目　录

一、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，其他。有些题来自网上搜集整理，有些题来自某某培训学员面试后的反馈，说真的，少数一些网上的面试题，我真怀疑其是否还有存在价值！

1.1 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。

1.2 Java 有没有 goto?

java 中的保留字，现在没有在 java 中使用。

1.3 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。

1.4 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;
                }
            }
        }
```

1.5 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。

1.6 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。

1.7 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。

1.8 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。

1.9 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

（）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。

1.10 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");
```

1.11 "=="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 = 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。

1.12 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！

1.13 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，不可以一个 static 方法内部发出对非 static 方法的调用。

1.14 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。

1.15 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，所以，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，所以，Math.floor(11.6)的结果为 11,Math.floor(-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。

1.16 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;
```

答：第一个问题少了一个右括号;第二个问题没有错误

1.17 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。

1.18 Overload 和 Override 的区别。Overload 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。

1.19 4 月 21 号班同学贡献的一些题？

朱 wenchao，女：3500，21 岁

搞了多个重载方法，参数分别是 int ,char,和 double，然后将 double x = 2，传递进去，会选择哪个方法？

陈 yong，4000

一个房子里有椅子，椅子有腿和背，房子与椅子是什么关系，椅子与腿和背是什么关系？

如果房子有多个椅子，就是聚合关系，否则是一种关联关系，当然，聚合是一种特殊的关联。

椅子与腿和背时组合关系。

说说 has a 与 is a 的区别。

答：is-a 表示的是属于得关系。比如兔子属于一种动物（继承关系）。

has-a 表示组合，包含关系。比如兔子包含有腿，头等组件；

工厂模式的类图

1.20 5 月 15 号班同学贡献的一些题？

1.线程如何同步和通讯。

同学回答说 synchronized 方法或代码块！面试官似乎不太满意！只有多个 synchronized 代码块使用的是同一个监视器对象，这些 synchronized 代码块之间才具有线程互斥的效果，假如 a 代码块用 obj1 作为监视器对象，假如 b 代码块用 obj2 作为监视器对象，那么，两个并发的线程可以同时分别进入这两个代码块中。 …这里还可以分析一下同步的原理。

对于同步方法的分析，所用的同步监视器对象是 this。接着对于静态同步方法的分析，所用的同步监视器对象是该类的 Class 对象。接着对如何实现代码块与方法的同步进行分析。

2.ClassLoader 如何加载 class 。

jvm 里有多个类加载，每个类加载可以负责加载特定位置的类，例如，bootstrap 类加载负责加载 jre/lib/rt.jar 中的类， 我们平时用的 jdk 中的类都位于 rt.jar 中。

extclassloader 负责加载 jar/lib/ext/*.jar 中的类，appclassloader 负责 classpath 指定的目录或 jar 中的类。除了 bootstrap 之外，其他的类加载器本身也都是 java 类，它们的父类是 ClassLoader。

3.Servlet 的生命周期 init service destroy

4.抽象类的作用

5.ArrayList 如何实现插入的数据按自定义的方式有序存放

```
class MyBean implements Comparable{
    public int compareTo(Object obj){
        if(! obj instanceof MyBean)
            throw new ClassCastException() //具体异常的名称，我要查 jdk 文档。

        MyBean other = (MyBean) obj;
        return age > other.age?1:age== other.age?0:-1;
    }
}

class MyTreeSet {

    private ArrayList    datas = new ArrayList();

    public void add(Object obj){
        for(int i=0;i<datas.size();i++){
            if(obj.compareTo(datas.get(i) != 1){
                datas.add(i,obj);
            }
        }
    }
}
```

6.分层设计的好处

把各个功能按调用流程进行了模块化,模块化带来的好处就是可以随意组合,举例说明:如果要注册一个用户,流程为显示界面并通过界面接收用户的输入,接着进行业务逻辑处理,在处理业务逻辑又访问数据库,如果我们将这些步骤全部按流水帐的方式放在一个方法中编写,这也是可以的,但这其中的坏处就是,当界面要修改时,由于代码全在一个方法内,可能会碰坏业务逻辑和数据库访问的码,同样,当修改业务逻辑或数据库访问的代码时,也会碰坏其他部分的代码。分层就是要把界面部分、业务逻辑部分、数据库访问部分的代码放在各自独立的方法或类中编写,这样就不会出现牵一发而动全身的问题了。这样分层后,还可以方便切换各层,譬如原来的界面是 Swing,现在要改成 BS 界面,如果最初是按分层设计的,这时候不需要涉及业务和数据访问的代码,只需编写一条 web 界面就可以了。

下面的仅供参考,不建议照搬照套,一定要改成自己的语言,发现内心的感受:

分层的好处:

1.实现了软件之间的解耦;

2.便于进行分工

3.便于维护

4.提高软件组件的重用

5.便于替换某种产品,比如持久层用的是hibernate,需要更换产品用toplink,就不用该其他业务代码,直接把配置一改。

6.便于产品功能的扩展。

7。便于适用用户需求的不断变化

7.序列化接口的 id 有什么用?

对象经常要通过 IO 进行传送,让你写程序传递对象,你会怎么做?把对象的状态数据用某种格式写入到硬盘,Person->“zxx,male,28,30000”→Person,既然大家都要这么干,并且没有个统一的干法,于是,sun 公司就提出一种统一的解决方案,它会把对象变成某个格式进行输入和输出,这种格式对程序员来说是透明(transparent)的,但是,我们的某个类要想能被 sun 的这种方案处理,必须实现 Serializable 接口。

```
ObjectOutputStream.writeObject(obj);
Object obj = ObjectInputStream.readObject();
```

假设两年前我保存了某个类的一个对象,这两年来,我修改该类,删除了某个属性和增

加了另外一个属性，两年后，我又去读取那个保存的对象，或有什么结果？未知！sun 的 jdk 就会蒙了。为此，一个解决办法就是在类中增加版本后，每一次类的属性修改，都应该把版本号升级一下，这样，在读取时，比较存储对象时的版本号与当前类的版本号，如果不一致，则直接报版本号不同的错！

9.hashCode 方法的作用？

网友提供的一段，待改进：hashcode 这个方法是用来鉴定 2 个对象是否相等的。那你会说，不是还有 equals 这个方法吗？不错，这 2 个方法都是用来判断 2 个对象是否相等的。但是他们是有区别的。一般来讲，equals 这个方法是给用户调用的，如果你想判断 2 个对象是否相等，你可以重写 equals 方法，然后在代码中调用，就可以判断他们是否相等了。简单来讲，equals 方法主要是用来判断从表面上看或者从内容上看，2 个对象是不是相等。举个例子，有个学生类，属性只有姓名和性别，那么我们可以认为只要姓名和性别相等，那么就说这 2 个对象是相等的。

hashcode 方法一般用户不会去调用，比如在 hashmap 中，由于 key 是不可以重复的，他在判断 key 是不是重复的时候就判断了 hashcode 这个方法，而且也用到了 equals 方法。这里不可以重复是说 equals 和 hashcode 只要有一个不等就可以了!所以简单来讲,hashcode 相当于是一个对象的编码，就好像文件中的 md5，他和 equals 不同就在于他返回的是 int 型的，比较起来不直观。我们一般在覆盖 equals 的同时也要覆盖 hashcode，让他们的逻辑一致。举个例子，还是刚刚的例子，如果姓名和性别相等就算 2 个对象相等的话，那么 hashcode 的方法也要返回姓名的 hashcode 值加上性别的 hashcode 值，这样从逻辑上，他们就一致了。

要从物理上判断 2 个对象是否相等，用==就可以了。

10.webservice 问得很多

11.设计出计算任意正整数的阶层。

12.在 oracle 数据库中需要查询出前 8 条记录的 sql 语句怎么写？

13.什么是 SOA，谈谈你的 SOA 的理解。service orientied architecture

14.如何实现线程间的通讯。

新题目：编程:1.编写一个函数将一个十六进制数的字符串参数转换成整数返回。

```
String str = “13abf”;
int len = str.length;
```

```
int sum = 0;
for(int i=0;i<len;i++){
    char c = str.charAt(len-1-i);
    int n = Character.digit(c,16);
    sum += n * (1<<(4*i));
}
```

其实，也可以用 Integer.parseInt(str,16)，但面试官很可能是想考我们的编码基本功。

编程 2

:银行贷款的还款方式中最常用的是一种叫“等额本息”，还款法，即借款人在约定还款期限内的每一期（月）归还的金额（产生的利息+部分本金）都是相等的，现有一笔总额为 T 元的 N 年期住房贷款，年利率为 R，要求算出每一期的还款的本金和利息总额，请写出解决思路和任意一种编程语言实现的主要代码。

思路：既然是按月还款，那我就要将 N 年按月来计算，即要还 N*12 个月，这样就可以求出每月要还的本金。由于每月要还的那部分本金所欠的时间不同，所以，它们所产生的利息是不同的，该部分本金的利息为：部分本金额*所欠月数*月利率。应该是这么个算法，如果利息还计利息，如果月还款不按年利率来算，老百姓算不明白的。

```
int monthMoney = T/N/12;
float monthRate = R/12;
int totalMonth = N * 12;
float totalRate = 0;
for(int i=1;i<=totalMonth;i++){
    totalRate += monthMoney * monthRate * i;
}
int result = monthMoney + totalRate/N/12;
```

几道题：

1. Spring 的 DI 是什么（学员注：除了 IOC，AOP 这些概念，还不太清楚 DI 的概念）

2. 任意数字序列“123456”之类，输出它们所有的排列组合

3. 什么是 AOP（学员注：会用，但感觉说不清楚）

我注：由上面这些题，可以看出，思想很重要，只有琢磨思想和原理的人才能很好地回答这些问题！

2 题的答案：

```
String str = “xafdvs”;
char[] arr1 = str.toCharArray();
char[] arr2 = Arrays.copyOf(arr1,arr1.length);
for(int i=0;i<arr1.length-1;i++)
{
    for(int j = i+1;j<arr2.length;j++){
        syso: arr1[i] + “,” + arr2[j];
    }
}
```

3 题的答案：

1.概念介绍：所谓 AOP，即 Aspect orientied program,就是面向方面的编程，

2.解释什么是方面：贯穿到系统的各个模块中的系统一个功能就是一个方面，

比如，记录日志，统一异常处理，事务处理，全限检查，这些功能都是软件系统

的一个面，而不是一点，在各个模块中都要出现。

3.什么是面向方面编程：把系统的一个方面的功能封装成对象的形式来处理

4.怎么进行面向方面编程：把功能模块对应的对象作为切面嵌入到原来的各个系统模块中，

采用代理技术，代理会调用目标，同时把切面功能的代码（对象）加入进来，所以，

用 spring 配置代理对象时只要要配两个属性，分别表示目标和切面对象（Advisor）。

1.21 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。

1.22 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。

1.23 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。

1.24 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封装、继承 、抽象、多态等 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：

```
class Person
{
    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？

1.25 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。

1.26 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。

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。

1.27 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(String name) 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。

1.28 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.new Innner1();
```

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 = new Outer.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。

1.29 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。

1.30 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)?

答：可以继承其他类或实现其他接口。不仅是可以，而是必须!

1.31 super.getClass()方法调用

下面程序的输出结果是多少？

```
import java.util.Date;
public  class Test extends Date{
    public static void main(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();
```

1.32 jdk 中哪些类是不能继承的？

不能继承的是类是那些用 final 关键字修饰的类。一般比较基本的类型或防止扩展类无意间破坏原来方法的实现的类型都应该是 final 的，在 jdk 中 System,String,StringBuffer 等都是基本类型。

1.33 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

1.34 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。

1.35 是否可以继承 String 类?

不能，String 类是 final 类，故不可以继承。

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

两个对象，”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。

1.37 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 StringBuffer 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。

1.38 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。

1.39 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();}

1.40 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。

1.41 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。

1.42 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  class Test {
    /**
     * @param args add by zxx ,Dec 9, 2008
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(new Test().test());;
    }

    static int test()  {
        int x = 1;
        try  {
            return x;
        }
        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。

1.43 下面的程序代码输出的结果是多少？

```
public class  smallT {
    public static void  main(String args[])  {
        smallT t  = new  smallT();
        int  b  =  t.get();
        System.out.println(b);
    }

    public int  get() {
        try {
            return 1 ;
        }
        finally {
            return 2 ;
        }
    }
}
```

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  class Test {

    /**
```

```
     * @param args add by zxx ,Dec 9, 2008
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(new Test().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

1.44 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。

1.45 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。

1.46 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。

1.47 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；

1.48 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。

1.49 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？

Java 通过**面向对象**的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或其它子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可

以捕获到这个异常并进行处理。

Java 的异常处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throws）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理。用 try 来指定一块预防所有"异常"的程序。紧跟在 try 程序后面，应包含一个 catch 子句。来指定你想要捕捉的"异常"的类型。

throw 语句用来明确地抛出一个"异常"。

throws 用来标明一个成员函数可能抛出的各种"异常"。

finally 为确保一段代码不管发生什么"异常"都被执行一段代码。可以在一个成员函数调用的外面写一个 try 语句，在这个成员函数内部写另一个 try 语句保护其他代码。每当遇到一个 try 语句，"异常"的框架就放到堆栈上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种"异常"进行处理，堆栈就会展开，直到遇到有处理这种"异常"的 try 语句。

1.50 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 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。

1.51 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，b

把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;

public class MultiThread {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new Thread(new Thread1()).start();
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
        }
        new Thread(new Thread2()).start();
    }


    private static class Thread1 implements Runnable
    {

        @Override
        public void run() {
            // TODO Auto-generated method stub
//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("enter thread1...");

                System.out.println("thread1 is 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-generated catch block
                    e.printStackTrace();
                }
```

```
                System.out.println("thread1 is going on...");
                System.out.println("thread1 is being over!");
            }
        }

    }

    private static class Thread2 implements Runnable
    {

        @Override
        public void run() {
            // TODO Auto-generated method stub
            synchronized (MultiThread.class) {

                System.out.println("enter thread2...");

                System.out.println("thread2 notify 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("thread2 is sleeping ten millisecond...");
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
```

```
                }

                System.out.println("thread2 is going on...");
                System.out.println("thread2 is being over!");

            }
        }

    }

}
```

