？

- JDK: Java Development Kit 的简称, java 开发工具包, 提供了 java 的开发环境和运行环境。

- JRE: Java Runtime Environment 的简称, java 运行环境, 为 java 的运行提供了所需环境。
- JVM: Java Virtual Machine的简称, java虚拟机, JVM是一种用于计算设备的规范, 它是一个虚构出来的计算机, 是通过在实际的计算机上仿真模拟各种计算机功能来实现的。

具体来说 JDK 其实包含了 JRE, 同时还包含了编译 java 源码的编译器 javac, 还包含了很多 java 程序调试和分析的工具。简单来说: 如果你需要运行 java 程序, 只需安装 JRE 就可以了, 如果你需要编写 java 程序, 需要安装 JDK。JDK包含JRE和JVM

2.== 和 equals 的区别是什么?

== 解读

对于基本类型和引用类型 == 的作用效果是不同的, 如下所示:

- 基本类型: 比较的是值是否相同;
- 引用类型: 比较的是引用是否相同;

代码如下:

```
1 String x = "string";
2 String y = "string";
3 String z = new String("string");
4 System.out.println(x==y); // true
5 System.out.println(x==z); // false
6 System.out.println(x.equals(y)); // true
7 System.out.println(x.equals(z)); // true
```

代码解读: 因为 x 和 y 指向的是同一个引用, 所以 == 也是 true, 而 new String()方法则重写开辟了内存空间, 所以 == 结果为 false, 而 equals 比较的一直是值, 所以结果都为 true。

equals 解读

equals 本质上就是 ==, 只不过 String 和 Integer 等重写了 equals 方法, 把它变成了值比较。看下面的代码就明白了。

首先来看默认情况下 equals 比较一个有相同值的对象，代码如下：

```
1 class Cat {
2     public Cat(String name) {
3         this.name = name;
4     }
5
6     private String name;
7
8     public String getName() {
9         return name;
10    }
11
12    public void setName(String name) {
13        this.name = name;
14    }
15 }
16
17 Cat c1 = new Cat("王磊");
18 Cat c2 = new Cat("王磊");
19 System.out.println(c1.equals(c2)); // false
```

输出结果出乎我们的意料，竟然是 false？这是怎么回事，看了 equals 源码就知道了，源码如下：

```
1 public boolean equals(Object obj) {
2     return (this == obj);
3 }
```

原来 equals 本质上就是 ==。

那问题来了，两个相同值的 String 对象，为什么返回的是 true？代码如下：

```
1 String s1 = new String("老王");
```

```
2 | String s2 = new String("老王"); 3 | System.out.println(s1.equals(s2)); // true
```

同样的，当我们进入 String 的 equals 方法，找到了答案，代码如下：

```
1 | public boolean equals(Object anObject) {  
2 |     if (this == anObject) {  
3 |         return true;  
4 |     }  
5 |     if (anObject instanceof String) {  
6 |         String anotherString = (String)anObject;  
7 |         int n = value.length;  
8 |         if (n == anotherString.value.length) {  
9 |             char v1[] = value;  
10 |            char v2[] = anotherString.value;  
11 |            int i = 0;  
12 |            while (n-- != 0) {  
13 |                if (v1[i] != v2[i])  
14 |                    return false;  
15 |                i++;  
16 |            }  
17 |            return true;  
18 |        }  
19 |    }  
20 |    return false;  
21 | }
```

原来是 String 重写了 Object 的 equals 方法，把引用比较改成了值比较。

总结：== 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，只是很多类重写了 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。

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

不对，两个对象的 hashCode()相同，equals()不一定 true。

代码示例：

```
1 String str1 = "通话";
2 String str2 = "重地";
3 System.out.println(String.format("str1: %d | str2: %d", str1.hashCode(),str2.hashCode()));
4 System.out.println(str1.equals(str2));
```

执行的结果：

str1: 1179395 | str2: 1179395

false

代码解读：很显然“通话”和“重地”的 hashCode() 相同，然而 equals() 则为 false，因为在散列表中，hashCode()相等即两个键值对的哈希值相等，然而哈希值相等，并不一定能得出键值对相等。

为什么要重写equals和hashCode()方法？

- 1、如果两个对象相同（即用equals比较返回true），那么它们的hashCode值一定要相同；
- 2、如果两个对象的hashCode相同，它们并不一定相同(即用equals比较返回false)

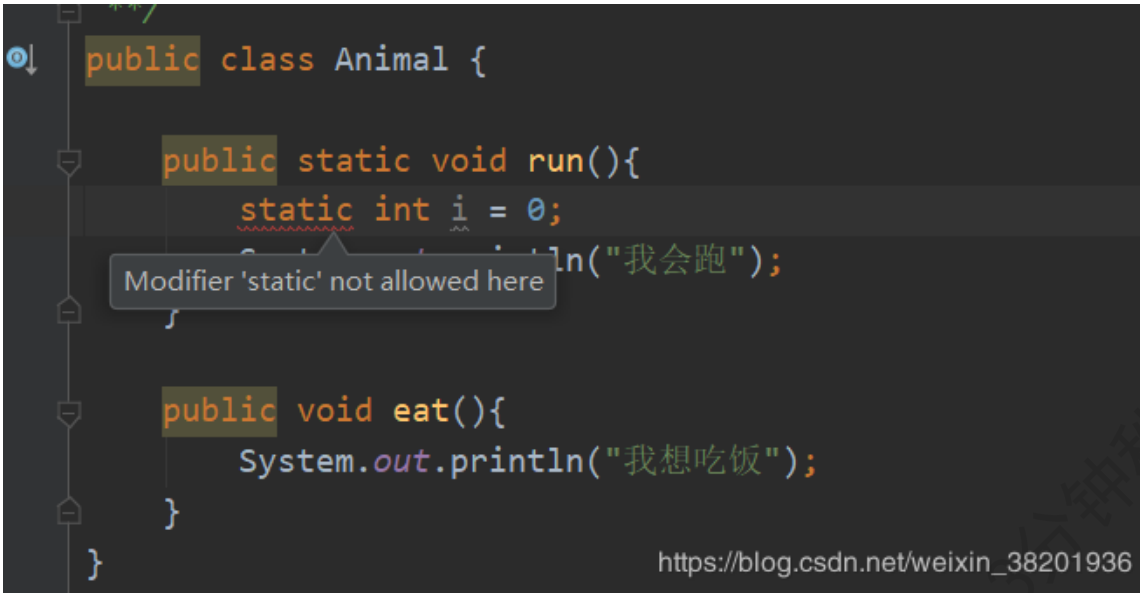
4.final 在 java 中有什么作用？

- final 修饰的类叫最终类，该类不能被继承。
- final 修饰的方法不能被重写。
- final 修饰的变量叫常量，常量必须初始化，初始化之后值就不能被修改。

Static关键字在java中有什么作用？

- **静态变量**：被static修饰的变量属于类变量，可以通过**类名.变量名**直接引用，而不需要new出一个类来。
- **静态方法**：被static修饰的方法属于类方法，可以通过**类名.方法名**直接引用，而不需要new出一个类来。

- **static是不允许用来修饰局部变量**
- **静态块：**静态块里面的代码只执行一次，且只在初始化类的时候执行。



```
public class Animal {  
    public static void run(){  
        static int i = 0;  
        System.out.println("我会跑");  
    }  
    public void eat(){  
        System.out.println("我想吃饭");  
    }  
}
```

Modifier 'static' not allowed here

https://blog.csdn.net/weixin_38201936

静态变量、静态方法和实例变量、实例方法有什么不同？

静态变量又称为类变量，在声明时必须加入static关键字，可以用类名直接打点调用

静态方法必须加入static关键字，在调用时，可以直接用类名调用，不需要创建实例对象。

实例变量，不需要加入static，调用时必须声明实例对象。

实例方法也不需要加入static关键字，调用时必须创建对象。

为什么要用静态方法？

静态方法的好处就是不用生成类的实例就可以直接调用，类第一次加载的时候，static就已经在内存中了，直到程序结束后，该内存才会释放。

如果不是static修饰的成员方法，在使用完之后就会立即被JVM回收。

5.java 中的 Math.round(-1.5) 等于多少？

Math.round()方法即为我们常说的“四舍五入”方法

Math.round(1.0)	1	Math.round(-1.0)	-1
Math.round(1.4)	1	Math.round(-1.4)	-1
Math.round(1.5)	1	Math.round(-1.5)	-1
Math.round(1.6)	2	Math.round(-1.6)	-2

```
public class TestDemo {  
    public static void main(String[] args) {  
  
        System.out.println(Math.round(-1.5));  
  
    }  
}
```

https://blog.csdn.net/weixin_38201936

```
"C:\Program Files\java\jdk1.8.0_151\bin\java.exe"  
-1  
  
Process finished with exit code 0
```

