重要的问题

第一题：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

第二题：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。

第三题："=="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 = new Object();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。

第四题：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);
  }
}
```


第五题：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。

第六题：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。

第七题：请说出作用域 public，private，protected，以及不写时的区别

修饰符	当前类	同 包	子 类	其他包

public	√	√	√	√
protected	√	√	√	×
default	√	√	×	×
private	√	×	×	×

类的成员不写访问修饰时默认为 default。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。Java 中，外部类的修饰符只能是 public 或默认，类的成员（包括内部类）的修饰符可以是以上四种。

第八题：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。

第九题：如何实现线程间的通讯

Java 提供了 3 个非常重要的方法来巧妙地解决线程间的通信问题。这 3 个方法分别是：wait()、notify()和 notifyAll()。

其中，调用 wait()方法可以使调用该方法的线程释放共享资源的锁，然后从运行态退出，进入等待队列，直到被再次唤醒。而调用 notify()方法可以唤醒等待队列中第一个等待同一共享资源的线程，并使该线程退出等待队列，进入可运行态。调用 notifyAll()方法可以使所有正在等待队列中等待同一共享资源的线程从等待状态退出，进入可运行状态，此时，优先级最高的那个线程最先执行。显然，利用这些方法就不必再循环检测共享资源的状态，而是在需要的时候直接唤醒等待队列中的线程就可以了。这样不但节省了宝贵的 CPU 资源，也提高了程序的效率。

由于 wait()方法在声明的时候被声明为抛出 InterruptedException 异常，因此，在调用 wait()方法时，需要将它放入 try…catch 代码块中。此外，使用该方法时还需要把它放到一个同步代码段中，否则会出现如下异常：

"java.lang.IllegalMonitorStateException: current thread not owner"

第十题：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 class BaseServlet 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 extends BaseServlet
{
    protected void doService(HttpServletRequest request,
HttpServletResponse response) throws
IOExcetion,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。

第十一题：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

第十二题：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。

第十三题：是否可以继承 String 类?

String 类是 final 类故不可以继承。

第十四题：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。String 类表示内容不可改变的字符串。而 StringBuffer 类表示内容可以被修改的字符串。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(new String(“abc”)的结果为 true，而 StringBuffer 没有实现 equals 方法，所以，new StringBuffer(“abc”).equals(new StringBuffer(“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。

第十五题：StringBuffer 与 StringBuilder 的区别

StringBuffer 和 StringBuilder 类都表示内容可以被修改的字符串，StringBuilder 是线程不安全的，运行效率高，如果一个字符串变量是在方法里面定义，这种情况只可能有一个线程访问

它，不存在不安全的因素了，则用 StringBuilder。如果要在类里面定义成员变量，并且这个类的实例对象会在多线程环境下使用，那么最好用 StringBuffer。

第十六题：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。

第十七题：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？