1.52 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。

1.53 下面两个方法同步吗？（自己发明）

```
class Test
{
    synchronized static void sayHello3()
    {

    }

    synchronized void getX(){}
}
```

1.54 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。

1.55 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。

1.56 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。

1.57 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。

1.58 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;

public class ThreadTest {

    /**
     * @param args
     */

    private int j;
    private Lock lock = new ReentrantLock();
```

```
public static void main(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 Subtractor implements Runnable
{

    @Override
    public void run() {
        // TODO Auto-generated method stub
        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 Adder implements Runnable
    {

        @Override
        public void run() {
            // TODO Auto-generated method stub
            while(true)
            {
                /*synchronized (ThreadTest.this) {
                System.out.println("j++=" + j++);
                }*/
                lock.lock();
                try
                {
                    System.out.println("j++=" + j++);
                }finally
                {
                    lock.unlock();
                }
            }
        }

    }
}
```

1.59 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 = new ThreadTest1();
    Inc inc = tt.new Inc();
    Dec dec = tt.new Dec();
    for(int i = 0;i<2;i++){
        Thread t=new Thread(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(int i=0;i<100;i++){
          inc();
        }
```

```
    }
}
class  Dec implements Runnable{
    public void run(){
        for(int i=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(
          new Runnable(){ public void run(){while(true){j.accumulate()}}}
      ).start();
      new Thread(new Runnable(){ public void run(){while(true){j.sub()}}}).start();
  }
}
```

```
}

class JManager
{
  private j = 0;

  public synchronized void subtract()
  {
      j--
  }

  public synchronized void accumulate()
  {
      j++;
  }

}
```

1.60 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：

```
public class ThreadTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
```

```
        new ThreadTest().init();


    }


    public void init()
    {
        final Business business = new Business();
        new Thread(
                new Runnable() {


                    public void run() {
                        for(int i=0;i<50;i++)
                        {
                            business.SubThread(i);
                        }
                    }


                }

        ).start();


        for(int i=0;i<50;i++)
        {
            business.MainThread(i);
        }
    }


    private class Business
    {
        boolean bShouldSub = true;//这里相当于定义了控制该谁执行的一个信号灯
```

```
        public synchronized void MainThread(int i)
        {
            if(bShouldSub)
                try {
                    this.wait();
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch 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 void SubThread(int i)
        {
            if(!bShouldSub)
                try {
                    this.wait();
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
```

```
            for(int j=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;

public class ThreadTest {

    private static boolean bShouldMain = 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 = new String("");

    new Thread(
            new Runnable()
            {
                public void run()
                {
                    for(int i=0;i<50;i++)
                    {
                        synchronized (ThreadTest.class) {
                            if(bShouldMain)
                            {
                                try {
                                    ThreadTest.class.wait();}
                                catch (InterruptedException e) {
                                    e.printStackTrace();
                                }
                            }
                            for(int j=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(int j=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 Lock lock = new ReentrantLock();
    private static Condition subThreadCondition = lock.newCondition();
    private static boolean bBhouldSubThread = false;
    public static void main(String [] args)
    {
        ExecutorService threadPool = Executors.newFixedThreadPool(3);
        threadPool.execute(new Runnable(){
            public void run()
            {
                for(int i=0;i<50;i++)
                {
                    lock.lock();
                    try
                    {
                        if(!bBhouldSubThread)
```

```
                    subThreadCondition.await();
                for(int j=0;j<10;j++)
                {
                    System.out.println(Thread.currentThread().getName() + ",j=" +
j);
                }
                bBhouldSubThread = false;
                subThreadCondition.signal();
            }catch(Exception e)
            {
            }
            finally
            {
                lock.unlock();
            }
        }
    }

});
threadPool.shutdown();
for(int i=0;i<50;i++)
{
        lock.lock();
        try
        {
            if(bBhouldSubThread)
                    subThreadCondition.await();

            for(int j=0;j<10;j++)
            {
```

```
                    System.out.println(Thread.currentThread().getName() + ",j=" + j);
                }
                bBhouldSubThread = true;
                subThreadCondition.signal();
            }catch(Exception e)
            {
            }
            finally
            {
                lock.unlock();
            }
        }
    }
}
```

1.61 介绍 Collection 框架的结构

答：集合框架(Collection Framework)泛指 java.util 包的若干个类和接口。如 Collection，List，ArrayList，LinkedList，Vector(自动增长数组),HashSet,HashMap 等。

集合框架中的类主要封装的是典型的数据结构,如动态数组,链表,堆栈,集合，哈希表等。

集合框架类似编程中经常用到的工具类，使得编码这专注于业务层的实现，不需要从底层实现相关细节—“数据结构的封装”和”典型算法的实现”。

1.62 Collection 框架中实现比较要实现什么接口

SortedSet 和 SortedMap 接口对元素按指定规则排序，SortedMap 是对 key 列进行排序

要实现 comparable 接口,把你的自定义类实现以上接口，实现 compareTo 方法就 OK 了

comparable/comparator

区别：用 Comparable 简单, 只要实现 Comparable 接口的对象直接就成为一个可以比较

的对象,但是需要修改源代码, 用 Comparator 的好处是不需要修改源代码, 而是另外实现一个比较器, 当某个自定义的对象需要作比较的时候,把比较器和对象一起传递过去就可以比大小了, 并且在 Comparator 里面用户可以自己实现复杂的可以通用的逻辑,使其可以匹配一些比较简单的对象,那样就可以节省很多重复劳动了。

1.63 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。

1.64 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

1.65 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。

1.66 List, Set, Map 是否继承自 Collection 接口?

List，Set 是，Map 不是

1.67 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,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。

1.68 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。

1.69 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);

1.70 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。

1.71 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。

1.72 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→List

→Collection

HashSet/TreeSet→Set

Propeties→HashTable

→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。

1.73 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。

1.74 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：

```
public class Parent implements Comparable {
    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 of parent");
        Parent o1 = (Parent)o;
        return age>o1.age?1:age<o1.age?-1:0;
    }

}

public class Child extends Parent {

    public Child(){
        super(3);
    }
    public int compareTo(Object o) {

            // TODO Auto-generated method stub
            System.out.println("method of child");
//          Child o1 = (Child)o;
            return 1;

    }
```

```
}

public class TreeSetTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        TreeSet set = new TreeSet();
        set.add(new Parent(3));
        set.add(new Child());
        set.add(new Parent(4));
        System.out.println(set.size());
    }

}
```

1.75 说出一些常用的类，包，接口，请各举 5 个

要让人家感觉你对java ee开发很熟，所以，不能仅仅只列core java中的那些东西，要多列你在做ssh项目中涉及的那些东西。就写你最近写的那些程序中涉及的那些类。

常用的类：BufferedReader BufferedWriter FileReader FileWirter String Integer java.util.Date，System，Class，List,HashMap

常用的包：java.lang java.io java.util java.sql ,javax.servlet,org.apache.strtuts.action,org.hibernate

常用的接口：Remote List Map Document NodeList ,Servlet,HttpServletRequest,HttpServletResponse,Transaction(Hibernate)、

Session(Hibernate),HttpSession

1.76 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。

1.77 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。

讲解字节流与字符流关系的代码案例：

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.InputStreamReader;
```

```
import java.io.PrintWriter;


public class IOTest {
   public static void main(String[] args) throws Exception {
        String str = "中国人";
        /*FileOutputStream fos    = new FileOutputStream("1.txt");


        fos.write(str.getBytes("UTF-8"));
        fos.close();*/


        /*FileWriter fw = new FileWriter("1.txt");
        fw.write(str);
        fw.close();*/
        PrintWriter pw = new PrintWriter("1.txt","utf-8");
        pw.write(str);
        pw.close();


        /*FileReader fr = new FileReader("1.txt");
        char[] buf = new char[1024];
        int len = fr.read(buf);
        String myStr = new String(buf,0,len);
        System.out.println(myStr);*/
        /*FileInputStream fr = new FileInputStream("1.txt");
        byte[] buf = new byte[1024];
        int len = fr.read(buf);
        String myStr = new String(buf,0,len,"UTF-8");
        System.out.println(myStr);*/
        BufferedReader br = new BufferedReader(
                    new InputStreamReader(
                        new FileInputStream("1.txt"),"UTF-8"
```

```
                    )
                );
        String myStr = br.readLine();
        br.close();
        System.out.println(myStr);
    }

}
```

1.78 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。

1.79 描述一下 JVM 加载 class 文件的原理机制?

Java 语言是一种具有动态性的解释型语言，类（class）只有被加载到 JVM 后才能运行。当运行指定程序时，JVM 会将编译生成的.class 文件按照需求和一定的规则加载到内存中，并组织成为一个完整的 Java 应用程序。这个加载过程是由类加载器完成，具体来说，就是由

ClassLoader 和它的子类来实现的。类加载器本身也是一个类，其实质是把类文件从硬盘读取到内存中。

类的加载方式分为隐式加载和显示加载。隐式加载指的是程序在使用 new 等方式创建对象时，会隐式地调用类的加载器把对应的类加载到 JVM 中。显示加载指的是通过直接调用 class.forName()方法来把所需的类加载到 JVM 中。

任何一个工程项目都是由许多类组成的，当程序启动时，只把需要的类加载到 JVM 中，其他类只有被使用到的时候才会被加载，采用这种方法一方面可以加快加载速度，另一方面可以节约程序运行时对内存的开销。此外，在 Java 语言中，每个类或接口都对应一个.class 文件，这些文件可以被看成是一个个可以被动态加载的单元，因此当只有部分类被修改时，只需要重新编译变化的类即可，而不需要重新编译所有文件，因此加快了编译速度。

在 Java 语言中，类的加载是动态的，它并不会一次性将所有类全部加载后再运行，而是保证程序运行的基础类（例如基类）完全加载到 JVM 中，至于其他类，则在需要的时候才加载。

类加载的主要步骤：

1>装载。根据查找路径找到相应的 class 文件，然后导入。

2>链接。链接又可分为 3 个小步：

1)检查，检查待加载的 class 文件的正确性。

2)准备，给类中的静态变量分配存储空间。

3)解析，将符号引用转换为直接引用（这一步可选）

3>初始化。对静态变量和静态代码块执行初始化工作。

1.80 heap 和 stack 有什么区别。

java 的内存分为两类，一类是**栈内存**，一类是**堆内存**。栈内存是指程序进入一个方法时，会为这个方法单独分配一块私属存储空间，用于存储这个方法内部的局部变量，当这个方法结束时，分配给这个方法的栈会释放，这个栈中的变量也将随之释放。

堆是与栈作用不同的内存，一般用于存放不放在当前方法栈中的那些数据，例如，使用 new 创建的对象都放在堆里，所以，它不会随方法的结束而消失。方法中的局部变量使用 final 修饰后，放在堆中，而不是栈中。

1.81 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。

1.82 垃圾回收的优点和原理。并考虑 2 种回收机制。

Java 语言中一个显著的特点就是引入了垃圾回收机制，使 c++程序员最头疼的内存管理的问题迎刃而解，它使得 Java 程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java 中的对象不再有"作用域"的概念，只有对象的引用才有"作用域"。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清楚和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。

回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

1.83 垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？

对于 GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当 GC 确定一些对象为"不可达"时，GC 就有责任回收这些内存空间。可以。程序员可以手动执行 System.gc()，通知 GC 运行，但是 Java 语言规范并不保证 GC 一定会执行。

1.84 什么时候用 assert?

assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。在实现中，assertion 就是在程序中的一条语句，它对一个 boolean 表达式进行检查，一个正确程序必须保证这个 boolean 表达式的值为 true；如果该值为 false，说明程序已经处于不正确的状态下，assert 将给出警告或退出。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion 检查通常是关闭的。

```
package com.hzit.interview;
public class AssertTest {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int i = 0;
        for(i=0;i<5;i++)
        {
            System.out.println(i);
        }
        //假设程序不小心多了一句--i;
        --i;
        assert i==5;
    }

}
```

1.85 java 中会存在内存泄漏吗，请简单描述。

所谓**内存泄露**就是指一个不再被程序使用的对象或变量一直被占据在内存中。java 中有垃圾回收机制，它可以保证一对象不再被引用的时候，即对象变成了孤儿的时候，对象将

自动被垃圾回收器从内存中清除掉。由于 Java 使用有向图的方式进行垃圾回收管理，可以消除引用循环的问题，例如有两个对象，相互引用，只要它们和根进程不可达的，那么 GC 也是可以回收它们的，例如下面的代码可以看到这种情况的内存回收：

```
package com.huawei.interview;
import java.io.IOException;
public class GarbageTest {
    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        try {
            gcTest();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println("has exited gcTest!");
        System.in.read();
        System.in.read();
        System.out.println("out begin gc!");
        for(int i=0;i<100;i++)
        {
            System.gc();
            System.in.read();
            System.in.read();
        }
    }
    private static void gcTest() throws IOException {
```

```
        System.in.read();
        System.in.read();
        Person p1 = new Person();
        System.in.read();
        System.in.read();
        Person p2 = new Person();
        p1.setMate(p2);
        p2.setMate(p1);
        System.out.println("before exit gctest!");
        System.in.read();
        System.in.read();
        System.gc();
        System.out.println("exit gctest!");
    }
    private static class Person
    {
        byte[] data = new byte[20000000];
        Person mate = null;
        public void setMate(Person other)
        {
            mate = other;
        }
    }
}
```

java 中的内存泄露的情况：长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露，尽管短生命周期对象已经不再需要，但是因为长生命周期对象持有它的引用而导致不能被回收，这就是 java 中内存泄露的发生场景，通俗地说，就是程序员可能创建了一个对象，以后一直不再使用这个对象，这个对象却一直被引用，即这个对象无用但是却无法被垃圾回收器回收的，这就是 java 中可能出现内存泄露的情况，例如，缓存系统，我们加载了一个对象放在缓存中(例如放在一个全局 map 对象中)，然后一直不再使用它，这个对象一

直被缓存引用，但却不再被使用。

检查 java 中的内存泄露，一定要让程序将各种分支情况都完整执行到程序结束，然后看某个对象是否被使用过，如果没有，则才能判定这个对象属于内存泄露。

如果一个外部类的实例对象的方法返回了一个内部类的实例对象,这个内部类对象被长期引用了，即使那个外部类实例对象不再被使用，但由于内部类持久外部类的实例对象，这个外部类对象将不会被垃圾回收，这也会造成内存泄露。

下面内容来自于网上(主要特点就是清空堆栈中的某个元素,并不是彻底把它从数组中拿掉，而是把存储的总数减少，本人写得可以比这个好，在拿掉某个元素时，顺便也让它从数组中消失，将那个元素所在的位置的值设置为 null 即可）：

我实在想不到比那个堆栈更经典的例子了,以致于我还要引用别人的例子，下面的例子不是我想到的，是书上看到的，当然如果没有在书上看到，可能过一段时间我自己也想的到，可是那时我说是我自己想到的也没有人相信的。