6.String 属于基础的数据类型吗？

String 不属于基础类型，基础类型有 8 种：byte、boolean、char、short、int、float、long、double，而 String 属于对象。

7.java 中操作字符串都有哪些类？它们之间有什么区别？

操作字符串的类有：String、StringBuffer、StringBuilder。

String 和 StringBuffer、StringBuilder 的区别在于 String 声明的是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象，而 StringBuffer、StringBuilder 可以在原有对象的基础上进行操作，所以在经常改变字符串内容的情况下最好不要使用 String。

StringBuffer 和 StringBuilder 最大的区别在于，StringBuffer 是线程安全的，而 StringBuilder 是非线程安全的，但 StringBuilder 的性能却高于 StringBuffer，所以在单线程环境下推荐使用 StringBuilder，多线程环境下推荐使用 StringBuffer。

8.String str="i"与 String str=new String("i")一样吗？

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

9.如何将字符串反转？

使用 StringBuilder 或者 stringBuffer 的 reverse() 方法。

```
1 // StringBuffer reverse
2 StringBuffer stringBuffer = new StringBuffer();
3 stringBuffer.append("abcdefg");
4 System.out.println(stringBuffer.reverse()); // gfedcba
5 // StringBuilder reverse
6 StringBuilder stringBuilder = new StringBuilder();
7 stringBuilder.append("abcdefg");
8 System.out.println(stringBuilder.reverse()); // gfedcba
```

10.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。

11.抽象类必须要有抽象方法吗?

不需要，抽象类不一定非要有抽象方法。

示例代码：

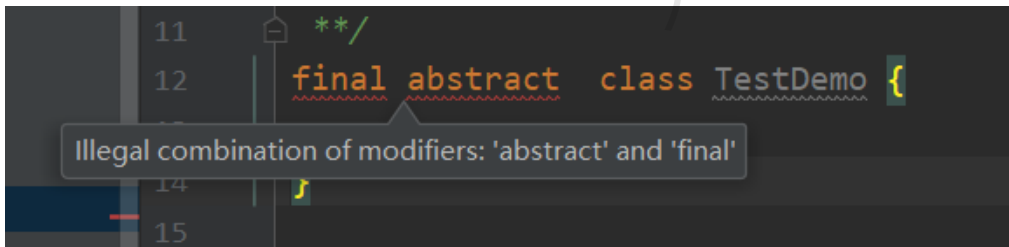
```
1 abstract class Cat {  
2     public static void sayHi() {  
3         System.out.println("hi~");  
4     }  
5 }
```

12.普通类和抽象类有哪些区别?

- 普通类不能包含抽象方法，抽象类可以包含抽象方法。
- 抽象类不能直接实例化，普通类可以直接实例化。

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，这样彼此就会产生矛盾，所以 final 不能修饰抽象类，如下图所示，编辑器也会提示错误信息：



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。



```
2 public abstract class Animal {
3
4     public Animal(){
5
6         System.out.println("我是动物");
7     }
8
9     public static void run(){
10         System.out.println("我会跑");
11     }
12
13     public void eat(){
14         System.out.println("我想吃饭");
15     }
16 }
17
18 https://blog.csdn.net/weixin_38201936
```

```
11 /**
12  public interface Cat {
13
14     public Cat();
15
16 }
17
18 https://blog.csdn.net/weixin_38201936
```

- 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。



```
1 interface Cat{
2
3 }
4
5 interface Dog{
6
7 }
8
9 public class TestDemo2 implements Cat,Dog{
10
11 }
12
13 https://blog.csdn.net/weixin_38201936
```

```
12  abstract class Animal{
13
14  }
15
16  abstract class Animals{
17
18  }
19  |
20  public class TestDemo extends Animal,Animals{
21
22  }
23
```

Class cannot extend multiple classes

https://blog.csdn.net/weixin_38201936

- 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。

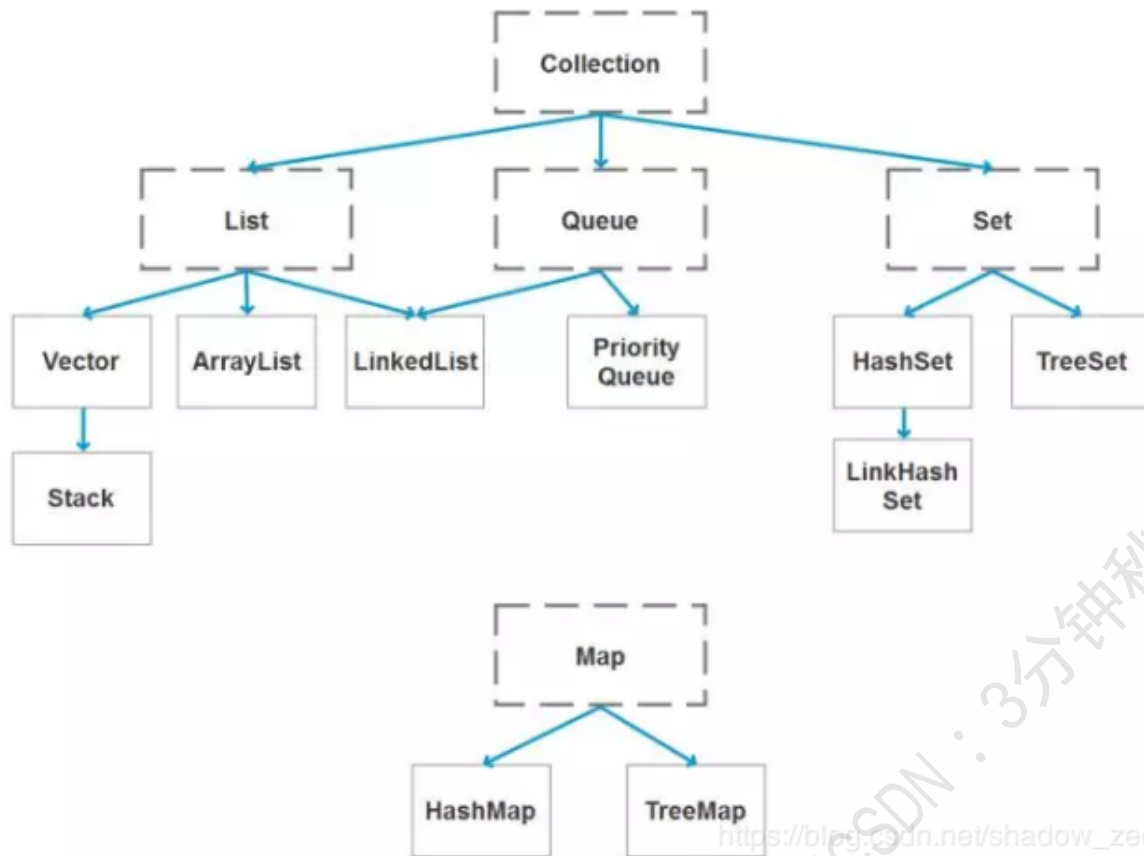
17.Files的常用方法都有哪些？

- Files.exists(): 检测文件路径是否存在。
- Files.createFile(): 创建文件。
- Files.createDirectory(): 创建文件夹。
- Files.delete(): 删除一个文件或目录。
- Files.copy(): 复制文件。
- Files.move(): 移动文件。
- Files.size(): 查看文件个数。
- Files.read(): 读取文件。
- Files.write(): 写入文件。

集合

18. java 容器都有哪些?

常用容器的图录:



19. Collection 和 Collections 有什么区别?

- java.util.Collection 是一个集合接口（集合类的一个顶级接口）。它提供了对集合对象进行基本操作的通用接口方法。Collection接口在Java 类库中有很多具体的实现。Collection接口的意义是为各种具体的集合提供了最大化的统一操作方式，其直接继承接口有List与Set。
- Collections则是集合类的一个工具类/帮助类，其中提供了一系列静态方法，用于对集合中元素进行排序、搜索以及线程安全等各种操作。

20. List、Set、Map 之间的区别是什么?

比较	List	Set	Map
继承接口	Collection	Collection	
常见实现类	AbstractList(其常用子类有 ArrayList、LinkedList、Vector)	AbstractSet(其常用子类有 HashSet、LinkedHashSet、TreeSet)	HashMap、HashTable
常见方法	add(), remove(), clear(), get(), contains(), size()	add(), remove(), clear(), contains(), size()	put(), get(), remove(), clear(), containsKey(), containsValue(), keySet(), values(), size()
元素	可重复	不可重复(用 equals() 判断)	不可重复
顺序	有序	无序(实际上由HashCode决定)	
线程安全	Vector线程安全		Hashtable线程安全