```
public  class Test {

    public static void main(String[] args) {

        System.out.println(new Test().test());;
    }

    static int test() {
        int x = 1;
        try{
            return x;
        }finally{
           ++x;
        }
    }
}
```

---------执行结果 ---------

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。

第十八题：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

第十九题：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。

第二十题：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？

throws 是获取异常

throw 是抛出异常

try 是将会发生异常的语句括起来，从而进行异常的处理，

catch 是如果有异常就会执行他里面的语句，

而 finally 不论是否有异常都会进行执行的语句。

throw 和 throws 的详细区别如下：

throw 是语句抛出一个异常。

语法：throw (异常对象);

throw e;

throws 是方法可能抛出异常的声明。(用在声明方法时，表示该方法可能要抛出异常)

语法：[(修饰符)](返回值类型)(方法名)([参数列表])[throws(异常类)]{.....}

public void doA(int a) throws Exception1,Exception3{.....}

第二十一题：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

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 void run(){
    }
}).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(new Runable(){public void run(){}});
}
Executors.newCachedThreadPool().execute(new
Runable(){public void run(){}});
Executors.newSingleThreadExecutor().execute(new
Runable(){public void 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。

第二十二题：sleep() 和 wait() 有什么区别？

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。

第二十三题：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。

第二十四题：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。

第二十五题：List, Set, Map 是否继承自 Collection 接口？

List，Set 是，Map 不是

第二十六题：说出 ArrayList,Vector, LinkedList 的存储性能和特性

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,obj value)，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

第二十七题：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

第二十八题：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/Vector

Collection

HashSet/TreeSetSet

PropetiesHashTable

Map

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。

第二十九题：java 中有几种类型的流？JDK 为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类

字节流，字符流。字节流继承于 InputStream OutputStream，字符流继承于 InputStreamReader OutputStreamWriter。在 java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。

第三十题：字节流与字符流的区别

要把一片二进制数据数据逐一输出到某个设备中，或者从某个设备中逐一读取一片二进制数据，不管输入输出设备是什么，我们要用统一的方式来完成这些操作，用一种抽象的方式进行描述，这个抽象描述方式起名为 IO 流，对应的抽象类为 OutputStream 和 InputStream ，不同的实现类就代表不同的输入和输出设备，它们都是针对字节进行操作的。

在应用中，经常要完全是字符的一段文本输出去或读进来，用字节流可以吗？计算机中的一切最终都是二进制的字节形式存在。对于“中国”这些字符，首先要得到其对应的字节，然后将字节写入到输出流。读取时，首先读到的是字节，可是我们要把它显

示为字符，我们需要将字节转换成字符。由于这样的需求很广泛，人家专门提供了字符流的包装类。

底层设备永远只接受字节数据，有时候要写字符串到底层设备，需要将字符串转成字节再进行写入。字符流是字节流的包装，字符流则是直接接受字符串，它内部将串转成字节，再写入底层设备，这为我们向 IO 设别写入或读取字符串提供了一点点方便。

字符向字节转换时，要注意编码的问题，因为字符串转成字节数组，

其实是转成该字符的某种编码的字节形式，读取也是反之的道理。

第三十一题：什么是 java 序列化，如何实现 java 序列化？或者请解释 Serializable 接口的作用

我们有时候将一个 java 对象变成字节流的形式传出去或者从一个字节流中恢复成一个 java 对象，例如，要将 java 对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个 java 对象变成某个格式的字节流再传输，但是，jre 本身就提供了这种支持，我们可以调用 OutputStream 的 writeObject 方法来做，如果要让 java 帮我们做，要被传输的对象必须实现 serializable 接口，这样，javac 编译时就会进行特殊处理，编译的类才可以被 writeObject 方法操作，这就是所谓的序列化。需要被序列化的类必须实现 Serializable 接口，该接口是一个 mini 接口，其中没有需要实现的方法，implements Serializable 只是为了标注该对象是可被序列化的。

例如，在 web 开发中，如果对象被保存在了 Session 中，tomcat 在重启时要把 Session 对象序列化到硬盘，这个对象就必须实现 Serializable 接口。如果对象要经过分布式系统进行网络传输或通过 rmi 等远程调用，这就需要在网络上传输对象，被传输的对象就必须实现 Serializable 接口。

第三十二题：能不能自己写个类，也叫 java.lang.String？

可以，但在应用的时候，需要用自己的类加载器去加载，否则，系统的类加载器永远只是去加载 jre.jar 包中的那个 java.lang.String。由于在 tomcat 的 web 应用程序中，都是由 webapp 自己的类加载器先自己加载 WEB-INF/classess 目录中的类，然后才委托上级的类加载器加载，如果我们在 tomcat 的 web 应用程序中写一个 java.lang.String，这时候 Servlet 程序加载的就是我们自己写的 java.lang.String，但是这么干就会出很多潜在的问题，原来所有用了 java.lang.String 类的都将出现问题。

虽然 java 提供了 endorsed 技术，可以覆盖 jdk 中的某些类，具体做法是….。但是，能够被覆盖的类是有限制范围，反正不包括 java.lang 这样的包中的类。

（下面的例如主要是便于大家学习理解只用，不要作为答案的一部分，否则，人家怀疑是题目泄露了）例如，运行下面的程序：