```
public class Stack {
private Object[] elements=new Object[10];
private int size = 0;
public void push(Object e){
ensureCapacity();
elements[size++] = e;
}
public Object pop(){
if( size == 0)

throw new EmptyStackException();
return elements[--size];
}
private void ensureCapacity(){
if(elements.length == size){
Object[] oldElements = elements;
elements = new Object[2 * elements.length+1];
```

```
System.arraycopy(oldElements,0, elements, 0, size);
}
}
}
```

上面的原理应该很简单，假如堆栈加了 10 个元素，然后全部弹出来，虽然堆栈是空的，没有我们要的东西，但是这是个对象是无法回收的，这个才符合了内存泄露的两个条件：无用，无法回收。

但是就是存在这样的东西也不一定会导致什么样的后果，如果这个堆栈用的比较少，也就浪费了几个 K 内存而已，反正我们的内存都上 G 了，哪里会有什么影响，再说这个东西很快就会被回收的，有什么关系。下面看两个例子。

例子 1

```
public class Bad{
public static Stack s=Stack();
static{
s.push(new Object());
s.pop(); //这里有一个对象发生内存泄露
s.push(new Object()); //上面的对象可以被回收了，等于是自愈了
}
}
```

因为是 static，就一直存在到程序退出，但是我们也可以看到它有自愈功能，就是说如果你的 Stack 最多有 100 个对象，那么最多也就只有 100 个对象无法被回收其实这个应该很容易理解，Stack 内部持有 100 个引用，最坏的情况就是他们都是无用的，因为我们一旦放新的进取，以前的引用自然消失！

内存泄露的另外一种情况：当一个对象被存储进 HashSet 集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进 HashSet 集合中时的哈希值就不同了，在这种情况下，即使在 contains 方法使用该对象的当前引用作为的参数去 HashSet 集合中检索对象，也将返回找不到对象的结果，这也会导致无法从 HashSet 集合中单独删除当前对象，造成内存泄露。

1.86 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使用，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 java.lang;
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。

1.87 Java 代码查错

题 1.

```
abstract class Name {
   private String name;
   public abstract boolean isStupidName(String name) {}
}
```

大侠们，这有何错误?

答案: 错。abstract method 必须以分号结尾，且不带花括号。

题 2.

```
public class Something {
   void doSomething () {
      private String s = "";
      int l = s.length();
   }
}
```

有错吗?

答案: 错。局部变量前不能放置任何访问修饰符 (private，public，和 protected)。final 可以用来修饰局部变量

(final 如同 abstract 和 strictfp，都是非访问修饰符，strictfp 只能修饰 class 和 method 而非 variable)。

题 3.

```
abstract class Something {
   private abstract String doSomething ();
}
```

这好像没什么错吧?

答案: 错。abstract 的 methods 不能以 private 修饰。abstract 的 methods 就是让子类 implement(实现)具体细节的，怎么可以用 private 把 abstract

method 封锁起来呢? (同理，abstract method 前不能加 final)。

题 4.

```
public class Something {
   public int addOne(final int x) {
      return ++x;
   }
}
```

这个比较明显。

答案: 错。int x 被修饰成 final，意味着 x 不能在 addOne method 中被修改。

题 5.

```
public class Something {
   public static void main(String[] args) {
      Other o = new Other();
      new Something().addOne(o);
   }
   public void addOne(final Other o) {
      o.i++;
   }
}
class Other {
   public int i;
}
```

和上面的很相似，都是关于 final 的问题，这有错吗?

答案: 正确。在 addOne method 中，参数 o 被修饰成 final。如果在 addOne method 里我们修改了 o 的 reference

(比如: o = new Other();)，那么如同上例这题也是错的。但这里修改的是 o 的 member vairable (成员变量)，而 o 的 reference 并没有改变。

题 6.

```
class Something {
    int i;
    public void doSomething() {
       System.out.println("i = " + i);
```

```
    }
}
```

有什么错呢？看不出来啊。

答案：正确。输出的是"i = 0"。int i 属於 instant variable (实例变量，或叫成员变量)。instant variable 有 default value。int 的 default value 是 0。

题 7.

```
class Something {
    final int i;
    public void doSomething() {
        System.out.println("i = " + i);
    }
}
```

和上面一题只有一个地方不同，就是多了一个 final。这难道就错了吗？

答案：错。final int i 是个 final 的 instant variable (实例变量，或叫成员变量)。final 的 instant variable 没有 default value，必须在 constructor (构造器)结束之前被赋予一个明确的值。可以修改为"final int i = 0;"。

题 8.

```
public class Something {
     public static void main(String[] args) {
        Something s = new Something();
        System.out.println("s.doSomething() returns " + doSomething());
    }
    public String doSomething() {
        return "Do something ...";
    }
}
```

看上去很完美。

答案：错。看上去在 main 里 call doSomething 没有什么问题，毕竟两个 methods 都在同一个 class 里。但仔细看，main 是 static 的。static method 不能直接 call non-static methods。可改成"System.out.println("s.doSomething() returns " + s.doSomething());"。同理，static method 不

能访问 non-static instant variable。

题 9.

此处，Something 类的文件名叫 OtherThing.java

```
class Something {
    private static void main(String[] something_to_do) {
        System.out.println("Do something ...");
    }
}
```

这个好像很明显。

答案: 正确。从来没有人说过 Java 的 Class 名字必须和其文件名相同。但 public class 的名字必须和文件名相同。

题 10.

```
interface   A{
   int x = 0;
}
class B{
   int x =1;
}
class C extends B implements A {
   public void pX(){
      System.out.println(x);
   }
   public static void main(String[] args) {
      new C().pX();
   }
}
```

答案：错误。在编译时会发生错误(错误描述不同的 JVM 有不同的信息，意思就是未明确的 x 调用，两个 x 都匹配（就象在同时 import java.util 和 java.sql 两个包时直接声明 Date 一样）。对于父类的变量,可以用 super.x 来明确，而接口的属性默认隐含为 public static final.所以可以通过 A.x 来明确。

题 11.

```
interface Playable {
    void play();
}
interface Bounceable {
    void play();
}
interface Rollable extends Playable, Bounceable {
    Ball ball = new Ball("PingPang");
}
class Ball implements Rollable {
    private String name;
    public String getName() {
        return name;
    }
    public Ball(String name) {
        this.name = name;
    }
   public void play() {
        ball = new Ball("Football");
        System.out.println(ball.getName());
    }
}
```

这个错误不容易发现。

答案: 错。"interface Rollable extends Playable, Bounceable"没有问题。interface 可继承多个 interfaces，所以这里没错。问题出在 interface Rollable 里的"Ball ball = new Ball("PingPang");"。任何在 interface 里声明的 interface variable (接口变量，也可称成员变量)，默认为 public static final。也就是说"Ball ball = new Ball("PingPang");"实际上是"public static final Ball ball = new Ball("PingPang");"。在 Ball 类的 Play()方法中，"ball = new Ball("Football");"改变了 ball 的 reference，而这里的 ball 来自 Rollable interface，Rollable interface 里的 ball 是 public static final

的，final 的 object 是不能被改变 reference 的。因此编译器将在"ball = new Ball("Football");"这里显示有错。

二、算法与编程 1

2.1 判断身份证：要么是 15 位，要么是 18 位，最后一位可以为字母，并写程序提出其中的年月日。

答：我们可以用正则表达式来定义复杂的字符串格式，(\d{17}[0-9a-zA-Z]|\d{14}[0-9a-zA-Z])可以用来判断是否为合法的 15 位或 18 位身份证号码。

因为 15 位和 18 位的身份证号码都是从 7 位到第 12 位为身份证为日期类型。这样我们可以设计出更精确的正则模式，使身份证号的日期合法，这样我们的正则模式可以进一步将日期部分的正则修改为[12][0-9]{3}[01][0-9][123][0-9]，当然可以更精确的设置日期。在 jdk 的 java.util.Regex 包中有实现正则的类,Pattern 和 Matcher。以下是实现代码：

```
import java.util.regex.Matcher;
import java.util.regex.P attern;
public class RegexTest {
    /**
     * @param args
```

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

    // 测试是否为合法的身份证号码
    String[] strs = { "130681198712092019", "13068119871209201x",
            "13068119871209201", "123456789012345", "12345678901234x",
            "1234567890123" };
    Pattern p1 = Pattern.compile("(\\d{17}[0-9a-zA-Z]|\\d{14}[0-9a-zA-Z])");
    for (int i = 0; i < strs.length; i++) {
        Matcher matcher = p1.matcher(strs[i]);
        System.out.println(strs[i] + ":" + matcher.matches());
    }
    Pattern p2 = Pattern.compile("\\d{6}(\\d{8}).*"); // 用于提取出生日字符串
    Pattern p3 = Pattern.compile("(\\d{4})(\\d{2})(\\d{2})");// 用于将生日字符串进行分解为年月日
    for (int i = 0; i < strs.length; i++) {
        Matcher matcher = p2.matcher(strs[i]);
        boolean b = matcher.find();
        if (b) {
            String s = matcher.group(1);
            Matcher matcher2 = p3.matcher(s);
            if (matcher2.find()) {
                System.out
                        .println("生日为" + matcher2.group(1) + "年"
                                + matcher2.group(2) + "月"
                                + matcher2.group(3) + "日");
            }
        }

    }
```

```
    }
}
```

2.2 编写一个程序，将 a.txt 文件中的单词与 b.txt 文件中的单词交替合并到 c.txt 文件中，a.txt 文件中的单词用回车符分隔，b.txt 文件中用回车或空格进行分隔。

答：

```
package cn.itcast;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
public class MainClass{
    public static void main(String[] args) throws Exception{
        FileManager a = new FileManager("a.txt",new char[]{'\n'});
        FileManager b = new FileManager("b.txt",new char[]{'\n',' '});
        FileWriter c = new FileWriter("c.txt");
        String aWord = null;
        String bWord = null;
        while((aWord = a.nextWord()) !=null ){
            c.write(aWord + "\n");
            bWord = b.nextWord();
            if(bWord != null)
                c.write(bWord + "\n");
        }
        while((bWord = b.nextWord()) != null){
            c.write(bWord + "\n");
        }
        c.close();
    }
```

```
}
class FileManager{
    String[] words = null;
    int pos = 0;
    public FileManager(String filename,char[] seperators) throws Exception{
        File f = new File(filename);
        FileReader reader = new FileReader(f);
        char[] buf = new char[(int)f.length()];
        int len = reader.read(buf);
        String results = new String(buf,0,len);
        String regex = null;
        if(seperators.length >1 ){
            regex = "" + seperators[0] + "|" + seperators[1];
        }else{
            regex = "" + seperators[0];
        }
        words = results.split(regex);
    }

    public String nextWord(){
        if(pos == words.length)
            return null;
        return words[pos++];
    }

}
```

2.3 编写一个程序，将 d:\java 目录下的所有.java 文件复制到 d:\jad 目录下，并将原来文件的扩展名从.java 改为.jad。

（大家正在做上面这道题，网上迟到的朋友也请做做这道题，找工作必须能编写这些简单问题的代码！）

答：**listFiles 方法接受一个 FileFilter 对象，这个 FileFilter 对象就是过虑的策略对象，不同的人提供不同的 FileFilter 实现，即提供了不同的过滤策略。**

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FilenameFilter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
public class Jad2Java {
	public static void main(String[] args) throws Exception {
		File srcDir = new File("java");
		if(!(srcDir.exists() && srcDir.isDirectory()))
			throw new Exception("目录不存在");
		File[] files = srcDir.listFiles(
			new FilenameFilter(){
				public boolean accept(File dir, String name) {
					return name.endsWith(".java");
				}

			}
		);

		System.out.println(files.length);
		File destDir = new File("jad");
```

```
            if(!destDir.exists()) destDir.mkdir();
            for(File f :files){
                FileInputStream    fis = new FileInputStream(f);
                String destFileName = f.getName().replaceAll("\\.java$", ".jad");
                FileOutputStream fos = new FileOutputStream(new File(destDir,destFileName));
                copy(fis,fos);
                fis.close();
                fos.close();
            }
        }
        private static void copy(InputStream ips,OutputStream ops) throws Exception{
            int len = 0;
            byte[] buf = new byte[1024];
            while((len = ips.read(buf)) != -1){
                ops.write(buf,0,len);
            }
        }
}
```

由本题总结的思想及策略模式的解析：

1.class jad2java{

1. 得到某个目录下的所有的 java 文件集合

1.1 得到目录 File srcDir = new File("d:\\java");

1.2 得到目录下的所有 java 文件：File[] files = srcDir.listFiles(new MyFileFilter());

1.3 只想得到.java 的文件： class MyFileFilter implememyts FileFilter{

```
public boolean accept(File pathname){
    return pathname.getName().endsWith(".java")
}
```

}

2.将每个文件复制到另外一个目录，并改扩展名

2.1 得到目标目录，如果目标目录不存在，则创建之

2.2 根据源文件名得到目标文件名，注意要用正则表达式，注意.的转义。

2.3 根据表示目录的 File 和目标文件名的字符串，得到表示目标文件的 File。

//要在硬盘中准确地创建出一个文件，需要知道文件名和文件的目录。

2.4 将源文件的流拷贝成目标文件流，拷贝方法独立成为一个方法，方法的参数采用抽象流的形式。

//方法接受的参数类型尽量面向父类，越抽象越好，这样适应面更宽广。

}

分析 listFiles 方法内部的策略模式实现原理

