**<!-- 2016/8/4 -->**

**1.Java中如何把一个byte[]类型的数组转换成一个String对象**

**new String(byte[] data)**

**或者使用**

**new String(byte[] data,String csn)  //硬编码的时候使用**

**===================================================================================**

**2.请写出执行读取且以UTF-8编码格式读取text.txt文件的过程**

**a: 传统写法，关键在于只有桥转换器能够指定字符编码~**

**FileInputStream fis = new FileInputStream("test.txt");**

**InputStreamReader isr = new InputStreamReader(fis,"UTF-8"); //\*\*\*\*\***

**BufferedReader br = new BufferedReader(isr);**

**String str;**

**while((str = br.readLine())!=null){**

**System.out.println(str);**

**}**

**br.close();**

**b: 使用JDK7.0 try-with-resources 也就是自动关闭资源的try catch语法**

**try(BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("test.txt"),"UTF-8"))){**

**String str;**

**while((str = br.readLine())!=null){**

**System.out.println(str);**

**}**

**}catch(Exception e){**

**e.printStackTrace();**

**}**

**c: 使用JDK7.0的nio2 （也就是新io2代） 需要导nio包**

**import java.nio.file.\*;**

**import java.nio.charset.\*;**

**import java.util.\*;**

**List<String> lines = Files.readAllLines(Paths.get("focus.txt"),Charset.forName("UTF-8"));**

**这一行代码能够将focus.txt文件当中的所有内容以行为单位读取到List集合当中 每一个元素就是一行文本~**

**===================================================================================**

**3.当classpath下有一个config.properties文件,在config.properties文件中**

**配置了一些JDBC的参数,请写出IO读取jdbcURL参数过程**

**a:使用java.util.Properties类**

**Properites pro = new Properties();**

**pro.load(new FileInputStream("config.properties"));**

**//pro.load(new FileReader("config.properites"));     //字符流读取 为了应对配置文件当中的中文 这样就不用native2ascii了！**

**String value = pro.getProperty("jdbcURL");**

**b:只使用io流完成读取和判断**

**BufferedReader br = new BufferedReader(new FileReader("config.properties"));**

**String str;**

**while((str.readLine())!=null){**

**if(str.contains("jdbcURL")){**

**System.out.println(str.split(" ")[1].trim());**

**break;**

**}**

**}**

**br.close();**

**===================================================================================**

**4.请写出把含有中文的a.txt文件内容转为unicode码的命令**

**native2ascii a.txt               //native2ascii是jdk安装目录下bin目录当中的工具命令**

**另外的用法：**

**native2ascii回车**

**我要转换的文字**

**===================================================================================**

**5.写出遍历一个HashMap的过程**

**a:keySet() 遍历所有主键对象 然后通过Map的get(k)方法得到对应的值**

**Set<K> ks = map.keySet();**

**for(K k : ks){**

**System.out.println(k + " = " + map.get(k));**

**}**

**b:values() 遍历所有的值对象 由于值对象不唯一 所以不能通过值得到主键**

**Collection<V> vs = map.values();**

**for(V v :vs){**

**System.out.println(v);**

**}**

 **c:entrySet() 得到所有键值对记录并且遍历**

**Set<Map.Entry<K,V>> es = map.entrySet();**

**for(Map.Entry<K,V> e: es){**

**K k = e.getKey();**

**V v = e.getValue();**

**System.out.println(k + " = " + v);**

**}**

**===================================================================================**

**6.当运行com.test.TestMain类需要依赖test.jar时,请写出运行com.test.TestMain类的完整流程**

**cmd命令行当中设置环境变量CLASSPATH**

**set CLASSPATH=.;test.jar //无比保证.;开头 防止骑着驴找驴**

**java com.test.TestMain**

**===================================================================================**

**7.请写出Java实现多线程的方式以及过程样例**

**我她妹！几点了这么一大片！**

**a: extends Thread**

**class ThreadOne extends Thread{**

**@Override**

**public void run(){**

**System.out.println("first");**

**}**

**}**

**然后需要创建该线程的位置：**

**ThreadOne t1 = new ThreadOne();**

**t1.start();**

**b:implements Runnable**

**class ThreadTwo implements Runnable{**

**@Override**

**public void run(){**

**System.out.println("second");**

**}**

**}**

**然后需要创建线程的位置：**

**ThreadTwo t = new ThreadTwo();**

**Thread t2 = new Thread(t);**

**t2.start();**

**c:implements Callable          //必须结合线程池执行器服务**

**class ThreadThree implements Callable<String>{**

**@Override**

**public void call()throws Exception{**

**System.out.println("third");**

**return "over";**

**}**

**}**

**然后需要创建线程池执行器服务并且提交任务**

**ExecutorService es = Executors.newFixedThreadPool(3);          //可重用的线程池**

**//Executors.newCachedThreadPool();        //缓存机制的线程池**

**//Executors.newSingleThreadExecutor();   //单一实例的线程执行器**

**Future<String> f = es.submit(new ThreadThree());**

**//然后视需求决定是否调用自我阻塞方法get()来得到返回的结果**

**String ok = f.get();**

**===================================================================================**

**8.平时有没有接触过服务器，linux常用的命令？**

**ls**

**pwd**

**mkdir**

**cp**

**rm**

**pkill**

**可以直接回答没接触过Linux 不是做运维的才涉及这个嘛 我们只做开发~**

**===================================================================================**

**9.请说明ArrayList、Vector、LinkedList的存储性能和特性，HashMap和Hashtable的区别？**

**ArrayList底层是数组实现    优势在于查找遍历 随机访问! 