```
package cn.itcast.entity;

public class String {

/**
 * @param args
 */
  public static void main(String[] args) {
     // TODO Auto-generated method stub
     System.out.println("string");
  }

}
```

报告的错误如下：

java.lang.NoSuchMethodError: main

Exception in thread "main"

这是因为加载了 jre 自带的 java.lang.String，而该类中没有 main 方法。

第三十三题：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。

第三十四题：Jvm 如何调优？

观察内存释放情况、集合类检查、对象树

可查看堆空间大小分配（年轻代、年老代、持久代分配）

提供即时的垃圾回收功能

垃圾监控（长时间监控回收情况）

查看堆内类、对象信息查看：数量、类型等

对象引用情况查看

有了堆信息查看方面的功能，我们一般可以顺利解决以下问题：

--年老代年轻代大小划分是否合理

--内存泄漏

--垃圾回收算法设置是否合理

线程信息监控：系统线程数量。

线程状态监控：各个线程都处在什么样的状态下

Dump 线程详细信息：查看线程内部运行情况

死锁检查

内存泄漏检查

内存泄漏是比较常见的问题，而且解决方法也比较通用，这里可以重点说一下，而线程、热点方面的问题则是具体问题具体分析了。

内存泄漏一般可以理解为系统资源(各方面的资源，堆、

栈、线程等）在错误使用的情况下，导致使用完毕的资源无法回收（或没有回收），从而导致新的资源分配请求无法完成，引起系统错误。

内存泄漏对系统危害比较大，因为他可以直接导致系统的崩溃。

需要区别一下，内存泄漏和系统超负荷两者是有区别的，虽然可能导致的最终结果是一样的。内存泄漏是用完的资源没有回收引起错误，而系统超负荷则是系统确实没有那么多资源可以分配了（其他的资源都在使用）

解决：

这种方式解决起来也比较容易，一般就是根据垃圾回收前后情况对比，同时根据对象引用情况（常见的集合对象引用）分析，基本都可以找到泄漏点。

持久代被占满

异常： java.lang.OutOfMemoryError: PermGen space

说明：

Perm 空间被占满。无法为新的 class 分配存储空间而引发的异常。这个异常以前是没有的，但是在 Java 反射大量使用的今天这个异常比较常见了。主要原因就是大量动态反射生成的类不断被加载，最终导致 Perm 区被占满。

更可怕的是，不同的 classLoader 即便使用了相同的类，但是都会对其进行加载，相当于同一个东西，如果有 N 个 classLoader 那么他将会被加载 N 次。因此，某些情况下，这个问题基本视为无解。当然，存在大量 classLoader 和大量反射类的情况其实也不多。

解决：

1. -XX:MaxPermSize=16m
2. 换用 JDK。比如 JRocket。

堆栈溢出

异常： java.lang.StackOverflowError

说明： 这个就不多说了，一般就是递归没返回，或者循环调用造成

线程堆栈满

异常： Fatal: Stack size too small

说明： java 中一个线程的空间大小是有限制的。JDK5.0 以后这个值是 1M。与这个线程相关的数据将会保存在其中。但是当线程空间满了以后，将会出现上面异常。

解决： 增加线程栈大小。-Xss2m。但这个配置无法解决根本问题，还要看代码部分是否有造成泄漏的部分。

系统内存被占满

异常： java.lang.OutOfMemoryError: unable to create new native thread

说明：

这个异常是由于操作系统没有足够的资源来产生这个线程造成的。系统创建线程时，除了要在 Java 堆中分配内存外，操作系统本身也需要分配资源来创建线程。因此，当线程数量大到一定程度以后，堆中或许还有空间，但是操作系统分配不出资源来了，就出现这个异常了。

分配给 Java 虚拟机的内存愈多，系统剩余的资源就越少，因此，当系统内存固定时，分配给 Java 虚拟机的内存越多，那么，系统总共能够产生的线程也就越少，两者成反比的关系。同时，可以通过修改-Xss 来减少分配给单个线程的空间，也可以增加系统总共内生产的线程数。

解决：

1. 重新设计系统减少线程数量。
2. 线程数量不能减少的情况下，通过-Xss 减小单个线

程大小。以便能生产更多的线程。

第三十五题：Jvm 如何加载类？如何分配空间。

指的是将 class 文件的二进制数据读入到运行时数据区（JVM 在内存中划分的）

中，并在方法区内创建一个 class 对象

JVM 运行起来时就给内存划分空间，这块空间就称为运行时数据区。

运行时数据区被划分为以下几块内容

1. 栈：

每一个线程运行起来的时候就会对应一个栈（线程栈），栈中存放的数据被当前

线程所独享（不会产生资源共享情况，所以线程是安全的）。而栈当中存放的是栈帧，

当线程调用方法时，就是形成一个栈帧，并将这个栈帧进行压栈操作。方法执行完后，

进行出栈操作。这个栈帧里面包括（局部变量，操作数栈，指向当前方法对应类的常