```
File[] listFiles(FileFilter filter){
    File[] files = listFiles();
    //Arraylist acceptedFilesList = new ArrayList();
    File[] acceptedFiles = new File[files.length];
    int pos = 0;
    for(File file: files){
        boolean accepted = filter.accept(file);
        if(accepted){
            //acceptedFilesList.add(file);
            acceptedFiles[pos++] = file;
        }
    }
    Arrays.copyOf(acceptedFiles,pos);
    //return (File[])accpetedFilesList.toArray();
}
```

2.4 编写一个截取字符串的函数，输入为一个字符串和字节数，输出为按字节截取的字符串，但要保证汉字不被截取半个，如“我 ABC”，4，应该截取“我 AB”，输入“我 ABC 汉 DEF”，6，应该输出“我 ABC”，而不是“我 ABC+汉的半个”。

答：

首先要了解中文字符有多种编码及各种编码的特征。

假设 n 为要截取的字节数。

```
public static void main(String[] args) throws Exception{
        String str = "我 a 爱中华 abc 我爱传智 def";
        String str = "我 ABC 汉";
        int num = trimGBK(str.getBytes("GBK"),5);
        System.out.println(str.substring(0,num) );
}

public static int    trimGBK(byte[] buf,int n){
        int num = 0;
        boolean bChineseFirstHalf = false;
        for(int i=0;i<n;i++)
        {
            if(buf[i]<0 && !bChineseFirstHalf){
                bChineseFirstHalf = true;
            }else{
                num++;
                bChineseFirstHalf = false;
            }
        }
        return num;
}
```

2.5 有一个字符串，其中包含中文字符、英文字符和数字字符，请统计和打印出各个字符的个数。

答：哈哈，其实包含中文字符、英文字符、数字字符原来是出题者放的烟雾弹。

```
String content = “中国 aadf 的 111 萨 bbb 菲的 zz 萨菲”;
HashMap map = new HashMap();
for(int i=0;i<content.length;i++)
{
```

```
    char c = content.charAt(i);
    Integer num = map.get(c);
    if(num == null)
        num = 1;
    else
        num = num + 1;
    map.put(c,num);
}
for(Map.EntrySet entry : map)
{
    system.out.println(entry.getkey() + ":" + entry.getValue());
}
```

估计是当初面试的那个学员表述不清楚，问题很可能是：

如果一串字符如"aaaabbc中国1512"要分别统计英文字符的数量，中文字符的数量，和数字字符的数量，假设字符中没有中文字符、英文字符、数字字符之外的其他特殊字符。

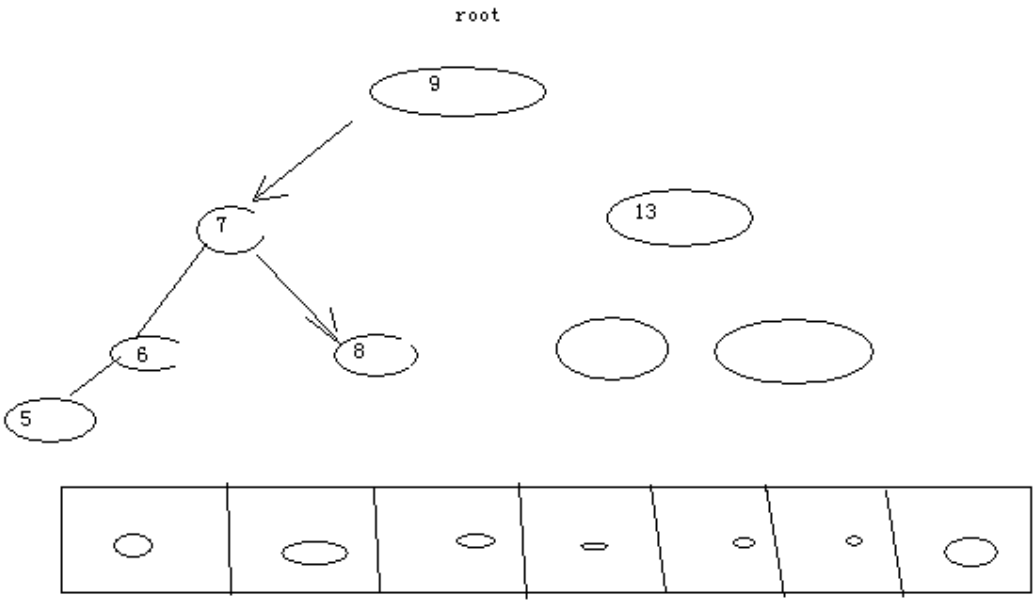
```
int engishCount;
int chineseCount;
int digitCount;
for(int i=0;i<str.length;i++)
{
    char ch = str.charAt(i);
    if(ch>='0' && ch<='9')
    {
        digitCount++
    }
    else if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))
    {
        engishCount++;
    }
    else
```

```
    {
        chineseCount++;
    }
}
System.out.println(.............);
```

2.6 说明生活中遇到的二叉树，用 java 实现二叉树

这是组合设计模式。

我有很多个(假设 10 万个)数据要保存起来，以后还需要从保存的这些数据中检索是否存在某个数据，（我想说出二叉树的好处，该怎么说呢？那就是说别人的缺点），假如存在数组中，那么，碰巧要找的数字位于 99999 那个地方，那查找的速度将很慢，因为要从第 1 个依次往后取，取出来后进行比较。平衡二叉树（构建平衡二叉树需要先排序，我们这里就不作考虑了）可以很好地解决这个问题，但二叉树的遍历（前序，中序，后序）效率要比数组低很多，原理如下图：



代码如下：

```
package com.huawei.interview;
public class Node {
    public int value;
    public Node left;
```

```
public Node right;
public void store(int value)
{
    if(value<this.value)
    {
        if(left == null)
        {
            left = new Node();
            left.value=value;
        }
        else
        {
            left.store(value);
        }
    }
    else if(value>this.value)
    {
        if(right == null)
        {
            right = new Node();
            right.value=value;
        }
        else
        {
            right.store(value);
        }
    }
}
public boolean find(int value)
{
```

```
        System.out.println("happen " + this.value);
        if(value == this.value)
        {
            return true;
        }
        else if(value>this.value)
        {
            if(right == null) return false;
            return right.find(value);
        }else
        {
            if(left == null) return false;
            return left.find(value);
        }
    }
    public  void preList()
    {
        System.out.print(this.value + ",");
        if(left!=null) left.preList();
        if(right!=null) right.preList();
    }
    public void middleList()
    {
        if(left!=null) left.preList();
        System.out.print(this.value + ",");
        if(right!=null) right.preList();
    }
    public void afterList()
    {
        if(left!=null) left.preList();
```

```
        if(right!=null) right.preList();
        System.out.print(this.value + ",");
    }
    public static void main(String [] args)
    {
        int [] data = new int[20];
        for(int i=0;i<data.length;i++)
        {
            data[i] = (int)(Math.random()*100) + 1;
            System.out.print(data[i] + ",");
        }
        System.out.println();
        Node root = new Node();
        root.value = data[0];
        for(int i=1;i<data.length;i++)
        {
            root.store(data[i]);
        }
        root.find(data[19]);
        root.preList();
        System.out.println();
        root.middleList();
        System.out.println();
        root.afterList();
    }
}
```

----------又一次临场写的代码----------

```
import java.util.Arrays;
import java.util.Iterator;
public class Node {
```

```
    private Node left;
    private Node right;
    private int value;
    //private int num;
    public Node(int value){
        this.value = value;
    }
    public void add(int value){
        if(value > this.value)
        {
            if(right != null)
                right.add(value);
            else
            {
                Node node = new Node(value);
                right = node;
            }
        }
        else{
            if(left != null)
                left.add(value);
            else
            {
                Node node = new Node(value);
                left = node;
            }
        }
    }
    public boolean find(int value){
        if(value == this.value) return true;
```

```
        else if(value > this.value){
            if(right == null) return false;
            else return right.find(value);
        }else{
            if(left == null) return false;
            else return left.find(value);
        }
    }
    public void display(){
        System.out.println(value);
        if(left != null) left.display();
        if(right != null) right.display();
    }
    /*public Iterator iterator(){
    }*/
    public static void main(String[] args){
        int[] values = new int[8];
        for(int i=0;i<8;i++){
            int num = (int)(Math.random() * 15);
            //System.out.println(num);
            //if(Arrays.binarySearch(values, num)<0)
            if(!contains(values,num))
                values[i] = num;
            else
                i--;
        }
        System.out.println(Arrays.toString(values));
        Node root  = new Node(values[0]);
        for(int i=1;i<values.length;i++){
            root.add(values[i]);
```

```
        }
        System.out.println(root.find(13));
        root.display();
    }
    public static boolean contains(int [] arr, int value){
        int i = 0;
        for(;i<arr.length;i++){
            if(arr[i] == value) return true;
        }
        return false;
    }
}
```

2.7 从类似如下的文本文件中读取出所有的姓名，并打印出重复的姓名和重复的次数，并按重复次数排序：

1,张三,28

2,李四,35

3,张三,28

4,王五,35

5,张三,28

6,李四,35

7,赵六,28

8,田七,35

程序代码如下（答题要博得用人单位的喜欢，包名用该公司，面试前就提前查好该公司的网址，如果查不到，现场问也是可以的。还要加上实现思路的注释）：

```
package com.huawei.interview;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
```

```
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.TreeSet;
public class GetNameTest {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        //InputStream ips =
GetNameTest.class.getResourceAsStream("/com/huawei/interview/info.txt
");
        //用上一行注释的代码和下一行的代码都可以，因为info.txt与GetNameTest类在
同一包下面，所以，可以用下面的相对路径形式

        Map results = new HashMap();
        InputStream ips =
GetNameTest.class.getResourceAsStream("info.txt");
        BufferedReader in = new BufferedReader(new
InputStreamReader(ips));
        String line = null;
        try {
            while((line=in.readLine())!=null)
            {
                dealLine(line,results);
            }
            sortResults(results);
        } catch (IOException e) {
```

```
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    static class User
    {
        public  String name;
        public Integer value;
        public User(String name,Integer value)
        {
            this.name = name;
            this.value = value;
        }
        @Override
        public boolean equals(Object obj) {
            // TODO Auto-generated method stub

            //下面的代码没有执行，说明往treeset中增加数据时，不会使用到equals方
法。
            boolean result = super.equals(obj);
            System.out.println(result);
            return result;
        }
    }
    private static void sortResults(Map results) {
        // TODO Auto-generated method stub
        TreeSet sortedResults = new TreeSet(
                new Comparator(){
                    public int compare(Object o1, Object o2) {
                        // TODO Auto-generated method stub
```

```
                    User user1 = (User)o1;
                    User user2 = (User)o2;
                    /*如果compareTo返回结果0，则认为两个对象相等，新的对象不
会增加到集合中去
                     * 所以，不能直接用下面的代码，否则，那些个数相同的其他姓
名就打印不出来。
                     * */

                    //return user1.value-user2.value;
                    //return
user1.value<user2.value?-1:user1.value==user2.value?0:1;
                    if(user1.value<user2.value) {
                        return -1;
                    }else if(user1.value>user2.value) {
                        return 1;
                    }else{
                        return user1.name.compareTo(user2.name);
                    }
                }
            }
        );
        Iterator iterator = results.keySet().iterator();
        while(iterator.hasNext())
        {
            String name = (String)iterator.next();
            Integer value = (Integer)results.get(name);
            if(value > 1)
            {
                sortedResults.add(new User(name,value));
            }
```

```
        }
        printResults(sortedResults);
    }
    private static void printResults(TreeSet sortedResults)
    {
        Iterator iterator  = sortedResults.iterator();
        while(iterator.hasNext())
        {
            User user = (User)iterator.next();
            System.out.println(user.name + ":" + user.value);
        }
    }
    public static void dealLine(String line,Map map)
    {
        if(!"".equals(line.trim()))
        {
            String [] results = line.split(",");
            if(results.length == 3)
            {
                String name = results[1];
                Integer value = (Integer)map.get(name);
                if(value == null) value = 0;
                map.put(name,value + 1);
            }
        }
    }
}
```

2.8 写一个 Singleton 出来。

第一种：饱汉模式

```
public class SingleTon {
    private SingleTon(){
        }
    //实例化放在静态代码块里可提高程序的执行效率，但也可能更占用空间
    private final static SingleTon instance = new SingleTon();
    public static SingleTon getInstance(){
        return instance;
    }
}
```

第二种：饥汉模式

```
public class SingleTon {
    private SingleTon(){}
    private static instance = null;//new SingleTon();
    public static synchronized SingleTon getInstance(){
        if(instance == null)
            instance = new SingleTon();
        return instance;
    }
}
```

第三种：用枚举

```
public enum SingleTon{ONE;}
```

第三：更实际的应用（在什么情况用单例）

```
public class SequenceGenerator{
    //下面是该类自身的业务功能代码
    private int count = 0;
    public synchronized int getSequence(){
        ++count;
```

```
    }
    //下面是把该类变成单例的代码
    private SequenceGenerator(){}
    private final static instance = new SequenceGenerator();
    public static SingleTon getInstance(){
        return instance;
    }
}
```

第四：

```
    public class MemoryDao
    {
    private HashMap map = new HashMap();
    public void add(Student stu1){
        map.put(SequenceGenerator.getInstance().getSequence(),stu1);
    }
   //把MemoryDao变成单例
  }
```

Singleton 模式主要作用是保证在 Java 应用程序中，一个类 Class 只有一个实例存在。

一般 Singleton 模式通常有几种种形式:

第一种形式: 定义一个类，它的构造函数为 private 的，它有一个 static 的 private 的该类变量，在类初始化时实例话，通过一个 public 的 getInstance 方法获取对它的引用,继而调用其中的方法。

```
public class Singleton {
private Singleton(){}
        //在自己内部定义自己一个实例，是不是很奇怪？
        //注意这是 private 只供内部调用
        private static Singleton instance = new Singleton();
        //这里提供了一个供外部访问本 class 的静态方法，可以直接访问
        public static Singleton getInstance() {
            return instance;
```

```
        }
    }
```

第二种形式:

```
public class Singleton {
    private static Singleton instance = null;
    public static synchronized Singleton getInstance() {
    //这个方法比上面有所改进，不用每次都进行生成对象，只是第一次
    //使用时生成实例，提高了效率!
    if (instance==null)
        instance＝new Singleton();
        return instance;
  }
}
```

其他形式:

定义一个类，它的构造函数为 private 的，所有方法为 static 的。

一般认为第一种形式要更加安全些

2.9 递归算法题 1

一个整数，大于 0，不用循环和本地变量，按照 n，2n，4n，8n 的顺序递增，当值大于 5000 时，把值按照指定顺序输出来。

例：n=1237

则输出为：

1237，

2474，

4948，

9896，

9896，

4948，

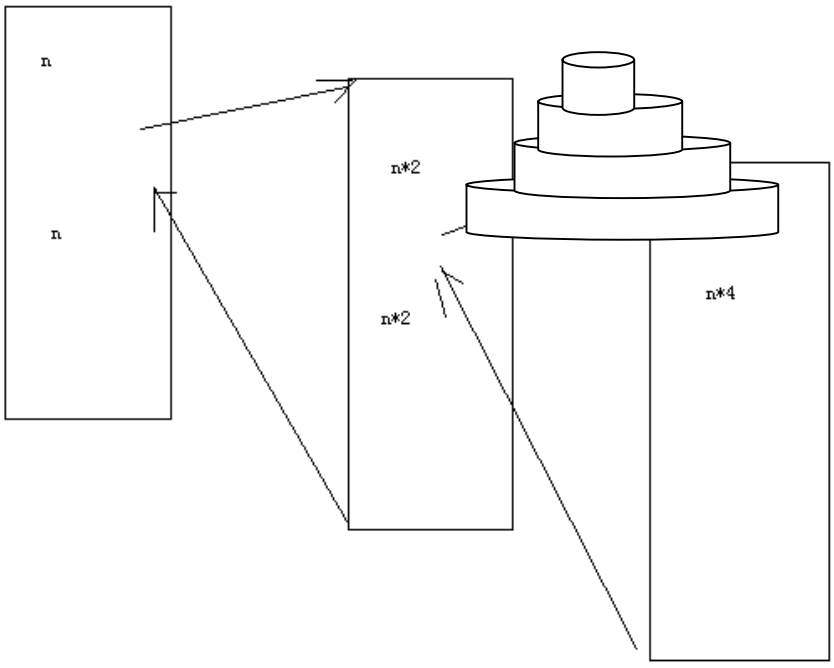
2474，

1237，

提示：写程序时，先致谢按递增方式的代码，写好递增的以后，再增加考虑递减部分。

```
public static void doubleNum(int n)
{
    System.out.println(n);
    if(n<=5000)
        doubleNum(n*2);
    System.out.println(n);
}
```

Gaibaota(N) = Gaibaota(N-1) + n



2.10 递归算法题 2

第 1 个人 10，第 2 个比第 1 个人大 2 岁，依次递推，请用递归方式计算出第 8 个人多大?

```
package cn.itcast;
import java.util.Date;
```

```
public class A1 {
    public static void main(String [] args)
    {
        System.out.println(computeAge(8));
    }

    public static int computeAge(int n)
    {
        if(n==1) return 10;
        return computeAge(n-1) + 2;
    }
}
    public static void toBinary(int n,StringBuffer result)
    {
        if(n/2 != 0)
            toBinary(n/2,result);
        result.append(n%2);
    }
```

2.11 排序都有哪几种方法？请列举。用 JAVA 实现一个快速排序。

本人只研究过**冒泡排序、选择排序和快速排序**，下面是快速排序的代码：

```
public class QuickSort {
/**
* 快速排序
* @param strDate
* @param left
* @param right
*/
public void quickSort(String[] strDate,int left,int right){
```

```
String  middle,tempDate;
int  i,j;
i=left;
j=right;
middle=strDate[(i+j)/2];
do{
while(strDate[i].compareTo(middle)<0&&  i<right)
i++;  //找出左边比中间值大的数
while(strDate[j].compareTo(middle)>0&&  j>left)
j--;  //找出右边比中间值小的数
if(i<=j){  //将左边大的数和右边小的数进行替换
tempDate=strDate[i];
strDate[i]=strDate[j];
strDate[j]=tempDate;
i++;
j--;
}
}while(i<=j);  //当两者交错时停止

if(i<right){
quickSort(strDate,i,right);//从
}
if(j>left){
quickSort(strDate,left,j);
}
}
/**
   *  @param  args
   */
public  static  void  main(String[]  args){
```

```
String[] strVoid=new String[]{"11","66","22","0","55","22","0","32"};
QuickSort sort=new QuickSort();
sort.quickSort(strVoid,0,strVoid.length-1);
for(int i=0;i<strVoid.length;i++){
System.out.println(strVoid[i]+" ");
}
}
}
```

2.12 有数组 a[n]，用 java 代码将数组元素顺序颠倒

```
//用下面的也可以
//for(int i=0,int j=a.length-1;i<j;i++,j--) 是否等效于 for(int i=0;i<a.length/2;i++)呢?
import java.util.Arrays;
public class SwapDemo{
    public static void main(String[] args){
        int [] a = new int[]{
                        (int)(Math.random() * 1000),
                        (int)(Math.random() * 1000),
                        (int)(Math.random() * 1000),
                        (int)(Math.random() * 1000),
                        (int)(Math.random() * 1000)

        };
        System.out.println(a);
        System.out.println(Arrays.toString(a));
        swap(a);
        System.out.println(Arrays.toString(a));
    }
    public static void swap(int a[]){
```

```
        int len = a.length;
        for(int i=0;i<len/2;i++){
            int tmp = a[i];
            a[i] = a[len-1-i];
            a[len-1-i] = tmp;
        }
    }
}
```

2.13 金额转换，阿拉伯数字的金额转换成中国传统的形式如：（￥1011）—>（一千零一拾一元整）输出。

去零的代码：

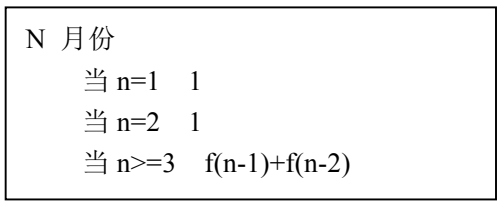
```
    return sb.reverse().toString().replaceAll("零[拾佰仟]","零").replaceAll("零+万","万").replaceAll("零+元","元").replaceAll("零+","零");
public class RenMingBi {
    /**
     * @param args add by zxx ,Nov 29, 2008
     */
    private static final char[] data = new char[]{ '零','壹','贰','叁','肆','伍','陆','柒','捌','玖'};
    private static final char[] units = new char[]{'元','拾','佰','仟','万','拾','佰','仟','亿'};
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(
                convert(135689123));
    }
    public static String convert(int money)
    {
        StringBuffer sbf = new StringBuffer();
        int unit = 0;
```

```
        while(money!=0)
        {
            sbf.insert(0,units[unit++]);
            int number = money%10;
            sbf.insert(0, data[number]);
            money /= 10;
        }
        return sbf.toString();
    }
}
```

三、算法与编程 2

3.1 题目：古典问题：有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第四个月后每个月又生一对兔子，假如兔子都不死，问第月的兔子总数为多少？

1. 程序分析：　　兔子的规律为数列 1,1,2,3,5,8,13,21....




```
public class exp1 {
  public static void main(String args[]) {
    int i = 0;
    for (i = 1; i <= 20; i++)
      System.out.println(f(i));    }
  public static int f(int x) {
    if (x == 1 || x == 2)
      return 1;      else
      return f(x - 1) + f(x - 2);
  } }或  public class exp1 {
```

```
  public static void main(String args[]) {
    int i = 0;
    math mymath = new math();
    for (i = 1; i <= 20; i++)        System.out.println(mymath.f(i));
  }
}
class math {
  public int f(int x) {
    if (x == 1 || x == 2)
      return 1;
    else        return f(x - 1) + f(x - 2);
  }
}
```

3.2 题目：判断 101-200 之间有多少个素数，并输出所有素数。

1.程序分析：判断素数的方法：用一个数分别去除 2 到 sqrt(这个数)，如果能被整除，则表明此数不是素数，反之是素数。

```
public class exp2 {
  public static void main(String args[]) {
    int i = 0;      math mymath = new math();
    for (i = 2; i <= 200; i++)
      if (mymath.iszhishu(i) == true)
        System.out.println(i);
  }
}
class math {
  public int f(int x) {
    if (x == 1 || x == 2)
      return 1;      else
```

```
    return f(x - 1) + f(x - 2);
  }
  public boolean iszhishu(int x) {      for (int i = 2; i <= x / 2; i++)
    if (x % 2 == 0)
      return false;
   return true;
  }
  }
```

3.3 题目：打印出所有的 "水仙花数 "，所谓 "水仙花数 "是指一个三位数，其各位数字立方和等于该数本身。例 如：153 是一个 "水仙花数 "，因为 153=1 的三次方＋5 的三次方＋3 的三次方。在 2000 以内的数字

1.程序分析：利用 for 循环控制 100-999 个数，每个数分解出个位，十位，百位。

```
public class exp2 {
  public static void main(String args[]) {
    int i = 0;
    math mymath = new math();
    for (i = 100; i <= 999; i++)        if (mymath.shuixianhua(i) == true)
      System.out.println(i);
  }
}
class math {
  public int f(int x) {
    if (x == 1 || x == 2)
      return 1;
    else       return f(x - 1) + f(x - 2);
  }
  public boolean iszhishu(int x) {
    for (int i = 2; i <= x / 2; i++)       if (x % 2 == 0)
```

```
      return false;
   return true;
  }
  public boolean shuixianhua(int x) {
   int i = 0, j = 0, k = 0;
   i = x / 100;     j = (x % 100) / 10;
   k = x % 10;
   if (x == i * i * i + j * j * j + k * k * k)
    return true;
   else       return false;
  }
}
```

3.4 题目：将一个正整数分解质因数。例如：输入 90,打印出 90=2*3*3*5。

程序分析：对 n 进行分解质因数，应先找到一个最小的质数 k，然后按下述步骤完成：

(1)如果这个质数恰等于 n，则说明分解质因数的过程已经结束，打印出即可。
(2)如果 n <> k，但 n 能被 k 整除，则应打印出 k 的值，并用 n 除以 k 的商,作为新的正整数你,重复执行第一步。

(3)如果 n 不能被 k 整除，则用 k+1 作为 k 的值,重复执行第一步。

```
public class exp2 {
  public exp2() {
  }
  public void fengjie(int n) {     for (int i = 2; i <= n / 2; i++) {
    if (n % i == 0) {
      System.out.print(i + "*");
      fengjie(n / i);
    }    }
```

```
   System.out.print(n);      System.exit(0);// /不能少这句，否则结果会出错
  }
  public static void main(String[] args) {
   String str = "";
   exp2 c = new exp2();      str = javax.swing.JOptionPane.showInputDialog("请输入 N 的值（输
入 exit 退出）：");
   int N;      N = 0;
   try {
     N = Integer.parseInt(str);
   } catch (NumberFormatException e) {
     e.printStackTrace();      }      System.out.print(N + "分解质因数：" + N + "=");
   c.fengjie(N);
  }
}
```

3.5 题目：利用条件运算符的嵌套来完成此题：学习成绩> =90 分的同学用 A 表示，60-89 分之间的用 B 表示，60 分以下的用 C 表示。

1.程序分析：(a> b)?a:b 这是条件运算符的基本例子。

```
import javax.swing.*;
public class ex5 {
  public static void main(String[] args) {
   String str = "";      str = JOptionPane.showInputDialog("请输入 N 的值（输入 exit 退出）：");
   int N;
   N = 0;
   try {
     N = Integer.parseInt(str);      } catch (NumberFormatException e) {
     e.printStackTrace();
   }
   str = (N > 90 ? "A" : (N > 60 ? "B" : "C"));
```

```
    System.out.println(str);    }
}
```

3.6 题目：输入两个正整数 m 和 n，求其最大公约数和最小公倍数。

1.程序分析：利用辗除法。　　最大公约数：

```
public class CommonDivisor {
  public static void main(String args[]) {      commonDivisor(24, 32);
  }
  static int commonDivisor(int M, int N) {
    if (N < 0 || M < 0) {        System.out.println("ERROR!");
      return -1;
    }
    if (N == 0) {
      System.out.println("the biggest common divisor is :" + M);        return M;
    }
    return commonDivisor(N, M % N);
  }
} 最小公倍数和最大公约数：  import java.util.Scanner;
public class CandC {    // 下面的方法是求出最大公约数
  public static int gcd(int m, int n) {
    while (true) {
      if ((m = m % n) == 0)          return n;
      if ((n = n % m) == 0)
        return m;
    }
  }
  public static void main(String args[]) throws Exception {     // 取得输入值
    // Scanner chin = new Scanner(System.in);
    // int a = chin.nextInt(), b = chin.nextInt();      int a = 23;
```

```
   int b = 32;
   int c = gcd(a, b);      System.out.println("最小公倍数：" + a * b / c + "\n 最大公约数：" + c);
 } }
```

3.7 题目：输入一行字符，分别统计出其中英文字母、空格、数字和其它字符的个数。

1.程序分析：利用 while 语句,条件为输入的字符不为 '\n '.