由于底层没有synchronized修饰 所以同时允许多个线程操作 可能导致并发错误**

**Vector底层同样使用数组实现 只是当中主要方法都使用了synchronized修饰 同一时间只能有一个线程进行访问 不会出现并发错误 但效率低**

**LinkedList底层是双向循环链表实现 优势在于添加和删除**

**劣势在于查找遍历 最大的劣势在随机访问 (get())**

**\*:使用LinkedList务必回避它的get(int)**

**HashMap和Hashtable的区别**

**同步特性不同**

**首先HashMap同一时间允许多个线程同时进行访问，效率相对较高，但是可能出现并发错误**

**Hashtable底层大量的使用了synchronized修饰方法，同一时间只允许一个线程对其操作，效率相对较低，但是不会出现并发错误**

**\*：从jdk5.0开始集合的工具类Collections当中出现了一个静态方法synchronizedMap方法可以将一个不安全的map变成线程安全的**

**\*：在高并发的场景下推荐使用java.util.concurrent.ConcurrentHashMap 有更高的性能~**

**对于null的处理不同**

**然后，他们对null的处理方式不同，HashMap无论是主键还是值都能存放null，**

**但是由于主键要求唯一，所以主键只能存放一个null，但是值能存放多个空**

**Hashtable无论是主键还是值都不能添加null，会触发空指针异常**

**底层实现的区别**

**底层有点不同 HashMap分组组数可以指定，默认分为16个小组，但是最终的分组组数一定是2的n次方数，**

**在计算散列的时候，底层使用&(分组组数-1)  [按位与运算]**

**Hashtable分组组数可以随意指定，默认分11组，可以随意指定分组模分组组数**

**出现的版本不同**

**HashMap jdk1.2     Hashtable jdk1.0**

**===================================================================================**

**10.interface和abstract class的区别？**

**interface是接口 abstract class是抽象类**

**接口当中定义的变量 默认添加三个修饰符public static final**

**接口当中定义的变量默认就是常量**

**抽象类当中定义的变量是普通的属性 每个这个类型的对象都有一份**

**接口当中定义的方法 自动添加两个修饰符 public abstract**

**抽象类当中定义的方法可以是非抽象方法 也可以是抽象方法**

**\*:从JDK8.0开始 接口当中的方法可以有具体的方法实现**

**a> static修饰的静态方法**

**b> default修饰的默认方法**

**===================================================================================**

**11.overload和override的区别？**

**Overload是方法重载 就是同一个类当中的两个方法 方法名字不同但是参数列表不同**

**参数可以是类型不同 个数不同或者顺序不同**

**方法重载能够让一个方法有多种形式存在 所以方法适用范围更广 功能更加强大**

**这体现了Java所谓的静态多态 也叫编译时多态 - 一个方法也有多种不同的存在形态**

**Override指的是方法覆盖 是子类继承得到父类的方法之后对方法实现并不满意 进行了重新实现**

**方法覆盖也叫方法重写 顾名思义 在继承得到父类方法后重新给出具体实现**

**方法覆盖要求 访问控制权限修饰符不能更加严格**

**返回类型 方法名字 参数列表 必须相同**

**\*:JDK5.0开始支持协变返回类型 也就是说子类方法返回类型可以是父类方法返回类型的子类类型**

**抛出的非运行时异常不能更加广泛**

**===================================================================================**

**12.jdk版本用的哪些，有什么区别？**

**JDK7.0**

**switch case 支持String类型**

**数值类型变量定义可以加下划线增强数值可读性 例如:int a = 123\_456\_789;**

**\*其实可以随便加 并不像千位分割那样规范 也就是说你可以 int a = 1\_\_\_\_2\_\_3\_4\_5;增强数值的不可读性**

**数值类型类型允许使用0b开头代表二进制赋值 例如:int a = 0b1011;**

**集合泛型的自动推断 例如:List<Integer> list = new ArrayList<>(); //后面的<>会自动推断类型**

**更加重要的内容,体现在应用上大变革：**

**NIO2(new io 第二代)**

**java.nio.files.Path**

**Path取代File代表文件或者目录的抽象路径 工具类Paths.get("父目录","又一级目录","文件");//可变参**

**java.nio.files.Files的出现**

**Files提供了很强大的方法让我们可以一行读取文件的所有字节或者所有行 一行完成文件复制**

**\*:参见昨天群内共享**

**try-with-resources**

**try(需要自动释放关闭的资源例如IO流或者JDBC的连接对象){   //括号内的类型需要实现过AutoCloseable接口**

**...;**

**}catch(Exception e){**

**e.printStackTrace();**

**}**

**其实还有很多，但是我在日常开发当中常用的就这些吧**

**JDK8.0**

**lambda表达式的存在颠覆了很多传统写法**

**例如在实现比较器的时候**

**传统写法：**

**Set<Integer> set = new TreeSet<>(new Comparator<Integer>(){**

**@Override**

**public int compare(Integer i1,Integer i2){**

**return i2 - i1;**

**}**

**});**

**改用8.0的lambda表达式**

**Set<Integer> set = new TreeSet<>((x,y) -> y-x);**

**再例如文件过滤器的实现**

**File[] fs = new File("dir").listFiles(new FileFilter(){**

**@Override**

**public boolean accept(File file){**

**return file.getName().toLowerCase().endsWith(".java");**

**}**

**});**

**改用8.0的lambda表达式**

**File[] fs = new File("dir").listFiles((x)->x.getName().toLowerCase().endsWith(".java"));**

**再例如集合的遍历**

**ArrayList<String> list = new ArrayList<>();**

**Collections.addAll(list,"Andy","Jacky","Leon","Jay");**

**for(String name : list){**

**if(str.length() > 3){**

**System.out.println(name);**

**}**

**}**

**改用8.0新特性**

**ArrayList<String> list = new ArrayList<>();**

**list.strem().filter((x->x.length()>3)).forEach(System.out::println);**

**list.forEach(System.out::println);//没有条件的遍历**

**===================================================================================**

**13.泛型用过吗？用过，ArrayList<String> 如何循环打印。**

**............................................**

**for(String name : list){**

**System.out.println(name);**

**}**

**8.0**

**list.forEach(System.out::println);**

**===================================================================================**

**14.设计模式有哪些？请简单说一下常用的设计模式**

**单例模式**

**工厂模式 SessionFactory**

**观察者模式 监听器的实现**

**包装模式 IO流的一层套一层就是包装模式**

**代理模式 Spring的AOP**

**命令模式**

**.....**

**===================================================================================**

**<!--  2016/7/29  -->**

**1. 静态变量是怎么用的？**

**static修饰的属性叫静态变量，静态变量表示整个类型共享一份的值，不依赖于对象而存在，所以需要拿着类名去调用**

**2. static的用法？**

**static表示一个修饰符，可以修饰属性，方法，代码快**

**修饰完的属性叫静态属性，表示整个类型共享一份的值，不依赖于对象而存在，所以需要拿着类名去调用**

**修饰完的方法叫静态方法，需要拿着类名去调用，静态方法里面只能直接的访问静态成员，如果想要访问非静态成员，需要找创建对象，拿着对象去调用**

**修饰完的代码块叫静态代码块，当类第一次被加载的时候执行，而且只执行一次**

**3. 讲一下IO流，常用的有哪些？IO流的批量操作？**

**InputStream**

**OutputStream**

**FileInputStream**

**FileOutputStream**

**BufferedInputStream**

**BufferedOutputStream**

**DataInputStream**

**DataOutputStream**

**ObjectInputStream**

**ObjectOutputStream**

**Reader**

**Writer**

**FileReader**

**FileWriter**

**BufferedReader**

**BufferedWriter**

**PrintStream**

**PrintWriter**

**InputStreamReader**

**OutputStreamWriter**

**第二问批量操作没懂...批量操作流还是批量操作文件..听了就蛋疼**

**"你听过青轴的机械电脑吗？"**

**4. 多线程用过么？讲一下你的理解**

**用过....理解...很@#￥@#￥！@#￥**

**5. io流怎么解决并发问题？**

**加锁**

**1.int和Integer的区别**

**首先int表示基本数据类型中的整数类型，是整数类型的默认类型，由32个位存储 其中一个符号位 31个数值位**

**而Integer是一个引用类型，它是int类型的包装类，能够将基本数据类型的int包装成一个引用类型的对象，其中还提供了很多静态方法都是int类型常用的功能 例如     Integer.parseInt()  Integer.toHexString() Integer.toBinaryString() 等等**