List集合：

arrayList集合：

- ArrayList是基于动态数组的数据结构，因为地址连续，所以一旦数据存储了查询效率比较高，但因为地址连续，所以插入和删除效率低。
- arrayList数组长度默认10，超过长度后， 每次扩容至原来容量的 1.5 倍。
- ArrayList不同步，线程不安全，因为线程的同步必然要影响性能，所以性能好。

linkedList集合：

- LinkedList 是基于双向链表的数据结构，地址随意，开辟内存空间时，不需要等一个连续的地址，所以插入和删除效率高，相反，数据量特别大的时候，查询效率特别低，不支持随机访问，使用下标访问一个元素。
- ArrayList不同步，线程不安全

21. 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List：调用Arrays的asList方法。

```
public class Test3 {  
    public static void main(String[] args) {  
        String[] arr = {"a","b"};  
        System.out.println(Arrays.toString(arr));  
        //将数组转成List  
        System.out.println(Arrays.asList(arr));  
    }  
}
```

https://blog.csdn.net/weixin_38201936

```
"C:\Program Files\java\jdk1.8.0_151\bin\java.exe" ...  
[a, b]  
[a, b]
```

22. Array 和 ArrayList 有何区别？（数组和集合的区别）

- Array可以容纳基本类型和对象，而ArrayList只能容纳对象。
- Array是指定大小后不可变的，而ArrayList大小是可变的。
- Array没有提供ArrayList那么多功能，比如addAll、removeAll和iterator等。

23. ArrayList 和 LinkedList 的区别是什么？

- ArrayList是基于动态数组的数据结构，因为地址连续，所以一旦数据存储了查询效率比较高，但因为地址连续，所以插入和删除效率低。
- LinkedList 是基于双向链表的数据结构，地址随意，开辟内存空间时，不需要等一个连续的地址，所以插入和删除效率高，相反，数据量特别大的时候，查询效率特别低，不支持随机访问，使用下标访问一个元素。

- ArrayList 的时间复杂度是 $O(1)$ ，而 LinkedList 是 $O(n)$ 。

24. ArrayList 和 Vector 的区别是什么？

- Vector是同步的，线程安全的，而ArrayList不同步的，线程不安全的。由于线程的同步必然要影响性能。因此，ArrayList的性能比Vector好。
- ArrayList比Vector查询效率高。
- Vector会扩容2倍，而ArrayList只增加50%的大小，这样。ArrayList就有利于节约内存空间。
- Vector可以设置增长因子，而ArrayList不可以。
- ArrayList更加通用，因为我们可以使用Collections工具类轻易地获取同步列表和只读列表。

适用场景分析：

- 1、Vector是线程同步的，所以它也是线程安全的，而ArrayList是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用ArrayList效率比较高。
- 2、如果集合中的元素的数目大于目前集合数组的长度时，在集合中使用数据量比较大的数据，用Vector有一定的优势。

Set集合：

25. 说一下 HashSet 的实现原理？

- HashSet底层由HashMap实现
- HashSet的值存放于HashMap的key上
- HashMap的value统一为PRESENT

26. HashSet与TreeSet的比较

- TreeSet 是二叉树实现的，TreeSet中的数据是自动排好序的，不允许放入null值。
- HashSet 是哈希表实现的，HashSet中的数据是无序的，可以放入null，但只能放入一个null，两者中的值都不能重复，就如数据库中唯一约束。

- HashSet要求放入的对象必须实现HashCode()方法，放入的对象，是以hashcode码作为标识的，而具有相同内容的String对象，hashcode是一样，所以放入的内容不能重复。但是同一个类的对象可以放入不同的实例。

Map集合：

在遍历map进行删除操作时，为什么经常出现ConcurrentModificationException异常？

为什么呢？

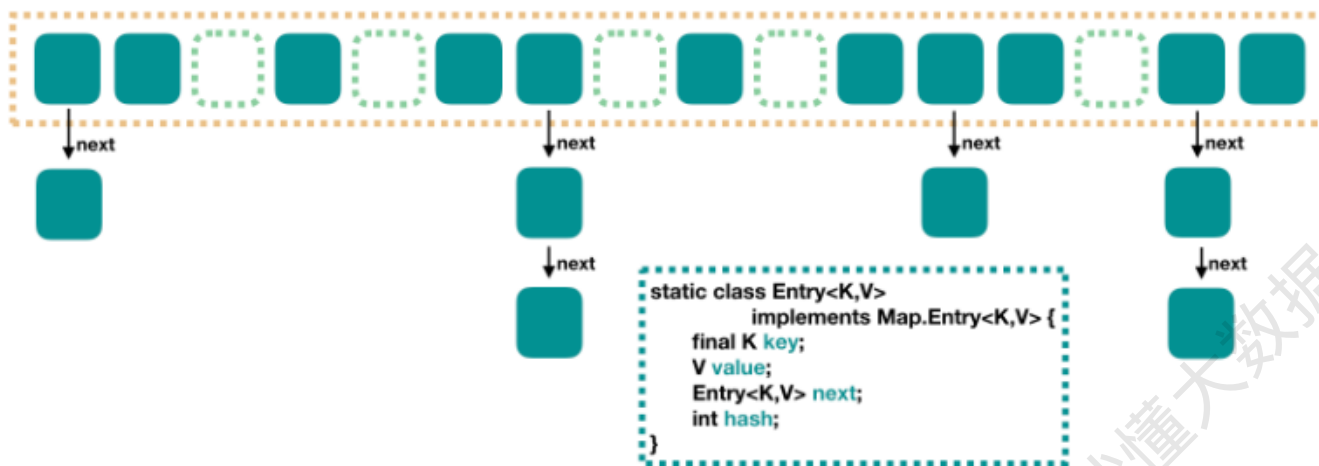
使用iterator迭代删除时没有问题的，在每一次迭代时都会调用hasNext（）方法判断是否有下一个，是允许集合中数据增加和减少的。

使用forEach删除时，会报错ConcurrentModificationException，因为在forEach遍历时，是不允许map元素进行删除和增加。

所以，遍历删除map集合中元素时，必须使用迭代iterator

HashMap

Java7 HashMap 结构



这个仅仅是示意图，因为没有考虑到数组要扩容的情况，具体的后面再说。

大方向上，HashMap 里面是一个数组，然后数组中每个元素是一个单向链表。

上图中，每个绿色的实体是嵌套类 Entry 的实例，Entry 包含四个属性：key, value, hash 值和用于单向链表的 next。

capacity: 当前数组容量，始终保持 2^n ，可以扩容，扩容后数组大小为当前的 2 倍。

loadFactor: 负载因子，默认为 0.75。

threshold: 扩容的阈值，等于 $\text{capacity} * \text{loadFactor}$

https://blog.csdn.net/weixin_38201936

Java8 HashMap

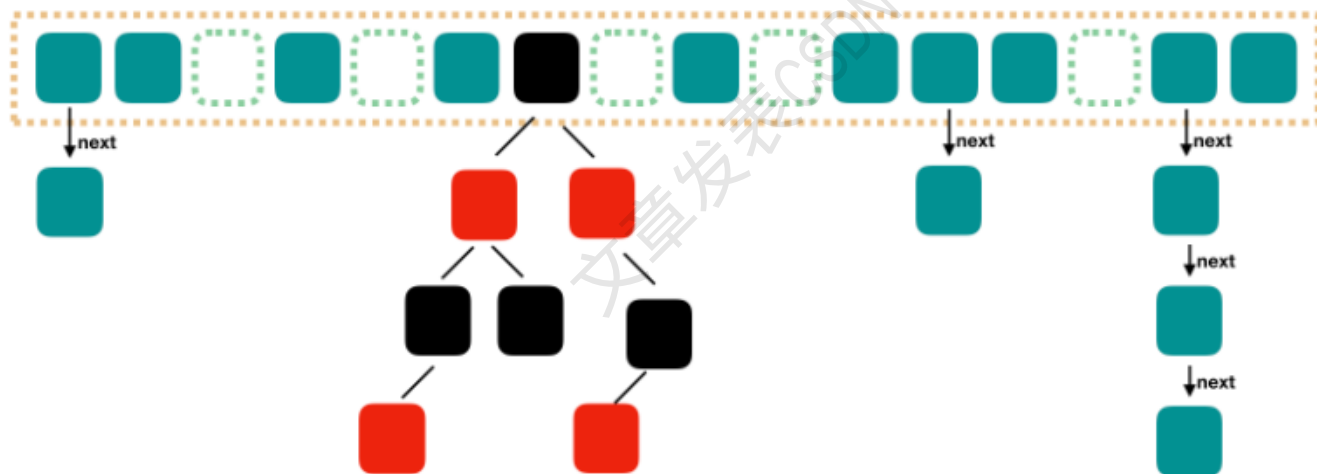
Java8 对 HashMap 进行了一些修改，最大的不同就是利用了红黑树，所以其由 数组+链表+红黑树 组成。