```
import java.util.Scanner;
public class ex7 {
  public static void main(String args[]) {     System.out.println("请输入字符串：");     Scanner scan = new Scanner(System.in);
    String str = scan.next();
    String E1 = "[\u4e00-\u9fa5]";
    String E2 = "[a-zA-Z]";
    int countH = 0;      int countE = 0;
    char[] arrChar = str.toCharArray();
    String[] arrStr = new String[arrChar.length];
    for (int i = 0; i < arrChar.length; i++) {
      arrStr[i] = String.valueOf(arrChar[i]);      }
    for (String i : arrStr) {
      if (i.matches(E1)) {
        countH++;
      }       if (i.matches(E2)) {
        countE++;
      }
    }      System.out.println("汉字的个数" + countH);      System.out.println("字母的个数" + countE);
  }
}
```

3.8 题目：求 s=a+aa+aaa+aaaa+aa...a 的值，其中 a 是一个数字。例如 2+22+222+2222+22222(此时 共有 5 个数相加)，几个数相加有键盘控制。

1.程序分析：关键是计算出每一项的值。

```
import java.io.*;
public class Sumloop {
 public static void main(String[] args) throws IOException {
  int s = 0;     String output = "";
  BufferedReader stadin = new BufferedReader(new InputStreamReader(
     System.in));     System.out.println("请输入 a 的值");
  String input = stadin.readLine();     for (int i = 1; i <= Integer.parseInt(input); i++) {
   output += input;
   int a = Integer.parseInt(output);
   s += a;
  }     System.out.println(s);
 }
} 另解: import java.io.*;
public class Sumloop {
 public static void main(String[] args) throws IOException {     int s = 0;
  int n;
  int t = 0;
  BufferedReader stadin = new BufferedReader(new InputStreamReader(
     System.in));     String input = stadin.readLine();
  n = Integer.parseInt(input);
  for (int i = 1; i <= n; i++) {
   t = t * 10 + n;
   s = s + t;       System.out.println(t);
  }
  System.out.println(s);
 }
```

```
}
```

3.9 题目：一个数如果恰好等于它的因子之和，这个数就称为 "完数 "。例如 6=1＋2＋3.编程　找出 1000 以内 的所有完数。

```
public class Wanshu {    public static void main(String[] args) {
   int s;
   for (int i = 1; i <= 1000; i++) {
    s = 0;
    for (int j = 1; j < i; j++)         if (i % j == 0)
       s = s + j;
    if (s == i)
      System.out.print(i + " ");
   }
   System.out.println();
 } }
```

3.10 题目：一球从 100 米高度自由落下，每次落地后反跳回原高度的一半；再落下，求它在　第 10 次落地时，共 经过多少米？第 10 次反弹多高？

```
public class Ex10 {
  public static void main(String[] args) {
    double s = 0;
    double t = 100;
    for (int i = 1; i <= 10; i++) {       s += t;
     t = t / 2;
    }
    System.out.println(s);
    System.out.println(t);
  }
}
```

3.11 题目：有 1、2、3、4 个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？

1.程序分析：可填在百位、十位、个位的数字都是 1、2、3、4。组成所有的排列后再去 掉不满足条件的排列。

```
public class Wanshu {
 public static void main(String[] args) {      int i = 0;
   int j = 0;
   int k = 0;
   int t = 0;
   for (i = 1; i <= 4; i++)        for (j = 1; j <= 4; j++)
      for (k = 1; k <= 4; k++)
       if (i != j && j != k && i != k) {
         t += 1;
         System.out.println(i * 100 + j * 10 + k);            }
   System.out.println(t);
 }
}
```

3.12 题目：企业发放的奖金根据利润提成。利润(I)低于或等于 10 万元时，奖金可提 10%；利润高于 10 万元， 低于 20 万元时，低于 10 万元的部分按 10%提成，高于 10 万元的部分，可可提成 7.5%；20 万到 40 万之间时，高于 20 万元的部分，可提成 5%；40 万到 60 万之间时高于 40 万元的部分，可提成 3%；60 万到 100 万之间时，高于 60 万元 的部分，可提成 1.5%，高于 100 万元时，超过 100 万元的部分按 1%提成，从键盘输入当月利润 I，求应发放奖金总数？

1.程序分析：请利用数轴来分界，定位。注意定义时需把奖金定义成长整型。

```
import java.util.*;
public class test {
```

```
 public static void main(String[] args) {     double sum;// 声明要储存的变量应发的奖金
Scanner input = new Scanner(System.in);// 导入扫描器
   System.out.print("输入当月利润");     double lirun = input.nextDouble();// 从控制台录入利
润
   if (lirun <= 100000) {
     sum = lirun * 0.1;     } else if (lirun <= 200000) {
     sum = 10000 + lirun * 0.075;
   } else if (lirun <= 400000) {
     sum = 17500 + lirun * 0.05;
   } else if (lirun <= 600000) {       sum = lirun * 0.03;
   } else if (lirun <= 1000000) {
     sum = lirun * 0.015;
   } else {
     sum = lirun * 0.01;     }     System.out.println("应发的奖金是" + sum);
 }
} 后面其他情况的代码可以由读者自行完善.
```

3.13 题目：一个整数，它加上 100 后是一个完全平方数，加上 168 又是一个完全平方数，请问该数是多少？

1.程序分析：在 10 万以内判断，先将该数加上 100 后再开方，再将该数加上 268 后再开方，如果开方后的结果满足如下条 件，即是结果。请看具体分析：

```
public class test {
 public static void main(String[] args) {
   long k = 0;     for (k = 1; k <= 100000l; k++)
     if (Math.floor(Math.sqrt(k + 100)) == Math.sqrt(k + 100)
         && Math.floor(Math.sqrt(k + 168)) == Math.sqrt(k + 168))
       System.out.println(k);
 } }
```

3.14 题目：输入某年某月某日，判断这一天是这一年的第几天？

1.程序分析：以 3 月 5 日为例，应该先把前两个月的加起来，然后再加上 5 天即本年的第几天，特殊情况，闰年且输入月份 大于 3 时需考虑多加一天。

```
import java.util.*;
public class test {    public static void main (String[]args){
   int day=0;
   int month=0;
   int year=0;
   int sum=0;      int leap;          System.out.print("请输入年,月,日\n");
   Scanner input = new Scanner(System.in);
   year=input.nextInt();
   month=input.nextInt();      day=input.nextInt();      switch(month)  /*先计算某月以前月份的总天数*/
   {
   case 1:
    sum=0;break;
   case 2:       sum=31;break;
   case 3:
    sum=59;break;
   case 4:
    sum=90;break;          case 5:
    sum=120;break;
   case 6:
    sum=151;break;
   case 7:       sum=181;break;
   case 8:
    sum=212;break;
   case 9:
    sum=243;break;          case 10:
```

```
    sum=273;break;
   case 11:
    sum=304;break;
   case 12:       sum=334;break;
   default:
    System.out.println("data error");break;
   }                              sum=sum+day;    /* 再 加 上 某 天 的 天 数 */
if(year%400==0||(year%4==0&&year%100!=0))/*判断是不是闰年*/
    leap=1;
   else
    leap=0;          if(leap==1 && month>2)/*如果是闰年且月份大于 2,总天数应该加一天*/
sum++;
   System.out.println("It is the the day:" + sum);
  }
}
```

3.15 题目：输入三个整数 x,y,z，请把这三个数由小到大输出。

1.程序分析：我们想办法把最小的数放到 x 上，先将 x 与 y 进行比较，如果 x> y 则将 x 与 y 的值进行交换，然后再用 x 与 z 进行比较，如果 x> z 则将 x 与 z 的值进行交换，这样能使 x 最小。

```
import java.util.*;
public class test {
 public static void main(String[] args) {      int i = 0;
   int j = 0;
   int k = 0;
   int x = 0;      System.out.print("请输入三个数\n");
   Scanner input = new Scanner(System.in);
   i = input.nextInt();      j = input.nextInt();
   k = input.nextInt();
```

```
    if (i > j) {
      x = i;
      i = j;       j = x;
    }
    if (i > k) {
      x = i;
      i = k;       k = x;
    }
    if (j > k) {
      x = j;
      j = k;       k = x;
    }
    System.out.println(i + ", " + j + ", " + k);
  }
}
```

3.16 题目：输出 9*9 口诀。

1.程序分析：分行与列考虑，共 9 行 9 列，i 控制行，j 控制列。

```
public class jiujiu {    public static void main(String[] args) {
    int i = 0;
    int j = 0;
    for (i = 1; i <= 9; i++) {
      for (j = 1; j <= 9; j++)          System.out.print(i + "*" + j + "=" + i * j + "\t");
      System.out.println();
    }
  }
} 不出现重复的乘积(下三角) public class jiujiu {
  public static void main(String[] args) {      int i = 0;
    int j = 0;
```

```
   for (i = 1; i <= 9; i++) {
     for (j = 1; j <= i; j++)
       System.out.print(i + "*" + j + "=" + i * j + "\t");       System.out.println();
   }
  }
} 上三角
public class jiujiu {
  public static void main(String[] args) {      int i = 0;
   int j = 0;
   for (i = 1; i <= 9; i++) {
     for (j = i; j <= 9; j++)
       System.out.print(i + "*" + j + "=" + i * j + "\t");       System.out.println();
   }
  }
}
```

3.17 题目：猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个　第二天早上又 将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下　的一半零一个。到第 10 天早上想再吃时，见只 剩下一个桃子了。求第一天共摘了多少。

1.程序分析：采取逆向思维的方法，从后往前推断。

```
public class 猴子吃桃 {
  static int total(int day) {      if (day == 10) {
     return 1;
   } else {
     return (total(day + 1) + 1) * 2;
   }   }
  public static void main(String[] args) {
   System.out.println(total(1));
```

} }

3.18 题目：两个乒乓球队进行比赛，各出三人。甲队为 a,b,c 三人，乙队为 x,y,z 三人。已抽签决定比赛名单。 有人向队员打听比赛的名单。a 说他不和 x 比，c 说他不和 x,z 比，请编程序找出三队赛手的名单。

1.程序分析：判断素数的方法：用一个数分别去除 2 到 sqrt(这个数)，如果能被整除， 则表明此数不是素数，反之是素数。

```
import java.util.ArrayList;
public class pingpang {    String a, b, c;
 public static void main(String[] args) {
  String[] op = { "x", "y", "z" };
  ArrayList<pingpang> arrayList = new ArrayList<pingpang>();      for (int i = 0; i < 3; i++)
   for (int j = 0; j < 3; j++)
    for (int k = 0; k < 3; k++) {
     pingpang a = new pingpang(op[i], op[j], op[k]);
     if (!a.a.equals(a.b) && !a.b.equals(a.c)              && !a.a.equals("x") && !a.c.equals("x")
        && !a.c.equals("z")) {
      arrayList.add(a);
     }
    }
  for (Object a : arrayList) {
   System.out.println(a);     }
 }
 public pingpang(String a, String b, String c) {
  super();     this.a = a;
  this.b = b;
  this.c = c;
 }
 @Override
```

```
 public String toString() {
  // TODO Auto-generated method stub      return "a 的对手是" + a + "," + "b 的对手是" + b + ","
+ "c 的对手是" + c + "\n";
 } }
```

3.19 题目：打印出如下图案（菱形）

```
          *
         ***
        *****
       *******
        *****
         ***
          *
```

1.程序分析：先把图形分成两部分来看待，前四行一个规律，后三行一个规律，利用双重 for 循环，第一层控制行，第二 层控制列。 三角形：

```
public class StartG {
 public static void main(String[] args) {      int i = 0;
  int j = 0;
  for (i = 1; i <= 4; i++) {
   for (j = 1; j <= 2 * i - 1; j++)
    System.out.print("*");       System.out.println("");
  }
  for (i = 4; i >= 1; i--) {
   for (j = 1; j <= 2 * i - 3; j++)
    System.out.print("*");       System.out.println("");
  }
 }
}
```

菱形：

```
public class StartG {
 public static void main(String[] args) {
  int i = 0;
```

```
    int j = 0;
    for (i = 1; i <= 4; i++) {        for (int k = 1; k <= 4 - i; k++)
        System.out.print(" ");
      for (j = 1; j <= 2 * i - 1; j++)
        System.out.print("*");
      System.out.println("");      }
    for (i = 4; i >= 1; i--) {
      for (int k = 1; k <= 5 - i; k++)
        System.out.print(" ");
      for (j = 1; j <= 2 * i - 3; j++)          System.out.print("*");
      System.out.println("");
    }
  }
}
```

3.20 题目：有一分数序列：2/1，3/2，5/3，8/5，13/8，21/13...求出这个数列的前 20 项之和。

1.程序分析：请抓住分子与分母的变化规律。

```
public class test20 {
  public static void main(String[] args) {
    float fm = 1f;
    float fz = 1f;      float temp;
    float sum = 0f;
    for (int i = 0; i < 20; i++) {
      temp = fm;
      fm = fz;        fz = fz + temp;
      sum += fz / fm;
      // System.out.println(sum);
    }
```

```
    System.out.println(sum);   }
}
```

【程序 21】　　题目：求 1+2!+3!+...+20!的和

1.程序分析：此程序只是把累加变成了累乘。

```
public class Ex21 {
  static long sum = 0;
  static long fac = 0;
  public static void main(String[] args) {
    long sum = 0;
    long fac = 1;
    for (int i = 1; i <= 10; i++) {
      fac = fac * i;
      sum += fac;
    }      System.out.println(sum);
  }
}
```

【程序 22】　　题目：利用递归方法求 5!。

1.程序分析：递归公式：fn=fn_1*4!