**[这是之前发的原题！！！]**

**2.作用域public、private、protected的范围还有不写的区别。**

**public 公共的 任何位置都可以使用的**

**private 私有的 只有本类当中可以使用 只能用于修饰成员（内部类属于成员）**

**protected 受保护的 本包内可见 保外有继承关系的子类中也可见**

**不写 默认的 本包内可见**

**3.你所知道的集合类型和主要的方法。**

**Collection                         Map**

**List               Set                    SortedMap**

**SortedSet**

**List: ArrayList LinkedList Vector Stack**

**Set: HashSet**

**SortedSet: TreeSet**

**Map: HashMap Hashtable**

**SortedMap: TreeMap**

**Collection : add(obj) remove(obj) contains(obj) clear() iterator() size()**

**List: get(int) remove(int)**

**Map: put(k,v) remove(k) get(k) containsKey(k) containsValue(v) clear() size()**

**keySet() values() entrySet()**

**不用都回答出来 只是把常用的回答出来就好**

**9.【综合题】两个整数类型数组A、B，要求将A、B共有的元素按升序排列，并打印。**

**综合个P 就是考察集合呗~**

**import java.util.\*;**

**public class Go{**

**public static void main(String[] args){**

**Integer[] src1 = {1,5,2};**

**Integer[] src2 = {5,1,4,3};**

**List<Integer> list1 = new ArrayList<>(src1.length);**

**List<Integer> list2 = new ArrayList<>(src2.length);**

**Collections.addAll(list1,src1);**

**Collections.addAll(list2,src2);**

**list1.retainAll(list2);//求交集~ 也就是list1和list2共有的部分**

**Collections.sort(list1);**

**for(Integer x : list1)**

**System.out.println(x);**

**}**

**}**

**<！-- 2016/7/27 -->**

**1 下面哪些是Thread类的方法:start() run() exit() getPriority()**

**start()**

**run()**

**getPriority()**

**===================================================================================**

**2 下面程序运行结果是:**

**String str1 = "hello";**

**String str2 = "he" + new String("llo");**

**System.out.println(str1==str2);**

**false.........**

**===================================================================================**

**3 volatile 关键字是否能保证线程安全?**

**不能     它只能保证多个线程访问的是同一个变量（不是线程复制的） 但是没法保证操作的原子性 也就是说依然可能出现并发错误**

**===================================================================================**

**4 0.6332的数据类型是**

**double**

**===================================================================================**

**5 下面哪个流是面向字符的输入流:**

**BufferedWriter  FileInputStream ObjectInputStream InputStreamReader**

**BufferedWriter**

**InputStreamReader需要的参数是字节流 但是经过包装转换之后得到一个字符流(Reader)**

**不太明白面向字符流什么意思**

**===================================================================================**

**6 java中的++操作是线程安全的码?**

**不是**

**++操作涉及到三个步骤 取值(read) 自增(inc) 重新赋值(write)**

**===================================================================================**

**7 a=a+b 与 a+=b的区别?**

**对于+=来说它能够保证运算符的左侧的数据类型不发生改变，也就是有一个隐形的强转**

**假如a是short类型，b是int类型，那么第一个赋值语句就会出错，而第二个语句没有任何问题**

**===================================================================================**

**8 int 和Integer哪个会占用更多的内存?**

**包装类类型，因为包装类类型属于引用类型，因为Integer对象当中也要保存int类型的value属性**

**===================================================================================**

**9 为什么java中的String 是不可变的(Immutable)?**

**所有的方法都不会改变原有字符串内容 而是将符合要求的新字符串返回回来**

**究其原因，String底层的char类型数组并不是一定有条件完成修改操作...**

**===================================================================================**

**10 64位JVM中,int的长度是多数?**

**32位，4个字节**

**===================================================================================**

**11 你能保证GC执行吗?**

**不能 我们只能主动的调用gc 但是gc作为一个系统级别的守护线程 它的执行机制跟普通线程是类似的**

**主动调用gc ： System.gc();    Runtime.getRuntime().gc();**

**===================================================================================**

**12 ArrayList与LinkedList的不同?**

**下面有**

**===================================================================================**

**13 除了单例模式,你在生产环境中还是用哪些设计模式?**

**工厂模式**

**代理模式**

**观察者模式**

**适配器模式**

**包装模式**

**....**

**===================================================================================**

**14 继承和组合之间有什么不同?**

**继承 描述的是 is a 关系 是一个**

**组合 描述的是 have a 关系**

**面向对象的编程语言 这种oo思想的问题是没有尽头的 也没有满分答案**

**===================================================================================**

**15 一般异常与运行异常的区别，分别怎样处理**

**运行时异常继承自RuntimeException 在编译的时候即便不去明确如何处理编译也能通过**

**一般继承直接继承Exception 在编译的时候必须要给出异常的处理方式 要么捕获要么声明抛出...**

**try catch将异常捕获,因为程序运行当中一旦有异常出现 虚拟机就会结束当前线程的执行 所以捕获处理异常是必须的**

**===================================================================================**

**<！-- 2016/7/24 -->**

**多线程是怎么处理的?**

**.......................该怎么处理怎么处理...**

**处理寓意好广泛啊...不懂~**

**===================================================================================**

**<！-- 2016/7/18 -->**

**1.java的面向对象的思想有什么特征**

**封装 继承 多态面向对象的三大特点**

**2.基本数据类型有哪些**

**布尔类型boolean 字符类型 char 整数类型byte short int long 浮点类型 float double**

**3.String StringBuffer StringBuilder 的区别**

**首先它们三个都表示Java中的字符串，底层都是使用char类型的数组实现的**

**String和它们两者直接的区别是对于String来说创建对象的时候底层开辟出来的数组对象几块空间看字符串里面的字符个数，而对于stringbuffer和StringBuilder来说底层空间大小不旦要看字符串的字符个数，它还会多预留16块空间用做缓冲区**

**对于StringBuffer和StringBuilder来说，它们的区别在于线程是否安全，对于StringBuffer来说底层大量使用了修饰符synchronized，由此同一时间只允许一个线程同时进行访问，效率较低，但是不会出现并发错误，而StringBuilder正好相反，它同一时间允许多个线程同时进行访问，效率较高，但是可能出现并发错误**

**4.int 和 integer 有什么区别**

**首先int表示基本数据类型中的整数类型，是整数类型的默认类型，由32个位存储 其中一个符号位 31个数值位**

**而Integer是一个引用类型，它是int类型的包装类，能够将基本数据类型的int包装成一个引用类型的对象，其中还提供了很多静态方法都是int类型常用的功能 例如     Integer.parseInt()  Integer.toHexString() Integer.toBinaryString() 等等**

**5.栈和堆的区别**

**首先这是底层不同的两块空间**

**如果我们创建基本数据类型，底层只在栈内存里面开辟空间**

**如果我们创建引用数据类型，底层需要在栈内存开辟空间装对象的引用，在堆内存开辟空间装对象**

**6.构造方法的一些内容**

**首先构造方法是创建对象的时候调用的方法，而且是一个收尾的工作，它能够在创建对象的同时完成对属性的赋值，构造方法的首行可以出现super()或者this（），super表示要执行本构造方法之前先去执行父类的构造方法，默认找父类的无参构造方法，this表示要执行本构造方法之前先去执行本类的其它的构造方法**

**7.内部类**

**定义在一个类当中的类 Java当中的内部类分为成员内部类、静态内部类、局部内部类以及特殊的匿名内部类**

**另外使用内部类可以很方便的共享到外部类当中的成员**

**8.HashMap和LinkedHashMap的区别**

**LinkedHashMap继承了HashMap，我们都知道HashMap是要对元素进行散列的，而散列导致遍历的顺序并不是我们添加记录的顺序，而LinkedHashMap在以HashMap为基础的前提下，引入了双向循环链表来存储添加的键值对记录的顺序，让HashMap的遍历能够实现按照添加顺序或者访问顺序（最常用的在最后）依次遍历。**

