**JAVA笔试题**

1. 某服务节举办的一次比武大赛中，来自东北战区、西北战区、西南战区、东南战区和华东战区的5名服务顾问(每战区1名)相遇在一起，有张三、李四、王五、马六、洪七，5名服务顾问之间的比赛信息满足以下条件：  
   (1)张三仅与2名选手比赛过。  
   (2)华东战区的选手和3名选手比赛过。  
   (3)李四不是西北战区的，也没有和西北战区的选手对阵过。  
   (4)西南战区的选手和王五比赛过。  
   (5)西北战区，西南战区，东南战区的选手互相都交过手。  
   (6)马六仅与1名选手比赛过。  
   根据以上条件，请问王五来自哪个战区（ 东南 ）
2. Java的四个基本特性，包括继承、抽象、继承、多态。

继承是\_子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。。

1. Java重载在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。

每个重载的方法（或者构造函数）都必须有一个独一无二的参数类型列表;

重写子类对父类的允许访问的方法的实现过程进行重新编写, 返回值和形参都不能改变。**即外壳不变，核心重写！**。

1. Java访问控制符：

Public 对所有类可见。使用对象：类、接口、变量、方法；

Private 在同一类内可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）**；

Protected 对同一包内的类和所有子类可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）**。

1. Java”==”和equal的区别

**”==”**

* 当比较的变量是基本类型，且都是数值类型，且’=='比较的是其数值。
* 当比较的是引用变量，只有当他们都指向同一个对象的时候才会返回true。
* 其不可以用去在类型上比较没有继承关系的的两个变量，编译器会报错。

**Equal**

* 该方法属于object类，因此所有引用变量都可以调用这个方法来判断是否与其他引用变量相等，但是如果这样的话该方法和"=="来判断两个对象相等的标准和使用没有区别。
* 很多时候，程序判断两个引用变量是否相等，也希望有一种类似于”值相等“的判断规则，并不严格要求两个引用变量指向同一个对象。这时候可以通过重写equal方法。
* String类就重新了equal方法，该方法可以判断两个字符串对象所引用的字符串直接量是否相等。

1. Java实例变量和static静态变量区别，程序运行时的区别
2. **生命周期不同**

实例变量随着对象的创建而存在，随着对象的回收而释放。

静态变量随着类的加载而存在，随着类的消失而消失。

1. **调用方式不同**

实例变量只能被对象调用。  
静态变量可以被对象调用，还可以被类名调用

1. **数据存储位置不同**

实例变量存储在堆内存的对象中，所以也叫对象的特有数据。  
静态变量数据存储在方法区（共享数据区）的静态区，所以也叫对象的共享数据。

1. Java自动装箱和拆箱：

装箱\_\_\_自动将基本数据类型转换为包装器类型;

拆箱\_\_\_自动将包装器类型转换为基本数据类型。

1. 什么情况会造成内存泄漏

static字段引起的内存泄漏；

未关闭的资源导致内存泄漏；

不正确的equals()和hashCode()；

引用了外部类的内部类；

finalize方法导致的内存泄漏；

常量字符串造成的内存泄漏；

改变哈希值；

变量不合理的作用域；

**等等... ...**

1. Java集合类：

List\_\_\_元素按进入先后有序保存，可重复;

set\_\_\_\_仅接收一次，不可重复，并做内部排序;

map\_\_\_\_键值对的集合 （双列集合）;

1. Java中ArrayList和linkedlist区别，

ArrayList：

基于动态数组的数据结构

使用在查询比较多，但是插入和删除比较少的情况

linkedlist：

基于链表结构实现。

用在查询比较少而插入删除比较多的情况

1. Java多线程的实现方法

四种方式，一般本地调试情况下只用前两种，实际开发建议使用线程池：

继承Thread类创建线程

实现Runnable接口创建线程

实现Callable接口通过FutureTask包装器来创建Thread线程

使用线程池接口ExecutorService结合Callable、Future实现有返回结果的线程

1. Java多线程wait()和sleep()区别，

通常，我们应该使用sleep（）来控制一个线程的执行时间，使用wait（）来进行多线程同步。

**Object.wait（）**是一个用于线程同步的实例方法。

它可以在任何对象上调用，因为它在java.lang.Object上定义，但它只能从synchronized块中调用。它释放对象的锁定，以便另一个线程可以跳入并获取锁。

**Thread.sleep（）**是一个可以从任何上下文调用的静态方法。

暂停当前线程并且不释放任何锁。

当我们使用sleep（）方法时，线程在指定的时间间隔后启动，除非它被中断。

对于wait（）可以通过调用正在等待的监视器上的notify（）或notifyAll（）方法来唤醒线程。

1. Spring IOC的理解

全称为 Inversion of Control，翻译为 “控制反转”，它还有一个别名为 DI（Dependency Injection）,即依赖注入。Spring IOC 容器来负责对象的生命周期和对象之间的关系。