```
import java.util.Scanner;
public class Ex22 {    public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int n = s.nextInt();
    Ex22 tfr = new Ex22();
    System.out.println(tfr.recursion(n));
  }
  public long recursion(int n) {
    long value = 0;      if (n == 1 || n == 0) {
      value = 1;
    } else if (n > 1) {
      value = n * recursion(n - 1);
```

```
    }      return value;
  }
}
```

3.21 题目：有 5 个人坐在一起，问第五个人多少岁？他说比第 4 个人大 2 岁。问第 4 个人岁数，他说比第 3 个人 大 2 岁。问第三个人，又说比第 2 人大两岁。问第 2 个人，说比第一个人大两岁。最后问第一个人，他说是 10 岁。请问第 五个人多大？

1.程序分析：利用递归的方法，递归分为回推和递推两个阶段。要想知道第五个人岁数，需知道第四人的岁数，依次类推， 推到第一人（10 岁），再往回推。

```
public class Ex23 {
  static int getAge(int n) {      if (n == 1) {
      return 10;
    }
    return 2 + getAge(n - 1);
  }
  public static void main(String[] args) {      System.out.println("第五个的年龄为:" + getAge(5));
  }
}
```

3.22 题目：给一个不多于 5 位的正整数，要求：一、求它是几位数，二、逆序打印出各位数字。

```
import java.util.Scanner;
public class Ex24 {
  public static void main(String[] args) {
    Ex24 tn = new Ex24();
    Scanner s = new Scanner(System.in);      long a = s.nextLong();
    if (a < 0 || a > 100000) {
```

```
    System.out.println("Error Input, please run this program Again");
    System.exit(0);
  }    if (a >= 0 && a <= 9) {      System.out.println(a + "是一位数");      System.out.println("
按逆序输出是" + '\n' + a);
  } else if (a >= 10 && a <= 99) {          System.out.println(a + "是二位数");
System.out.println("按逆序输出是");
    tn.converse(a);
  } else if (a >= 100 && a <= 999) {          System.out.println(a + "是三位数");
System.out.println("按逆序输出是");      tn.converse(a);
  } else if (a >= 1000 && a <= 9999) {          System.out.println(a + "是四位数");
System.out.println("按逆序输出是");
    tn.converse(a);    } else if (a >= 10000 && a <= 99999) {      System.out.println(a + "是五
位数");      System.out.println("按逆序输出是");
    tn.converse(a);
  }  }
 public void converse(long l) {
  String s = Long.toString(l);
  char[] ch = s.toCharArray();     for (int i = ch.length - 1; i >= 0; i--) {
   System.out.print(ch[i]);
  }
 }
}
```

3.23 题目：一个 5 位数，判断它是不是回文数。即 12321 是回文数，个位与万位相同，十位与千位相同。

```
import java.util.Scanner;
public class Ex25 {   static int[] a = new int[5];
 static int[] b = new int[5];
 public static void main(String[] args) {
```

```
   boolean is = false;     Scanner s = new Scanner(System.in);     System.out.print("输入一个
10000-99999 之间的整数:");
   long l = s.nextLong();
   if (l > 99999 || l < 10000) {
     System.out.println("Input error, please input again!");       l = s.nextLong();
   }
   for (int i = 4; i >= 0; i--) {
     a[i] = (int) (l / (long) Math.pow(10, i));
     l = (l % (long) Math.pow(10, i));      }
   System.out.println();
   for (int i = 0, j = 0; i < 5; i++, j++) {
     b[j] = a[i];
   }      for (int i = 0, j = 4; i < 5; i++, j--) {
     if (a[i] != b[j]) {
       is = false;
       break;
     } else {          is = true;
     }
   }
   if (is == false) {
     System.out.println("is not a Palindrom!");      } else if (is == true) {
     System.out.println("is a Palindrom!");
   }
  }
}
```

3.24 题目：请输入星期几的第一个字母来判断一下是星期几，如果第一个字母一样，则继续　判断第二个字母。

1.程序分析：用情况语句比较好，如果第一个字母一样，则判断用情况语句或 if 语句判断

第二个字母。

```
import java.util.Scanner;
public class Ex26 {
 public static void main(String[] args) {     // 保存用户输入的第二个字母
  char weekSecond;     // 将 Scanner 类示例化为 input 对象，用于接收用户输入
  Scanner input = new Scanner(System.in);      // 开始提示并接收用户控制台输入
System.out.print("请输入星期值英文的第一个字母，我来帮您判断是星期几：");
  String letter = input.next();     // 判断用户控制台输入字符串长度是否是一个字母
  if (letter.length() == 1) {      // 利用取第一个索引位的字符来实现让 Scanner 接收 char 类型
输入
    char weekFirst = letter.charAt(0);
    switch (weekFirst) {      case 'm':       // 当输入小写字母时，利用 switch 结构特性执行下
一个带 break 语句的 case 分支，以实现忽略用户控 制台输入大小写敏感的功能
    case 'M':       System.out.println("星期一(Monday)");       break;
    case 't':       // 当输入小写字母时，利用 switch 结构特性执行下一个带 break 语句的 case
分支，以实现忽略用户控 制台输入大小写敏感的功能
    case 'T':       System.out          .print("由于星期二(Tuesday)与星期四(Thursday)均以字母
T 开头，故需输入第二个字母 才能正确判断：");
     letter = input.next();       // 判断用户控制台输入字符串长度是否是一个字母       if
(letter.length() == 1) {       // 利用取第一个索引位的字符来实现让 Scanner 接收 char 类型输
入
      weekSecond = letter.charAt(0);       // 利用或（||）运算符来实现忽略用户控制台输入
大小写敏感的功能
      if (weekSecond == 'U' || weekSecond == 'u') {          System.out.println("星期二
(Tuesday)");
       break;       // 利用或（||）运算符来实现忽略用户控制台输入大小写敏感的功能
      } else if (weekSecond == 'H' || weekSecond == 'h') {          System.out.println("星期四
(Thursday)");       break;       // 控制台错误提示
      } else {       System.out.println("输入错误，不能识别的星期值第二个字母，程序结
束！");
```

```
            break;          }
        } else {        // 控制台错误提示          System.out.println("输入错误，只能输入一个字母，程序结束！");
            break;      }
        case 'w':       // 当输入小写字母时，利用 switch 结构特性执行下一个带 break 语句的 case 分支，以实现忽略用户控 制台输入大小写敏感的功能
        case 'W':       System.out.println("星期三(Wednesday)");
            break;
        case 'f':
        // 当输入小写字母时，利用 switch 结构特性执行下一个带 break 语句的 case 分支，以实现忽略用户控 制台输入大小写敏感的功能
        case 'F':       System.out.println("星期五(Friday)");       break;
        case 's':       // 当输入小写字母时，利用 switch 结构特性执行下一个带 break 语句的 case 分支，以实现忽略用户控 制台输入大小写敏感的功能
        case 'S':       System.out          .print("由于星期六(Saturday)与星期日(Sunday)均以字母 S 开头，故需输入第二个字母 才能正确判断：");
            letter = input.next();      // 判断用户控制台输入字符串长度是否是一个字母      if (letter.length() == 1) {         // 利用取第一个索引位的字符来实现让 Scanner 接收 char 类型输入
            weekSecond = letter.charAt(0);      // 利用或（||）运算符来实现忽略用户控制台输入大小写敏感的功能
            if (weekSecond == 'A' || weekSecond == 'a') {           System.out.println("星期六(Saturday)");
                break;          // 利用或（||）运算符来实现忽略用户控制台输入大小写敏感的功能
            } else if (weekSecond == 'U' || weekSecond == 'u') {        System.out.println("星期日(Sunday)");          break;          // 控制台错误提示
            } else {        System.out.println("输入错误，不能识别的星期值第二个字母，程序结束！");
                break;          }
        } else {        // 控制台错误提示          System.out.println("输入错误，只能输入一个字
```

```
母，程序结束！");
        break;        }
    default:       // 控制台错误提示        System.out.println("输入错误，不能识别的星期值第一个字母，程序结束！");
      break;      }
  } else {      // 控制台错误提示       System.out.println("输入错误，只能输入一个字母，程序结束！");
  }  }
}
```

3.25 题目：求 100 之内的素数

```
public class Ex27 {
 public static void main(String args[]) {
  int sum, i;
  for (sum = 2; sum <= 100; sum++) {       for (i = 2; i <= sum / 2; i++) {
     if (sum % i == 0)
       break;
    }
    if (i > sum / 2)         System.out.println(sum + "是素数");
  }
 }
}
```

3.26 题目：对 10 个数进行排序

1.程序分析：可以利用选择法，即从后 9 个比较过程中，选择一个最小的与第一个元素交换，　　下次类推，即用第二个元素 与后 8 个进行比较，并进行交换。

```
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;
```

```
public class Ex28 {
  public static void main(String[] args) {
    int arr[] = new int[11];
    Random r = new Random();
    for (int i = 0; i < 10; i++) {        arr[i] = r.nextInt(100) + 1;// 得到 10 个 100 以内的整数
    }
    Arrays.sort(arr);
    for (int i = 0; i < arr.length; i++) {
      System.out.print(arr[i] + "\t");      }
    System.out.print("\nPlease Input a int number: ");
    Scanner sc = new Scanner(System.in);      arr[10] = sc.nextInt();// 输入一个 int 值
    Arrays.sort(arr);      for (int i = 0; i < arr.length; i++) {
      System.out.print(arr[i] + "\t");
    }
  }
}
```

3.27 题目：求一个 3*3 矩阵对角线元素之和

1.程序分析：利用双重 for 循环控制输入二维数组，再将 a[i][i]累加后输出。

```
public class Ex29 {    public static void main(String[] args) {
    double sum = 0;
    int array[][] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 7, 8 } };
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++) {          if (i == j)
          sum = sum + array[i][j];
      }
    System.out.println(sum);
  } }
```

3.28 题目：有一个已经排好序的数组。现输入一个数，要求按原来的规律将它插入数组中。

1. 程序分析：首先判断此数是否大于最后一个数，然后再考虑插入中间的数的情况，插入后此元素之后的数，依次后移一 个位置。

```
import java.util.Random;
public class ArraySort {
  public static void main(String[] args) {
    int temp = 0;
    int myarr[] = new int[12];
    Random r = new Random();      for (int i = 1; i <= 10; i++)
      myarr[i] = r.nextInt(1000);
    for (int k = 1; k <= 10; k++)
      System.out.print(myarr[k] + ",");
    for (int i = 1; i <= 9; i++)       for (int k = i + 1; k <= 10; k++)
       if (myarr[i] > myarr[k]) {
         temp = myarr[i];
         myarr[i] = myarr[k];
         myarr[k] = temp;          }
    System.out.println("");
    for (int k = 1; k <= 10; k++)
      System.out.print(myarr[k] + ",");
    myarr[11] = r.nextInt(1000);
    for (int k = 1; k <= 10; k++)
      if (myarr[k] > myarr[11]) {
        temp = myarr[11];
        for (int j = 11; j >= k + 1; j--)           myarr[j] = myarr[j - 1];
        myarr[k] = temp;
      }
    System.out.println("");
```

```
   for (int k = 1; k <= 11; k++)       System.out.print(myarr[k] + ",");
  }
}
```

3.29 题目：将一个数组逆序输出。

程序分析:用第一个与最后一个交换。　　其实,用循环控制变量更简单：for(int k=11;k>=1;k--) System.out.print(myarr[k]+",");

【程序 32】　　题目：取一个整数 a 从右端开始的 4～7 位。

程序分析：可以这样考虑：　　(1)先使 a 右移 4 位。　　(2)设置一个低 4 位全为 1,其余全为 0 的数。可用~(~0 < <4)　　(3)将上面二者进行&运算。

```
public class Ex32 {
  public static void main(String[] args) {
    int a = 0;
    long b = 18745678;      a = (int) Math.floor(b % Math.pow(10, 7) / Math.pow(10, 3));
    System.out.println(a);
  }
}
```

3.30 题目：打印出杨辉三角形（要求打印出 10 行如下图）

1.程序分析：

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5   10  10  5   1
```

```
public class Ex33 {
  public static void main(String args[]) {      int i, j;
    int a[][];
```

```
    a = new int[8][8];
    for (i = 0; i < 8; i++) {
      a[i][i] = 1;        a[i][0] = 1;
    }
    for (i = 2; i < 8; i++) {
      for (j = 1; j <= i - 1; j++) {
        a[i][j] = a[i - 1][j - 1] + a[i - 1][j];        }
    }
    for (i = 0; i < 8; i++) {
      for (j = 0; j < i; j++) {
        System.out.printf("    " + a[i][j]);        }
      System.out.println();
    }
  }
}
```

3.31 题目：输入 3 个数 a,b,c，按大小顺序输出。

1.程序分析：利用指针方法。

```
public class Ex34 {
  public static void main(String[] args) {
    int[] arrays = { 800, 56, 500 };
    for (int i = arrays.length; --i >= 0;) {
      for (int j = 0; j < i; j++) {
        if (arrays[j] > arrays[j + 1]) {
          int temp = arrays[j];            arrays[j] = arrays[j + 1];
          arrays[j + 1] = temp;
        }
      }
    }      for (int n = 0; n < arrays.length; n++)
```

```
    System.out.println(arrays[n]);
  }
}
```

3.32 题目：输入数组，最大的与第一个元素交换，最小的与最后一个元素交换，输出数组。

```
import java.util.*;
public class Ex35 {    public static void main(String[] args) {
   int i, min, max, n, temp1, temp2;
   int a[];      System.out.println("输入数组的长度:");
   Scanner keyboard = new Scanner(System.in);      n = keyboard.nextInt();
   a = new int[n];
   for (i = 0; i < n; i++) {        System.out.print("输入第" + (i + 1) + "个数据");
    a[i] = keyboard.nextInt();     }     // 以上是输入整个数组
   max = 0;
   min = 0;     // 设置两个标志,开始都指向第一个数      for (i = 1; i < n; i++) {
    if (a[i] > a[max])         max = i; // 遍历数组,如果大于 a[max]，就把他的数组下标赋给 max
    if (a[i] < a[min])         min = i; // 同上，如果小于 a[min],就把他的数组下标赋给 min     }
// 以上 for 循环找到最大值和最小值，max 是最大值的下标，min 是最小值的下标
   temp1 = a[0];      temp2 = a[min]; // 这两个 temp 只是为了在交换时使用
   a[0] = a[max];      a[max] = temp1; // 首先交换 a[0]和最大值 a[max]
   if (min != 0) { // 如果最小值不是 a[0]，执行下面
    a[min] = a[n - 1];
    a[n - 1] = temp2; // 交换 a[min]和 a[n-1]     } else { // 如果最小值是 a[0],执行下面
    a[max] = a[n - 1];
    a[n - 1] = temp1;     }
   for (i = 0; i < n; i++) { // 输出数组
    System.out.print(a[i] + " ");
   }   }
```

}

3.33 题目：有 n 个整数，使其前面各数顺序向后移 m 个位置，最后 m 个数变成最前面的 m 个数

```
import java.util.Scanner;
public class Ex36 {
  public void change(int[] arr, int m) {     // 判断 m 的值,如果超过数组长度,则循环移动,其结果相当于移动了 m%arr.length 次.
    if (m >= arr.length) {        m = m % arr.length;
    }
    /*       * 实现步骤:   1.先将最后一个数取出保存,前面的所有数依次后移一位   2.前面的数移动完后再将保存的数存到第一个位置         3.依次循环前面的步骤 m 次,直到完成.
      */
    for (int i = 0; i < m; i++) {
      int temp = arr[arr.length - 1];        for (int j = arr.length - 1; j > 0; j--) {
        arr[j] = arr[j - 1];
      }
      arr[0] = temp;
    }   }
  // 测试数据
  public static void main(String[] args) {
    int[] arr = { 2, 4, 8, 3, 56, 12, 43, 23, 33, 17, 63, 52 };
    System.out.println("移动前的数组是:");
    for (int i : arr) {
      System.out.print(i + " ");     }      System.out.println("其整数个数为" + arr.length);
    Scanner input = new Scanner(System.in);     System.out.print("输入顺序后移的次数(如超过整数个数,则会循环移动):");
    int m = input.nextInt();
    new Ex36().change(arr, m);     System.out.println("移动后的数组是:");
```

```
   for (int i : arr) {
     System.out.print(i + " ");     }
  }
}
```