**9.打印**

**1**

**2 3**

**4 5 6**

**7 8 9 10**

**11 12 13 14 15**

**int cur = 0,line = 1;//cur=当前行打印完成几个 line=当前是第几行**

**for(int i = 1;i<=15;i++){**

**System.out.print(i + " ");**

**if(++cur == line){**

**System.out.println();**

**line++;**

**cur = 0;**

**}**

**}**

**<！-- 2016/7/13 -->**

**1.怎么做一个jar包 将自己的一些工具类分享给其他人**

**首先需要先将这些工具类对应的.class得到，如何得到.class文件可以通过开发工具的编译器得到，或者在命令行里面输入指令**

**javac xxx.java，**

**然后将对应的.class放进jar包里面，同样的需要在命令行里面输入指令**

**jar cvf 标志符.jar xxx.class( yyy.class...)**

**注意：是如何制作Jar包而不是如何制作可执行的Jar包 这个世界上有99%的jar包是用来共享类库的 而不是执行的**

**<！-- 2016/7/12 -->**

**java集合  List Set Map区别**

**首先List和Set是Collection接口的子接口，属于单值类型，List集合要求元素有序，不唯一，而Set集合要求元素唯一 Map是键值对集合的父接口**

**===================================================================================**

**集合类型**

**单值类型(Collection)和键值对类型(Map)**

**List的排序**

**通过写一个比较器实现某种排序规则，然后Collections.sort(list,比较器对象);**

**JDK8.0可以直接使用lambda表达式指定排序规则**

**例如:Collections.sort(list,(x,y) -> y-x );**

**Set与List的区别？**

**List集合要求元素有序，不唯一，而Set集合要求元素唯一**

**用没用过比较器？**

**用过 TreeSet和TreeMap在指定元素排序规则的时候可以使用比较器Comparator**

**另外List集合可以使用工具类Collections.sort()方法对元素进行排序**

**<！-- 2016/7/11 -->**

**1: 多线程了解么？创建线程有哪些方式？具体说一下继承Callable接口的方式？**

**1.extends Thread  @Override public void run(){...}**

**2.implements Runnable @Override public void run(){...}**

**3.implements Callable<X> @Override public X call(){...}**

**第三种实现Callable接口的方式弥补了原本Runnable接口当中run()的两个不足**

**run()方法定义为void方法 所以无法返回结果**

**方法没有throws声明所以没法抛出异常**

**另外这种方式必须结合线程池执行器服务 利用submit将任务提交给执行器**

**如果需要返回的数据 可以得到提交任务后返回的Future对象 然后调用自我阻塞的get()方法[这个方法会导致当前线程阻塞]**

**最后结束执行器的时候需要调用shutdown()或者shutdownNow()...**

**\*:你所知道的线程池    java.util.concurrent.\*;**

**ExecutorService es = Executors.newFixedThreadPool(3);**

**..newCachedThreadPool();**

**..newSingleThreadExecutor();**

**===================================================================================**

**2: java基本数据类型**

**Java中的基本数据类型分为四类八种**

**布尔类型：boolean只能使用true false进行赋值**

**字符类型：char**

**整数类型：byte short int long**

**浮点类型：float double**

**===================================================================================**

**<！-- 2016/7/10 -->**

**1.值传递和引用传递的区别？**

**Java中有两种传递方式一个是值传递一个是引用传递**

**另外也有人说Java当中只有值传递，因为引用类型的值就是内存指向的地址~**

**所谓的值传递指的是两个基本数据类型相互赋值，赋的就是这个变量保存的值**

**所谓的引用传递指的是两个引用数据类型相互赋值，赋的是这个变量里面保存的地址**

**===================================================================================**

**<！-- 2016/7/7 -->**

**1.如何跳出循环**

**使用关键字break;能直接跳出当前所在的循环**

**但是如果在嵌套的两层循环当中想要直接跳出外层循环 需要使用循环标签 就是在循环前给出一个合法的标识符然后冒号**

**out : for(int x = 0;x<10;x++){**

**for(int y = 0;y<10;y++){**

**if(....)**

**break out;**

**}**

**}**

**===================================================================================**

**2.排序的方式**

**首先可以使用数组工具类提供的排序方法，Arrays.sort(要排序的数组对象)**

**但是这种排序规则只能从小到大进行排序**

**第二种是手动排序，可以使用多种不同的算法实现排序 例如最常用的 冒泡排序**

**for(int x = 0;x < data.length -1;x++){**

**for(int y = 0;y < data.length -1 -x;y++){**

**if(data[y] > data[y + 1]){//比较规则可能改变 现在从小到大继续排序**

**数据类型 z = data[y];**

**data[y] = data[y + 1];**

**data[y + 1] = z;**

**}**

**}**

**}**

**===================================================================================**

**3.线程的实现方式**

**1.extends Thread  @Override public void run(){...}**

**2.implements Runnable @Override public void run(){...}**

**3.implements Callable<X> @Override public X call(){...}**

**\*:弥补了原本run()返回类型定义为void**

**方法签名后有throws Exception**

**\*:你所知道的线程池    java.util.concurrent.\*;**

**ExecutorService es = Executors.newFixedThreadPool(3);**

**..newCachedThreadPool();**

**..newSingleThreadExecutor();**

**===================================================================================**

**4.全局变量 和局部变量**

**首先Java中没有全局变量的概念，那应该叫成员变量，也叫实例变量，也叫属性**

**成员变量和局部变量这是Java中的变量下面的两大分支**

**（定义的位置不同）** **对于成员变量定义在类体里面方法体外面，对于局部变量定义在方法体里面**

**（作用范围不同）其次成员变量依赖于对象而存在，表示每个对象都有的属性，只要对象还没有被回收那么成员变量都可以访问，在本类类体里面**

**直接的进行访问,但是在其他的类里面通过对象去访问，而对于局部变量他的有效期只能从定义的那一行开始到所在的方法体结束之前可以访问，**

**其他的地方都访问不了**

**（默认值不同）最后对于成员变量来说，即使不赋值也有默认值，默认值为多少看属性对应的类型，而对于局部变量来说没有默认值，局部变量要**

**求使用之前必须先赋值**

**<！-- 2016/7/6 -->**

**===================================================================================**

**1.在你以前的项目中常用的数组是什么？**

**我能说不常用数组吗？**

**Object[]**

**===================================================================================**

**2.ArrayList和LinkedList有什么区别？**

**首先他们都表示List接口的实现类 所以它们都是有序而且不唯一的单值类型集合**

**对于ArrayList来说它的底层基于数组实现，导致它的优势是遍历** **查找，随机访问(get(int))，**

**它的劣势是向集合里面插入一个或是删除一个元素，因为这可能会涉及重新开辟新的数组对象。**

**而对于LinkedList来说，它的底层基于双向循环链表实现的，而链表这种结构导致它的优势是往集合里面添加或者删除元素，**

**它的劣势在于随机访问，遍历查找，特别是它的get里面传下标的方法效率特别低。**

**===================================================================================**

**3.String 和 StringBuffer**

**首先它们都表示Java中的字符串，底层都是基于char类型的数组实现的**

**对于它们之间的区别，第一个底层开辟的内存空间的差距，对于String来说，每创建一个String类型的对象，底层都会开辟一个数组对象，这个数组对象多大空间完全取决于字符串的字符个数，而对于StringBuffer来说，每创建一个StringBuffer对象，底层也是会开辟一个char类型的数组，只是这个数组对象的空间大小不但要看字符串的字符个数，还要多预留16块缓冲区，用来追加信息，所以当我们不断的在字符串的后面追加新内容的时候，StringBuffer的效率更高。**

**之后还有区别 可改变性**

**对于String类里面那些改变字符串内容的方法都不会直接操作原本的字符串，而且将符合条件的字符串返回给我们，所以需要拿着一个变量去接收 比如substring方法**

**而对于StringBuffer里面的一些改变字符串的方法就能直接的对原本字符串进行操作，不需要再接收 比如reverse() insert() append()**

**===================================================================================**

**4.Fit用过吗?**

**没有**

**===================================================================================**

**5.HashMap存储数据过程中遇到重复数据，怎么手动设置回滚？**

**我都不想骂娘了 这个问题得骂姥姥！**

**装B装出新高度了 为什么把一个CoreJava问题和事务当中的概念结合起来？回滚？！不会滚！**

**HashMap在存储数据过程中遇到重复数据，主键对象会直接舍弃，而新的值对象会替换原本的值对象**

**如果想要防止这种问题的出现，可以使用Map集合提供的containsKey()方法先进行判断是否包含指定的主键对象**

**\*:HashMap主键验证唯一的方式依赖于:hashCode() == equals()三步比较机制~**

**至于您说的回滚，是什么意思呢？滚到哪？不会滚。**

**===================================================================================**

**6.java虚拟机的底层实现**

**还记得我们讲过的类加载器+字节码校验器+解释执行器 底层实现~好样的装的好 你写一个吧**

**其实实现一个JVM就是要从加载.class文件到解析执行其中的JVM命令，问这个问题的是要干嘛？？让我们开发虚拟机吗？**

**下面是部分虚拟机指令**

**0x00 nop             什么都不做**

**0x01 aconst\_null 将null推送至栈顶**

**0x02 iconst\_m1 将int型-1推送至栈顶**

**0x03 iconst\_0   将int型0推送至栈顶**

**0x04 iconst\_1   将int型1推送至栈顶**

**0x05 iconst\_2   将int型2推送至栈顶**

**0x06 iconst\_3   将int型3推送至栈顶**

**0x07 iconst\_4   将int型4推送至栈顶**

**0x08 iconst\_5   将int型5推送至栈顶**

**0x09 lconst\_0   将long型0推送至栈顶**

**0x0a lconst\_1   将long型1推送至栈顶**

**0x0b fconst\_0   将float型0推送至栈顶**

**0x0c fconst\_1   将float型1推送至栈顶**

**0x0d fconst\_2   将float型2推送至栈顶**

**0x0e dconst\_0   将do le型0推送至栈顶**

**0x0f dconst\_1   将do le型1推送至栈顶**

**===================================================================================** 

**7.java中的代理与反射**

**怎么了呢？**

**代理是一种设计模式 反射是一种动态获取class当中属性和方法等信息，并且能动态调用方法其方法**

**java.lang.reflect.\*;**

**所有框架的实现必然离不开反射**

**而SpringAOP的实现涉及到动态代理以及反射**

**===================================================================================**

**8.消息队列**

**找ET001王栋 了解IBM MQ的用法~**

**消息是在两台计算机间传递的数据单位，消息可以非常简单，例如只包含文本字符串，也可以很复杂，可能包含嵌入对象**

**消息被发送到队列中，消息队列是在消息的传输过程中保存消息的容器，消息队列管理器在将消息从它的源中续到它的目标时充当中间人，队列的主要目的是提供路由并保证消息的传递，如果发送消息时接受者不可用，消息队列会保留消息，直到可以成功地传递它。**