根据 Java7 HashMap 的介绍，我们知道，查找的时候，根据 hash 值我们能够快速定位到数组的具体下标，但是之后的话，需要顺着链表一个个比较下去才能找到我们需要的，时间复杂度取决于链表的长度，为 $O(n)$ 。

为了降低这部分的开销，在 Java8 中，当链表中的元素超过了 8 个以后，会将链表转换为红黑树，在这些位置进行查找的时候可以降低时间复杂度为 $O(\log N)$ 。

来一张图简单示意一下吧：

Java8 HashMap 结构

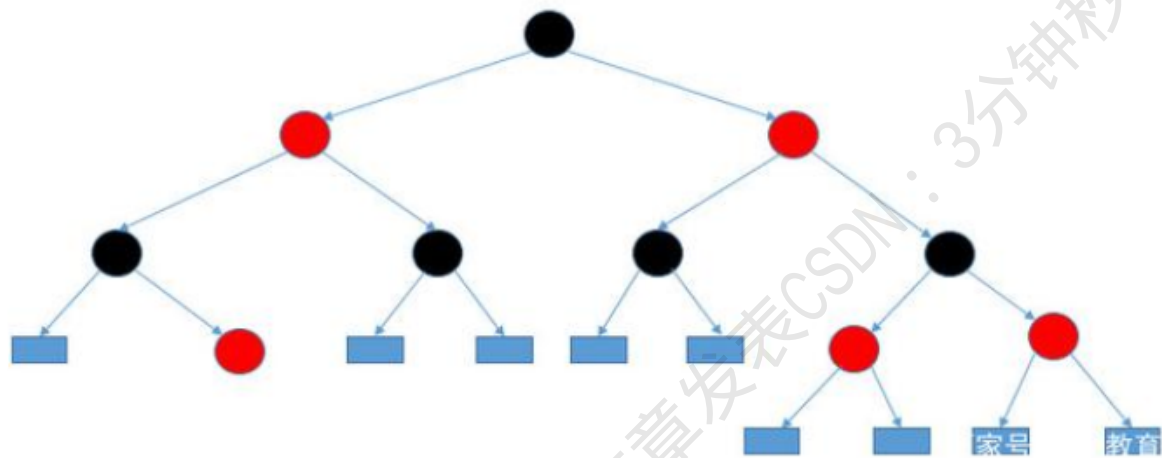


https://blog.csdn.net/weixin_38201936

TreeMap

1、TreeMap 的存储结构

TreeMap使用的存储结构是平衡二叉树，也称为红黑树。它首先是一棵二叉树，具有二叉树所有的特性，即树中的任何节点的值大于它的左子节点，且小于它的右子节点，如果是一棵左右完全均衡的二叉树，元素的查找效率将获得极大提高。最坏的情况就是一边倒，只有左子树或只有右子树，这样势必会导致二叉树的检索效率大大降低。为了维持二叉树的平衡，程序员们提出了各种实现的算法，其中平衡二叉树就是其中的一种算法。平衡二叉树的数据结构如下图所示：



TreeMap存储结构

https://blog.csdn.net/weixin_38201936

请介绍一下TreeMap

TreeMap是一个有序的key-value集合，基于红黑树（Red-Black tree）的 NavigableMap实现。该映射根据其键的自然顺序进行排序，或者根据创建映射时提供的 Comparator进行排序，具体取决于使用的构造方法。

TreeMap的特性：

根节点是黑色

每个节点都只能是红色或者黑色

每个叶节点（NIL节点，空节点）是黑色的。

如果一个节点是红色的，则它的两个子节点都是黑色的，也就是说在一条路径上不能出现两个红色的节点。

从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。

27. 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。基于你的collection的大小，也许向HashMap中添加元素会更快，将map换为TreeMap进行有序key的遍历。

28. 你用过HashMap 吗？

用过，HashMap可以接受null键值和值，而HashTable不能，如果HashTable的value存放null,则会报空指针异常。

HashMap是非Synchronized；HashMap很快，以及HashMap存储的是键值对。

那你说说HashMap的工作原理

HashMap基于hashing的原理，我们通过put()和get()方法存储和获取对象。当我们将键值对传递给put()方法时，它调用键对象的hashCode()方法来计算hashCode,然后找到bucket位置来存储值对象。当获取对象时，通过键对象的equals()方法，找到正确的键值对，然后返回值对象。

HashMap使用LinkedList来解决碰撞问题，当发生碰撞了，对象将会存储在LinkedList的下一个节点。HashMap在每一个LinkedList节点中存储键值对对象

当两个对象的Hashcode相同时，会发生什么？你如何获取值对象？

他们会存储在同一个bucket位置的LinkedList中，键对象的equal()方法用来找到键对象

调用get()方法，HashMap会使用键对象的HashCode找到Bucket位置，然后调用key.equals()方法去找到LinkedList中正确的节点，最终找到要找的值对象。

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

默认负载因子为0.75，当一个map填满了75%的bucket的时候，就会将原来的HashMap扩大2倍来重新调整map的大小，并将原来的对象放入新的bucket数组中。这个过程叫做rehashing。

重新调整HashMap大小会存在什么问题吗？

当在多线程情况下重新调整时，会存在条件竞争，因为两个线程都发现HashMap需要调整大小，它们会试着调整大小，在调整大小时，存储在LinkedList中的元素的次序会反过来，因为移动到新的bucket位置的时候，HashMap并不会将元素放在LinkedList的尾部，而是放在头部，为了避免尾部遍历，如果条件竞争了，就会死循环了。

Hash函数算法：

index=HashCode(Key)&(hashMap.length-1); 将key值得HashCode值与HashMap的长度-1进行位运算

29. 你用过HashTable 吗？

hashTable 数据结构 数组+链表 默认容量为11

put操作：首先进行索引计算(key.hashCode()&0x7FFFFFFF)%table.length;若在链表中找到了，则替换旧值，若未找到则继续，当总元素个数超过容量*加载因子时，扩大为原来的2倍并重新散列；将新元素加入到链表头部。

对修改HashTable内部共享数据的方法添加了Synchronized,保证了线程安全。

30. HashMap 和 Hashtable 有什么区别？

- hashMap没加锁，所以线程不安全，hashTable加锁 线程安全
- hashTable同步的，而HashMap是非同步的，效率上比hashTable要高。
- hashMap允许空键值，而hashTable不允许，如果hashTable 值为null,就报空指针异常。
- hashMap 初始值为16，负载因子为0.75 扩容为原来的2倍，hashTable 初始值为11，负载因子0.75 扩容为2倍，所以他两最后的扩容不同。
- hashMap去掉了HashTable 的contains方法，但是加上了containsValue () 和containsKey () 方法。

31. 迭代器 Iterator 是什么？

Iterator提供了统一遍历操作集合元素的统一接口, Collection接口实现Iterable接口,

每个集合都通过实现Iterable接口中iterator()方法返回Iterator接口的实例, 然后对集合的元素进行迭代操作.

有一点需要注意的是：在迭代元素的时候不能通过集合的方法删除元素, 否则会抛出ConcurrentModificationException 异常. 但是可以通过Iterator接口中的remove()方法进行删除.

32. Iterator 怎么使用？有什么特点？

Java中的Iterator功能比较简单，并且只能单向移动：

- (1) 使用方法iterator()要求容器返回一个Iterator。第一次调用Iterator的next()方法时，它返回序列的第一个元素。注意：iterator()方法是java.lang.Iterable接口，被Collection继承。
- (2) 使用next()获得序列中的下一个元素。
- (3) 使用hasNext()检查序列中是否还有元素。
- (4) 使用remove()将迭代器新返回的元素删除。

Iterator是Java迭代器最简单的实现，为List设计的ListIterator具有更多的功能，它可以从两个方向遍历List，也可以从List中插入和删除元素。

33. Iterator 和 ListIterator 有什么区别？

- Iterator可用来遍历Set和List集合，但是ListIterator只能用来遍历List。
- Iterator对集合只能是前向遍历，ListIterator既可以前向也可以后向。
- ListIterator实现了Iterator接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等等。

