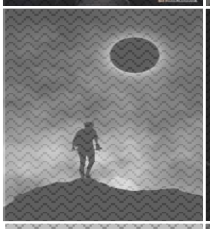
2016

Hello World



Whether sixty or sixteen, there is in every human being's heart the lure of wonders, the unfailing childlike appetite of what's next and the joy of the game of living. In the center of your heart and my heart there is a wireless station: so long as it receives messages of beauty, hope, cheer, courage and power from men and from the infinite, so long are you young.

An individual human existence should be like a river—small at first, narrowly contained within its banks, and rushing passionately past boulders and over waterfalls. Gradually the river grows wider, the banks recede, the waters flow more quietly, and in the end, without any visible break, they become merged in the sea, and painlessly lose their individual being.

Youth means a temperamental predominance of courage over timidity,of the appetite for adventure over the love of ease.This often exists in a man of sixty more than a boy of twenty.Nobody grows old merely by a number of years.We grow old by deserting our ideals.

Years may wrinkle the skin,but to give up enthusiasm wrinkles the soul. Worry, fear, self-distrust bows the heart and turns the spirit back to dust.

zhanglei
Shandong University of Finance and Economics
2016/03/05

1. Java 基础部分

基础部分的顺序：基本语法，类相关的语法，内部类的语法，继承相关的语法，异常的语法，线程的语法，集合的语法，io 的语法，虚拟机方面的语法。

1、一个".java"源文件中是否可以包括多个类（不是内部类）？有什么限制？

可以有多个类，但只能有一个 public 的类，并且 public 的类名必须与文件名相一致。

2、Java 有没有 goto?

java 中的保留字，现在没有在 java 中使用。

3、说说&和&&的区别。

&和&&都可以用作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，则结果为 false。

&&还具有短路的功能，即如果第一个表达式为 false，则不再计算第二个表达式，例如，对于 if(str != null&& !str.equals(“”))表达式，当 str 为 null 时，后面的表达式不会执行，所以不会出现 NullPointerException 如果将&&改为&，则会抛出 NullPointerException 异常。If(x==33 &++y>0) y 会增长，If(x==33 && ++y>0)不会增长

&还可以用作位运算符，当&操作符两边的表达式不是 boolean 类型时，&表示按位与操作，我们通常使用 0x0f 来与一个整数进行&运算，来获取该整数的最低 4 个 bit 位，例如，0x31 & 0x0f 的结果为 0x01。

备注：这道题先说两者的共同点，再说出&&和&的特殊之处，并列举一些经典的例子来表明自己理解透彻深入、实际经验丰富。 f

4、在 JAVA 中如何跳出当前的多重嵌套循环？

在 Java 中，要想跳出多重循环，可以在外面的循环语句前定义一个标号，然后在里层循环体的代码中使用带有标号的 break 语句，即可跳出外层循环。例如，

```
ok:
for(int i=0;i<10;i++) {
for(int j=0;j<10;j++) {
System.out.println(“i=” + i + “,j=” + j);
if(j == 5) break ok;
}
}
```

另外，我个人通常并不使用标号这种方式，而是让外层的循环条件表达式的结果可以受到里层循环体代码的控制，例如，要在二维数组中查找到某个数字。

```
int arr[][] ={{1,2,3},{4,5,6,7},{9}};
boolean found = false;
for(int i=0;i<arr.length&& !found;i++) {
for(int j=0;j<arr[i].length;j++){
System.out.println(“i=” + i + “,j=” + j);
if(arr[i][j] ==5) {
found = true;
break;
}
}
}
```

5、switch 语句能否作用在 byte 上，能否作用在 long 上，能否作用在 String 上?

在 switch（expr1）中，expr1 只能是一个整数表达式或者枚举常量（更大字体），整数表达式可以是 int 基本类型或 Integer 包装类型，由于，byte,short,char 都可以隐含转换为 int，所以，这些类型以及这些类型的包装类型也是可以的。显然，long 和 String 类型都不符合 switch 的语法规定，并且不能被隐式转换成 int 类型，所以，它们不能作用于 swtich 语句中。

6、short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 += 1;有什么错?

对于 short s1 = 1; s1 = s1 + 1;由于 s1+1 运算时会自动提升表达式的类型，所以结果是 int 型，再赋值给 short 类型 s1 时，编译器将报告需要强制转换类型的错误。

对于 short s1 = 1; s1 += 1;由于　+=是 java 语言规定的运算符，java 编译器会对它进行特殊处理，因此可以正确编译。

7、char 型变量中能不能存贮一个中文汉字?为什么?

char 型变量是用来存储 Unicode 编码的字符的，unicode 编码字符集中包含了汉字，所以，char 型变量中当然可以存储汉字啦。不过，如果某个特殊的汉字没有被包含在 unicode 编码字符集中，那么，这个 char 型变量中就不能存储这个特殊汉字。补充说明：unicode 编码占用两个字节，所以，char 类型的变量也是占用两个字节。

备注：后面一部分回答虽然不是在正面回答题目，但是，为了展现自己的学识和表现自己对问题理解的透彻深入，可以回答一些相关的知识，做到知无不言，言无不尽。

8、用最有效率的方法算出 2 乘以 8 等於几?

2 << 3，

因为将一个数左移 n 位，就相当于乘以了 2 的 n 次方，那么，一个数乘以 8 只要将其左移 3 位即可，而位运算 cpu 直接支持的，效率最高，所以，2 乘以 8 等於几的最效率的方法是 2 << 3。

9、请设计一个一百亿的计算器

首先要明白这道题目的考查点是什么，一是大家首先要对计算机原理的底层细节要清楚、要知道加减法的位运算原理和知道计算机中的算术运算会发生越界的情况，二是要具备一定的面向对象的设计思想。

首先，计算机中用固定数量的几个字节来存储的数值，所以计算机中能够表示的数值是有一定的范围的，为了便于讲解和理解，我们先以 byte 类型的整数为例，它用 1 个字节进行存储，表示的最大数值范围为-128 到+127。-1 在内存中对应的二进制数据为 11111111，如果两个-1 相加，不考虑 Java 运算时的类型提升，运算后会产生进位，二进制结果为 1,11111110，由于进位后超过了 byte 类型的存储空间，所以进位部分被舍弃，即最终的结果为 11111110，也就是-2，这正好利用溢位的方式实现了负数的运算。-128 在内存中对应的二进制数据为 10000000，如果两个-128 相加，不考虑 Java 运算时的类型提升，运算后会产生进位，二进制结果为 1,00000000，由于进位后超过了 byte 类型的存储空间，所以进位部分被舍弃，即最终的结果为 00000000，也就是 0，这样的结果显然不是我们期望的，这说明计算机中的算术运算是会发生越界情况的，两个数值的运算结果不能超过计算机中的该类型的数值范围。由于 Java 中涉及表达式运算时的类型自动提升，我们无法用 byte 类型来做演示这种问题和现象的实验，大家可以用下面一个使用整数做实验的例子程序体验一下：

```
int a = Integer.MAX_VALUE;
int b = Integer.MAX_VALUE;
int sum = a + b;
System.out.println("a="+a+",b="+b+",sum="+sum);
```

先不考虑 long 类型，由于 int 的正数范围为 2 的 31 次方，表示的最大数值约等于 2*1000*1000*1000，也就是 20 亿的大小，所以，要实现一个一百亿的计算器，我们得自己设计一个类可以用于表示很大的整数，并且提供了与另外一个整数进行加减乘除的功能，大概功能如下：

（）这个类内部有两个成员变量，一个表示符号，另一个用字节数组表示数值的二进制数

（）有一个构造方法，把一个包含有多位数值的字符串转换到内部的符号和字节数组中

（）提供加减乘除的功能

```
public class BigInteger{
int sign;
byte[] val;
public Biginteger(String val) {
sign = ;
val = ;
}
public BigInteger add(BigInteger other) {
}
public BigInteger subtract(BigInteger other) {
}
public BigInteger multiply(BigInteger other){
}
public BigInteger divide(BigInteger other){
}
}
```

备注：要想写出这个类的完整代码，是非常复杂的，如果有兴趣的话，可以参看 jdk 中自带的 java.math.BigInteger 类的源码。面试的人也知道谁都不可能在短时间内写出这个类的完整代码的，他要的是你是否有这方面的概念和意识，他最重要的还是考查你的能力，所以，你不要因为自己无法写出完整的最终结果就放弃答这道题，你要做的就是你比别人写得多，证明你比别人强，你有这方面的思想意识就可以了，毕竟别人可能连题目的意思都看不懂，什么都没写，你要敢于答这道题，即使只答了一部分，那也与那些什么都不懂的人区别出来，拉开了距离，算是矮子中的高个，机会当然就属于你了。另外，答案中的框架代码也很重要，体现了一些面向对象设计的功底，特别是其中的方法命名很专业，用的英文单词很精准，这也是能力、经验、专业性、英语水平等多个方面的体现，会给人留下很好的印象，在编程能力和其他方面条件差不多的情况下，英语好除了可以使你获得更多机会外，薪水可以高出一千元。

10、使用 final 关键字修饰一个变量时，是引用不能变，还是引用的对象不能变？

使用 final 关键字修饰一个变量时，是指引用变量不能变，引用变量所指向的对象中的内容还是可以改变的。例如，对于如下语句：

```
final StringBuffer a=new StringBuffer("immutable");
```

执行如下语句将报告编译期错误：

```
a=new StringBuffer("");
```

但是，执行如下语句则可以通过编译：

```
a.append(" broken!");
```

有人在定义方法的参数时，可能想采用如下形式来阻止方法内部修改传进来的参数对象：

```
public void method(final StringBuffer param){
}
```

实际上，这是办不到的，在该方法内部仍然可以增加如下代码来修改参数对象：

```
param.append("a");
```

11、"=="和 equals 方法究竟有什么区别？

（单独把一个东西说清楚，然后再说清楚另一个，这样，它们的区别自然就出来了，混在一起说，则很难说清楚）

==操作符专门用来比较两个变量的值是否相等，也就是用于比较变量所对应的内存中所存储的数值是否相同，要比较两个基本类型的数据或两个引用变量是否相等，只能用==操作符。

如果一个变量指向的数据是对象类型的，那么，这时候涉及了两块内存，对象本身占用一块内存（堆内存），变量也占用一块内存，例如 Objet obj = newObject();变量 obj 是一个内存，new Object()是另一个内存，此时，变量 obj 所对应的内存中存储的数值就是对象占用的那块内存的首地址。对于指向对象类型的变量，如果要比较两个变量是否指向同一个对象，即要看这两个变量所对应的内存中的数值是否相等，这时候就需要用==操作符进行比较。

equals 方法是用于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的两个对象是独立的。例如，对于下面的代码：

String a=new String("foo");

String b=new String("foo");

两条 new 语句创建了两个对象，然后用 a/b 这两个变量分别指向了其中一个对象，这是两个不同的对象，它们的首地址是不同的，即 a 和 b 中存储的数值是不相同的，所以，表达式 a==b 将返回 false，而这两个对象中的内容是相同的，所以，表达式 a.equals(b)将返回 true。

在实际开发中，我们经常要比较传递进行来的字符串内容是否等，例如，String input =…;input.equals(“quit”)，许多人稍不注意就使用==进行比较了，这是错误的，随便从网上找几个项目实战的教学视频看看，里面就有大量这样的错误。记住，字符串的比较基本上都是使用 equals 方法。

如果一个类没有自己定义 equals 方法，那么它将继承 Object 类的 equals 方法，Object 类的 equals 方法的实现代码如下：

```
boolean equals(Object o){
return this==o;
}
```

这说明，如果一个类没有自己定义 equals 方法，它默认的 equals 方法（从 Object 类继承的）就是使用==操作符，也是在比较两个变量指向的对象是否是同一对象，这时候使用 equals 和使用==会得到同样的结果，如果比较的是两个独立的对象则总返回 false。如果你编写的类希望能够比较该类创建的两个实例对象的内容是否相同，那么你必须覆盖 equals 方法，由你自己写代码来决定在什么情况即可认为两个对象的内容是相同的。

12、静态变量和实例变量的区别？

在语法定义上的区别：静态变量前要加 static 关键字，而实例变量前则不加。

在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

例如，对于下面的程序，无论创建多少个实例对象，永远都只分配了一个 staticVar 变量，并且每创建一个实例对象，这个 staticVar 就会加 1；但是，每创建一个实例对象，就会分配一个 instanceVar，即可能分配多个 instanceVar，并且每个 instanceVar 的值都只自加了 1 次。

```
public class VariantTest{
public static int staticVar = 0;
public int instanceVar = 0;
public VariantTest(){
staticVar++;
instanceVar++;
System.out.println(“staticVar=” + staticVar + ”,instanceVar=”+ instanceVar);
}
}
```

备注：这个解答除了说清楚两者的区别外，最后还用一个具体的应用例子来说明两者的差异，体现了自己有很好的解说问题和设计案例的能力，思维敏捷，超过一般程序员，有写作能力！

13、是否可以从一个 static 方法内部发出对非 static 方法的调用？

不可以。因为非 static 方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而 static 方法调用时不需要创建对象，可以直接调用。也就是说，当一个 static 方法被调用时，可能还没有创建任何实例对象，如果从一个 static 方法中发出对非 static 方法的调用，那个非 static 方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个 static 方法内部发出对非 static 方法的调用。

14、Integer 与 int 的区别

int 是 java 提供的 8 种原始数据类型之一。Java 为每个原始类型提供了封装类，Integer 是 java 为 int 提供的封装类。int 的默认值为 0，而 Integer 的默认值为 null，即 Integer 可以区分出未赋值和值为 0 的区别，int 则无法表达出未赋值的情况，例如，要想表达出没有参加考试和考试成绩为 0 的区别，则只能使用 Integer。在 JSP 开发中，Integer 的默认为 null，所以用 el 表达式在文本框中显示时，值为空白字符串，而 int 默认的默认值为 0，所以用 el 表达式在文本框中显示时，结果为 0，所以，int 不适合作为 web 层的表单数据的类型。

在 Hibernate 中，如果将 OID 定义为 Integer 类型，那么 Hibernate 就可以根据其值是否为 null 而判断一个对象是否是临时的，如果将 OID 定义为了 int 类型，还需要在 hbm 映射文件中设置其 unsaved-value 属性为 0。

另外，Integer 提供了多个与整数相关的操作方法，例如，将一个字符串转换成整数，Integer 中还定义了表示整数的最大值和最小值的常量。

15、Math.round(11.5)等於多少? Math.round(-11.5)等於多少?