**===================================================================================**

**11.对多线程熟悉吗**

**非常熟悉 谢谢**

**===================================================================================**

**12.HashMap和Hashtable哪个线程安全**

**Hashtable 蛋疼的问题 这个已经out了N年的题目**

**JDK5.0开始可以使用Collections.synchronizedMap() 将一个线程不安全的Map集合转换成线程安全的Map**

**JDK5.0的并发包(java.util.concurrent.\*)当中还提供了高并发场景下效率更好的ConcurrentHashMap**

**===================================================================================**

**<！-- 2016/7/5 -->**

**1.设计模式?单例模式?能不能new?观察者模式?**

**单例模式不能在当前类体之外new对象，因为单例需要私有化构造方法，外界是不能new对象的，但是在所在单例类体里面是可以new对象的**

**单例模式设计的类无论懒汉式还是醉汉式都会提供一个公共的静态的返回当前类型的方法来得到对象 通常叫做getInstance()**

**Java当中监听器的实现就是标准的观察者模式 英文Observer ... 具体内容待续 ... 未来几天单独讲23种设计模式**

**===================================================================================**

**2.HashMap和Hashtable的区别**

**同步特性不同**

**首先HashMap同一时间允许多个线程同时进行访问，效率相对较高，但是可能出现并发错误**

**Hashtable底层大量的使用了synchronized修饰方法，同一时间只允许一个线程对其操作，效率相对较低，但是不会出现并发错误**

**\*：从jdk5.0开始集合的工具类Collections当中出现了一个静态方法synchronizedMap方法可以将一个不安全的map变成线程安全的**

**\*：在高并发的场景下推荐使用java.util.concurrent.ConcurrentHashMap 有更高的性能~**

**对于null的处理不同**

**然后，他们对null的处理方式不同，HashMap无论是主键还是值都能存放null，**

**但是由于主键要求唯一，所以主键只能存放一个null，但是值能存放多个空**

**Hashtable无论是主键还是值都不能添加null，会触发空指针异常**

**底层实现的区别**

**底层有点不同 HashMap分组组数可以指定，默认分为16个小组，但是最终的分组组数一定是2的n次方数，**

**在计算散列的时候，底层使用&(分组组数-1)  [按位与运算]**

**Hashtable分组组数可以随意指定，默认分11组，可以随意指定分组模分组组数**

**出现的版本不同**

**HashMap jdk1.2     Hashtable jdk1.0**

**===================================================================================**

**3.StringBuffer怎么用?为什么会比String效率高?(StringBuffer和String哪个效率高?**

**首先字符串的底层都是基于char类型的数组实现的 当在字符串后面不断的追加新内容的时候，使用StringBuffer效率高，因为对于StringBuffer的对象来说底层开辟的数组对象会预留16块空间，当我们不断的在字符串的后面追加新的内容，他是在缓冲区里面追加，当缓冲区满了那么才开辟一个新的数组对象，然后再预留当前数组空间大小一半的空间当作缓冲区**

**而对于一个String类型的对象来说，没有缓冲区的概念，当前字符串有多少个字符，底层数组就开辟多大空间，如果我们想在String类型的后面不断的追加新的内容的话，底层其实不断的在创建新的数组对象，相对来说创建数组的频率会高很多，效率较低**