量池引用，方法返回地址等信息）。

2. 本地方法栈：

本地方法栈的机制和栈的相似，区别在于，栈运行的是 Java 实现的方法，而本地

方法栈运行的是本地方法。本地方法指的是 JVM 需要调用非 Java 语言所实现的方法，

例如 C 语言。在 JVM 规范中，没有强化性要求实现方一定要划分出本地方法栈（例如：

HotSpot 虚拟机将本地方法栈和栈合二为一）和具体实现（不同的操作系统，对 JVM

规范的具体实现都不一样）。

3.程序计数器：

程序计数器也可以称为 PC 寄存器(通俗讲就是　指令缓存)。它主要用于缓存当前

程序下一条指令的指令地址，CPU 根据这个地址找到将要执行的指令。这个寄存器是 JVM

内部实现的，不是物理概念上的计数器，不过和 JVM 的实现逻辑一样。

4.堆：

堆内存主要存放创建的对象和数组。堆内存在 JVM 中是唯一的，能被多个线程所共享。

堆里面的每一个对象都存放着实例的实例变量。堆内存的对象没有被引用，会自动被 Java

垃圾回收机制回收。

当在方法中定义了局部变量，如果这个变量是基本数据类型，那么这个变量的值就直接

存放在栈中；如果这个变量是引用数据类型，那么变量值就存放在堆内存中，而栈中存放的是

指向堆中的引用地址。

5.方法区：

方法区在 JVM 也是一个非常重要的一块内存区域,和堆一样,可以被多个线程多共享。

主要存放每一个加载 class 的信息。class 信息主要包含魔数(确定是否是一个 class 文件），常量

池，访问标志（当前的类是普通类还是接口，是否是抽象类，是否被 public 修饰，是否使用了 final

修饰等描述信息.....），字段表集合信息（使用什么访问修饰符，是实例变量还是静态变量，是否

使用了 final 修饰等描述信息.....），方法表集合信息（使用什么访问修饰符，是否静态方法，是否

使用了 final 修饰，是否使用了 synchronized 修饰，是否是 native 方法.....）等内容。当一个类加

载器加载了一个类的时候，会根据这个 class 文件创建一个 class 对象，class 对象就包含了上述的信息。

后续要创建这个类的实例，都根据这个 class 对象创建出来的。

6.常量池：

常量池是方法区中的一部分，存放 class 对象中最重要的资源。JVM 为每一个 class 对象都维护一个

常量池。

第三十六题：八个基本类型各占多少字节？

byte 1 字节

short 2 字节

int 4 字节

long 8 字节

float 4 字节

double 8 字节

char 2 字节

boolean 1 字节（理论上是 1 比特位八分之一个字节，但是计算机存储的最小单位是 1 个字节）

第三十七题：HashMap、HashSet、HashTable 的区别？

区别一：继承的父类不同

Hashtable 继承自 Dictionary 类，而 HashMap 继承自 AbstractMap 类。但二者都实现了 Map 接口。

区别二：线程安全性不同

Hashtable 中的方法是 Synchronize 的，而 HashMap 中的方法在缺省情况下是非 Synchronize 的。

区别三：是否提供 contains 方法

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 则保留了 contains，containsValue 和 containsKey 三个方法，其中 contains 和 containsValue 功能相同。

区别四：**key 和 value 是否允许 null 值 （面试比较喜欢问）

其中 key 和 value 都是对象，并且不能包含重复 key，但可以包含重复的 value。

Hashtable 中，key 和 value 都不允许出现 null 值。

HashMap 中，null 可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为 null。当 get()方法返回 null 值时，可能是 HashMap 中没有该键，也可能使该键所对应的值为 null。因此，在 HashMap 中不能由 get()方法来判断 HashMap 中是否存在某个键， 而应该用 containsKey()方法来判断。

区别五：哈希值的计算方法不同，Hashtable 直接使用的是对象

的 hashCode,而 HashMap 则是在对象的 hashCode 的基础上还进行了一些变化。

区别六：内部实现使用的数组初始化和扩容方式不同，内存初始大小不同，HashTable 初始大小是 11，而 HashMap 初始大小是 16

第三十八题：Hashcode 和 equals

equals：

Object 类中默认的实现方式是 ： return this == obj 。那就是说，只有 this 和 obj 引用同一个对象，才会返回 true。

而我们往往需要用 equals 来判断 2 个对象是否等价，而非验证他们的唯一性。这样我们在实现自己的类时，就要重写 equals

按照约定，equals 要满足以下规则。

自反性： x.equals(x) 一定是 true

对 null： x.equals(null) 一定是 false

对称性： x.equals(y) 和 y.equals(x)结果一致