Math 类中提供了三个与取整有关的方法：ceil、floor、round，这些方法的作用与它们的英文名称的含义相对应，例如，ceil 的英文意义是天花板，该方法就表示向上取整，Math.ceil(11.3)的结果为 12,Math.ceil(-11.3)的结果是-11；floor 的英文意义是地板，该方法就表示向下取整，Math.ceil(11.6)的结果为 11,Math.ceil(-11.6)的结果是-12；最难掌握的是 round 方法，它表示“四舍五入”，算法为 Math.floor(x+0.5)，即将原来的数字加上 0.5 后再向下取整，所以，Math.round(11.5)的结果为 12，Math.round(-11.5)的结果为-11。

16、下面的代码有什么不妥之处?

1. if(username.equals(“zxx”){}

2. int x = 1;

return x==1?true:false;

17、请说出作用域 public，private，protected，以及不写时的区别

这四个作用域的可见范围如下表所示。

说明：如果在修饰的元素上面没有写任何访问修饰符，则表示 friendly。

作用域 当前类同一 package 子孙类其他 package

public √ √ √ √

protected √ √ √ ×

friendly √ √ × ×

private √ × × ×

备注：只要记住了有 4 种访问权限，4 个访问范围，然后将全选和范围在水平和垂直方向上分别按排从小到大或从大到小的顺序排列，就很容易画出上面的图了。

18、Overload 和 Override 的区别。Overloaded 的方法是否可以改变返回值的类型?

Overload 是重载的意思，Override 是覆盖的意思，也就是重写。

重载 Overload 表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写 Override 表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是 private 类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

至于 Overloaded 的方法是否可以改变返回值的类型这个问题，要看你倒底想问什么呢？这个题目很模糊。如果几个 Overloaded 的方法的参数列表不一样，它们的返回者类型当然也可以不一样。但我估计你想问的问题是：如果两个方法的参数列表完全一样，是否可以让它们的返回值不同来实现重载 Overload。这是不行的，我们可以用反证法来说明这个问题，因为我们有时候调用一个方法时也可以不定义返回结果变量，即不要关心其返回结果，例如，我们调用 map.remove(key)方法时，虽然 remove 方法有返回值，但是我们通常都不会定义接收返回结果的变量，这时候假设该类中有两个名称和参数列表完全相同的方法，仅仅是返回类型不同，java 就无法确定编程者倒底是想调用哪个方法了，因为它无法通过返回结果类型来判断。

override 可以翻译为覆盖，从字面就可以知道，它是覆盖了一个方法并且对其重写，以求达到不同的作用。对我们来说最熟悉的覆盖就是对接口方法的实现，在接口中一般只是对方法进行了声明，而我们在实现时，就需要实现接口声明的所有方法。除了这个典型的用法以外，我们在继承中也可能会在子类覆盖父类中的方法。在覆盖要注意以下的几点：

1、覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；

2、覆盖的方法的返回值必须和被覆盖的方法的返回一致；

3、覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；

4、被覆盖的方法不能为 private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

overload 对我们来说可能比较熟悉，可以翻译为重载，它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，VM 就会根据不同的参数样式，来选择合适的方法执行。在使用重载要注意以下的几点：

1、在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是 fun(int,float)，但是不能为 fun(int,int)）；

2、不能通过访问权限、返回类型、抛出的异常进行重载；

3、方法的异常类型和数目不会对重载造成影响；

4、对于继承来说，如果某一方法在父类中是访问权限是 priavte，那么就不能在子类对其进行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

19、构造器 Constructor 是否可被 override?

构造器 Constructor 不能被继承，因此不能重写 Override，但可以被重载 Overload。

20、接口是否可继承接口?抽象类是否可实现(implements)接口?抽象类是否可继承具体类(concrete class)?抽象类中是否可以有静态的 main 方法?

接口可以继承接口。抽象类可以实现(implements)接口，抽象类是否可继承具体类。抽象类中可以有静态的 main 方法。

备注：只要明白了接口和抽象类的本质和作用，这些问题都很好回答，你想想，如果你是 java 语言的设计者，你是否会提供这样的支持，如果不提供的话，有什么理由吗？如果你没有道理不提供，那答案就是肯定的了。

只有记住抽象类与普通类的唯一区别就是不能创建实例对象和允许有 abstract 方法。

21、写 clone()方法时，通常都有一行代码，是什么？

clone 有缺省行为，super.clone();因为首先要把父类中的成员复制到位，然后才是复制自己的成员。

22、面向对象的特征有哪些方面

计算机软件系统是现实生活中的业务在计算机中的映射，而现实生活中的业务其实就是一个个对象协作的过程。面向对象编程就是按现实业务一样的方式将程序代码按一个个对象进行组织和编写，让计算机系统能够识别和理解用对象方式组织和编写的程序代码，这样就可以把现实生活中的业务对象映射到计算机系统中。

面向对象的编程语言有，吗等 4 个主要的特征。

1 封装：

封装是保证软件部件具有优良的模块性的基础，封装的目标就是要实现软件部件的“高内聚、低耦合”，防止程序相互依赖性而带来的变动影响。在面向对象的编程语言中，对象是封装的最基本单位，面向对象的封装比传统语言的封装更为清晰、更为有力。面向对象的封装就是把描述一个对象的属性和行为的代码封装在一个“模块”中，也就是一个类中，属性用变量定义，行为用方法进行定义，方法可以直接访问同一个对象中的属性。通常情况下，只要记住让变量和访问这个变量的方法放在一起，将一个类中的成员变量全部定义成私有的，只有这个类自己的方法才可以访问到这些成员变量，这就基本上实现对象的封装，就很容易找出要分配到这个类上的方法了，就基本上算是会面向对象的编程了。把握一个原则：把对同一事物进行操作的方法和相关的方法放在同一个类中，把方法和它操作的数据放在同一个类中。

例如，人要在黑板上画圆，这一共涉及三个对象：人、黑板、圆，画圆的方法要分配给哪个对象呢？由于画圆需要使用到圆心和半径，圆心和半径显然是圆的属性，如果将它们在类中定义成了私有的成员变量，那么，画圆的方法必须分配给圆，它才能访问到圆心和半径这两个属性，人以后只是调用圆的画圆方法、表示给圆发给消息而已，画圆这个方法不应该分配在人这个对象上，这就是面向对象的封装性，即将对象封装成一个高度自治和相对封闭的个体，对象状态（属性）由这个对象自己的行为（方法）来读取和改变。一个更便于理解的例子就是，司机将火车刹住了，刹车的动作是分配给司机，还是分配给火车，显然，应该分配给火车，因为司机自身是不可能有那么大的力气将一个火车给停下来的，只有火车自己才能完成这一动作，火车需要调用内部的离合器和刹车片等多个器件协作才能完成刹车这个动作，司机刹车的过程只是给火车发了一个消息，通知火车要执行刹车动作而已。

抽象：

抽象就是找出一些事物的相似和共性之处，然后将这些事物归为一个类，这个类只考虑这些事物的相似和共性之处，并且会忽略与当前主题和目标无关的那些方面，将注意力集中在与当前目标有关的方面。例如，看到一只蚂蚁和大象，你能够想象出它们的相同之处，那就是抽象。抽象包括行为抽象和状态抽象两个方面。例如，定义一个 Person 类，如下：

```
classPerson{
String name;
int age;
}
```

人本来是很复杂的事物，有很多方面，但因为当前系统只需要了解人的姓名和年龄，所以上面定义的类中只包含姓名和年龄这两个属性，这就是一种抽像，使用抽象可以避免考虑一些与目标无关的细节。我对抽象的理解就是不要用显微镜去看一个事物的所有方面，这样涉及的内容就太多了，而是要善于划分问题的边界，当前系统需要什么，就只考虑什么。

继承：

在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并可以加入若干新的内容，或修改原来的方法使之更适合特殊的需要，这就是继承。继承是子类自动共享父类数据和方法的机制，这是类之间的一种关系，提高了软件的可重用性和可扩展性。

多态：

多态是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这

就是多态性。多态性增强了软件的灵活性和扩展性。例如，下面代码中的 UserDao 是一个接口，它定义引用变量 userDao 指向的实例对象由 daofactory.getDao()在执行的时候返回，有时候指向的是 UserJdbcDao 这个实现，有时候指向的是 UserHibernateDao 这个实现，这样，不用修改源代码，就可以改变 userDao 指向的具体类实现，从而导致 userDao.insertUser()方法调用的具体代码也随之改变，即有时候调用的是 UserJdbcDao 的 insertUser 方法，有时候调用的是 UserHibernateDao 的 insertUser 方法：

UserDao userDao =daofactory.getDao();

userDao.insertUser(user);

比喻：人吃饭，你看到的是左手，还是右手？

23、java 中实现多态的机制是什么？

靠的是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

24、abstract class 和 interface 有什么区别?

含有 abstract 修饰符的 class 即为抽象类，abstract 类不能创建的实例对象。含有 abstract 方法的类必须定义为 abstract class，abstract class 类中的方法不必是抽象的。abstract class 类中定义抽象方法必须在具体(Concrete)子类中实现，所以，不能有抽象构造方法或抽象静态方法。如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为 abstract 类型。

接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为 public abstract 类型，接口中的成员变量类型默认为 public static final。

下面比较一下两者的语法区别：

1.抽象类可以有构造方法，接口中不能有构造方法。

2.抽象类中可以有普通成员变量，接口中没有普通成员变量

3.抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。

4. 抽象类中的抽象方法的访问类型可以是 public，protected 和（默认类型,虽然

eclipse 下不报错，但应该也不行），但接口中的抽象方法只能是 public 类型的，并且默认即为 public abstract 类型。

5. 抽象类中可以包含静态方法，接口中不能包含静态方法

6. 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是 public static final 类型，并且默认即为 public static final 类型。

7. 一个类可以实现多个接口，但只能继承一个抽象类。

下面接着再说说两者在应用上的区别：

接口更多的是在系统架构设计方法发挥作用，主要用于定义模块之间的通信契约。而抽象类在代码实现方面发挥作用，可以实现代码的重用，例如，模板方法设计模式是抽象类的一个典型应用，假设某个项目的所有 Servlet 类都要用相同的方式进行权限判断、记录访问日志和处理异常，那么就可以定义一个抽象的基类，让所有的 Servlet 都继承这个抽象基类，在抽象基类的 service 方法中完成权限判断、记录访问日志和处理异常的代码，在各个子类中只是完成各自的业务逻辑代码，伪代码如下：

```
public abstract classBaseServlet extends HttpServlet{
public final void service(HttpServletRequest request,HttpServletResponse response) throws IOExcetion,ServletException {
记录访问日志
进行权限判断
if(具有权限){
try{
doService(request,response);
}
catch(Excetpion e) {
```

记录异常信息

}

}

}

protected abstract void doService(HttpServletRequest request,HttpServletResponse response) throws

IOExcetion,ServletException;

//注意访问权限定义成 protected，显得既专业，又严谨，因为它是专门给子类用的

}

public class MyServlet1 extendsBaseServlet

{

protected voiddoService(HttpServletRequest request, HttpServletResponse response) throwsIOExcetion,ServletException

{

本 Servlet 只处理的具体业务逻辑代码

}

}

父类方法中间的某段代码不确定，留给子类干，就用模板方法设计模式。

备注：这道题的思路是先从总体解释抽象类和接口的基本概念，然后再比较两者的语法细节，最后再说两者的应用区别。比较两者语法细节区别的条理是：先从一个类中的构造方法、普通成员变量和方法（包括抽象方法），静态变量和方法，继承性等 6 个方面逐一去比较回答，接着从第三者继承的角度的回答，特别是最后用了一个典型的例子来展现自己深厚的技术功底。

25、abstract 的 method 是否可同时是 static,是否可同时是 native，是否可同时是 synchronized?

abstract 的 method 不可以是 static 的，因为抽象的方法是要被子类实现的，而 static 与子类扯不上关系！

native 方法表示该方法要用另外一种依赖平台的编程语言实现的，不存在着被子类实现的问题，所以，它也不能是抽象的，不能与 abstract 混用。例如，FileOutputSteam 类要硬件打交道，底层的实现用的是操作系统相关的 api 实现，例如，在 windows 用 c 语言实现的，所以，查看 jdk 的源代码，可以发现 FileOutputStream 的 open 方法的定义如下：

private native void open(Stringname) throws FileNotFoundException;

如果我们要用 java 调用别人写的 c 语言函数，我们是无法直接调用的，我们需要按照 java 的要求写一个 c 语言的函数，又我们的这个 c 语言函数去调用别人的 c 语言函数。由于我们的 c 语言函数是按 java 的要求来写的，我们这个 c 语言函数就可以与 java 对接上，java 那边的对接方式就是定义出与我们这个 c 函数相对应的方法，java 中对应的方法不需要写具体的代码，但需要在前面声明 native。