**因为每次开辟新数组之后都需要复制老元素 改变引用指向 然后将来还要回收旧的数组...**

**===================================================================================**

**4.java中有哪些工具包？**

**不太理解他说的工具包是指官方提供的包结构的包还是常用的开发组件的jar包**

**java.util.\*;          //顾名思义 工具包 当中提供的各种集合 日期 日历 ZipInputStream ZipOutputStream 都是工具**

**java.util.concurrent.\*; //工具包当中的并发包 当中提供多线程高并发所涉及到的各种工具 包括支持并发的集合等等**

**jar包就更多了**

**log4j   dom4j   common-xxxx   ... 无数个答案**

**===================================================================================**

**5.如何取得当前时间?**

**java.text.SimpleDateFormat**

**SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");**

**String ok = sdf.format(Sytsem.currentTimeMillis());**

**System.out.println(ok);**

**java.util.Date**

**Date now = new Date();**

**System.out.println(now);**

**System.out.println(now.getYear()+1900);**

**System.out.println(now.getMonth() + 1);**

**System.out.println(now.getDate());**

**System.out.println(now.getHours());**

**System.out.println(now.getMinutes());**

**System.out.println(now.getSeconds());**

**java.util.Calendar**

**Calendar now = Calendar.getInstance();**

**System.out.println(now.get(1));**

**System.out.println(now.get(2)+1);**

**System.out.println(now.get(5));**

**System.out.println(now.get(11));**

**System.out.println(now.get(12));**

**System.out.println(now.get(13));**

**===================================================================================**

**6.能不能读懂英文API？**

**能！我就想知道我说能你还要考察英语水平吗？！**

**===================================================================================**

**7.平时浏览什么网站?从什么网站上学习?**

**你管得着吗？我不告诉你！**

**===================================================================================**

**8.性能优化?**

**这玩意能讲一辈子...你确定要听吗？**

**===================================================================================**

**<！-- 2016/7/4 -->**

**1.多线程出现高并发会怎样?**

**我都懒得理他，会怎样呢~直接甩脸上很好啊不会怎样！**

**要么没事     - 或者没有并发错误的情况下你根本不把慢当回事~**

**要么慢        - 可以使用线程池减少大量创建和销毁线程的时间**

**要么出错     - 并发错误 各种加锁呗**

**===================================================================================**

**2.怎么解决死锁?**

**如果可行的话 改变锁标记出现的位置 顺序的排列锁标记让每个线程依次拿去锁标记防止死锁 但是这可能需要改变逻辑**

**通常要使用等待队列 也就是使用Object类的wait() notify() notifyAll()三个方法**

**[注意这三个方法都必须已经持有锁标记才能调用 所以他们只能出现在synchronized代码块当中]**

**使用wait()让当前线程(a)放弃锁标记进入等待池当中阻塞，**

**从而成全另外的线程(b)能够成功获得它(b)需要的锁标记之后再调用notify()或者notifyAll()唤醒线程(a)，**

**让线程(a)从等待池进入锁池等待获得锁标记.**

**===================================================================================**

**3.synchronized 用法?**

**修饰代码块或者修饰整个方法**

**修饰代码块**

**synchronized(临界资源){ //进入大括号需要先拿到临界资源的锁标记 [又叫：互斥锁 互斥锁标记 锁标记 锁旗标 监视器 英文:Monitor]**

**需要连续执行的操作1; //如果时间片在此耗尽则线程返回就绪 但是锁标记不会归还 所以其它线程中相同的代码块无法进入从而互斥**

**需要连续执行的操作2;**

**}//离开大括号释放临界资源的锁标记**

**修饰方法**

**public synchronized void add(Object obj){**

**//等价于对调用方法的当前对象加锁 如果是静态方法则对类的元对象(meta)加锁[你可以理解成对.class加锁]**

**//可以理解成从方法的第一行到最后一行统统对this进行加锁**

**}**

**===================================================================================**

**4.char和byte的区别，各几个字节**

**侮辱人的题目**

**char类型是字符型 16个位 两个字节 没有符号位 16个位全部都是数值位，所以取值范围0到2的16次方-1**

**byte类型是整数类型当中的一种 8个位 一个字节 其中1个符号位 15个数值位，所以取值范围-2的15次方到+2的15次方-1**

**===================================================================================**

**5.HashMap的put值的过程，hashCode一样怎么处理**

**所谓的哈希冲突，哈希码相同则会导致元素散列到相同的小组（桶）当中，但是接下来会使用 == 和equals来判断这两个对象究竟是否是相同的**

**所以Java当中的HashMap实现是不会惧怕这个问题的 Josh Bloch[Java之母]没那么菜**

**所以我们课上不停的讲hashCode() == equals() 而且是 1st && (2nd || 3rd)**

**1st.新元素.hashCode() == 老元素.hashCode()**

**相等                                                                                                    不等**

**继续比较                                                                                          新老元素不同 比较下一个或者添加进集合**

**2nd.新元素 == 老元素**

**相等                            不等**

**真正的同一对象-舍弃        继续比较**

**3rd.新元素.equals(老元素)**

**相等                                  不等**

**本身不同 但是程序员               哈希冲突**

**需要视作相同 覆盖了方法         既不是同一对象也不是逻辑相等 还它青白**

**逻辑相等 舍弃新元素                比较下一个或者添加进集合**

**===================================================================================**

**6.内存溢出的原因和解决方法**

**内存溢出和内存泄漏不是同一个问题，所谓溢出是程序根本无法申请到所需要的内存空间而导致的Error**

**其实有很多种Error都是因为内存溢出**

**java.lang.OutOfMemoryError : Java heap space**

**堆内存溢出，堆内存就是存放对象的内存空间**

**如果不是代码写的有错误(例如：无限循环的创建对象并加入集合当中~)，**

**那么就通过设置虚拟机参数解决 set JAVA\_OPTS=-Xms512m -Xmx512m**

**java.lang.OutOfMemoryError：PermGen space**

**全称是Permanent Generation space 也就是永久保存区**

**当中存放类的元对象(meta信息) 常量和静态变量**

**经常见到Spring项目由于cglib代理导致 或者大量使用反射加载.class导致 因为它默认只有4mb**

**可以通过调整虚拟机参数解决...**

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

**java.lang.StackOverflowError**

**栈内存溢出，这种情况多数因为递归调用层次太深或者就是不停的递归调用自己导致**

**还是从代码着手解决问题吧**

**再度重申下 Error不是Exception！它们是Throwable的两个分支 Error通常指由于硬件环境或者系统原因导致的问题**

**===================================================================================**

**7.全局变量和局部变量哪个能被覆盖**

**直接回答Java当中不允许定义全局变量 [其它编程语言当中类体之外定义的变量在之后每一个类都能访问的 = 全局变量！]**

**脑瘫问题！Java当中不存在全局变量的概念，只有类体当中的成员变量(实例变量)和方法体当中的局部变量**

**另外覆盖是只有方法才存在的概念 子类继承得到父类方法后对其进行重新实现叫做方法覆盖**

**而属性根本不存在覆盖的概念**

**如果父类当中定义的属性在子类当中又再次定义 那么子类对象当中其实有两个不同的属性同时存在**

**例如：**

**class A{**

**int x = 3;**

**}**

**class C extends A{**

**int x = 5;**

**public void test(){**

**System.out.println(x); //5的那个x**

**System.out.println(this.x);//5的那个x**

**System.out.println(super.x);//3的那个x           这T喵D根本不叫覆盖**

**}**

**}**

**===================================================================================**

**8.如何跳出循环**

**使用关键字break;能直接跳出当前所在的循环**

**但是如果在嵌套的两层循环当中想要直接跳出外层循环 需要使用循环标签 就是在循环前给出一个合法的标识符然后冒号**

**out : for(int x = 0;x<10;x++){**

**for(int y = 0;y<10;y++){**

**if(....)