34.请说说快速失败(fail-fast)和安全失败(fail-safe)的区别？

Iterator的安全失败是基于对底层集合做拷贝，因此，它不受源集合上修改的影响。java.util包下面的所有的集合类都是快速失败的，而java.util.concurrent包下面的所有的类都是安全失败的。快速失败的迭代器会抛出ConcurrentModificationException异常，而安全失败的迭代器永远不会抛出这样的异常。

java7 ConcurrentHashMap 和 java8 ConcurrentHashMap

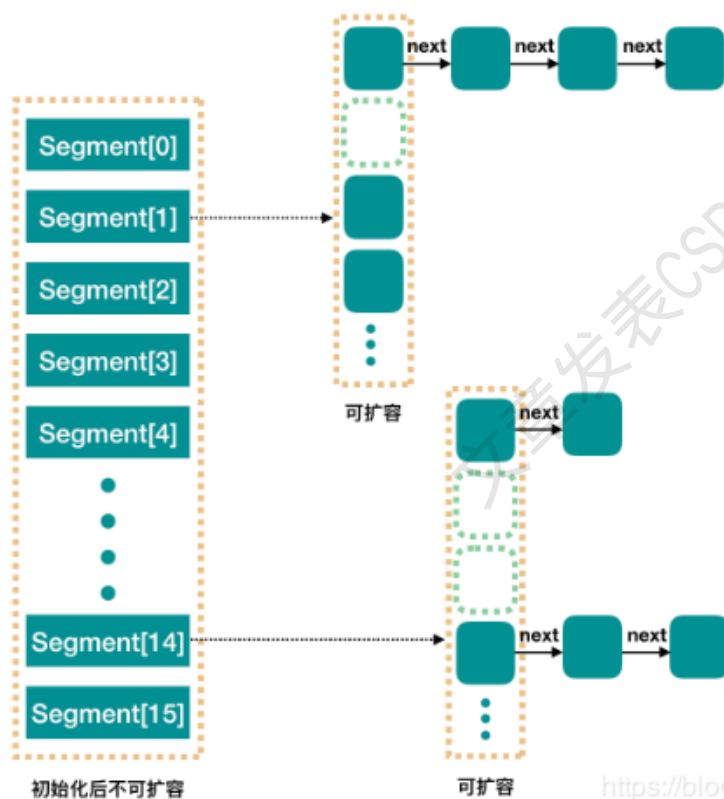
Java7 ConcurrentHashMap

ConcurrentHashMap 和 HashMap 思路是差不多的，但是因为它支持并发操作，所以要复杂一些。

整个 ConcurrentHashMap 由一个个 Segment 组成，Segment 代表“部分”或“一段”的意思，所以很多地方都会将其描述为分段锁。注意，行文中，我很多地方用了“槽”来代表一个 segment。

简单理解就是，ConcurrentHashMap 是一个 Segment 数组，Segment 通过继承 ReentrantLock 来进行加锁，所以每次需要加锁的操作锁住的是一个 segment，这样只要保证每个 Segment 是线程安全的，也就实现了全局的线程安全。

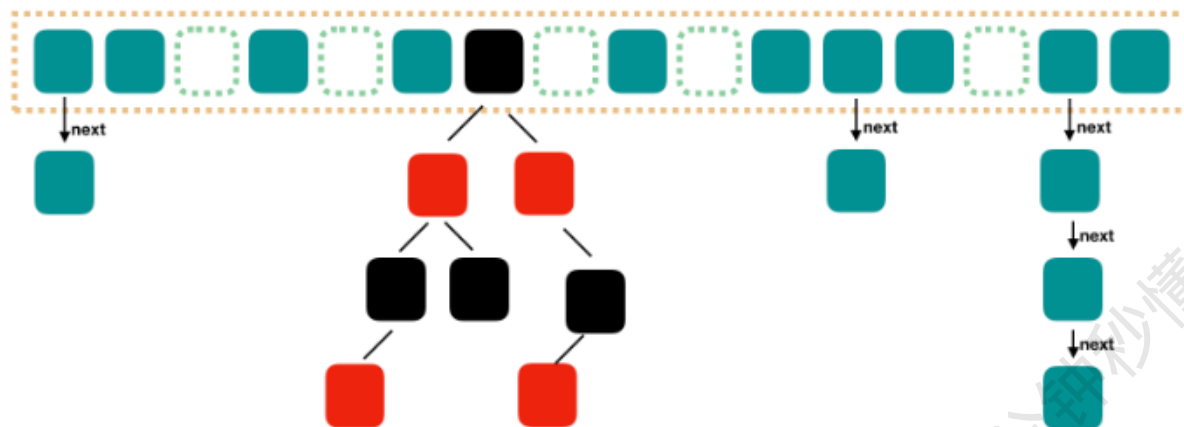
Java7 ConcurrentHashMap 结构



https://blog.csdn.net/weixin_38201936

- Segment 数组长度为 16, 不可以扩容
- Segment[i] 的默认大小为 2, 负载因子是 0.75, 得出初始阈值为 1.5, 也就是以后插入第一个元素不会触发扩容, 插入第二个会进行第一次扩容

Java8 ConcurrentHashMap 结构



结构上和 Java8 的 HashMap 基本上一样, 不过它要保证线程安全性, 所以在源码上确实要复杂一些。

https://blog.csdn.net/weixin_38201936

34.请你说明concurrenthashmap有什么优势以及1.7和1.8区别?

Concurrenthashmap线程安全的, 1.7是在jdk1.7中采用Segment + HashEntry的方式进行实现的, lock加在Segment上面。1.7size计算是先采用不加锁的方式, 连续计算元素的个数, 最多计算3次: 1、如果前后两次计算结果相同, 则说明计算出来的元素个数是准确的; 2、如果前后两次计算结果都不同, 则给每个Segment进行加锁, 再计算一次元素的个数;

1.8中放弃了Segment臃肿的设计, 取而代之的是采用Node + CAS + Synchronized来保证并发安全进行实现, 1.8中使用一个volatile类型的变量baseCount记录元素的个数, 当插入新数据或则删除数据时, 会通过addCount()方法更新baseCount, 通过累加baseCount和CounterCell数组中的数量, 即可得到元素的总个数;

35.请说一下ConcurrentHashMap的原理?

ConcurrentHashMap 类中包含两个静态内部类 HashEntry 和 Segment。HashEntry 用来封装映射表的键 / 值对；Segment 用来充当锁的角色，每个 Segment 对象守护整个散列映射表的若干个桶。每个桶是由若干个 HashEntry 对象链接起来的链表。一个 ConcurrentHashMap 实例中包含由若干个 Segment 对象组成的数组。HashEntry 用来封装散列映射表中的键值对。在 HashEntry 类中，key，hash 和 next 域都被声明为 final 型，value 域被声明为 volatile 型。

```
1 static final class HashEntry<K,V> {  
2     final K key;                // 声明 key 为 final 型  
3     final int hash;             // 声明 hash 值为 final 型  
4     volatile V value;           // 声明 value 为 volatile 型  
5     final HashEntry<K,V> next;  // 声明 next 为 final 型  
6  
7     HashEntry(K key, int hash, HashEntry<K,V> next, V value) {  
8         this.key = key;  
9         this.hash = hash;  
10        this.next = next;  
11        this.value = value;  
12    }  
13 }
```

在ConcurrentHashMap 中，在散列时如果产生“碰撞”，将采用“分离链接法”来处理“碰撞”：把“碰撞”的 HashEntry 对象链接成一个链表。由于 HashEntry 的 next 域为 final 型，所以新节点只能在链表的表头处插入。

36.请说一下ConcurrentHashMap和HashMap的区别？

- 1、HashMap 不是线程安全的，而 ConcurrentHashMap 是线程安全的。
- 2、ConcurrentHashMap 采用锁分段技术，将整个 Hash 桶进行了分段 segment，也就是将这个大的数组分成了几个小的片段 segment，而且每个小的片段 segment 上面都有锁存在，那么在插入元素的时候就需要先找到应该插入到哪一个片段 segment，然后再在这个片段上面进行插入，而且这里还需要获取 segment 锁。
- 3、ConcurrentHashMap 让锁的粒度更精细一些，并发性能更好。

37 Hashtable 和 ConcurrentHashMap 的区别

它们都可以用于多线程的环境，但是当 Hashtable 的大小增加到一定的时候，性能会急剧下降，因为迭代时需要被锁定很长的时间。因为 ConcurrentHashMap 引入了分割 (segmentation)，不论它变得多么大，仅仅需要锁定 map 的某个部分，而其它的线程不需要等到迭代完成才能访问map。简而言之，在迭代的过程中，ConcurrentHashMap仅仅锁定map的某个部分，而Hashtable则会锁定整个map。

线程、多线程、线程池

38 线程从创建到死亡有哪些状态？

- 1.初始状态 (new) :新建一个线程
- 2.就绪状态：其他线程调用了这个线程的start()方法，这个线程被放到可运行的线程池中，等待获取CPU的使用权。
- 3.运行状态：获得CPU进入运行状态
- 4.阻塞状态：当调用sleep()方法或者调用了wait()方法等待某个通知，此时线程处于阻塞状态。阻塞状态是指线程因为某种原因放弃了 cpu 使用权，也即让出了 cpu timeslice，暂时停止运行。直到线程进入可运行(runnable)状态，才有 机会再次获得 cpu timeslice 转到运行(running)状态。阻塞的情况分三种：
 - (一). 等待阻塞：运行(running)的线程执行 o . wait ()方法， JVM 会把该线程放 入等待队列(waiting queue)中。
 - (二). 同步阻塞：运行(running)的线程在获取对象的同步锁时，若该同步锁 被别的线程占用，则 JVM 会把该线程放入锁池(lock pool)中。
 - (三). 其他阻塞: 运行(running)的线程执行 Thread . sleep (long ms)或 t . join ()方法，或者发出了 I / O 请求时， JVM 会把该线程置为阻塞状态。当 sleep ()状态超时、 join ()等待线程终止或者超时、或者 I / O 处理完毕时，线程重新转入可运行(runnable)状态。
- 5.终止状态：线程执行完所有指令序列，便进入终止状态。