关于 synchronized 与 abstract 合用的问题，我觉得也不行，因为在我几年的学习和开发中，从来没见到过这种情况，并且我觉得 synchronized 应该是作用在一个具体的方法上才有意义。而且，方法上的 synchronized 同步所使用的同步锁对象是 this，而抽象方法上无法确定 this 是什么。

26、什么是内部类？Static Nested Class 和 Inner Class 的不同。

内部类就是在一个类的内部定义的类，内部类中不能定义静态成员（静态成员不是对象的特性，只是为了找一个容身之处，所以需要放到一个类中而已，这么一点小事，你还要把它放到类内部的一个类中，过分了啊！提供内部类，不是为让你干这种事情，无聊，不让你干。我想可能是既然静态成员类似 c 语言的全局变量，而内部类通常是用于创建内部对象用的，所以，把“全局变量”放在内部类中就是毫无意义的事情，既然是毫无意义的事情，就应该被禁止），内部类可以直接访问外部类中的成员变量，内部类可以定义在外部类的方法外面，也可以定义在外部类的方法体中，如下所示：

public class Outer

{

```
int out_x = 0;
public void method()
{
Inner1 inner1 = new Inner1();
public class Inner2 //在方法体内部定义的内部类
{
public method()
{
out_x = 3;
}
}
Inner2 inner2 = new Inner2();
}
public class Inner1 //在方法体外面定义的内部类
{
}
}
```

在方法体外面定义的内部类的访问类型可以是 public,protecte,默认的，private 等 4 种类型，这就好像类中定义的成员变量有 4 种访问类型一样，它们决定这个内部类的定义对其他类是否可见；对于这种情况，我们也可以在外面创建内部类的实例对象，创建内部类的实例对象时，一定要先创建外部类的实例对象，然后用这个外部类的实例对象去创建内部类的实例对象，代码如下：

```
Outer outer = new Outer();
Outer.Inner1 inner1 = outer.newInnner1();
```

在方法内部定义的内部类前面不能有访问类型修饰符，就好像方法中定义的局部变量一样，但这种内部类的前面可以使用 final 或 abstract 修饰符。这种内部类对其他类是不可见的其他类无法引用这种内部类，但是这种内部类创建的实例对象可以传递给其他类访问。这种内部类必须是先定义，后使用，即内部类的定义代码必须出现在使用该类之前，这与方法中的局部变量必须先定义后使用的道理也是一样的。这种内部类可以访问方法体中的局部变量，但是，该局部变量前必须加 final 修饰符。

对于这些细节，只要在 eclipse 写代码试试，根据开发工具提示的各类错误信息就可以马上了解到。

在方法体内部还可以采用如下语法来创建一种匿名内部类，即定义某一接口或类的子类的同时，还创建了该子类的实例对象，无需为该子类定义名称：

```
public class Outer
{
public void start()
{
new Thread(
new Runable(){
public void run(){};
}
).start();
}
}
```

最后，在方法外部定义的内部类前面可以加上 static 关键字，从而成为 Static Nested Class，它不再具有内部类的特性，所有，从狭义上讲，它不是内部类。Static Nested Class 与普通类在运行时的行为和功能上没有什么区别，只是在编程引用时的语法上有一些差别，它可以定义成 public、protected、默认的、private 等多种类型，而普通类只能定义成 public 和默认的这两种类型。在外面引用 Static Nested Class 类的名称为“外部类名.内部类名”。在外面不需要创建外部类的实例对象，就可以直接创建 Static Nested Class，例如，假设 Inner 是定义在 Outer 类中的 Static Nested Class，那么可以使用如下语句创建 Inner 类：

Outer.Inner inner = newOuter.Inner();

由于 static Nested Class 不依赖于外部类的实例对象，所以，static Nested Class 能访问外部类的非 static 成员变量。当在外部类中访问 Static Nested Class 时，可以直接使用 Static Nested Class 的名字，而不需要加上外部类的名字了，在 Static Nested Class 中也可以直接引用外部类的 static 的成员变量，不需要加上外部类的名字。

在静态方法中定义的内部类也是 Static Nested Class，这时候不能在类前面加 static 关键字，静态方法中的 Static Nested Class 与普通方法中的内部类的应用方式很相似，它除了可以直接访问外部类中的 static 的成员变量，还可以访问静态方法中的局部变量，但是，该局部变量前必须加 final 修饰符。

备注：首先根据你的印象说出你对内部类的总体方面的特点：例如，在两个地方可以定义，可以访问外部类的成员变量，不能定义静态成员，这是大的特点。然后再说一些细节方面的知识，例如，几种定义方式的语法区别，静态内部类，以及匿名内部类。

27、内部类可以引用它的包含类的成员吗？有没有什么限制？

完全可以。如果不是静态内部类，那没有什么限制！

如果你把静态嵌套类当作内部类的一种特例，那在这种情况下不可以访问外部类的普通成员变量，而只能访问外部类中的静态成员，例如，下面的代码：

```
class Outer
{
static int x;
static class Inner
{
void test()
{
syso(x);
}
}
}
```

答题时，也要能察言观色，揣摩提问者的心思，显然人家希望你说的是静态内部类不能访问外部类的成员，但你一上来就顶牛，这不好，要先顺着人家，让人家满意，然后再说特殊情况，让人家吃惊。

28、Anonymous Inner Class (匿名内部类)是否可以 extends(继承)其它类，是否可以 implements(实现)interface(接口)?

可以继承其他类或实现其他接口。不仅是可以，而是必须!

29、super.getClass()方法调用

下面程序的输出结果是多少？

```
importjava.util.Date;
public classTestextends Date{
public static voidmain(String[] args) {
new Test().test();
}
```

```
public void test(){
System.out.println(super.getClass().getName());
}
}
```

很奇怪，结果是 Test

这属于脑筋急转弯的题目，在一个 qq 群有个网友正好问过这个问题，我觉得挺有趣，就研究了一下，没想到今天还被你面到了，哈哈。

在 test 方法中，直接调用 getClass().getName()方法，返回的是 Test 类名

由于 getClass()在 Object 类中定义成了 final，子类不能覆盖该方法，所以，在 test 方法中调用 getClass().getName()方法，其实就是在调用从父类继承的 getClass()方法，等效于调用 super.getClass().getName()方法，所以，super.getClass().getName()方法返回的也应该是 Test。

如果想得到父类的名称，应该用如下代码：

```
getClass().getSuperClass().getName();
```

30、String 是最基本的数据类型吗?

基本数据类型包括 byte、int、char、long、float、double、boolean 和 short。

java.lang.String 类是 final 类型的，因此不可以继承这个类、不能修改这个类。为了提高效率节省空间，我们应该用 StringBuffer 类

31、String s = "Hello";s = s + " world!";这两行代码执行后，原始的 String 对象中的内容到底变了没有？

没有。因为 String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。在这段代码中，s 原先指向一个 String 对象，内容是 "Hello"，然后我们对 s 进行了+操作，那么 s 所指向的那个对象是否发生了改变呢？答案是没有。这时，s 不指向原来那个对象了，而指向了另一个 String 对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的话会引起很大的内存开销。因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始化，把它设置为初始值，应当这样做：

```
public class Demo {
private String s;
...
public Demo {
s = "Initial Value";
}
...
}
```

而非

```
s = new String("Initial Value");
```

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个 String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性 s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。而用关键字 new 调用构造器，总是会创建一个新的对象，无论内容是否相同。

至于为什么要把 String 类设计成不可变类，是它的用途决定的。其实不只 String，很多 Java 标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以 Java 标准类库还提供了一个可变版本，即 StringBuffer。

32、是否可以继承 String 类?

String 类是 final 类故不可以继承。

33、String s = new String("xyz");创建了几个 String Object?二者之间有什么区别？

两个或一个，”xyz”对应一个对象，这个对象放在字符串常量缓冲区，常量”xyz”不管出现多少遍，都是缓冲区中的那一个。New String 每写一遍，就创建一个新的对象，它一句那个常量”xyz”对象的内容来创建出一个新 String 对象。如果以前就用过’xyz’，这句代表就不会创建”xyz”自己了，直接从缓冲区拿。

34、String 和 StringBuffer 的区别

JAVA 平台提供了两个类：String 和 StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个 String 类提供了数值不可改变的字符串。而这个 StringBuffer 类提供的字符串进行修改。当你知道字符数据要改变的时候你就可以使用 StringBuffer。典型地，你可以使用 StringBuffers 来动态构造字符数据。另外，String 实现了 equals 方法，new String(“abc”).equals(newString(“abc”)的结果为 true,而 StringBuffer 没有实现 equals 方法，所以，new StringBuffer(“abc”).equals(newStringBuffer(“abc”)的结果为 false。

接着要举一个具体的例子来说明，我们要把 1 到 100 的所有数字拼起来，组成一个串。

```
StringBuffer sbf = new StringBuffer();
for(int i=0;i<100;i++)
{
sbf.append(i);
}
```

上面的代码效率很高，因为只创建了一个 StringBuffer 对象，而下面的代码效率很低，因为创建了 101 个对象。

```
String str = new String();
for(int i=0;i<100;i++)
{
str = str + i;
}
```

在讲两者区别时，应把循环的次数搞成 10000，然后用 endTime-beginTime 来比较两者执行的时间差异，最后还要讲讲 StringBuilder 与 StringBuffer 的区别。

String 覆盖了 equals 方法和 hashCode 方法，而 StringBuffer 没有覆盖 equals 方法和 hashCode 方法，所以，将 StringBuffer 对象存储进 Java 集合类中时会出现问题。

35、如何把一段逗号分割的字符串转换成一个数组?

如果不查 jdk api，我很难写出来！我可以说说我的思路：

1 用正则表达式，代码大概为：String [] result = orgStr.split(“,”);

2 用 StingTokenizer ,代码为：StringTokenizer tokener = StringTokenizer(orgStr,”,”);

String [] result =new String[tokener .countTokens()];

Int i=0;

while(tokener.hasNext(){result[i++]=toker.nextToken();}

36、数组有没有 length()这个方法? String 有没有 length()这个方法？

数组没有 length()这个方法，有 length 的属性。String 有有 length()这个方法。

37、下面这条语句一共创建了多少个对象：String s="a"+"b"+"c"+"d";

答：对于如下代码：

String s1 = "a";

String s2 = s1 + "b";

String s3 = "a" + "b";

System.out.println(s2 == "ab");

System.out.println(s3 == "ab");

第一条语句打印的结果为 false，第二条语句打印的结果为 true，这说明 javac 编译可以对字符串常量直接相加的表达式进行优化，不必要等到运行期去进行加法运算处理，而是在编译时去掉其中的加号，直接将其编译成一个这些常量相连的结果。

题目中的第一行代码被编译器在编译时优化后，相当于直接定义了一个”abcd”的字符串，所以，上面的代码应该只创建了一个 String 对象。写如下两行代码，

String s ="a" + "b" + "c" + "d";

System.out.println(s== "abcd");

最终打印的结果应该为 true。

38、try {}里有一个 return 语句，那么紧跟在这个 try 后的 finally {}里的 code 会不会被执行，什么时候被执行，在 return 前还是后?

也许你的答案是在 return 之前，但往更细地说，我的答案是在 return 中间执行，请看下面程序代码的运行结果：

public classTest {

/**

* @paramargs add by zxx ,Dec 9, 2008

*/

public static voidmain(String[] args) {

// TODO Auto-generated method stub

System.out.println(newTest().test());;

}

static int test()

{

int x = 1;

try

{

returnx;

}

finally

{

++x;

}

```
}
}
```

---------执行结果 ---------

1

运行结果是 1，为什么呢？主函数调用子函数并得到结果的过程，好比主函数准备一个空罐子，当子函数要返回结果时，先把结果放在罐子里，然后再将程序逻辑返回到主函数。所谓返回，就是子函数说，我不运行了，你主函数继续运行吧，这没什么结果可言，结果是在说这话之前放进罐子里的。

39、下面的程序代码输出的结果是多少？

```
public class smallT
{
public staticvoid main(String args[])
{
smallT t = new smallT();
int b = t.get();
System.out.println(b);
}
public int get()
{
try
{
return1 ;
}
finally
{
return2 ;
}
}
}
```

返回的结果是 2。

我可以通过下面一个例子程序来帮助我解释这个答案，从下面例子的运行结果中可以发现，try 中的 return 语句调用的函数先于 finally 中调用的函数执行，也就是说 return 语句先执行，finally 语句后执行，所以，返回的结果是 2。Return 并不是让函数马上返回，而是 return 语句执行后，将把返回结果放置进函数栈中，此时函数并不是马上返回，它要执行 finally 语句后才真正开始返回。

在讲解答案时可以用下面的程序来帮助分析：

```
public classTest {
/**
* @paramargs add by zxx ,Dec 9, 2008
*/
public static voidmain(String[] args) {
// TODO Auto-generated method stub
System.out.println(newTest().test());;
```

```
}
int test()
{
try
{
return func1();
}
finally
{
return func2();
}
}
int func1()
{
System.out.println("func1");
return 1;
}
int func2()
{
System.out.println("func2");
return 2;
}
}
```

----------执行结果----------

```
func1
func2
2
```

结论：finally 中的代码比 return 和 break 语句后执行

40、final, finally, finalize 的区别。

final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。

内部类要访问局部变量，局部变量必须定义成 final 类型，例如，一段代码……

finally 是异常处理语句结构的一部分，表示总是执行。

finalize 是 Object 类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。JVM 不保证此方法总被调用

41、运行时异常与一般异常有何异同？

异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

42、error 和 exception 有什么区别?