**

**break out;**

**}**

**}**

**===================================================================================**

**9.JDK6和7的区别**

**JDK7.0版本更新自然提供了更多的新特性可以使用**

**比较常用的包括:**

**switch case 支持String类型**

**数值类型变量定义可以加下划线增强数值可读性 例如:int a = 123\_456\_789;**

**\*其实可以随便加 并不像千位分割那样规范 也就是说你可以 int a = 1\_\_\_\_2\_\_3\_4\_5;增强数值的不可读性**

**数值类型类型允许使用0b开头代表二进制赋值 例如:int a = 0b1011;**

**集合泛型的自动推断 例如:List<Integer> list = new ArrayList<>(); //后面的<>会自动推断类型**

**更加重要的内容,体现在应用上大变革：**

**NIO2(new io 第二代)**

**java.nio.files.Path**

**Path取代File代表文件或者目录的抽象路径 工具类Paths.get("父目录","又一级目录","文件");//可变参**

**java.nio.files.Files的出现**

**Files提供了很强大的方法让我们可以一行读取文件的所有字节或者所有行 一行完成文件复制**

**\*:参见昨天群内共享**

**try-with-resources**

**try(需要自动释放关闭的资源例如IO流或者JDBC的连接对象){   //括号内的类型需要实现过AutoCloseable接口**

**...;**

**}catch(Exception e){**

**e.printStackTrace();**

**}**

**其实还有很多，但是我在日常开发当中常用的就这些吧**

**===================================================================================**

**<！-- 2016/7/3 -->**

**01：jdk6.0与7.0的区别**

**JDK7.0版本更新自然提供了更多的新特性可以使用**

**比较常用的包括:**

**switch case 支持String类型**

**数值类型变量定义可以加下划线增强数值可读性 例如:int a = 123\_456\_789;**

**\*其实可以随便加 并不像千位分割那样规范 也就是说你可以 int a = 1\_\_\_\_2\_\_3\_4\_5;增强数值的不可读性**

**数值类型类型允许使用0b开头代表二进制赋值 例如:int a = 0b1011;**

**集合泛型的自动推断 例如:List<Integer> list = new ArrayList<>(); //后面的<>会自动推断类型**

**更加重要的内容,体现在应用上大变革：**

**NIO2(new io 第二代)**

**java.nio.files.Path**

**Path取代File代表文件或者目录的抽象路径 工具类Paths.get("父目录","又一级目录","文件");//可变参**

**java.nio.files.Files的出现**

**Files提供了很强大的方法让我们可以一行读取文件的所有字节或者所有行 一行完成文件复制**

**\*:参见昨天群内共享**

**try-with-resources**

**try(需要自动释放关闭的资源例如IO流或者JDBC的连接对象){   //括号内的类型需要实现过AutoCloseable接口**

**...;**

**}catch(Exception e){**

**e.printStackTrace();**

**}**

**其实还有很多，但是我在日常开发当中常用的就这些吧**

**===================================================================================**

**02：List如何更改比较机制**

**我说的话：这问题问的有歧义，首先List作为不唯一的集合添加过程当中根本不涉及比较，那么这么问的意思不外乎两种可能**

**1.删除元素也就是调用remove(obj)或者contains(obj)的时候如何让原本不同的两个对象视作同一对象**

**2.如何让List集合当中的元素按照需求进行排序**

**所以如果问我我会先问对方：你他妹的什么意思**

**请问您是指对List元素进行排序还是问如何让不同元素视作相同对象的“逻辑相等”的问题**

**如果是蓝字提到的1：可以通过覆盖equals方法实现逻辑相等**

**如果是蓝字提到的2：可以使用集合工具类的排序方法结合自定义一个比较器完成**

**Collections.sort(list,new Comparator<Integer>(){**

**@Override**

**public int compareTo(Integer i1,Integer i2){**

**return i2 - i1;**

**}**

**});**

**JDK8.0之后 代码可以直接使用Lambda表达式完成排序 代码如下：**

**Collections.sort(list,(x,y) -> y - x);**

**//问这问题的绝对是个半瓶子**

**===================================================================================**

**03：HashMap底层扩容机制**

**每次扩容的时候分组组数会直接\*2 也就是变成原本的2倍大小 例如默认16组那么扩容一次之后变成32组**

**首先HashMap构造方法可以传入两个参数，分别是int类型的分组组数和float类型的加载因子**

**而其扩容跟阈值有关 只是在不同的JDK版本当中HashMap的实现也有区别     所以这个答案问的很脑瘫**

**所谓阈值 = 分组组数\*加载因子 = 这是HashMap达到扩容条件的最小临界值**

**\*：当然在不同的JDK7.0当中，即便达到或者超过阈值，如果新元素要去的小组不为空，也会暂时不扩容**

**以默认的分组组数16和加载因子0.75F为例 也就是Map<String,Integer> map = new HashMap<>();**

**那么阈值就是12 也就是第13个元素添加的时候可能会导致扩容，那么底层将从分16组变成分32组 然后所有元素进行重新散列**

**毫无疑问，重新散列会消耗时间，在使用HashMap的时候我们应该让分组组数\*加载因子>总共要存放的键值对总量**

**以避免添加过程当中触发扩容操作**

**===================================================================================**

**04：怎么提高HashMap的效率**

**我能说最简单的办法是换电脑吗？开玩笑的，在考你分组组数和加载因子**

**我们可以让HashMap底层开辟更多的分组，然后在元素总量不变的情况下，分组组数越多每一组元素个数自然越少，效率自然越高。**

**HashMap构造方法是可以传参数来指定分组组数和加载因子(又名装填因子)的，**

**这两者相乘得到阈值也就是达到扩容条件的最小临界值**

**我们使用HashMap应该在知道元素总量的情况下保证分组组数\*加载因子>元素总量 从而避免添加过程当中触发扩容操作影响效率**

**另外在这个条件成立的前提下只需要**

**分组组数越大，数据将被散列到越多个小组，查找效率越高**

**加载因子越小，扩容会发生的越早，也就是在尽可能的保证效率**

**===================================================================================**

**05：HashMap 支持并发操作吗？如果多个线程同时操作一个hashmap,会出现什么问题，会报异常吗**

**我想先问问这个家伙支持是啥意思呢？是底层是否允许发生还是会不会导致并发错误？**

**在这里你要明白HashMap底层没有synchronized所以是允许多个线程同时操作的，但是由于会导致并发错误，很多人就会说它不支持**

**但是底层真的是多个线程进行了操作才会导致的错误 妹的，很像20厘米宽的大雪糕很好吃但是太难吃了这句话当中好吃和难吃的意思**

**大家自己体会吧...再例如：足球场上前锋能不能用手触球？【能啊！不就是犯规嘛！】**

**同样的问题还出现在StringBuffer和StringBuilder   ArrayList和Vector的身上 所以你答对答错看他怎么理解的了 或者你们先沟通下！**

**常规答案：HashMap不支持多个线程并发操作，会出现并发错误，在并发的场景下，传统应该使用Hashtable**

**但是从JDK5.0开始集合的工具类Collections当中提供了一个静态方法synchronizedMap();**

**它能够将一个线程不安全的HashMap变成线程安全的Map集合，而且效率高于Hashtable**

**另外java.util.concurrent.包也就是并发包当中提供了支持多线程同时操作而且拥有更好的并发性能的ConcurrentHashMap**

**[多线程大师Doug Lea（李狗）作品 这爷爷除了类名写的都很长 水平还是很强di]**

**多个线程共同操作一个HashMap会导致并发错误，当然HashMap的底层实现中对并发错误做了判断和校验**

**如果发现有并发操作的情况会第一时间(fail-fast快速失败)的抛出ConcurrentModificationException**

**但是这依然不能保证可怕的并发错误不会出现**

**问这个问题的家伙绝对被ConcurrentModificationException搞郁闷过，但是问的问题出卖了他的水平**

**并发错误和并发修改异常根本不是一个层面的东西**

**并发错误是指多个线程共享数据的时候由于无法保证线程体当中对数据的连续操作一定完整**

**而另一个线程很可能拿走处理了一半的错误数据，这根本不会报异常！（你可知道CME是主动throw的？）**

**但是由于数据是错误的，接下来会死的非常惨！**

**而并发修改异常也就是ConcurrentModificationException是集合底层实现的时候为了防止并发错误而主动进行的验证和判断**

**主动抛出异常防止出现并发错误**

**===================================================================================** 

**06：两个线程如何实现数据共享**

**共享数据？这个问题跟线程有多大关系呢？**

**首先Java当中的线程也是一个对象，而创建对象是需要模板的也就是类**

**如果两个线程要共享数据，可以简单的理解为两个类要访问同一个对象嘛~**

**所以我们可以：**

**将要共享的数据定义成静态的 然后使用类名.属性名访问它**

**将要共享的对象通过线程类的构造方法或者setter方法赋值给一个属性**

**将要共享的数据定义在一个类普通成员的位置 然后线程作为成员内部类定义**

**\*：这个问题跟线程没有半毛钱关系，事实上在CoreJava课程当中还没讲线程就已经讲董卓吕布怎么共享貂蝉的故事了**

**还有电影院当中小男孩小女孩一杯可乐两人共享的故事....**

**===================================================================================** 

**07：讲讲Java队列queue**

**多亏他没面试我，要不然我得骂死他，首先Queue是队列 是数据结构**

**数据结构 算法 甚至包括大家整天听的设计模式 这些都是高于语言的**

**也就是说即便不用Java也有可能会涉及到这些概念**

**那么题目当中的Java队列是他妹儿的啥意思？是Java当中的队列结构的实现有哪些？还是用Java实现队列？蛋疼！你怎么连话都说不清楚！**

**Java当中的Queue是一个作为一个接口存在的 指的是先进先出（FIFO）** **的队列结构 这个接口出现于JDK5.0 由Doug Lea实现**

**我们经常使用的LinkedList就实现了这个接口**

**（但是从JDK6.0开始实现LinkedList直接实现Deque也就是双端队列，而Deque是Queue的子接口）**

**\*：个人感觉，一种结构定义为一种规范（一个接口）是极为不恰当的，或者说这么做对哈希表和红黑树极为不公平！