39 Thread类中的start()方法和run()方法有什么区别？

start()方法被用来启动新创建的线程，而且start()内部调用了run()，这和直接调用run()方法的效果不一样，而调用run()方法时，只会在原线程中调用，没有新的线程启动，start()方法才会启动新的线程。

40.wait和sleep的区别

sleep()方法是属于Thread类中的，而wait方法则是属于object类中的

sleep()方法导致程序暂停执行指定的时间，让出CPU给其他线程，但他的监控状态依旧保持着，当指定的时间到了之后又会自动恢复运行状态，所以调用sleep()方法的过程中，线程不会释放对象锁。

调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后，本线程才能进入对象锁定池准备获取对象锁进入运行状态。

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。

线程的sleep()方法和yield()方法有什么区别？

- ① sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；
- ② 线程执行sleep()方法后转入阻塞（blocked）状态，而执行yield()方法后转入就绪（ready）状态；
- ③ sleep()方法声明抛出InterruptedException，而yield()方法没有声明任何异常；
- ④ sleep()方法比yield()方法（跟操作系统CPU调度相关）具有更好的可移植性。

stop()和suspend()方法为何不推荐使用，请说明原因？

反对使用stop()，是因为它不安全。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在。suspend()方法容易发生死锁。调用suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被“挂起”的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。所以不应该使用suspend()，而应在自己的Thread类中置入一个标志，指出线程应该活动还是挂起。若标志指出线程应该挂起，便用 wait()命其进入等待状态。若标志指出线程应当恢复，则用一个notify()重新启动线程。

说一下 runnable 和 callable 有什么区别？

有点深的问题了，也看出一个Java程序员学习知识的广度。

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

41.创建线程有几种不同的方式？你喜欢哪一种？为什么？

有三种：

一.继承java.lang.Thread类，

- (1) 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。
- (2) 创建Thread子类的实例，即创建了线程对象。
- (3) 调用线程对象的start()方法来启动该线程。

```
1 package com.thread;
2 public class FirstThreadTest extends Thread{
3     int i = 0;
4     // 重写run方法，run方法的方法体就是现场执行体
5     public void run()
6     {
7         for(;i<100;i++){
8             System.out.println(getName()+" "+i);
9         }
10    }
11 }
12 public static void main(String[] args)
13 {
14     for(int i = 0;i< 100;i++)
15     {
16         System.out.println(Thread.currentThread().getName()+" : "+i);
```

```
17         if(i==20)18         {
19             new FirstThreadTest().start();
20             new FirstThreadTest().start();
21         }
22     }
23 }
24
25 }
```

二.实现java.lang.Runnable接口

- (1) 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。
- (2) 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。
- (3) 调用线程对象的start()方法来启动该线程。

```
1 package com.thread;
2
3 public class RunnableThreadTest implements Runnable
4 {
5
6     private int i;
7     public void run()
8     {
9         for(i = 0;i <100;i++)
10         {
11             System.out.println(Thread.currentThread().getName()+" "+i);
12         }
13     }
14     public static void main(String[] args)
15     {
16         for(int i = 0;i < 100;i++)
17         {
18             System.out.println(Thread.currentThread().getName()+" "+i);
19             if(i==20)
```

```
20     {
21         RunnableThreadTest rtt = new RunnableThreadTest();
22         new Thread(rtt,"新线程1").start();
23         new Thread(rtt,"新线程2").start();
24     }
25 }
26
27 }
28
29 }
```

三.通过Callable和Future创建线程

- (1) 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。
- (2) 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。
- (3) 使用FutureTask对象作为Thread对象的target创建并启动新线程。
- (4) 调用FutureTask对象的get()方法来获得子线程执行结束后的返回值

```
1 package com.thread;
2 import java.util.concurrent.Callable;
3 import java.util.concurrent.ExecutionException;
4 import java.util.concurrent.FutureTask;
5
6 public class CallableThreadTest implements Callable<Integer>
7 {
8
9     public static void main(String[] args)
10    {
11        CallableThreadTest ctt = new CallableThreadTest();
12        FutureTask<Integer> ft = new FutureTask<>(ctt);
13        for(int i = 0;i < 100;i++)
14        {
15            System.out.println(Thread.currentThread().getName()+" 的循环变量i的值"+i);
```

```
16         if(i==20)
17             {
18                 new Thread(ft,"有返回值的线程").start();
19             }
20     }
21     try
22     {
23         System.out.println("子线程的返回值: "+ft.get());
24     } catch (InterruptedException e)
25     {
26         e.printStackTrace();
27     } catch (ExecutionException e)
28     {
29         e.printStackTrace();
30     }
31 }
32
33
34 @Override
35 public Integer call() throws Exception
36 {
37     int i = 0;
38     for(;i<100;i++)
39     {
40         System.out.println(Thread.currentThread().getName()+" "+i);
41     }
42     return i;
43 }
44
45 }
```

实现Runnable接口这种方式更受欢迎，因为这不需继承Thread类。在应用设计中已经继承了别的对象的情况下，这需要多继（而Java不支持多继承），只能实现接口。同时，线程池也是非常高效的，很容易实现和使用。

42.线程与进程的区别？

线程是操作系统能够进行运算调度的最小单元。它被包含在进程之中，是进程中的实际运作单位。一个进程可以有多个线程。每个线程并行执行不同的任务，所有线程共享一片相同的内存空间。

43.什么是线程安全？

在多线程环境下，无论多个线程如何访问目标对象，目标对象的状态始终保持一致，线程的行为也总是正确，则证明线程安全

44.如何保证线程安全？

通过合理的时间调度，避开共享资源的存取冲突。另外，在并行任务设计上可以通过适当的策略，保证任务与任务之间不存在共享资源，设计一个规则来保证一个客户的计算工作和数据访问只会被一个线程或一台工作机完成，而不是把一个客户的计算工作分配给多个线程去完成。

45.请使用内部类实现线程设计4个线程，其中两个线程每次对j增加1，另外两个线程对j每次减少1

```
1 public class ThreadTest1{
2     private int j;
3     public static void main(String args[]){
4         ThreadTest1 tt=new ThreadTest1();
5         Inc inc=tt.new Inc();
6         Dec dec=tt.new Dec();
7         for(int i=0;i<2;i++){
8             Thread t=new Thread(inc);
9             t.start();
10            t=new Thread(dec);
11            t.start();
12        }
13    }
14    private synchronized void inc(){
15        j++;
16        System.out.println(Thread.currentThread().getName()+"-inc:"+j);
17    }
18    private synchronized void dec(){
19        j--;
20        System.out.println(Thread.currentThread().getName()+"-dec:"+j);
21    }
22    class Inc implements Runnable{
```

```
23 |         public void run(){
24 |             for(int i=0;i<100;i++){
25 |                 inc();
26 |             }
27 |         }
28 |     }
29 |     class Dec implements Runnable{
30 |         public void run(){
31 |             for(int i=0;i<100;i++){
32 |                 dec();
33 |             }
34 |         }
35 |     }
36 | }
```

46 请说明一下线程中的同步和异步有何异同？并且请举例说明在什么情况下会使用到同步和异步？

同步：数据将在线程间共享。 银行多线程对一个账户进行存取钱。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

多线程并发时，多个线程同时请求同一个资源，必然导致此资源的数据不安全，A线程修改了B线程处理的数据，而B线程又修改了A线程处理的数据。显然是由于全局资源造成的，有时为了解决此问题，优先考虑使用局部变量，退而求其次使用同步代码块，出于这样的安全考虑就必须牺牲系统处理性能，加在多线程并发时资源争夺最激烈的地方，这就实现了线程的同步机制

同步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为同步机制存在，A线程请求不到，怎么办，A线程只能等待下去

异步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为没有同步机制存在，A线程仍然请求的到，A线程无需等待

47.请说出线程同步的方法

考察点：线程同步

wait():使一个线程处于等待状态，并且释放所持有的对象的lock。

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException异常。

notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先

级。

Allnotity():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

48.请你说明一下在监视器(Monitor)内部，是如何做到线程同步的？在程序又应该做哪种级别的同步呢？

考察点：JAVA线程同步

监视器和锁在Java虚拟机中是一块使用的。监视器监视一块同步代码块，确保一次只有一个线程执行同步代码块。每一个监视器都和一个对象引用相关联。线程在获取锁之前不允许执行同步代码。