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。 exception 表示一种设计或实现问题。也就是说，它表示如果程序运行正常，从不会发生的情况。

43、Java 中的异常处理机制的简单原理和应用。

异常是指 java 程序运行时（非编译）所发生的非正常情况或错误，与现实生活中的事件很相似，现实生活中的事件可以包含事件发生的时间、地点、人物、情节等信息，可以用一个对象来表示，Java 使用面向对象的方式来处理异常，它把程序中发生的每个异常也都分别封装到一个对象来表示的，该对象中包含有异常的信息。

Java 对异常进行了分类，不同类型的异常分别用不同的 Java 类表示，所有异常的根类为 java.lang.Throwable，Throwable 下面又派生了两个子类：Error 和 Exception，Error 表示应用程序本身无法克服和恢复的一种严重问题，程序只有死的份了，例如，说内存溢出和线程死锁等系统问题。Exception 表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常，系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件死掉，例如，数组脚本越界（ArrayIndexOutOfBoundsException），空指针异常（NullPointerException）、类转换异常（ClassCastException）；普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。

java 为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须 try..catch 处理或用 throws 声明继续抛给上层调用方法处理，所以普通异常也称为 checked 异常，而系统异常可以处理也可以不处理，所以，编译器不强制用 try..catch 处理或用 throws 声明，所以系统异常也称为 unchecked 异常。

提示答题者：就按照三个级别去思考：虚拟机必须宕机的错误，程序可以死掉也可以不死掉的错误，程序不应该死掉的错误；

44、请写出你最常见到的 5 个 runtime exception。

这道题主要考你的代码量到底多大，如果你长期写代码的，应该经常都看到过一些系统方面的异常，你不一定真要回答出 5 个具体的系统异常，但你要能够说出什么是系统异常，以及几个系统异常就可以了，当然，这些异常完全用其英文名称来写是最好的，如果实在写不出，那就用中文吧，有总比没有强！

所谓系统异常，就是.....，它们都是 RuntimeException 的子类，在 jdk doc 中查 RuntimeException 类，就可以看到其所有的子类列表，也就是看到了所有的系统异常。我比较有印象的系统异常有：NullPointerException、ArrayIndexOutOfBoundsException、ClassCastException。

45、JAVA 语言如何进行异常处理，关键字：throws,throw,try,catch,finally 分别代表什么意义？在 try 块中可以抛出异常吗？

46、java 中有几种方法可以实现一个线程？用什么关键字修饰同步方法? stop()和 suspend()方法为何不推荐使用？

java5 以前，有如下两种：

第一种：

new Thread(){}.start();这表示调用 Thread 子类对象的 run 方法，new Thread(){}表示一个 Thread 的匿名子类的实例对象，子类加上 run 方法后的代码如下：

```
new Thread(){
public void run(){
}
}.start();
```

第二种：

new Thread(new Runnable(){}).start();这表示调用 Thread 对象接受的 Runnable 对象的 run 方法，new Runnable(){}表示一个 Runnable 的匿名子类的实例对象,runnable 的子类加上 run 方法后的代码如下：

```
new Thread(new Runnable(){
public voidrun(){
}
}
).start();
```

从 java5 开始，还有如下一些线程池创建多线程的方式：

ExecutorService pool = Executors.newFixedThreadPool(3)

for(int i=0;i<10;i++)

{

pool.execute(newRunable(){public void run(){}});

}

Executors.newCachedThreadPool().execute(new Runable(){publicvoid run(){}});

Executors.newSingleThreadExecutor().execute(new Runable(){publicvoid run(){}});

有两种实现方法，分别使用 new Thread()和 new Thread(runnable)形式，第一种直接调用 thread 的 run 方法，所以，我们往往使用 Thread 子类，即 new SubThread()。第二种调用 runnable 的 run 方法。

有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口

用 synchronized 关键字修饰同步方法

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。

47、sleep()和 wait()有什么区别?

（网上的答案：sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用 sleep 不会释放对象锁。 wait 是 Object 类的方法，对此对象调用 wait 方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出 notify 方法（或 notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。）

sleep 就是正在执行的线程主动让出 cpu，cpu 去执行其他线程，在 sleep 指定的时间过后，cpu 才会回到这个线程上继续往下执行，如果当前线程进入了同步锁，sleep 方法并不会释放锁，即使当前线程使用 sleep 方法让出了 cpu，但其他被同步锁挡住了的线程也无法得到执行。wait 是指在一个已经进入了同步锁的线程内，让自己暂时让出同步锁，以便其他正在等待此锁的线程可以得到同步锁并运行，只有其他线程调用了 notify 方法（notify 并不释放锁，只是告诉调用过 wait 方法的线程可以去参与获得锁的竞争了，但不是马上得到锁，因为锁还在别人手里，别人还没释放。如果 notify 方法后面的代码还有很多，需要这些代码执行完后才会释放锁，可以在 notfiy 方法后增加一个等待和一些代码，看看效果），调用 wait 方法的线程就会解除 wait 状态和程序可以再次得到锁后继续向下运行。对于 wait 的讲解一定要配合例子代码来说明，才显得自己真明白。

package com.huawei.interview;

publicclass MultiThread {

/**

* @paramargs

*/

public static voidmain(String[] args) {

// TODO Auto-generated method stub

new Thread(newThread1()).start();

try {

Thread.sleep(10);

```
} catch (InterruptedException e) {
// TODO Auto-generated catchblock
e.printStackTrace();
}
new Thread(newThread2()).start();
}
private static classThread1implements Runnable
{
@Override
public void run() {
// TODO Auto-generated methodstub
//由于这里的 Thread1 和下面的 Thread2 内部 run 方法要用同一对象作为监视器，我们这里不能用 this，因为在 Thread2 里面的 this 和这个 Thread1 的 this 不是同一个对象。我们用 MultiThread.class 这个字节码对象，当前虚拟机里引用这个变量时，指向的都是同一个对象。
synchronized (MultiThread.class){
System.out.println("enterthread1...");
System.out.println("thread1is waiting");
try {
//释放锁有两种方式，第一种方式是程序自然离开监视器的范围，也就是离开了 synchronized 关键字管辖的代码范围，另一种方式就是在 synchronized 关键字管辖的代码内部调用监视器对象的 wait 方法。这里，使用 wait 方法释放锁。
MultiThread.class.wait();
} catch(InterruptedException e) {
// TODO Auto-generatedcatch block
e.printStackTrace();
}
System.out.println("thread1is going on...");
System.out.println("thread1is being over!");
}
}
}
private static classThread2implements Runnable
{
@Override
public void run() {
// TODO Auto-generated methodstub
synchronized (MultiThread.class){
System.out.println("enterthread2...");
System.out.println("thread2notify other thread can release wait status..");
//由于 notify 方法并不释放锁，即使 thread2 调用下面的 sleep 方法休息了 10 毫秒，但 thread1 仍然不会执行，因为 thread2 没有释放锁，所以 Thread1 无法得不到锁。
```

```
MultiThread.class.notify();
System.out.println("thread2is sleeping ten millisecond...");
try {
Thread.sleep(10);
} catch (InterruptedExceptione) {
// TODO Auto-generatedcatch block
e.printStackTrace();
}
System.out.println("thread2is going on...");
System.out.println("thread2is being over!");
}
}
}
}
```

48、同步和异步有何异同，在什么情况下分别使用他们？举例说明。

如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。

当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

49. 下面两个方法同步吗？（自己发明）

```
class Test
{
synchronizedstatic voidsayHello3()
{
}
synchronizedvoid getX(){}
}
```

50、多线程有几种实现方法?同步有几种实现方法?

多线程有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口

同步的实现方面有两种，分别是 synchronized,wait 与 notify

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。

51、启动一个线程是用 run()还是 start()? .

启动一个线程是调用 start()方法，使线程就绪状态，以后可以被调度为运行状态，一个线程必须关联一些具体的执行代码，run()方法是该线程所关联的执行代码。

52、当一个线程进入一个对象的一个 synchronized 方法后，其它线程是否可进入此对象的其它方法?

分几种情况：

1.其他方法前是否加了 synchronized 关键字，如果没加，则能。

2.如果这个方法内部调用了 wait，则可以进入其他 synchronized 方法。

3.如果其他个方法都加了 synchronized 关键字，并且内部没有调用 wait，则不能。

4.如果其他方法是 static，它用的同步锁是当前类的字节码，与非静态的方法不能同步，因为非静态的方法用的是 this。

53、线程的基本概念、线程的基本状态以及状态之间的关系

一个程序中可以有多条执行线索同时执行，一个线程就是程序中的一条执行线索，每个线程上都关联有要执行的代码，即可以有多段程序代码同时运行，每个程序至少都有一个线程，即 main 方法执行的那个线程。如果只是一个 cpu，它怎么能够同时执行多段程序呢？这是从宏观上来看的，cpu 一会执行 a 线索，一会执行 b 线索，切换时间很快，给人的感觉是 a,b 在同时执行，好比大家在同一个办公室上网，只有一条链接到外部网线，其实，这条网线一会为 a 传数据，一会为 b 传数据，由于切换时间很短暂，所以，大家感觉都在同时上网。

状态：就绪，运行，synchronize 阻塞，wait 和 sleep 挂起，结束。wait 必须在 synchronized 内部调用。

调用线程的 start 方法后线程进入就绪状态，线程调度系统将就绪状态的线程转为运行状态，遇到 synchronized 语句时，由运行状态转为阻塞，当 synchronized 获得锁后，由阻塞转为运行，在这种情况可以调用 wait 方法转为挂起状态，当线程关联的代码执行完后，线程变为结束状态。

54、简述 synchronized 和 java.util.concurrent.locks.Lock 的异同？

主要相同点：Lock 能完成 synchronized 所实现的所有功能

主要不同点：Lock 有比 synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而 Lock 一定要求程序员手工释放，并且必须在 finally 从句中释放。Lock 还有更强大的功能，例如，它的 tryLock 方法可以非阻塞方式去拿锁。

举例说明（对下面的题用 lock 进行了改写）：

```
package com.huawei.interview;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
publicclass ThreadTest {
/**
* @paramargs
*/
private int j;
private Lock lock =newReentrantLock();
public static voidmain(String[] args) {
// TODO Auto-generated method stub
ThreadTest tt = new ThreadTest();
for(int i=0;i<2;i++)
{
new Thread(tt.new Adder()).start();
new Thread(tt.new Subtractor()).start();
}
}
private class SubtractorimplementsRunnable
```

```
{
@Override
public void run() {
// TODO Auto-generated methodstub
while(true)
{
/*synchronized (ThreadTest.this) {
System.out.println("j--="+ j--);
//这里抛异常了，锁能释放吗？
}*/
lock.lock();
try
{
System.out.println("j--="+ j--);
}finally
{
lock.unlock();
}
}
}
}
private class AdderimplementsRunnable
{
@Override
public void run() {
// TODO Auto-generated methodstub
while(true)
{
/*synchronized (ThreadTest.this) {
System.out.println("j++="+ j++);
}*/
lock.lock();
try
{
System.out.println("j++="+ j++);
}finally
{
lock.unlock();
}
}
```

```
}
}
}
```

55、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。写出程序。

以下程序使用内部类实现线程，对 j 增减的时候没有考虑顺序问题。

```
public class ThreadTest1
{
private int j;
public static void main(String args[]){
ThreadTest1 tt=newThreadTest1();
Inc inc=tt.new Inc();
Dec dec=tt.new Dec();
for(inti=0;i<2;i++){
Thread t=newThread(inc);
t.start();
t=new Thread(dec);
t.start();
}
}
private synchronized void inc(){
j++;
System.out.println(Thread.currentThread().getName()+"-inc:"+j);
}
private synchronized void dec(){
j--;
System.out.println(Thread.currentThread().getName()+"-dec:"+j);
}
class Inc implements Runnable{
public void run(){
for(inti=0;i<100;i++){
inc();
}
}
}
class Dec implements Runnable{
public void run(){
for(inti=0;i<100;i++){
dec();
}
}
```

```
}
}
----------随手再写的一个----------
class A
{
JManger j =new JManager();
main()
{
new A().call();
}
void call
{
for(int i=0;i<2;i++)
{
new Thread(
newRunnable(){ public void run(){while(true){j.accumulate()}}}
).start();
new Thread(newRunnable(){ public void run(){while(true){j.sub()}}}).start();
}
}
}
class JManager
{
private j = 0;
public synchronized voidsubtract()
{
j--
}
public synchronized voidaccumulate()
{
j++;
}
}
```

56、子线程循环 10 次，接着主线程循环 100，接着又回到子线程循环 10 次，接着再回到主线程又循环 100，如此循环 50 次，请写出程序。

最终的程序代码如下：

```
publicclass ThreadTest {
/**
* @paramargs
*/
```

```
public static voidmain(String[] args) {
// TODO Auto-generated method stub
new ThreadTest().init();
}
public void init()
{
final Business business =newBusiness();
new Thread(
new Runnable()
{
public voidrun() {
for(inti=0;i<50;i++)
{
business.SubThread(i);
}
}
}
).start();
for(int i=0;i<50;i++)
{
business.MainThread(i);
}
}
private class Business
{
booleanbShouldSub =true;//这里相当于定义了控制该谁执行的一个信号灯
public synchronized voidMainThread(int i)
{
if(bShouldSub)
try {
this.wait();
} catch(InterruptedException e) {
// TODO Auto-generatedcatch block
e.printStackTrace();
}
for(int j=0;j<5;j++)
{
System.out.println(Thread.currentThread().getName()+ ":i=" + i +",j=" + j);
}
bShouldSub =true;
```

```
this.notify();
}
public synchronized voidSubThread(int i)
{
if(!bShouldSub)
try {
this.wait();
} catch (InterruptedExceptione) {
// TODO Auto-generatedcatch block
e.printStackTrace();
}
for(intj=0;j<10;j++)
{
System.out.println(Thread.currentThread().getName()+ ":i=" + i +",j=" + j);
}
bShouldSub =false;
this.notify();
}
}
}
```