**

**或许有个接口叫FIFO有个接口叫LIFO能更好让人接受**

**===================================================================================**

**08：线程和进程的区别**

**首先这都是操作系统当中的概念 而作为Java程序员我们不需要过分深究 那就直接一语戳破要害吧**

**进程简单理解就是一个程序运行起来就是一个进程，进程代表一个程序的一次执行过程。**

**线程是程序当中一条独立的执行线索，一个进程可以包含多个线程**

**简单来说，进程就是一场战争，那么线程就是这场战争中的一股作战部队。**

**一场战争可能涉及到多股部队分兵合作，每股作战单位彼此独立而都是这场战斗的一部分**

**我们可以把Java的主线程理解成诸葛亮带领的主力作战部队，**

**那么战斗开始的时候，我们可以让赵子龙和关云长各自领一股小部队执行不同的作战任务，而主力部队也会同时行动...**

**部署作战任务就是覆盖run()或者call() 而调用start()就是告诉部队开拔！但是主力作战部队(main())不要直接去做小部队的任务(调用run())**

**自己理解吧 线程的课开头的基础中的基础**

**===================================================================================**

**09 : 方法过载，方法重写**

**方法重载也叫方法过载 英文Overload Overloading Overloaded 最讨厌这种换名字问东西的 欺负人嘛**

**指的是一个类当中的两个方法拥有完全相同的名字和不同的参数列表 这能让一个方法有更强的适用性~**

**[构造方法也允许重载 例如File类提供了4个不同的构造方法~]**

**官方的定义比较蛋疼**

**发生在同一个类型当中  方法名字相同  参数列表必须不同**

**所谓参数不同可以是参数的类型不同 个数不同 或者顺序不同 但是参数名字不同不能算不同 因为形参和局部变量的名字没有实质意义**

**它们根本不被.class文件保存 换言之 方法参数叫a和叫abcdefghijklmnop 生成的.class文件一模一样（大小都一样）**

**方法覆盖也叫方法重写 英文Override Overriding Overrided**

**指的是子类继承得到父类的方法后对继承到的方法实现进行了重新实现**

**方法覆盖也有要求**

**首先访问控制权限修饰符不能更加严格 也就是说要么相等 要么比父类方法更宽松 [如果父类方法已经public 覆盖的时候就没有选择了]**

**方法返回类型 方法名和参数列表必须相同 当然在JDK5.0之后支持协变返回类型**

**[子类方法可以返回父类方法返回类型的子类类型 群共享有]**

**[例如父类test()返回Animal 子类test()可以返回任何Animal类型的子类类型如Bird Cat Dog...]**

**[进一步 如果父类方法返回Object 则子类可以返回任何引用类型 因为它们都是Object 例如Object类的clone]**

**[再进一步 如果父类方法返回基本数据类型 或者final修饰的最终类型 再或者void 那根本没法协变]**

**抛出的异常(非运行时异常)种类不能更加广泛(只范围不是数量)**

**[当然如果你要找茬的去throws N多运行时异常就能怼死这句话 但是throws运行时异常毫无意义]**

**===================================================================================**

**10：equals 底层怎么实现**

**我她妹儿啊！他到底问的是啥意思！你怎么连话都说不清楚！底层底层底层 底层你妹啊 是Object还是我自己写？这都不底层！**

**Object类的equals方法就是直接用==比较当前对象this和参数obj的**

**首先equals是Object类的一个方法，由于==是运算符而Java不允许运算符重载，所以==永远都是比较两端对象的地址是否相同**

**也就是在比较是不是内存当中的同一个对象，而在很多需求中，我们需要把内容相同甚至部分内容相同的（甚至完全不同的）两个对象视作同一个对象，也就是所谓的逻辑相等，这时候就可以通过覆盖equals方法来描述我们程序员当前自己的需求，例如String类的equals就会在两个字符串对象内容相同的情况下返回true。不过这并不意味着equals就是比较内容的，Object类的equals方法其实就是直接使用==比较当前对象和参数的，也就是说equals存在的意义是让我们通过覆盖它来描述我们的逻辑和需求而不是说equals一定是比较内容的！**

**底层怎么实现，一个方法，你他妹儿的爱怎么实现就怎么实现啊，我能说直接return true;就是实现嘛~**

**别一见底层就害pia，有些人所谓的底层还不如我们平常理解的底呢！！就是干！不能耸！**

**List集合的remove(obj) 就尊重元素的equals()来辨别每一个元素是不是程序员要删除的对象**

**如果你直接return false;那么这个方法将永远无法删除成功~**

**亲情提示：JDK7.0当中util包里面出现了一个叫做Objects的"工具类"当中有个静态的equals()方法**

**会在两个对象==的情况下直接返回true 一个为null的情况下直接返回false 否则返回调用equals()的结果**

**所以今后如果你确定调用你写的equals方法的人一定掌握并且100%的会用这个方法去比较你的对象 那么可以不写equals()开头的两个if**

**然而你根本没法确保这件事情的发生 所以亲情提示纯属坑爹，此致敬礼~**

**===================================================================================**

**<！-- 2016/7/1 -->**

**01.String StringBuffer StringBuilder的区别**

    keyword: 缓冲区 缓冲空间

    String 没有预留任何缓冲空间

    StringBuffer 底层大量使用了synchronized修饰符

            同一时间只允许一个线程进行操作

            效率相对较低 但是不会出现并发错误

    StringBuilder 同一时间允许多个线程同时进行操作

            效率较高 但是可能出现并发错误

    \*:String s1 = "abc";            //常量池

      String s2 = new String("abc");

**02.HashMap和Hashtable的区别**

    首先HashMap同一时间允许多个线程同时进行操作

        效率相对较高 但是可能出现并发错误

    Hashtable底层大量的使用了synchronized修饰方法

        同一时间只允许一个线程对其进行操作

        效率相对较低 但是不会出现并发错误

    \*:从JDK5.0开始集合的工具类Collections当中出现了一个静态方法

        Map<X> map = Collections.synchronizedMap(map);

    \*:java.util.concurrent.ConcurrentHashMap;

    然后 它们对于null的处理不同

    HashMap无论主键还是值都可以存放null 但是由于主键要求唯一

        所以主键只能存放一个null 但是值能存储无数个null

    Hashtable无论主键还是值 都不能添加null 会触发NullPointerException

    再然后 它们的底层实现有些许区别

    HashMap 分组组数可以指定 默认分16 但是最终结果一定是2的N次方数

        在计算散列分组的时候 底层使用&(分组组数-1)

    Hashtable 分组组数可以随意指定 默认分11组

        可以随意指定分组    %分组组数

    最后 HashMap sinceJDK1.2   Hashtable since JDK1.0

**03. == 和 equals的区别**

    ==是一个运算符 用于比较两端的内容是否相等

        基本数据类型 : 两端的值是否相等

        引用类型 : 比较的是引用的值(内存指向的地址)是否相等

    equals() : 它是Object类的一个方法 子类继承到这个方法之后

        可以按照自己所需要的逻辑需求 覆盖这个方法

        从而描述自己的"逻辑相等"的比较规则

        例如:String类就将equals()覆盖为比较其字符串内容的

**04.HashMap如何调整性能和空间的取舍?**

    构造方法的两个参数:int 分组组数 , float 加载因子

    在加载因子不变的情况下 如果我们让分组组数变大

        则效率更高 -> 但是需要更多的空间

    在分组组数不变的情况下 如果让加载因子变大

        则更加节省空间 -> 但是效率更低

    \*:我们在已经知道要存放的元素个数的前提下 应该让

        分组组数\*加载因子 > 元素总量

        避免添加过程当中触发扩容操作

**05.List Set和Map存取元素的特点以及你所知道的实现类**

    List和Set都是单值类型集合 Map是键值对集合

    List要求元素有序 不唯一    ArrayList LinkedList Vector Stack

    Set要求元素唯一 HashSet SortedSet 要求元素有序 唯一 TreeSet

    Map要求主键唯一 但是值没有要求 HashMap

    SortedMap要求主键有序 而且唯一 TreeMap

**06.创建线程有几种方式?**

    1.extends Thread  @Override public void run(){...}

    2.implements Runnable @Override public void run(){...}

    3.implements Callable<X> @Override public X call(){...}

        \*:弥补了原本run()返回类型定义为void

          方法签名后有throws Exception

    \*:你所知道的线程池    java.util.concurrent.\*;

    ExecutorService es = Executors.newFixedThreadPool(3);

                     ..newCachedThreadPool();

                     ..newSingleThreadExecutor();

**07.ArrayList和LinkedList的区别**

    ArrayList底层是数组实现    优势在于查找遍历 随机访问!

    LinkedList底层是双向循环链表实现 优势在于添加和删除

        劣势在于查找遍历 最大的劣势在随机访问 (get())

        \*:使用LinkedList务必回避它的get(int)

    \*:ArrayList 和 Vector的区别?

**08.接口和抽象类的区别**

    接口当中定义的变量 默认添加三个修饰符public static final

        接口当中定义的变量默认就是常量

    抽象类当中定义的变量是普通的属性 每个这个类型的对象都有一份

    接口当中定义的方法 自动添加两个修饰符 public abstract

    抽象类当中定义的方法可以是非抽象方法 也可以是抽象方法

    \*:从JDK8.0开始 接口当中的方法可以有具体的方法实现

        a> static修饰的静态方法

        b> default修饰的默认方法

**09.介绍下你所了解的设计模式**

    单例模式

    工厂模式 SessionFactory

    观察者模式 监听器的实现

    包装模式 IO流的一层套一层就是包装模式

    代理模式 Spring的AOP

    命令模式

**10:如何去遍历一个Map集合**

    keySet()    values()    entrySet()

    Map.Entry : getKey() getValue()