传递性： a 和 b equals ， b 和 c equals，那么 a 和 c 也一定 equals。

一致性： 在某个运行时期间，2 个对象的状态的改变不会不影响 equals 的决策结果，那么，在这个运行时期间，无论调用多少次 equals，都返回相同的结果。

Hashcode：

这个方法返回对象的散列码，返回值是 int 类型的散列码。
对象的散列码是为了更好的支持基于哈希机制的 Java 集合类，例如 Hashtable, HashMap, HashSet 等。

关于 hashCode 方法，一致的约定是：

重写了 euqls 方法的对象必须同时重写 hashCode()方法。

如果 2 个对象通过 equals 调用后返回是 true，那么这个 2 个对象的 hashCode 方法也必须返回同样的 int 型散列码

如果 2 个对象通过 equals 返回 false，他们的 hashCode 返回的值**允许相同**。(然而，程序员必须意识到，hashCode 返回独一无二的散列码，会让存储这个对象的 hashtables 更好地工作。)

在上面的例子中，Test 类对象有 2 个字段，num 和 data，这 2 个字段代表了对象的状态，他们也用在 equals 方法中作为评判的依据。那么， 在 hashCode 方法中，这 2 个字段也要参与 hash 值的运算，作为 hash 运算的中间参数。这点很关键，这是为了遵守：2 个对象 equals，那么 hashCode 一定相同规则。

也是说，参与 equals 函数的字段，也必须都参与 hashCode 的计算。

合乎情理的是：同一个类中的不同对象返回不同的散列码。典型的方式就是根据对象的地址来转换为此对象的散列码，但是这种方式对于 Java 来说并不是唯一的要求的
的实现方式。通常也不是最好的实现方式。

相比 于 equals 公认实现约定，hashCode 的公约要求是很容易理解的。有 2 个重点是 hashCode 方法必须遵守的。约定的第 3 点，其实就是第 2 点的
细化，下面我们就来看看对 hashCode 方法的一致约定要求。

第一：在某个运行时期间，只要对象的（字段的）变化不会影响 equals 方法的决策结果，那么，在这个期间，无论调用多少次 hashCode，都必须返回同一个散列码。

第二：通过 equals 调用返回 true 的 2 个对象的 hashCode 一定一样。

第三：通过 equasl 返回 false 的 2 个对象的散列码不需要不同，也就是他们的 hashCode 方法的返回值允许出现相同的情况。

总结一句话：等价的(调用 equals 返回 true)对象必须产生相同的散列码。不等价的对象，不要求产生的散列码不相同。

第三十九题：方法重载和重写的区别

1. 重写必须继承，重载不用。
2. 重写的方法名，参数数目相同，参数类型兼容，重载的方法名相同，参数列表不同。
3. 重写的方法修饰符大于等于父类的方法，重载和修饰符无关。
4. 重写不可以抛出父类没有抛出的一般异常，可以抛出运行时异常

第四十题：进程和线程的区别

1. 定义

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.

线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

2. 关系

一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.

相对进程而言，线程是一个更加接近于执行体的概念，它可以与同进程中的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。

3. 区别

进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。**但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。**

1） 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.

2） 线程的划分尺度小于进程，使得多线程程序的并发性高。

3） 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4） 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。**但是线程不能够独立执行，**必须依存在应用程序中，由应用程序提供多个线程执行控制。

5） 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看

做多个独立的应用，来实现进程的调度和管理以及资源分配。**这就是进程和线程的重要区别。**

4.优缺点

线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源的管理和保护；而进程正相反。同时，线程适合于在 SMP 机器上运行，而进程则可以跨机器迁移。

第四十一题：junit 中 before 和 beforeclass 区别

@before

在每个测试方法之前都会运行一次，只需声明成 public

@beforeclass

在类中只运行一次，必须声明成 public static

第四十二题：单例手写

懒汉式