备注：不可能一上来就写出上面的完整代码，最初写出来的代码如下，问题在于两个线程的代码要参照同一个变量，即这两个线程的代码要共享数据，所以，把这两个线程的执行代码搬到同一个类中去：

```
package com.huawei.interview.lym;
publicclass ThreadTest {
private static booleanbShouldMain=false;
public static void main(String[]args) {
// TODO Auto-generated method stub
/*new Thread(){
public void run()
{
for(int i=0;i<50;i++)
{
for(int j=0;j<10;j++)
{
System.out.println("i="+ i + ",j=" + j);
}
}
}
}.start();*/
//final String str = newString("");
```

```
new Thread(
new Runnable()
{
public voidrun()
{
for(inti=0;i<50;i++)
{
synchronized(ThreadTest.class) {
if(bShouldMain)
{
try {
ThreadTest.class.wait();}
catch(InterruptedException e) {
e.printStackTrace();
}
}
for(intj=0;j<10;j++)
{
System.out.println(
Thread.currentThread().getName()+
"i="+ i + ",j=" + j);
}
bShouldMain= true;
ThreadTest.class.notify();
}
}
}
}
).start();
for(int i=0;i<50;i++)
{
synchronized (ThreadTest.class){
if(!bShouldMain)
{
try {
ThreadTest.class.wait();}
catch(InterruptedException e) {
e.printStackTrace();
}
}
```

```
for(intj=0;j<5;j++)
{
System.out.println(
Thread.currentThread().getName()+
"i=" + i +",j=" + j);
}
bShouldMain =false;
ThreadTest.class.notify();
}
}
}
}
```

下面使用 jdk5 中的并发库来实现的：

```
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Condition;
public class ThreadTest
{
private static Locklock = new ReentrantLock();
private staticCondition subThreadCondition = lock.newCondition();
private staticboolean bBhouldSubThread = false;
public static voidmain(String [] args)
{
ExecutorServicethreadPool = Executors.newFixedThreadPool(3);
threadPool.execute(newRunnable(){
publicvoid run()
{
for(inti=0;i<50;i++)
{
lock.lock();
try
{
if(!bBhouldSubThread)
subThreadCondition.await();
for(intj=0;j<10;j++)
{
System.out.println(Thread.currentThread().getName()+ ",j=" + j);
}
```

```
bBhouldSubThread= false;
subThreadCondition.signal();
}catch(Exceptione)
{
}
finally
{
lock.unlock();
}
}
}
});
threadPool.shutdown();
for(inti=0;i<50;i++)
{
lock.lock();
try
{
if(bBhouldSubThread)
subThreadCondition.await();
for(intj=0;j<10;j++)
{
System.out.println(Thread.currentThread().getName()+ ",j=" + j);
}
bBhouldSubThread= true;
subThreadCondition.signal();
}catch(Exceptione)
{
}
finally
{
lock.unlock();
}
}
}
}
```

57、介绍 Collection 框架的结构

答：随意发挥题，天南海北谁便谈，只要让别觉得你知识渊博，理解透彻即可。

58、Collection 框架中实现比较要实现什么接口

comparable/comparator

59、ArrayList 和 Vector 的区别

答：

这两个类都实现了 List 接口（List 接口继承了 Collection 接口），他们都是有序集合，即存储在这两个集合中的元素的位置都是有顺序的，相当于一种动态的数组，我们以后可以按位置索引号取出某个元素，，并且其中的数据是允许重复的，这是 HashSet 之类的集合的最大不同处，HashSet 之类的集合不可以按索引号去检索其中的元素，也不允许有重复的元素（本来题目问的与 hashset 没有任何关系，但为了说清楚 ArrayList 与 Vector 的功能，我们使用对比方式，更有利于说明问题）。

接着才说 ArrayList 与 Vector 的区别，这主要包括两个方面：.

（1）同步性：

Vector 是线程安全的，也就是说是它的方法之间是线程同步的，而 ArrayList 是线程序不安全的，它的方法之间是线程不同步的。如果只有一个线程会访问到集合，那最好是使用 ArrayList，因为它不考虑线程安全，效率会高些；如果有多个线程会访问到集合，那最好是使用 Vector，因为不需要我们自己再去考虑和编写线程安全的代码。

备注：对于 Vector&ArrayList、Hashtable&HashMap，要记住线程安全的问题，记住 Vector 与 Hashtable 是旧的，是 java 一诞生就提供了的，它们是线程安全的，ArrayList 与 HashMap 是 java2 时才提供的，它们是线程不安全的。所以，我们讲课时先讲老的。

（2）数据增长：

ArrayList 与 Vector 都有一个初始的容量大小，当存储进它们里面的元素的个数超过了容量时，就需要增加 ArrayList 与 Vector 的存储空间，每次要增加存储空间时，不是只增加一个存储单元，而是增加多个存储单元，每次增加的存储单元的个数在内存空间利用与程序效率之间要取得一定的平衡。Vector 默认增长为原来两倍，而 ArrayList 的增长策略在文档中没有明确规定（从源代码看到的是增长为原来的 1.5 倍）。ArrayList 与 Vector 都可以设置初始的空间大小，Vector 还可以设置增长的空间大小，而 ArrayList 没有提供设置增长空间的方法。

总结：即 Vector 增长原来的一倍，ArrayList 增加原来的 0.5 倍。

60、HashMap 和 Hashtable 的区别

（条理上还需要整理，也是先说相同点，再说不同点）

HashMap 是 Hashtable 的轻量级实现（非线程安全的实现），他们都完成了 Map 接口，主要区别在于 HashMap 允许空（null）键值（key）,由于非线程安全，在只有一个线程访问的情况下，效率要高于 Hashtable。

HashMap 允许将 null 作为一个 entry 的 key 或者 value，而 Hashtable 不允许。

HashMap 把 Hashtable 的 contains 方法去掉了，改成 containsvalue 和 containsKey。因为 contains 方法容易让人引起误解。

Hashtable 继承自 Dictionary 类，而 HashMap 是 Java1.2 引进的 Map interface 的一个实现。

最大的不同是，Hashtable 的方法是 Synchronize 的，而 HashMap 不是，在多个线程访问 Hashtable 时，不需要自己为它的方法实现同步，而 HashMap 就必须为之提供外同步。

Hashtable 和 HashMap 采用的 hash/rehash 算法都大概一样，所以性能不会有很大的差异。

就 HashMap 与 HashTable 主要从三方面来说。

一.历史原因:Hashtable 是基于陈旧的 Dictionary 类的，HashMap 是 Java 1.2 引进的 Map 接口的一个实现

二.同步性:Hashtable 是线程安全的，也就是说是同步的，而 HashMap 是线程序不安全的，不是同步的

三.值：只有 HashMap 可以让你将空值作为一个表的条目的 key 或 value

61、List 和 Map 区别?

一个是存储单列数据的集合，另一个是存储键和值这样的双列数据的集合，List 中存储的数据是有顺序，并且允许重复；Map 中存储的数据是没有顺序的，其键是不能重复的，它的值是可以有重复的。

62、List, Set, Map 是否继承自 Collection 接口?

List，Set 是，Map 不是

63、List、Map、Set 三个接口，存取元素时，各有什么特点？

这样的题属于随意发挥题：这样的题比较考水平，两个方面的水平：一是要真正明白这些内容，二是要有较强的总结和表述能力。如果你明白，但表述不清楚，在别人那里则等同于不明白。

首先，List 与 Set 具有相似性，它们都是单列元素的集合，所以，它们有一个功共同的父接口，叫 Collection。Set 里面不允许有重复的元素，所谓重复，即不能有两个相等（注意，不是仅仅是相同）的对象，即假设 Set 集合中有了一个 A 对象，现在我要向 Set 集合再存入一个 B 对象，但 B 对象与 A 对象 equals 相等，则 B 对象存储不进去，所以，Set 集合的 add 方法有一个 boolean 的返回值，当集合中没有某个元素，此时 add 方法可成功加入该元素时，则返回 true，当集合含有与某个元素 equals 相等的元素时，此时 add 方法无法加入该元素，返回结果为 false。Set 取元素时，没法说取第几个，只能以 Iterator 接口取得所有的元素，再逐一遍历各个元素。

List 表示有先后顺序的集合，注意，不是那种按年龄、按大小、按价格之类的排序。当我们多次调用 add(Obj e)方法时，每次加入的对象就像火车站买票有排队顺序一样，按先来后到的顺序排序。有时候，也可以插队，即调用 add(int index,Obj e)方法，就可以指定当前对象在集合中的存放位置。一个对象可以被反复存储进 List 中，每调用一次 add 方法，这个对象就被插入进集合中一次，其实，并不是把这个对象本身存储进了集合中，而是在集合中用一个索引变量指向这个对象，当这个对象被 add 多次时，即相当于集合中有多个索引指向了这个对象，如图 x 所示。List 除了可以以 Iterator 接口取得所有的元素，再逐一遍历各个元素之外，还可以调用 get(index i)来明确说明取第几个。

Map 与 List 和 Set 不同，它是双列的集合，其中有 put 方法，定义如下：put(obj key,objvalue)，每次存储时，要存储一对 key/value，不能存储重复的 key，这个重复的规则也是按 equals 比较相等。取则可以根据 key 获得相应的 value，即 get(Object key)返回值为 key 所对应的 value。另外，也可以获得所有的 key 的结合，还可以获得所有的 value 的结合，还可以获得 key 和 value 组合成的 Map.Entry 对象的集合。

List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存 key-value 值，value 可多值。

HashSet 按照 hashcode 值的某种运算方式进行存储，而不是直接按 hashCode 值的大小进行存储。例如，"abc"---> 78，"def" ---> 62，"xyz" ---> 65 在 hashSet 中的存储顺序不是 62,65,78，这些问题感谢以前一个叫崔健的学员提出，最后通过查看源代码给他解释清楚，看本次培训学员当中有多少能看懂源码。LinkedHashSet 按插入的顺序存储，那被存储对象的 hashcode 方法还有什么作用呢？学员想想!hashset 集合比较两个对象是否相等，首先看 hashcode 方法是否相等，然后看 equals 方法是否相等。new 两个 Student 插入到 HashSet 中，看 HashSet 的 size，实现 hashcode 和 equals 方法后再看 size。

同一个对象可以在 Vector 中加入多次。往集合里面加元素，相当于集合里用一根绳子连接到了目标对象。往 HashSet 中却加不了多次的。

64、说出 ArrayList,Vector, LinkedList 的存储性能和特性

ArrayList 和 Vector 都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector 由于使用了 synchronized 方法（线程安全），通常性能上较 ArrayList 差，而 LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

LinkedList 也是线程不安全的，LinkedList 提供了一些方法，使得 LinkedList 可以被当作堆栈和队列来使用。

65、去掉一个 Vector 集合中重复的元素

```
Vector newVector = new Vector();
For (int i=0;i<vector.size();i++)
{
Object obj = vector.get(i);
if(!newVector.contains(obj);
newVector.add(obj);
```

}

还有一种简单的方式，HashSet set = new HashSet(vector);

66、Collection 和 Collections 的区别。

Collection 是集合类的上级接口，继承与他的接口主要有 Set 和 List.

Collections 是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

67、Set 里的元素是不能重复的，那么用什么方法来区分重复与否呢?是用==还是 equals()?它们有何区别?

Set 里的元素是不能重复的，元素重复与否是使用 equals()方法进行判断的。

equals()和==方法决定引用值是否指向同一对象 equals()在类中被覆盖，为的是当两个分离的对象的内容和类型相配的话，返回真值。

68、你所知道的集合类都有哪些？主要方法？

最常用的集合类是 List 和 Map。 List 的具体实现包括 ArrayList 和 Vector，它们是可变大小的列表，比较适合构建、存储和操作任何类型对象的元素列表。 List 适用于按数值索引访问元素的情形。

Map 提供了一个更通用的元素存储方法。 Map 集合类用于存储元素对（称作"键"和"值"），其中每个键映射到一个值。

ArrayList/VectoràList

àCollection

HashSet/TreeSetàSet

PropetiesàHashTable

àMap

Treemap/HashMap

我记的不是方法名，而是思想，我知道它们都有增删改查的方法，但这些方法的具体名称，我记得不是很清楚，对于 set，大概的方法是 add,remove, contains；对于 map，大概的方法就是 put,remove，contains 等，因为，我只要在 eclispe 下按点操作符，很自然的这些方法就出来了。我记住的一些思想就是 List 类会有 get(int index)这样的方法，因为它可以按顺序取元素，而 set 类中没有 get(int index)这样的方法。List 和 set 都可以迭代出所有元素，迭代时先要得到一个 iterator 对象，所以，set 和 list 类都有一个 iterator 方法，用于返回那个 iterator 对象。map 可以返回三个集合，一个是返回所有的 key 的集合，另外一个返回的是所有 value 的集合，再一个返回的 key 和 value 组合成的 EntrySet 对象的集合，map 也有 get 方法，参数是 key，返回值是 key 对应的 value。

69、两个对象值相同(x.equals(y) == true)，但却可有不同的 hash code，这句话对不对?