49 分析一下同步方法和同步代码块的区别是什么？

考察点：JAVA代码块同步

区别：

同步方法默认用this或者当前类class对象作为锁；

同步代码块可以选择以什么来加锁，比同步方法要更细颗粒度，我们可以选择只同步会发生同步问题的部分代码而不是整个方法。

50 请简要说明一下JAVA中cyclicbarrier和countdownlatch的区别分别是什么？

考察点：线程

CountDownLatch和CyclicBarrier都能够实现线程之间的等待，只不过它们侧重点不同：

CountDownLatch一般用于某个线程A等待若干个其他线程执行完任务之后，它才执行；

而CyclicBarrier一般用于一组线程互相等待至某个状态，然后这一组线程再同时执行；

另外，CountDownLatch是不能够重用的，而CyclicBarrier是可以重用的。

多线程

51. 请简述一下实现多线程同步的方法？

考察点：多线程

可以使用synchronized、lock、volatile和ThreadLocal来实现同步。

52 多线程中的i++线程安全吗？请简述一下原因？

考察点：多线程

不安全。i++不是原子性操作。i++分为读取i值，对i值加一，再赋值给i++，执行期中任何一步都是有可能被其他线程抢占的。

53 如何在线程安全的情况下实现一个计数器？

考察点：多线程

可以使用加锁，比如synchronized或者lock。也可以使用Concurrent包下的原子类。

54 请解释一下Java多线程回调是什么意思？

考察点：JAVA多线程

所谓回调，就是客户程序C调用服务程序S中的某个方法A，然后S又在某个时候反过来调用C中的某个方法B，对于C来说，这个B便叫做回调方法

请简短说明一下你对AQS的理解。

考察点：多线程

AQS其实就是一个可以给我们实现锁的框架

内部实现的关键是：先进先出的队列、state状态

定义了内部类ConditionObject

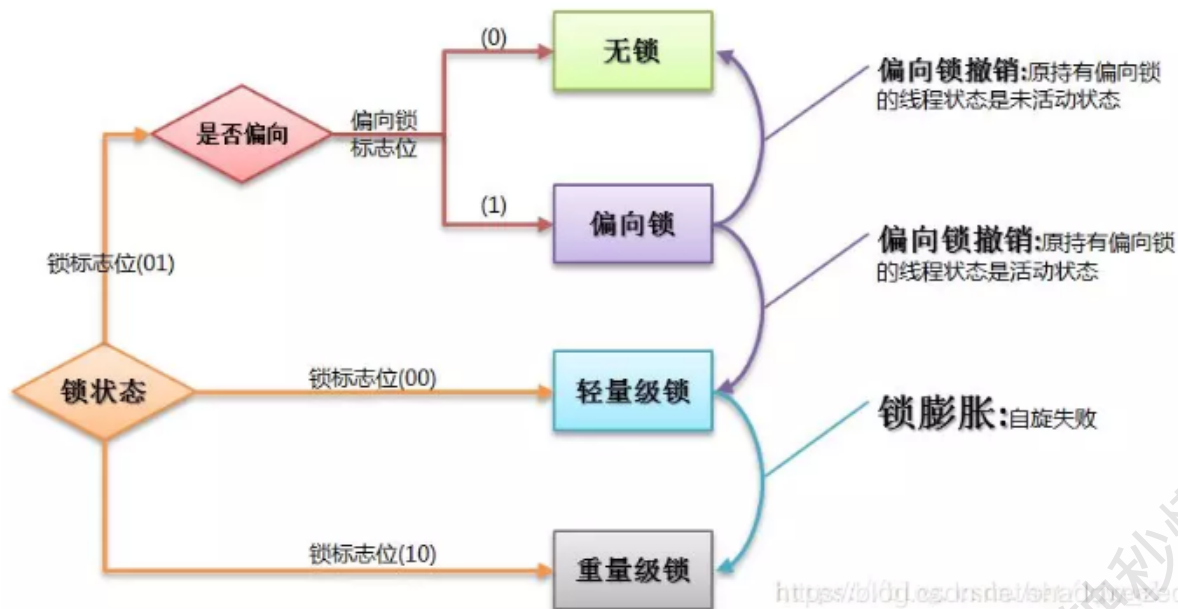
拥有两种线程模式独占模式和共享模式。

在LOCK包中的相关锁(常用的有ReentrantLock、ReadWriteLock)都是基于AQS来构建，一般我们叫AQS为同步器。

多线程锁的升级原理是什么？

在Java中，锁共有4种状态，级别从低到高依次为：无状态锁，偏向锁，轻量级锁和重量级锁状态，这几个状态会随着竞争情况逐渐升级。锁可以升级但不能降级。

锁升级的图示过程：



线程池

55 请你解释一下什么是线程池 (thread pool) ?

考察点：线程池

参考回答：

在面向对象编程中，创建和销毁对象是很费时间的，因为创建一个对象要获取内存资源或者其它更多资源。在Java中更是如此，虚拟机将试图跟踪每一个对象，以便能够在对象销毁后进行垃圾回收。所以提高服务程序效率的一个手段就是尽可能减少创建和销毁对象的次数，特别是一些很耗资源的对象创建和销毁，这就是“池化资源”技术产生的原因。

线程池顾名思义就是事先创建若干个可执行的线程放入一个池（容器）中，需要的时候从池中获取线程不用自行创建，使用完毕不需要销毁线程而是放回池中，从而减少创建和销毁线程对象的开销。

56 线程池的种类都有哪些？线程池的实现过程

线程池的种类有：

newCachedThreadPool 可缓存线程池, **newFixedThreadPool** 指定工作线程数量的线程,

newScheduledThreadPool 定长线程池 **newSingleThreadExecutor** 单线程化的线程池

Java通过Executors提供四种线程池,

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。

57 请说明一下线程池有什么优势?

考察点: 线程池

第一: 降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。

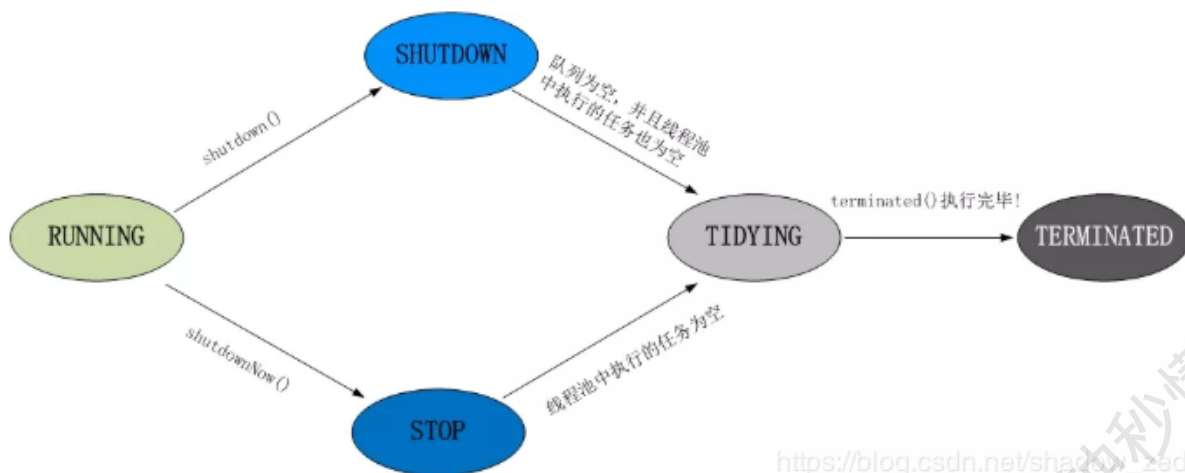
第二: 提高响应速度。当任务到达时, 任务可以不需要等到线程创建就能立即执行。

第三: 提高线程的可管理性

线程池都有哪些状态?