```
public class Singleton {
    private Singleton() {}
    private static Singleton single=null;
    //静态工厂方法
    public static Singleton getInstance() {
         if (single == null) {
             single = new Singleton();
         }
        return single;
    }
}
```

饿汉式

```
//饿汉式单例类.在类初始化时，已经自行实例化
public class Singleton1 {

    private Singleton1() {}
    private static final Singleton1 single = new
Singleton1();
    //静态工厂方法
    public static Singleton1 getInstance() {
        return single;
    }
}
```

第四十三题：触发器的作用是什么？

比如说你 emp 和 dept 两张表是有外键关联的，当 emp 存在相关数据时，dept 无法删除数据，这时候就可以写个触发器，让他可以删除的同时并对 emp 表的依赖数据发生变化

第四十四题：Static 局部变量与全局变量的区别，编译后映射文件是否包含此类变量的地址？

全局变量(外部变量)的说明之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式， 静态全局变量当然也是静态存储方式。 这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。 而静态全局变量则限制了其作用域， 即只在定义该变量的源文件内有效， 在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用，

因此可以避免在其它源文件中引起错误。

从以上分析可以看出， 把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域， 限制了它的使用范围。

static 函数与普通函数作用域不同。仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件

static 全局变量与普通的全局变量有什么区别：static 全局变量只初使化一次，防止在其他文件单元中被引用；

static 局部变量和普通局部变量有什么区别：static 局部变量只被初始化一次，下一次依据上一次结果值；

static 函数与普通函数有什么区别：static 函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝

程序的局部变量存在于（堆栈）中，全局变量存在于（静态区 ）中，动态申请数据存在于（ 堆）中。

extern 全局变量、static 全局变量和 static 局部变量的生存期都是“永久”，区别只是可见域不同。extern 全局变量可见区域是工程，static 全局变量可见区域是文件，而 static 局部变量的可见区域是块。

从代码维护角度来看，对 extern 变量的修改可能会影响所有代码，对 static 全局变量的修改可能影响一个文件中的代码，而对 static 变量的修改 可能影响一个块的代码；

因此在选择变量类型时，优先级是 static 局部>static 全局>extern 全局。但它们有着共同的缺点：使用 了这些类型变量的函数将是不可重入的，不是线程安全的。在 C/C++标准库中有很多函数都使用了 static 局部变量，目前的实现中都为它们提供了两套代 码，单线程版本使用 static 变量而多线程版本使用“线程全局变量”，比如 rand, strtok 等。

一个进程可用内存空间为 4G，可分在存放静态数据，代码，系统内存，堆，栈等。.活动记录一般存放调用参数、返回地址等内容。堆和栈最大的区别在于 堆是由低地址向高地址分配内存，而栈是由高向低。全局和静态数据存放在全局数据区，其余的在栈中，用 malloc 或 new 分配的内存位于堆中。一般来说栈在低地址，堆位于高地址

第四十五题：用 JAVA 实现一种排序。（要写出具体的算法实现，不要简单的调用 Arrays.sort 方法）。

冒泡排序（Bubble Sort）是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。

冒泡排序算法的运作如下:

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
3. 针对所有的元素重复以上的步骤，除了最后一个。
4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

代码：