对。

如果对象要保存在 HashSet 或 HashMap 中，它们的 equals 相等，那么，它们的 hashcode 值就必须相等。

如果不是要保存在 HashSet 或 HashMap，则与 hashcode 没有什么关系了，这时候 hashcode 不等是可以的，例如 arrayList 存储的对象就不用实现 hashcode，当然，我们没有理由不实现，通常都会去实现的。

70、TreeSet 里面放对象，如果同时放入了父类和子类的实例对象，那比较时使用的是父类的 compareTo 方法，还是使用的子类的 compareTo 方法，还是抛异常！

（应该是没有针对问题的确切的答案，当前的 add 方法放入的是哪个对象，就调用哪个对象的 compareTo 方法，至于这个 compareTo 方法怎么做，就看当前这个对象的类中是如何编写这个方法的）

实验代码：

```
publicclass ParentimplementsComparable {
private int age = 0;
public Parent(int age){
this.age = age;
```

```
}
public int compareTo(Object o){
// TODO Auto-generated method stub
System.out.println("method ofparent");
Parent o1 = (Parent)o;
returnage>o1.age?1:age<o1.age?-1:0;
}
}
publicclass Childextends Parent {
public Child(){
super(3);
}
public int compareTo(Object o){
// TODO Auto-generated methodstub
System.out.println("methodof child");
// Child o1 = (Child)o;
return 1;
}
}
publicclass TreeSetTest {
/**
* @paramargs
*/
public static voidmain(String[] args) {
// TODO Auto-generated method stub
TreeSet set = new TreeSet();
set.add(newParent(3));
set.add(new Child());
set.add(newParent(4));
System.out.println(set.size());
}
}
```

面向对象编程（OOP）

Java 是一个支持并发、基于类和面向对象的计算机编程语言。下面列出了面向对象软件开发的优点：

- 代码开发模块化，更易维护和修改。
- 代码复用。
- 增强代码的可靠性和灵活性。
- 增加代码的可理解性。

面向对象编程有很多重要的特性，比如：封装，继承，多态和抽象。下面的章节我们会逐个分析这些特性。

封装

封装给对象提供了隐藏内部特性和行为的能力。对象提供一些能被其他对象访问的方法来改变它内部的数据。在 Java 当中，有 3 种修饰符：public，private 和 protected。每一种修饰符给其他的位于同一个包或者不同包下面对象赋予了不同的访问权限。

下面列出了使用封装的一些好处：

- 通过隐藏对象的属性来保护对象内部的状态。
- 提高了代码的可用性和可维护性，因为对象的行为可以被单独的改变或者是扩展。
- 禁止对象之间的不良交互提高模块化。

参考这个文档获取更多关于封装的细节和示例。
多态

多态是编程语言给不同的底层数据类型做相同的接口展示的一种能力。一个多态类型上的操作可以应用到其他类型的值上面。

继承

继承给对象提供了从基类获取字段和方法的能力。继承提供了代码的重用行，也可以在不修改类的情况下给现存的类添加新特性。

抽象

抽象是把想法从具体的实例中分离出来的步骤，因此，要根据他们的功能而不是实现细节来创建类。Java 支持创建只暴漏接口而不包含方法实现的抽象的类。这种抽象技术的主要目的是把类的行为和实现细节分离开。

抽象和封装的不同点

抽象和封装是互补的概念。一方面，抽象关注对象的行为。另一方面，封装关注对象行为的细节。一般是通过隐藏对象内部状态信息做到封装，因此，封装可以看成是用来提供抽象的一种策略。

常见的 Java 问题
1.什么是 Java 虚拟机？为什么 Java 被称作是“平台无关的编程语言”？

Java 虚拟机是一个可以执行 Java 字节码的虚拟机进程。Java 源文件被编译成能被 Java 虚拟机执行的字节码文件。

Java 被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。Java 虚拟机让这个变为可能，因为它知道底层硬件平台的指令长度和其他特性。

2.JDK 和 JRE 的区别是什么？

Java 运行时环境(JRE)是将要执行 Java 程序的 Java 虚拟机。它同时也包含了执行 applet 需要的浏览器插件。Java 开发工具包(JDK)是完整的 Java 软件开发包，包含了 JRE，编译器和其他的工具(比如：JavaDoc，Java 调试器)，可以让开发者开发、编译、执行 Java 应用程序。

3.”static”关键字是什么意思？Java 中是否可以覆盖(override)一个 private 或者是 static 的方法？

“static”关键字表明一个成员变量或者是成员方法可以在没有所属的类的实例变量的情况下被访问。
Java 中 static 方法不能被覆盖，因为方法覆盖是基于运行时动态绑定的，而 static 方法是编译时静态绑定的。static 方法跟类的任何实例都不相关，所以概念上不适用。

4.是否可以在 static 环境中访问非 static 变量？

static 变量在 Java 中是属于类的，它在所有的实例中的值是一样的。当类被 Java 虚拟机载入的时候，会对 static 变量进行初始化。如果你的代码尝试不用实例来访问非 static 的变量，编译器会报错，因为这些变量还没有被创建出来，还没有跟任何实例关联上。

5.Java 支持的数据类型有哪些？什么是自动拆装箱？

Java 语言支持的 8 中基本数据类型是：

- byte
- short
- int
- long
- float
- double
- boolean
- char

自动装箱是 Java 编译器在基本数据类型和对应的对象包装类型之间做的一个转化。比如：把 int 转化成 Integer，double 转化成 double，等等。反之就是自动拆箱。

6.Java 中的方法覆盖(Overriding)和方法重载(Overloading)是什么意思？

Java 中的方法重载发生在同一个类里面两个或者是多个方法的方法名相同但是参数不同的情况。与此相对，方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和返回类型。覆盖者可能不会限制它所覆盖的方法的访问。

7.Java 中，什么是构造函数？什么是构造函数重载？什么是复制构造函数？

当新对象被创建的时候，构造函数会被调用。每一个类都有构造函数。在程序员没有给类提供构造函数的情况下，Java 编译器会为这个类创建一个默认的构造函数。

Java 中构造函数重载和方法重载很相似。可以为一个类创建多个构造函数。每一个构造函数必须有它自己唯一的参数列表。

Java 不支持像 C++中那样的复制构造函数，这个不同点是因为如果你不自己写构造函数的情况下，Java 不会创建默认的复制构造函数。

8.Java 支持多继承么？

不支持，Java 不支持多继承。每个类都只能继承一个类，但是可以实现多个接口。

9.接口和抽象类的区别是什么？

Java 提供和支持创建抽象类和接口。它们的实现有共同点，不同点在于：

- 接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。
- 类可以实现很多个接口，但是只能继承一个抽象类
- 类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。
- 抽象类可以在不提供接口方法实现的情况下实现接口。
- Java 接口中声明的变量默认都是 final 的。抽象类可以包含非 final 的变量。
- Java 接口中的成员函数默认是 public 的。抽象类的成员函数可以是 private，protected 或者是 public。
- 接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含 main 方法的话是可以被调用的。
  也可以参考 JDK8 中抽象类和接口的区别

10.什么是值传递和引用传递？

对象被值传递，意味着传递了对象的一个副本。因此，就算是改变了对象副本，也不会影响源对象的值。

对象被引用传递，意味着传递的并不是实际的对象，而是对象的引用。因此，外部对引用对象所做的改变会反映到所有的对象上。

Java 线程

11.进程和线程的区别是什么？

进程是执行着的应用程序，而线程是进程内部的一个执行序列。一个进程可以有多个线程。线程又叫做轻量级进程。

12.创建线程有几种不同的方式？你喜欢哪一种？为什么？

有三种方式可以用来创建线程：

- 继承 Thread 类
- 实现 Runnable 接口
- 应用程序可以使用 Executor 框架来创建线程池

实现 Runnable 接口这种方式更受欢迎，因为这不需要继承 Thread 类。在应用设计中已经继承了别的对象的情况下，这需要多继承（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。

13.概括的解释下线程的几种可用状态。

线程在执行过程中，可以处于下面几种状态：

- 就绪(Runnable):线程准备运行，不一定立马就能开始执行。
- 运行中(Running)：进程正在执行线程的代码。
- 等待中(Waiting):线程处于阻塞的状态，等待外部的处理结束。
- 睡眠中(Sleeping)：线程被强制睡眠。
- I/O 阻塞(Blocked on I/O)：等待 I/O 操作完成。
- 同步阻塞(Blocked on Synchronization)：等待获取锁。
- 死亡(Dead)：线程完成了执行。

14.同步方法和同步代码块的区别是什么？

在 Java 语言中，每一个对象有一把锁。线程可以使用 synchronized 关键字来获取对象上的锁。synchronized 关键字可应用在方法级别(粗粒度锁)或者是代码块级别(细粒度锁)。

15.在监视器(Monitor)内部，是如何做线程同步的？程序应该做哪种级别的同步？

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。

16.什么是死锁(deadlock)？

两个进程都在等待对方执行完毕才能继续往下执行的时候就发生了死锁。结果就是两个进程都陷入了无限的等待中。

17.如何确保 N 个线程可以访问 N 个资源同时又不导致死锁？

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。

Java 集合类

18.Java 集合类框架的基本接口有哪些？

Java 集合类提供了一套设计良好的支持对一组对象进行操作的接口和类。Java 集合类里面最基本的接口有：

- Collection：代表一组对象，每一个对象都是它的子元素。
- Set：不包含重复元素的 Collection。
- List：有顺序的 collection，并且可以包含重复元素。
- Map：可以把键(key)映射到值(value)的对象，键不能重复。

19.为什么集合类没有实现 Cloneable 和 Serializable 接口？

集合类接口指定了一组叫做元素的对象。集合类接口的每一种具体的实现类都可以选择以它自己的方式对元素进行保存和排序。有的集合类允许重复的键，有些不允许。

20.什么是迭代器(Iterator)？

Iterator 接口提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代器实例的迭代方法。迭代器可以在迭代的过程中删除底层集合的元素。

克隆(cloning)或者是序列化(serialization)的语义和含义是跟具体的实现相关的。因此，应该由集合类的具体实现来决定如何被克隆或者是序列化。

21.Iterator 和 ListIterator 的区别是什么？

下面列出了他们的区别：

- 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。

22.快速失败(fail-fast)和安全失败(fail-safe)的区别是什么？

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。

23.Java 中的 HashMap 的工作原理是什么？

Java 中的 HashMap 是以键值对(key-value)的形式存储元素的。HashMap 需要一个 hash 函数，它使用 hashCode()和 equals()方法来向集合/从集合添加和检索元素。当调用 put()方法的时候，HashMap 会计算 key 的 hash 值，然后把键值对存储在集合中合适的索引上。如果 key 已经存在了，value 会被更新成新值。HashMap 的一些重要的特性是它的容量(capacity)，负载因子(load factor)和扩容极限(threshold resizing)。

24.hashCode()和 equals()方法的重要性体现在什么地方？

Java 中的 HashMap 使用 hashCode()和 equals()方法来确定键值对的索引，当根据键获取值的时候也会用到这两个方法。如果没有正确的实现这两个方法，两个不同的键可能会有相同的 hash 值，因此，可能会被集合认为是相等的。而且，这两个方法也用来发现重复元素。所以这两个方法的实现对 HashMap 的精确性和正确性是至关重要的。

25.HashMap 和 Hashtable 有什么区别？

- HashMap 和 Hashtable 都实现了 Map 接口，因此很多特性非常相似。但是，他们有以下不同点：
- HashMap 允许键和值是 null，而 Hashtable 不允许键或者值是 null。
- Hashtable 是同步的，而 HashMap 不是。因此，HashMap 更适合于单线程环境，而 Hashtable 适合于多线程环境。
- HashMap 提供了可供应用迭代的键的集合，因此，HashMap 是快速失败的。另一方面，Hashtable 提供了对键的列举(Enumeration)。
    - 一般认为 Hashtable 是一个遗留的类。

26.数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用 Array 而不是 ArrayList？

下面列出了 Array 和 ArrayList 的不同点：

- Array 可以包含基本类型和对象类型，ArrayList 只能包含对象类型。
- Array 大小是固定的，ArrayList 的大小是动态变化的。
- ArrayList 提供了更多的方法和特性，比如：addAll()，removeAll()，iterator()等等。
- 对于基本类型数据，集合使用自动装箱来减少编码工作量。但是，当处理固定大小的基本数据类型的时候，这种方式相对比较慢。

27.ArrayList 和 LinkedList 有什么区别？

ArrayList 和 LinkedList 都实现了 List 接口，他们有以下的不同点：

- ArrayList 是基于索引的数据接口，它的底层是数组。它可以以 O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList 是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是 O(n)。

- 相对于 ArrayList，LinkedList 的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。

- LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

也可以参考 ArrayList vs. LinkedList。

28.Comparable 和 Comparator 接口是干什么的？列出它们的区别。

Java 提供了只包含一个 compareTo()方法的 Comparable 接口。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。

Java 提供了包含 compare()和 equals()两个方法的 Comparator 接口。compare()方法用来给两个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。equals()方法需要一个对象作为参数，它用来决定输入参数是否和 comparator 相等。只有当输入参数也是一个 comparator 并且输入参数和当前 comparator 的排序结果是相同的时候，这个方法才返回 true。

29.什么是 Java 优先级队列(Priority Queue)？

PriorityQueue 是一个基于优先级堆的无界队列，它的元素是按照自然顺序(natural order)排序的。在创建的时候，我们可以给它提供一个负责给元素排序的比较器。PriorityQueue 不允许 null 值，因为他们没有自然顺序，或者说他们没有任何的相关联的比较器。最后，PriorityQueue 不是线程安全的，入队和出队的时间复杂度是 O(log(n))。

30.你了解大 O 符号(big-O notation)么？你能给出不同数据结构的例子么？

大 O 符号描述了当数据结构里面的元素增加的时候，算法的规模或者是性能在最坏的场景下有多么好。
大 O 符号也可用来描述其他的行为，比如：内存消耗。因为集合类实际上是数据结构，我们一般使用大 O 符号基于时间，内存和性能来选择最好的实现。大 O 符号可以对大量数据的性能给出一个很好的说明。