线程池有5种状态：Running、ShutDown、Stop、Tidying、Terminated。

线程池各个状态切换框架图：



58.请简述一下线程池的运行流程，使用参数以及方法策略等

考察点：线程池

线程池主要就是指定线程池核心线程数大小，最大线程数，存储的队列，拒绝策略，空闲线程存活时长。当需要任务大于核心线程数时候，就开始把任务往存储任务的队列里，当存储队列满了的话，就开始增加线程池创建的线程数量，如果当线程数量也达到了最大，就开始执行拒绝策略，比如说记录日志，直接丢弃，或者丢弃最老的任务。

线程池中 submit()和 execute()方法有什么区别？

- 接收的参数不一样
- submit有返回值，而execute没有
- submit方便Exception处理

关键字、锁

锁的作用：

锁是多线程软件开发的必要工具之一，它的基本作用是保护临界区资源不会被多个线程同时访问而受到破坏。

59.synchronized, Lock, ReentrantLock, ReadWriteLock, stampedLock 的区别

- 1.synchronized：同步锁，用于同步方法和代码块 执行完后自动释放锁，可用object.wait() object.notify() 来操作线程等待唤醒
- 2.Lock:锁，提供获取锁和解锁的方法 (lock, trylock,unlock)
- 3.ReentrantLock:重入锁 Lock是ReentrantLock的接口，它是Lock的实现类
- 4.ReadWriteLock 读入锁 接口的核心方法是readLock(),writeLock(),实现了并发读，互斥写，但读锁会阻塞写锁，是悲观锁的策略，它的实现类是ReenTrantReadWriteLock 可重入读写锁，读锁共享，写锁独占，读与读不互斥，读和写，写与写才互斥

60.简单介绍一下Lock

Lock:使用Lock时，它不会像synchronized执行完成之后或者抛出异常之后，自动释放锁，而是需要你主动释放锁，所以我们必须在使用Lock的时候加上try{}Catch{}finally{}块，并且在finally中释放占用的锁资源。

61.synchronized 和 Lock 的区别

区别：

- 1 synchronized 是java内置关键字，在JVM层面，Lock是个java类
- 2 synchronized 无法判断是否获取锁的状态，Lock可以判断是否获取到锁
- 3 synchronized 自动释放锁 Lock 需要在finally中手工释放锁，否则容易造成线程死锁
- 4最大区别： synchronized 一个线程抢到锁之后，其他线程会一直等待下去

Lock锁，不一定会继续等待下去，如果尝试获取不到锁，就会做其他业务。

62.synchronized 和 ReentrantLock 的区别

- 1 synchronized 是java内置关键字，在JVM层面，ReentrantLock是个java类

2 ReentrantLock包含等待可中断锁，意味着持有锁的线程长期不释放锁，正在等待的线程可以选择放弃等待，可以避免死锁

3 synchronized 自动释放锁 ReentrantLock需要在finally中手工释放锁，否则容易造成线程死锁 Lock.unlock()

4 ReentrantLock可以实现公平锁，公平锁就是先等待的线程先获得锁。

63 对于synchronized关键字的理解

解决多线程之间访问资源你的同步性

保证被他修饰的方法或者代码块在任意时刻只能有一个线程执行，执行完后自动释放锁

64.如何使用synchronized关键字

三种使用方式：

- 1.修饰实例方法，作用于当前对象实例加锁，进入同步代码前要获得当前对象实例的锁
- 2.修饰静态方法：作用于当前类对象加锁，进入同步代码前要获得当前类对象的锁
- 3.修饰代码块：指定加锁对象，对给定对象加锁，进入同步代码块前要获得给定对象的锁

65.synchronized的底层原理

1 底层属于JVM层面 同步代码块情况

```
1 public class SynchronizedDemo{
2
3     public void method(){
4         synchronized(this){
5             System.out.println("synchronized代码块");
6         }
7     }
8 }
```

synchronized 同步代码块使用的monitorenter 和monitorexit 指令 其中 monitorenter 指向开始位置, monitorexit指向结束位置, 执行monitorenter 指令时, 线程试图获取锁, 计数器从0到1, 在执行monitorexit指令时, 锁计数器从1设为0, 表明锁释放。

2 修饰方法的情况

```
1 public class SynchronizedDemo{
2
3     public synchronized void method(){
4
5         System.out.println("synchronized代码块");
6
7     }
8 }
```

synchronized 修饰的方法并没有monitorenter 和monitorexit 指令,取而代之的是ACC_synchronized标识。该标识指明了该方法是一个同步方法, JVM通过该标识来判别一个方法是否为同步方法, 从而执行相应的同步调用。

66.synchronized和volatile的区别

- 1.volatile是线程同步的轻量级实现, 性能比synchronized好, 且它只能用于变量
- 2多线程访问volatile不会发生阻塞, 而synchronized可能会阻塞。
- 3.volatile能保证数据的可见性, 但不能保证原子性, synchronized两者都能保证
- 4.volatile主要解决变量在多个线程之间的可见性, 而synchronized解决多线程之间访问资源的同步性。

67.请问什么是死锁(deadlock)?

死锁是指多个线程因竞争资源而造成的一种僵局(互相等待), 若无外力作用, 这些进程都将无法向前推进。

例如, 如果线程1锁住了A, 然后尝试对B进行加锁, 同时线程2已经锁住了B, 接着尝试对A进行加锁, 这时死锁就发生了。线程1永远得不到B, 线程2也永远得不到A, 并且它们永远也不会知道发生了这样的事情。为了得到彼此的对象(A和B), 它们将永远阻塞下去。这种情况就是一个死锁。

68.死锁产生的原因

1. 系统资源的竞争

系统资源的竞争导致系统资源不足，以及资源分配不当，导致死锁。

2. 进程运行推进顺序不合适

进程在运行过程中，请求和释放资源的顺序不当，会导致死锁。

69.死锁的四个必要条件

互斥条件：一个资源每次只能被一个进程使用，即在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。

请求与保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

不可剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放（只能是主动释放）。

循环等待条件：若干进程间形成首尾相接循环等待资源的关系

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。

70. JAVA中如何确保N个线程可以访问N个资源，但同时又不导致死锁？

考察点：死锁

参考回答：

使用多线程的时候，一种非常简单的避免死锁的方式就是：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。因此，如果所有的线程都是以同样的顺序加锁和释放锁，就不会出现死锁了。

预防死锁，预先破坏产生死锁的四个条件。互斥不可能破坏，所以有如下三种方法：

1. 破坏“不可剥夺”条件：一个进程不能获得所需要的全部资源时便处于等待状态，等待期间他占有的资源将被隐式的释放重新加入到系统的资源列表中，可以被其他的进程使用，而等待的进程只有重新获得自己原有的资源以及新申请的资源才可以重新启动，执行。
2. 破坏“请求与保持条件”：第一种方法静态分配即每个进程在开始执行时就申请他所需要的全部资源。第二种是动态分配即每个进程在申请所需要的资源时他本身不占用系统资源。

3. 破坏“循环等待”条件：采用资源有序分配其基本思想是将系统中的所有资源顺序编号，将紧缺的，稀少的采用较大的编号，在申请资源时必须按照编号的顺序进行，一个进程只有获得较小编号的进程才能申请较大编号的进程。

71.请讲一下非公平锁和公平锁在reentrantlock里的实现过程是怎样的。

考察点：锁

参考回答：

如果一个锁是公平的，那么锁的获取顺序就应该符合请求的绝对时间顺序，FIFO。对于非公平锁，只要CAS设置同步状态成功，则表示当前线程获取了锁，而公平锁还需要判断当前节点是否有前驱节点，如果有，则表示有线程比当前线程更早请求获取锁，因此需要等待前驱线程获取并释放锁之后才能继续获取锁。

58. 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。

反射

59.什么是反射？

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语言编译之后会生成一个.class文件，反射就是通过字节码文件找到某一个类、类中的方法以及属性等。反射的实现主要借助以下四个类：Class：类的对象，Constructor：类的构造方法，Field：类中的属性对象，Method：类中的方法对象。

作用：反射机制指的是程序在运行时能够获取自身的信息。在JAVA中，只要给定类的名字，那么就可以通过反射机制来获取类的所有信息。

Java反射：

在Java运行时环境中，对于任意一个类，能否知道这个类有哪些属性和方法？对于任意一个对象，能否调用它的任意一个方法

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。

60. 什么是 java 序列化？ 什么情况下需要序列化？

对象序列化是一个用于将对象状态转换为字节流的过程，可以将其保存到磁盘文件中或通过网络发送到任何其他程序；从字节流创建对象的相反的过程称为反序列化。而创建的字节流是与平台无关的，在一个平台上序列化的对象可以在不同的平台上反序列化。

在Java中创建的对象，只要没有被回收就可以被复用，但是，创建的这些对象都是存在于JVM的堆内存中，JVM处于运行状态时候，这些对象可以复用，但是一旦JVM停止，这些对象的状态也就丢失了。

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；

如何使Java类可序列化？

通过实现java.io.Serializable接口，可以在Java类中启用可序列化。它是一个标记接口，意味着它不包含任何方法或字段，仅用于标识可序列化的语义。

61. 动态代理是什么？ 有哪些应用？

动态代理：

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。

动代理的应用：

- Spring的AOP
- 加事务
- 加权限
- 加日志

62. 怎么实现动态代理?

首先必须定义一个接口，还要有一个InvocationHandler(将实现接口的类的对象传递给它)处理类。再有一个工具类Proxy(习惯性将其称为代理类，因为调用他的newInstance()可以产生代理对象,其实他只是一个产生代理对象的工具类)。利用到InvocationHandler，拼接代理类源码，将其编译生成代理类的二进制码，利用加载器加载，并将其实例化产生代理对象，最后返回。

转接下个链接：<https://mp.csdn.net/postedit>



MBA排名MBA院校排行榜100强

MBA学校排名