从四个方面理解：

**A.谁控制谁：**在传统的开发模式下，我们都是采用直接 new 一个对象的方式来创建对象，也就是说你依赖的对象直接由你自己控制，但是有了 IOC 容器后，则直接由 IoC 容器来控制。所以“谁控制谁”，当然是 IoC 容器控制对象。

**B.控制什么：**控制对象。

**C.为何是反转：**没有 IoC 的时候我们都是在自己对象中主动去创建被依赖的对象，这是正转。但是有了 IoC 后，所依赖的对象直接由 IoC 容器创建后注入到被注入的对象中，依赖的对象由原来的主动获取变成被动接受，所以是反转。

**D.哪些方面反转了：**所依赖对象的获取被反转了。

**IOC注入方式：**构造方法注入、stter方法注入、接口注入。

1. Spring的AOP理解

全称Aspect Oriented Programming，即面向切面编程

**切面实现了横切关注点的模块化：**

AOP就是希望将这些分散在各个业务逻辑代码中的相同代码，通过横向切割的方式抽取到一个独立的模块中，让业务逻辑类依然保存整洁。

**Spring 中的 AOP 是通过动态代理实现的：**

Spring AOP 不能拦截对对象字段的修改，也不支持构造器连接点,我们无法在 Bean 创建时应用通知。

**定义一个切面：（伪代码示例）：**

@Aspect

@Component

public class BuyAspectJ {

@Before("execution(\* com.sharpcj.aopdemo.test1.IBuy.buy(..))")

public void haha(){

System.out.println("男孩女孩都买自己喜欢的东西");

}

}

1. #{}和${}的区别（）

#{}:表示一个占位符号，通过#{}可以实现preparedStatement向占位符中设置值，自动进行java类型和jdbc类型转换，#{}可以有效防止sql注入。 #{}可以接收简单类型值或pojo属性值。 如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。

${}:表示拼接sql串，通过${}可以将parameterType 传入的内容拼接在sql中且不进行jdbc类型转换， ${}可以接收简单类型值或pojo属性值，如果parameterType传输单个简单类型值，${}括号中只能是value。

1. 快速排序算法

*/\*\*  
 \* 递归方法-快速排序  
 \*  
 \** ***@param*** *array 原数组  
 \** ***@param*** *startIndex 开始下标  
 \** ***@param*** *endIndex 结束下标  
 \*/*public static void quickSort(int[] array, int startIndex, int endIndex) {  
 // 递归结束条件：startIndex大于或等于endIndex时  
 if (startIndex >= endIndex) {  
 return;  
 }  
  
 // 得到基准元素位置  
 int pivotIndex = *partition2*(array, startIndex, endIndex);  
 // 根据基准元素，分成两部分进行递归排序  
 *quickSort*(array, startIndex, pivotIndex - 1);  
 *quickSort*(array, pivotIndex + 1, endIndex);  
}

*/\*\*  
 \* 分治算法（单边循环）  
 \*  
 \** ***@param*** *array 原数组  
 \** ***@param*** *startIndex 起始下标  
 \** ***@param*** *endIndex 结束下标  
 \** ***@return*** *\*/*public static int partition2(int[] array, int startIndex, int endIndex) {  
 // 选取基准元素  
 int pivot = array[startIndex];  
 int mark = startIndex;  
 for (int i = startIndex + 1; i <= endIndex; i++) {  
 if (array[i] < pivot) {  
 mark++;  
 int p = array[mark];  
 array[mark] = array[i];  
 array[i] = p;  
 }  
 }  
 array[startIndex] = array[mark];  
 array[mark] = pivot;  
 return mark;  
}

**更多实现方式查看：**

<https://gitee.com/algebra/algebra_demo/blob/dev-aspect/demo-api/src/main/java/com/algebra/demo/util/sort/QuickSort.java>

1. Java实现层级树形菜单

**实现方式不唯一**

public class MapTreeUtil {  
  
 public List<Tree> menuCommon;  
 public List<Object> list = new ArrayList<>();  
  
 public List<Object> menuList(List<Tree> menu) {  
 this.menuCommon = menu;  
 for (Tree x : menu) {  
 Map<String, Object> mapArr = new LinkedHashMap<>();  
 if ("0".equals(x.getPId())) {  
 mapArr.put("id", x.getId());  
 mapArr.put("name", x.getName());  
 mapArr.put("pid", x.getPId());  
 mapArr.put("childList", menuChild(x.getId()));  
 list.add(mapArr);  
 }  
 }  
 return list;  
 }  
  
 public List<?> menuChild(String id) {  
 List<Object> lists = new ArrayList<>();  
 for (Tree a : menuCommon) {  
 Map<String, Object> childArray = new LinkedHashMap<>();  
 if (a.getPId().equals(id)) {  
 childArray.put("id", a.getId());  
 childArray.put("name", a.getName());  
 childArray.put("pid", a.getPId());  
 childArray.put("childList", menuChild(a.getId()));  
 lists.add(childArray);  
 }  
 }  
 return lists;  
 }  
  
}