31.如何权衡是使用无序的数组还是有序的数组？

有序数组最大的好处在于查找的时间复杂度是 O(log n)，而无序数组是 O(n)。有序数组的缺点是插入操作的时间复杂度是 O(n)，因为值大的元素需要往后移动来给新元素腾位置。相反，无序数组的插入时间复杂度是常量 O(1)。

32.Java 集合类框架的最佳实践有哪些？

- 根据应用的需要正确选择要使用的集合的类型对性能非常重要，比如：假如元素的大小是固定的，而且能事先知道，我们就应该用 Array 而不是 ArrayList。
- 有些集合类允许指定初始容量。因此，如果我们能估计出存储的元素的数目，我们可以设置初始容量来避免重新计算 hash 值或者是扩容。
- 为了类型安全，可读性和健壮性的原因总是要使用泛型。同时，使用泛型还可以避免运行时的 ClassCastException。
- 使用 JDK 提供的不变类(immutable class)作为 Map 的键可以避免为我们自己的类实现 hashCode()和 equals()方法。
- 编程的时候接口优于实现。
- 底层的集合实际上是空的情况下，返回长度是 0 的集合或者是数组，不要返回 null。

33.Enumeration 接口和 Iterator 接口的区别有哪些？

Enumeration 速度是 Iterator 的 2 倍，同时占用更少的内存。但是，Iterator 远远比 Enumeration 安全，因为其他线程不能够修改正在被 iterator 遍历的集合里面的对象。同时，Iterator 允许调用者删除底层集合里面的元素，这对 Enumeration 来说是不可能的。

34.HashSet 和 TreeSet 有什么区别？

HashSet 是由一个 hash 表来实现的，因此，它的元素是无序的。add()，remove()，contains()方法的时间复杂度是 O(1)。

另一方面，TreeSet 是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()，remove()，contains()方法的时间复杂度是 O(logn)。

垃圾收集器(Garbage Collectors)

35.Java 中垃圾回收有什么目的？什么时候进行垃圾回收？

垃圾回收的目的是识别并且丢弃应用不再使用的对象来释放和重用资源。

36.System.gc()和 Runtime.gc()会做什么事情？

这两个方法用来提示 JVM 要进行垃圾回收。但是，立即开始还是延迟进行垃圾回收是取决于 JVM 的。

37.finalize()方法什么时候被调用？析构函数(finalization)的目的是什么？

在释放对象占用的内存之前，垃圾收集器会调用对象的 finalize()方法。一般建议在该方法中释放对象持有的资源。

38.如果对象的引用被置为 null，垃圾收集器是否会立即释放对象占用的内存？

不会，在下一个垃圾回收周期中，这个对象将是可被回收的。

39.Java 堆的结构是什么样子的？什么是堆中的永久代(Perm Gen space)?

JVM 的堆是运行时数据区，所有类的实例和数组都是在堆上分配内存。它在 JVM 启动的时候被创建。对象所占的堆内存是由自动内存管理系统也就是垃圾收集器回收。

堆内存是由存活和死亡的对象组成的。存活的对象是应用可以访问的，不会被垃圾回收。死亡的对象是应用不可访问尚且还没有被垃圾收集器回收掉的对象。一直到垃圾收集器把这些对象回收掉之前，他们会一直占据堆内存空间。

40.串行(serial)收集器和吞吐量(throughput)收集器的区别是什么？

吞吐量收集器使用并行版本的新生代垃圾收集器，它用于中等规模和大规模数据的应用程序。而串行收集器对大多数的小应用(在现代处理器上需要大概 100M 左右的内存)就足够了。

41.在 Java 中，对象什么时候可以被垃圾回收？

当对象对当前使用这个对象的应用程序变得不可触及的时候，这个对象就可以被回收了。

42.JVM 的永久代中会发生垃圾回收么？

垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。如果你仔细查看垃圾收集器的输出信息，就会发现永久代也是被回收的。这就是为什么正确的永久代大小对避免 Full GC 是非常重要的原因。请参考下 Java8：从永久代到元数据区
(译者注：Java8 中已经移除了永久代，新加了一个叫做元数据区的 native 内存区)
43.Java 中的两种异常类型是什么？他们有什么区别？

Java 中有两种异常：受检查的(checked)异常和不受检查的(unchecked)异常。不受检查的异常不需要在方法或者是构造函数上声明，就算方法或者是构造函数的执行可能会抛出这样的异常，并且不受检查的异常可以传播到方法或者是构造函数的外面。相反，受检查的异常必须要用 throws 语句在方法或者是构造函数上声明。这里有 Java 异常处理的一些小建议。
44.Java 中 Exception 和 Error 有什么区别？

Exception 和 Error 都是 Throwable 的子类。Exception 用于用户程序可以捕获的异常情况。Error 定义了不期望被用户程序捕获的异常。

45.throw 和 throws 有什么区别？

throw 关键字用来在程序中明确的抛出异常，相反，throws 语句用来表明方法不能处理的异常。每一个方法都必须要指定哪些异常不能处理，所以方法的调用者才能够确保处理可能发生的异常，多个异常是用逗号分隔的。

45.异常处理的时候，finally 代码块的重要性是什么？(译者注：作者标题的序号弄错了)

无论是否抛出异常，finally 代码块总是会被执行。就算是没有 catch 语句同时又抛出异常的情况下，finally 代码块仍然会被执行。最后要说的是，finally 代码块主要用来释放资源，比如：I/O 缓冲区，数据库连接。

46.异常处理完成以后，Exception 对象会发生什么变化？

Exception 对象会在下一个垃圾回收过程中被回收掉。

47.finally 代码块和 finalize()方法有什么区别？

无论是否抛出异常，finally 代码块都会执行，它主要是用来释放应用占用的资源。finalize()方法是 Object 类的一个 protected 方法，它是在对象被垃圾回收之前由 Java 虚拟机来调用的。

Java 小应用程序(Applet)
48.什么是 Applet？

java applet 是能够被包含在 HTML 页面中并且能被启用了 java 的客户端浏览器执行的程序。Applet 主要用来创建动态交互的 web 应用程序。

49.解释一下 Applet 的生命周期

applet 可以经历下面的状态：

- Init：每次被载入的时候都会被初始化。
- Start：开始执行 applet。
- Stop：结束执行 applet。
- Destroy：卸载 applet 之前，做最后的清理工作。

50.当 applet 被载入的时候会发生什么？

首先，创建 applet 控制类的实例，然后初始化 applet，最后开始运行。

51.Applet 和普通的 Java 应用程序有什么区别？

applet 是运行在启用了 java 的浏览器中，Java 应用程序是可以在浏览器之外运行的独立的 Java 程序。但是，它们都需要有 Java 虚拟机。

进一步来说，Java 应用程序需要一个有特定方法签名的 main 函数来开始执行。Java applet 不需要这样的函数来开始执行。

最后，Java applet 一般会使用很严格的安全策略，Java 应用一般使用比较宽松的安全策略。

52.Java applet 有哪些限制条件？

主要是由于安全的原因，给 applet 施加了以下的限制：

- applet 不能够载入类库或者定义本地方法。
- applet 不能在宿主机上读写文件。
- applet 不能读取特定的系统属性。
- applet 不能发起网络连接，除非是跟宿主机。
- applet 不能够开启宿主机上其他任何的程序。

53.什么是不受信任的 applet？

不受信任的 applet 是不能访问或是执行本地系统文件的 Java applet，默认情况下，所有下载的 applet 都是不受信任的。

54.从网络上加载的 applet 和从本地文件系统加载的 applet 有什么区别？

当 applet 是从网络上加载的时候，applet 是由 applet 类加载器载入的，它受 applet 安全管理器的限制。

当 applet 是从客户端的本地磁盘载入的时候，applet 是由文件系统加载器载入的。

从文件系统载入的 applet 允许在客户端读文件，写文件，加载类库，并且也允许执行其他程序，但是，却通不过字节码校验。

55.applet 类加载器是什么？它会做哪些工作？

当 applet 是从网络上加载的时候，它是由 applet 类加载器载入的。类加载器有自己的 java 名称空间等级结构。类加载器会保证来自文件系统的类有唯一的名称空间，来自网络资源的类有唯一的名称空间。

当浏览器通过网络载入 applet 的时候，applet 的类被放置于和 applet 的源相关联的私有的名称空间中。然后，那些被类加载器载入进来的类都是通过了验证器验证的。验证器会检查类文件格式是否遵守 Java 语言规范，确保不会出现堆栈溢出(stack overflow)或者下溢(underflow)，传递给字节码指令的参数是正确的。

56.applet 安全管理器是什么？它会做哪些工作？

applet 安全管理器是给 applet 施加限制条件的一种机制。浏览器可以只有一个安全管理器。安全管理器在启动的时候被创建，之后不能被替换覆盖或者是扩展。

Swing
57.弹出式选择菜单(Choice)和列表(List)有什么区别

Choice 是以一种紧凑的形式展示的，需要下拉才能看到所有的选项。Choice 中一次只能选中一个选项。List 同时可以有多个元素可见，支持选中一个或者多个元素。

58.什么是布局管理器？

布局管理器用来在容器中组织组件。

59.滚动条(Scrollbar)和滚动面板(JScrollPane)有什么区别？

Scrollbar 是一个组件，不是容器。而 ScrollPane 是容器。ScrollPane 自己处理滚动事件。

60.哪些 Swing 的方法是线程安全的？

只有 3 个线程安全的方法： repaint(), revalidate(), and invalidate()。

61.说出三种支持重绘(painting)的组件。

Canvas, Frame, Panel,和 Applet 支持重绘。

62.什么是裁剪(clipping)？

限制在一个给定的区域或者形状的绘图操作就做裁剪。

63.MenuItem 和 CheckboxMenuItem 的区别是什么？

CheckboxMenuItem 类继承自 MenuItem 类，支持菜单选项可以选中或者不选中。

64.边缘布局(BorderLayout)里面的元素是如何布局的？

BorderLayout 里面的元素是按照容器的东西南北中进行布局的。

65.网格包布局(GridBagLayout)里面的元素是如何布局的？

GridBagLayout 里面的元素是按照网格进行布局的。不同大小的元素可能会占据网格的多于 1 行或一列。因此，行数和列数可以有不同的大小。

66.Window 和 Frame 有什么区别？

Frame 类继承了 Window 类，它定义了一个可以有菜单栏的主应用窗口。

67.裁剪(clipping)和重绘(repainting)有什么联系？

当窗口被 AWT 重绘线程进行重绘的时候，它会把裁剪区域设置成需要重绘的窗口的区域。

68.事件监听器接口(event-listener interface)和事件适配器(event-adapter)有什么关系？

事件监听器接口定义了对特定的事件，事件处理器必须要实现的方法。事件适配器给事件监听器接口提供了默认的实现。

69.GUI 组件如何来处理它自己的事件？

GUI 组件可以处理它自己的事件，只要它实现相对应的事件监听器接口，并且把自己作为事件监听器。

70.Java 的布局管理器比传统的窗口系统有哪些优势？

Java 使用布局管理器以一种一致的方式在所有的窗口平台上摆放组件。因为布局管理器不会和组件的绝对大小和位置相绑定，所以他们能够适应跨窗口系统的特定平台的不同。

71.Java 的 Swing 组件使用了哪种设计模式？
Java 中的 Swing 组件使用了 MVC(视图-模型-控制器)设计模式。
JDBC
72.什么是 JDBC？

JDBC 是允许用户在不同数据库之间做选择的一个抽象层。JDBC 允许开发者用 JAVA 写数据库应用程序，而不需要关心底层特定数据库的细节。
73.解释下驱动(Driver)在 JDBC 中的角色。

JDBC 驱动提供了特定厂商对 JDBC API 接口类的实现，驱动必须要提供 java.sql 包下面这些类的实现：
Connection, Statement, PreparedStatement,CallableStatement, ResultSet 和 Driver。
74.Class.forName()方法有什么作用？

这个方法用来载入跟数据库建立连接的驱动。

75.PreparedStatement 比 Statement 有什么优势？

PreparedStatements 是预编译的，因此，性能会更好。同时，不同的查询参数值，PreparedStatement 可以重用。

76.什么时候使用 CallableStatement？用来准备 CallableStatement 的方法是什么？

CallableStatement 用来执行存储过程。存储过程是由数据库存储和提供的。存储过程可以接受输入参数，也可以有返回结果。非常鼓励使用存储过程，因为它提供了安全性和模块化。准备一个 CallableStatement 的方法是：

```
1    CallableStament.prepareCall();
```

77.数据库连接池是什么意思？

像打开关闭数据库连接这种和数据库的交互可能是很费时的，尤其是当客户端数量增加的时候，会消耗大量的资源，成本是非常高的。可以在应用服务器启动的时候建立很多个数据库连接并维护在一个池中。连接请求由池中的连接提供。在连接使用完毕以后，把连接归还到池中，以用于满足将来更多的请求。

远程方法调用(RMI)
78.什么是 RMI？

Java 远程方法调用(Java RMI)是 Java API 对远程过程调用(RPC)提供的面向对象的等价形式，支持直接传输序列化的 Java 对象和分布式垃圾回收。远程方法调用可以看做是激活远程正在运行的对象上的方法的步骤。RMI 对调用者是位置透明的，因为调用者感觉方法是执行在本地运行的对象上的。看下 RMI 的一些注意事项。

79.RMI 体系结构的基本原则是什么？

RMI 体系结构是基于一个非常重要的行为定义和行为实现相分离的原则。RMI 允许定义行为的代码和实现行为的代码相分离，并且运行在不同的 JVM 上。

80.RMI 体系结构分哪几层？

RMI 体系结构分以下几层：

存根和骨架层(Stub and Skeleton layer)：这一层对程序员是透明的，它主要负责拦截客户端发出的方法调用请求，然后把请求重定向给远程的 RMI 服务。