```
public class BubbleSort{
        public static void main(String[] args){
            int score[] = {67, 69, 75, 87, 89, 90, 99,
100};
            for (int i = 0; i < score.length -1;
i++){    //最多做 n-1 趟排序
                for(int j = 0 ;j < score.length - i - 1;
j++){    //对当前无序区间 score[0......length-i-1]进行排
序(j 的范围很关键，这个范围是在逐步缩小的)
                    if(score[j] < score[j + 1]){    //
把小的值交换到后面
                        int temp = score[j];
                        score[j] = score[j + 1];
                        score[j + 1] = temp;
                    }
                }
                System.out.print("第" + (i + 1) + "次排
序结果：");
                for(int a = 0; a < score.length; a++){
                    System.out.print(score[a] + "\t");
                }
                System.out.println("");
            }
            System.out.print("最终排序结果：");
            for(int a = 0; a < score.length; a++){
                System.out.print(score[a] + "\t");
            }
        }
}
```


第四十六题：Java 中如何实现多继承关系？

1，java 中只能利用接口达到多实现而已，跟多继承相仿

2，java 中唯一可以实现多继承的 就是接口与接口之间了。

第四十七题：怎么进行数组排序？

分析：

对数组进行排序，常规的排序时用两个 for 循环，只是这样比较

的次数固定，会比较多；另外一种优化的方法是加一个标志位看是否进行了位置改变，如果进行了改变，则在该次比较完成之后，还需要从头再次比较。这样的好处是如果本来就是排好序的，则直接一遍比完就结束了。

代码如下：

双重 for 循环进行排序：

```
Mouse temp = new Mouse();
for (int i=0;i<mouse.length;i++)
{
for (int j=i+1;j<mouse.length;j++)
{
if (mouse[i].weight > mouse[j].weight)
{
temp = mouse[i];
mouse[i] = mouse[j];
mouse[j] = temp;
}
}
}
```


加标志位用 while 和 for 双重循环进行排序：

```
boolean flag=true;
while(flag)
{
flag=false;
for(j=0;j+1<mouse.length;j++)
{
if(mouse[j].weight>mouse[j+1].weight)
{
```

```
flag=true;
temp=mouse[j+1];
mouse[j+1]=mouse[j];
mouse[j]=temp;
}
}
}
```

第四十八题：当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?

是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的

第四十九题：请用 java 代码(或伪代码)实现字符串的反转，如：输入 abcde，输出 edcba。

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

    Scanner sc = new Scanner(System.in);
    System.out.println("请输入字符串 按回车键结束");
    char[] inputChars = sc.next().toCharArray();

    for (int i = inputChars.length-1; i >=0 ; i--) {
        System.out.print(inputChars[i]);
    }
    System.out.println();
```

}

第五十题：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

第五十一题：GC 是什么？为什么要有 GC?

GC 是垃圾收集的意思（Gabage Collection），内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。

第五十二题：Math.round(11.5)等於多少？Math.round(-11.5)等於多少？

Math.round(11.5)==12

Math.round(-11.5)==-11

round 方法返回与参数最接近的长整数，参数加 1/2 后求其 floor.