远程引用层(Remote Reference Layer)：RMI 体系结构的第二层用来解析客户端对服务端远程对象的引用。这一层解析并管理客户端对服务端远程对象的引用。连接是点到点的。

传输层(Transport layer)：这一层负责连接参与服务的两个 JVM。这一层是建立在网络上机器间的 TCP/IP 连接之上的。它提供了基本的连接服务，还有一些防火墙穿透策略。

81.RMI 中的远程接口(Remote Interface)扮演了什么样的角色？

远程接口用来标识哪些方法是可以被非本地虚拟机调用的接口。远程对象必须要直接或者是间接实现远程接口。实现了远程接口的类应该声明被实现的远程接口，给每一个远程对象定义构造函数，给所有远程接口的方法提供实现。

82.java.rmi.Naming 类扮演了什么样的角色？

java.rmi.Naming 类用来存储和获取在远程对象注册表里面的远程对象的引用。Naming 类的每一个方法接收一个 URL 格式的 String 对象作为它的参数。

83.RMI 的绑定(Binding)是什么意思？

绑定是为了查询找远程对象而给远程对象关联或者是注册以后会用到的名称的过程。远程对象可以使用 Naming 类的 bind()或者 rebind()方法跟名称相关联。

84.Naming 类的 bind()和 rebind()方法有什么区别？

bind()方法负责把指定名称绑定给远程对象，rebind()方法负责把指定名称重新绑定到一个新的远程对象。如果那个名称已经绑定过了，先前的绑定会被替换掉。

85.让 RMI 程序能正确运行有哪些步骤？

为了让 RMI 程序能正确运行必须要包含以下几个步骤：

- 编译所有的源文件。
- 使用 rmic 生成 stub。
- 启动 rmiregistry。
- 启动 RMI 服务器。
- 运行客户端程序。

86.RMI 的 stub 扮演了什么样的角色？

远程对象的 stub 扮演了远程对象的代表或者代理的角色。调用者在本地 stub 上调用方法，它负责在远程对象上执行方法。当 stub 的方法被调用的时候，会经历以下几个步骤：

- 初始化到包含了远程对象的 JVM 的连接。
- 序列化参数到远程的 JVM。
- 等待方法调用和执行的结果。
- 反序列化返回的值或者是方法没有执行成功情况下的异常。
- 把值返回给调用者。

87.什么是分布式垃圾回收(DGC)？它是如何工作的？

DGC 叫做分布式垃圾回收。RMI 使用 DGC 来做自动垃圾回收。因为 RMI 包含了跨虚拟机的远程对象的引用，垃圾回收是很困难的。DGC 使用引用计数算法来给远程对象提供自动内存管理。

88.RMI 中使用 RMI 安全管理器(RMISecurityManager)的目的是什么？

RMISecurityManager 使用下载好的代码提供可被 RMI 应用程序使用的安全管理器。如果没有设置安全管理器，RMI 的类加载器就不会从远程下载任何的类。

89.解释下 Marshalling 和 demarshalling。

当应用程序希望把内存对象跨网络传递到另一台主机或者是持久化到存储的时候，就必须要把对象在内存里面的表示转化成合适的格式。这个过程就叫做 Marshalling，反之就是 demarshalling。

90.解释下 Serialization 和 Deserialization。

Java 提供了一种叫做对象序列化的机制，他把对象表示成一连串的字节，里面包含了对象的数据，对象的类型信息，对象内部的数据的类型信息等等。因此，序列化可以看成是为了把对象存储在磁盘上或者是从磁盘上读出来并重建对象而把对象扁平化的一种方式。反序列化是把对象从扁平状态转化成活动对象的相反的步骤。

Servlet

91.什么是 Servlet？

Servlet 是用来处理客户端请求并产生动态网页内容的 Java 类。Servlet 主要是用来处理或者是存储 HTML 表单提交的数据，产生动态内容，在无状态的 HTTP 协议下管理状态信息。

92.说一下 Servlet 的体系结构。

所有的 Servlet 都必须要实现的核心的接口是 javax.servlet.Servlet。每一个 Servlet 都必须要直接或者是间接实现这个接口，或者是继承 javax.servlet.GenericServlet 或者 javax.servlet.http.HTTPServlet。最后，Servlet 使用多线程可以并行的为多个请求服务。

93.Applet 和 Servlet 有什么区别？

Applet 是运行在客户端主机的浏览器上的客户端 Java 程序。而 Servlet 是运行在 web 服务器上的服务端的组件。applet 可以使用用户界面类，而 Servlet 没有用户界面，相反，Servlet 是等待客户端的 HTTP 请求，然后为请求产生响应。

94.GenericServlet 和 HttpServlet 有什么区别？

GenericServlet 是一个通用的协议无关的 Servlet，它实现了 Servlet 和 ServletConfig 接口。继承自 GenericServlet 的 Servlet 应该要覆盖 service()方法。最后，为了开发一个能用在网页上服务于使用 HTTP 协议请求的 Servlet，你的 Servlet 必须要继承自 HttpServlet。这里有 Servlet 的例子。
95.解释下 Servlet 的生命周期。

对每一个客户端的请求，Servlet 引擎载入 Servlet，调用它的 init()方法，完成 Servlet 的初始化。然后，Servlet 对象通过为每一个请求单独调用 service()方法来处理所有随后来自客户端的请求，最后，调用 Servlet(译者注：这里应该是 Servlet 而不是 server)的 destroy()方法把 Servlet 删除掉。

96.doGet()方法和 doPost()方法有什么区别？

doGet：GET 方法会把名值对追加在请求的 URL 后面。因为 URL 对字符数目有限制，进而限制了用在客户端请求的参数值的数目。并且请求中的参数值是可见的，因此，敏感信息不能用这种方式传递。

doPOST：POST 方法通过把请求参数值放在请求体中来克服 GET 方法的限制，因此，可以发送的参数的数目是没有限制的。最后，通过 POST 请求传递的敏感信息对外部客户端是不可见的。

97.什么是 Web 应用程序？

Web 应用程序是对 Web 或者是应用服务器的动态扩展。有两种类型的 Web 应用：面向表现的和面向服务的。面向表现的 Web 应用程序会产生包含了很多种标记语言和动态内容的交互的 web 页面作为对请求的响应。而面向服务的 Web 应用实现了 Web 服务的端点(endpoint)。一般来说，一个 Web 应用可以看成是一组安装在服务器 URL 名称空间的特定子集下面的 Servlet 的集合。

98.什么是服务端包含(Server Side Include)？

服务端包含(SSI)是一种简单的解释型服务端脚本语言，大多数时候仅用在 Web 上，用 servlet 标签嵌入进来。SSI 最常用的场景把一个或多个文件包含到 Web 服务器的一个 Web 页面中。当浏览器访问 Web 页面的时候，Web 服务器会用对应的 servlet 产生的文本来替换 Web 页面中的 servlet 标签。

99.什么是 Servlet 链(Servlet Chaining)？

Servlet 链是把一个 Servlet 的输出发送给另一个 Servlet 的方法。第二个 Servlet 的输出可以发送给第三个 Servlet，依次类推。链条上最后一个 Servlet 负责把响应发送给客户端。

100.如何知道是哪一个客户端的机器正在请求你的 Servlet？

ServletRequest 类可以找出客户端机器的 IP 地址或者是主机名。getRemoteAddr()方法获取客户端主机的 IP 地址，getRemoteHost()可以获取主机名。看下这里的例子。
101.HTTP 响应的结构是怎么样的？

HTTP 响应由三个部分组成：

状态码(Status Code)：描述了响应的状态。可以用来检查是否成功的完成了请求。请求失败的情况下，状态码可用来找出失败的原因。如果 Servlet 没有返回状态码，默认会返回成功的状态码 HttpServletResponse.SC_OK。

HTTP 头部(HTTP Header)：它们包含了更多关于响应的信息。比如：头部可以指定认为响应过期的过期日期，或者是指定用来给用户安全的传输实体内容的编码格式。如何在 Serlet 中检索 HTTP 的头部看这里。
主体(Body)：它包含了响应的内容。它可以包含 HTML 代码，图片，等等。主体是由传输在 HTTP 消息中紧跟在头部后面的数据字节组成的。

102.什么是 cookie？session 和 cookie 有什么区别？

cookie 是 Web 服务器发送给浏览器的一块信息。浏览器会在本地文件中给每一个 Web 服务器存储 cookie。以后浏览器在给特定的 Web 服务器发请求的时候，同时会发送所有为该服务器存储的 cookie。下面列出了 session 和 cookie 的区别：

- 无论客户端浏览器做怎么样的设置，session 都应该能正常工作。客户端可以选择禁用 cookie，但是，session 仍然是能够工作的，因为客户端无法禁用服务端的 session。
- 在存储的数据量方面 session 和 cookies 也是不一样的。session 能够存储任意的 Java 对象，cookie 只能存储 String 类型的对象。

103.浏览器和 Servlet 通信使用的是什么协议？
浏览器和 Servlet 通信使用的是 HTTP 协议。

104.什么是 HTTP 隧道？

HTTP 隧道是一种利用 HTTP 或者是 HTTPS 把多种网络协议封装起来进行通信的技术。因此，HTTP 协议扮演了一个打通用于通信的网络协议的管道的包装器的角色。把其他协议的请求掩盖成 HTTP 的请求就是 HTTP 隧道。

105.sendRedirect()和 forward()方法有什么区别？

sendRedirect()方法会创建一个新的请求，而 forward()方法只是把请求转发到一个新的目标上。重定向(redirect)以后，之前请求作用域范围以内的对象就失效了，因为会产生一个新的请求，而转发(forwarding)以后，之前请求作用域范围以内的对象还是能访问的。一般认为 sendRedirect()比 forward()要慢。

106.什么是 URL 编码和 URL 解码？

URL 编码是负责把 URL 里面的空格和其他的特殊字符替换成对应的十六进制表示，反之就是解码。

JSP
107.什么是 JSP 页面？

JSP 页面是一种包含了静态数据和 JSP 元素两种类型的文本的文本文档。静态数据可以用任何基于文本的格式来表示，比如：HTML 或者 XML。JSP 是一种混合了静态内容和动态产生的内容的技术。这里看下 JSP 的例子。
108.JSP 请求是如何被处理的？

浏览器首先要请求一个以.jsp 扩展名结尾的页面，发起 JSP 请求，然后，Web 服务器读取这个请求，使用 JSP 编译器把 JSP 页面转化成一个 Servlet 类。需要注意的是，只有当第一次请求页面或者是 JSP 文件发生改变的时候 JSP 文件才会被编译，然后服务器调用 servlet 类，处理浏览器的请求。一旦请求执行结束，servlet 会把响应发送给客户端。这里看下如何在 JSP 中获取请求参数。
109.JSP 有什么优点？

下面列出了使用 JSP 的优点：

- JSP 页面是被动态编译成 Servlet 的，因此，开发者可以很容易的更新展现代码。
- JSP 页面可以被预编译。
- JSP 页面可以很容易的和静态模板结合，包括：HTML 或者 XML，也可以很容易的和产生动态内容的代码结合起来。
- 开发者可以提供让页面设计者以类 XML 格式来访问的自定义的 JSP 标签库。
- 开发者可以在组件层做逻辑上的改变，而不需要编辑单独使用了应用层逻辑的页面。

110.什么是 JSP 指令(Directive)？JSP 中有哪些不同类型的指令？

Directive 是当 JSP 页面被编译成 Servlet 的时候，JSP 引擎要处理的指令。Directive 用来设置页面级别的指令，从外部文件插入数据，指定自定义的标签库。Directive 是定义在<%@ 和 %>之间的。下面列出了不同类型的 Directive：

- 包含指令(Include directive)：用来包含文件和合并文件内容到当前的页面。
- 页面指令(Page directive)：用来定义 JSP 页面中特定的属性，比如错误页面和缓冲区。
- Taglib 指令： 用来声明页面中使用的自定义的标签库。

111.什么是 JSP 动作(JSP action)？

JSP 动作以 XML 语法的结构来控制 Servlet 引擎的行为。当 JSP 页面被请求的时候，JSP 动作会被执行。它们可以被动态的插入到文件中，重用 JavaBean 组件，转发用户到其他的页面，或者是给 Java 插件产生 HTML 代码。下面列出了可用的动作：

- jsp:include-当 JSP 页面被请求的时候包含一个文件。
- jsp:useBean-找出或者是初始化 Javabean。
- jsp:setProperty-设置 JavaBean 的属性。
- jsp:getProperty-获取 JavaBean 的属性。
- jsp:forward-把请求转发到新的页面。
- jsp:plugin-产生特定浏览器的代码。

112.什么是 Scriptlets？

JSP 技术中，scriptlet 是嵌入在 JSP 页面中的一段 Java 代码。scriptlet 是位于标签内部的所有的东西，在标签与标签之间，用户可以添加任意有效的 scriplet。

113.声明(Decalaration)在哪里？

声明跟 Java 中的变量声明很相似，它用来声明随后要被表达式或者 scriptlet 使用的变量。添加的声明必须要用开始和结束标签包起来。

114.什么是表达式(Expression)？

【列表很长，可以分上、中、下发布】

JSP 表达式是 Web 服务器把脚本语言表达式的值转化成一个 String 对象，插入到返回给客户端的数据流中。表达式是在<%=和%>这两个标签之间定义的。

115.隐含对象是什么意思？有哪些隐含对象？

JSP 隐含对象是页面中的一些 Java 对象，JSP 容器让这些 Java 对象可以为开发者所使用。开发者不用明确的声明就可以直接使用他们。JSP 隐含对象也叫做预定义变量。下面列出了 JSP 页面中的隐含对象：

- application
- page
- request
- response
- session
- exception
- out
- config
- pageContext