3.34 题目：有 n 个人围成一圈，顺序排号。从第一个人开始报数（从 1 到 3 报数），凡报到 3 的人退出圈子，问最后留下的是原 来第几号的那位。

```
import java.util.Scanner;
public class Ex37 {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int n = s.nextInt();      boolean[] arr = new boolean[n];
    for (int i = 0; i < arr.length; i++) {       arr[i] = true;// 下标为 TRUE 时说明还在圈里
    }
    int leftCount = n;      int countNum = 0;
    int index = 0;
    while (leftCount > 1) {       if (arr[index] == true) {// 当在圈里时          countNum++; // 报数递加          if (countNum == 3) {// 报道 3 时          countNum = 0;// 从零开始继续报数 arr[index] = false;// 此人退出圈子          leftCount--;// 剩余人数减一
      }       }       index++;// 每报一次数，下标加一       if (index == n) {// 是循环数数，当下标大于 n 时，说明已经数了一圈，          index = 0;// 将下标设为零重新开始。
     }     }
    for (int i = 0; i < n; i++) {
     if (arr[i] == true) {
       System.out.println(i);
     }     }
  }
}
```

3.35 题目：编写一个函数，输入 n 为偶数时，调用函数求 1/2+1/4+...+1/n,当输入 n 为奇数时，调用函数 1/1+1/3+...+1/n

```
import java.util.Scanner;
public class Ex38 {
 public double add() {     Scanner input = new Scanner(System.in);     System.out.print("请输入一个大于 0 的整数:");     int n = input.nextInt(); // 保存输入的整数
   while (n <= 0) {       System.out.print("输入错误,重新输入一个大于 0 的整数");       n = input.nextInt();
   }      double sum = addFraction(n); // 记录求和的结果
   return sum;
 }
 // 使用递归完成求和计算.
 public double addFraction(double n) {     if (n > 2) { // 保证分母不为 0 或负数
     double s = 1 / n + addFraction(n - 2);       return s;
   } else {
     return 1 / n;
   }
 }
 // 测试
 public static void main(String[] args) {
   double test = new Ex38().add();
   System.out.println(test);   } }
```

3.36 题目：字符串排序。

```
import java.util.*;       public class test{
 public     static     void     main(String[]     args{
         ArrayList<String> list=new ArrayList<String>();
list.add("010101");
```

```
        list.add("010003");                 list.add("010201");
        Collections.sort(list);
    for(int     i=0;i<list.size();i++){
      System.out.println(list.get(i));
    }           }
}
```

3.37 这只猴子把题目：海滩上有一堆桃子，五只猴子来分。第一只猴子把这堆桃子凭据分为五份，多了一个，多的一个扔入海中， 拿走了一份。第二只猴子把剩下的桃子又平均分成五份，又多了一个，它同样把多的一个扔入海中，拿走了一份，第三、第 四、第五只猴子都是这样做的，问海滩上原来最少有多少个桃子？

```
public class Dg {
    static int ts = 0;// 桃子总数
    int fs = 1;// 记录分的次数
    static int hs = 5;// 猴子数...
    int tsscope = 5000;// 桃子数的取值范围.太大容易溢出.

    public int fT(int t) {
        if (t == tsscope) { // 当桃子数到了最大的取值范围时取消递归
            System.out.println("结束");
            return 0;
        } else {
            if ((t - 1) % hs == 0 && fs <= hs) {
                if (fs == hs) {
                    System.out.println("桃子数 = " + ts + " 时满足分桃条件");
                }
                fs += 1;
                return fT((t - 1) / 5 * 4);// 返回猴子拿走一份后的剩下的总数
```

```
            } else { // 没满足条件
                fs = 1;// 分的次数重置为 1
                return fT(ts += 1);// 桃子数加+1
            }
        }
    }

    public static void main(String[] args) {
        new Dg().fT(0);
    }
}
```

四、Html & JavaScript & Ajax 部分

4.1 判断第二个日期比第一个日期大

如何用脚本判断用户输入的的字符串是下面的时间格式 2004-11-21 必须要保证用户的输入是此格式，并且是时间，比如说月份不大于 12 等等，另外我需要用户输入两个，并且后一个要比前一个晚，只允许用 JAVASCRIPT，请详细帮助作答，,

```
//这里可用正则表达式判断提前判断一下格式，然后按下提取各时间字段内容
    <script type="text/javascript">
      window.onload = function()
      {
//这么写是为了实现 js 代码与 html 代码的分离，当我修改 js 时，不能影响 html 代码。
        document.getElementById("frm1").onsubmit =
            function(){
                var d1 = this.d1.value;
                var d2 = this.d2.value;
                if(!verifyDate (d1)) {alert("第一个日期格式不对");return false;}
                if(!verifyDate (d2)) {alert("第二个日期格式不对");return false;}
                if(!compareDate(d1,d2)) {alert("第二个日期比第一日期小");return false;}
            };
        }

        function compareDate(d1,d2)
        {
            var arrayD1 = d1.split("-");
            var date1 = new Date(arrayD1[0],arrayD1[1],arrayD1[2]);
            var arrayD2 = d2.split("-");
            var date2 = new Date(arrayD2[0],arrayD2[1],arrayD2[2]);
            if(date1 > date2) return false;
            return true;
```

```
        }

        function verifyDate(d)
        {
            var datePattern = /^\d{4}-(0?[1-9]|1[0-2])-(0?[1-9]|[1-2]\d|3[0-1])$/;
            return datePattern.test(d);
        }
    </script>

    <form id="frm1" action="xxx.html">
    <input type="text" name="d1" />
    <input type="text" name="d2" />
    <input type="submit"/>
    </form>
```

4.2 用 table 显示 n 条记录，每 3 行换一次颜色，即 1，2，3 用红色字体，4，5，6 用绿色字体，7，8，9 用红颜色字体。

```
    <body>
        <table id="tbl">
            <tr><td>1</td></tr>
            <tr><td>2</td></tr>
            <tr><td>3</td></tr>
            <tr><td>4</td></tr>
            <tr><td>5</td></tr>
            <tr><td>6</td></tr>
            <tr><td>7</td></tr>
            <tr><td>8</td></tr>
            <tr><td>9</td></tr>
            <tr><td>10</td></tr>
```

```
        </table>
    </body>
    <script type="text/javascript">
      window.onload=function()
        {
            var tbl = document.getElementById("tbl");
            rows = tbl.getElementsByTagName("tr");
            for(i=0;i<rows.length;i++)
            {
                var j = parseInt(i/3);
                if(j%2==0) rows[i].style.backgroundColor="#f00";
                else    rows[i].style.backgroundColor="#0f0";
            }
        }
    </script>
```

4.3 HTML 的 form 提交之前如何验证数值文本框的内容全部为数字？否则的话提示用户并终止提交？

```
<form onsubmit=’return chkForm(this)’>
<input type="text" name="d1"/>
<input type="submit"/>
</form>
<script type=”text/javascript” />
function chkForm(this)
    {
        var value = thist.d1.value;
        var len = value.length;
        for(var i=0;i<len;i++)
        {
```

```
            if(value.charAt(i)>"9" || value.charAt(i)<"0")
            {
                alert("含有非数字字符");
                return false;
            }
        }
        return true;
    }
</script>
```

4.4 请写出用于校验 HTML 文本框中输入的内容全部为数字的 javascript 代码

```
<input type="text" id="d1" onblur=" chkNumber (this)"/>
<script type=”text/javascript” />
function    chkNumber(eleText)

    {
        var value = eleText.value;
        var len = value.length;
        for(var i=0;i<len;i++)
        {
            if(value.charAt(i)>"9" || value.charAt(i)<"0")
            {
                alert("含有非数字字符");

                eleText.focus();
                break;
            }
        }
```

```
    }
</script>
```

除了写完代码，还应该在网页上写出实验步骤和在代码中加入实现思路，让面试官一看就明白你的意图和检查你的结果。

4.5 说说你用过那些 ajax 技术和框架，说说它们的区别

答:AJAX（Asynchronous JavaScript And XML，异步 JavaScript 和 XML)，是创建交互式 Web 应用的主要开发技术。互联网中也有大量的关于 AJAX 的框架，本文汇总了最常用的 11 个框架。

1. jQuery

jQuery 是一个轻量级的 Javascript 库，兼容 CSS3，还兼容各种浏览器。jQuery 使用户能更方便地处理 HTML documents、events、实现动画效果，并且方便地为网站提供 AJAX 交互。

2. MooTools

MooTools 是一个简洁、模块化、面向对象的 JavaScript 库。它能够帮助你更快、更简单地编写可扩展和兼容性强的 JavaScript 代码。Mootools 跟 prototypejs 相类似，语法几乎一样。但它提供的功能要比 prototypejs 多，而且更强大。比如增加了动画特效、拖放操作等等。

3. Prototype

Prototype 是 Sam Stephenson 写的一个非常优雅的 JavaScript 基础类库，对 JavaScript 做了大量的扩展，旨在简化动态 Web 应用程序的开发。Prototype 很好的支持 AJAX，国内外有多个基于此类库实现的效果库，也做得很棒。

4. ASP.NET AJAX

ASP.NET AJAX 是一个完整的开发框架，容易与现有的 ASP.NET 程序相结合，通常实现复杂的功能只需要在页面中拖几个控件，而不必了解深层次的工作原理，除此之外服务器端编程的 ASP.NET AJAX Control Toolkit 含有大量的独立 AJAX 控件和对 ASP.NET 原有服务器控件的 AJAX 功能扩展，实现起来也非常简单。

5. Apache Wicket

Apache Wicket 是一个针对 Java 的 Web 开发框架，与 Struts、WebWork、Tapestry 类似。其特点在于对 HTML 和代码进行了有效的分离（有利于程序员和美工的合作），基于规则

的配置（减少了 XML 等配置文件的使用)，学习曲线较低（开发方式与 C/S 相似），更加易于调试（错误类型比较少，而且容易定位）。

6. Dojo Tookit

Dojo是一个强大的面向对象的JavaScript框架。主要由三大模块组成：Core、Dijit、DojoX。Core 提供 AJAX、events、packaging、CSS-based querying、animations、JSON 等相关操作 API；Dijit 是一个可更换皮肤、基于模板的 WEB UI 控件库；DojoX 包括一些创新/新颖的代码和控件：DateGrid、charts、离线应用、跨浏览器矢量绘图等。

7. DWR（Direct Web Remoting）

DWR 是一个 Java 库，可以帮助开发者轻松实现服务器端的 Java 和客户端的 JavaScript 相互操作、彼此调用。

8. Spry Framework

Adobe Spry 是一个面向 Web 设计人员而不是开发人员的 AJAX 框架，它使得设计人员不需要了解复杂的 AJAX 技巧也能在一个 HTML 页面中创建丰富体验成为了可能。

9. YUI （Yahoo User Interface） Library

YUI（Yahoo User Interface），是由雅虎开发的一个开源的 JavaScript 函数库，它采用了 AJAX、 DHTML 和 DOM 等诸多技术。YUI 包含多种程序工具、函数库以及网页操作界面，能够更快速地开发互动性高且丰富的网站应用程序。

10. Google Web Toolkit

Google Web Toolkit (GWT) 是一个开源的 Java 开发框架，可以使不会使用第二种浏览器语言的开发人员编写 Google 地图和 Gmail 等 AJAX 应用程序时更加轻松。

11. ZK Framework

ZK 是一套开源、兼容 XUL/HTML 标准、使用 Java 编写的 AJAX 框架，使用该框架，你无需编写 JavaScript 代码就可以创建一个支持 Web 2.0 的富互联网应用程序（RIA）。其最大的好处是，在设计 AJAX 网络应用程序时，轻松简便的操作就像设计桌面程序一样。ZK 包含了一个以 AJAX 为基础、事件驱动（event-driven）、高互动性的引擎，同时还提供了多样丰富、可重复使用的 XUL 与 HTML 组件，以及以 XML 为基础的使用接口设计语言 ZK User-interfaces Markup Language（ZUML）。

五、Java Web 部分

5.1 Tomcat 的优化经验

Tomcat 作为 Web 服务器，它的处理性能直接关系到用户体验，下面是几种常见的优化措施:

1，去掉对 web.xml 的监视，把 jsp 提前编辑成 Servlet。有富余物理内存的情况，加大 tomcat 使用的 jvm 的内存

2，服务器资源

服务器所能提供 CPU、内存、硬盘的性能对处理能力有决定性影响。

(1) 对于高并发情况下会有大量的运算，那么 CPU 的速度会直接影响到处理速度。

(2) 内存在大量数据处理的情况下，将会有较大的内存容量需求，可以用-Xmx -Xms -XX:MaxPermSize 等参数对内存不同功能块进行划分。我们之前就遇到过内存分配不足，导致虚拟机一直处于 full GC，从而导致处理能力严重下降。

(3) 硬盘主要问题就是读写性能，当大量文件进行读写时，磁盘极容易成为性能瓶颈。最好的办法还是利用下面提到的缓存。

3，利用缓存和压缩

对于静态页面最好是能够缓存起来，这样就不必每次从磁盘上读。这里我们采用了 Nginx 作为缓存服务器，将图片、css、js 文件都进行了缓存，有效的减少了后端 tomcat 的访问。

另外，为了能加快网络传输速度，开启 gzip 压缩也是必不可少的。但考虑到 tomcat 已经需要处理很多东西了，所以把这个压缩的工作就交给前端的 Nginx 来完成。

除了文本可以用 gzip 压缩，其实很多图片也可以用图像处理工具预先进行压缩，找到一个平衡点可以让画质损失很小而文件可以减小很多。曾经我就见过一个图片从 300 多 kb 压缩到几十 kb，自己几乎看不出来区别。

4，采用集群

单个服务器性能总是有限的，最好的办法自然是实现横向扩展，那么组建 tomcat 集群是有效提升性能的手段。我们还是采用了 Nginx 来作为请求分流的服务器，后端多个 tomcat 共享 session 来协同工作。可以参考之前写的《利用 nginx+tomcat+memcached 组建 web 服务器负载均衡》。

5，优化 tomcat 参数

这里以 tomcat7 的参数配置为例，需要修改 conf/server.xml 文件，主要是优化连接配置，关闭客户端 dns 查询。

```
<Connector port="8080"
   protocol="org.apache.coyote.http11.Http11NioProtocol"
   connectionTimeout="20000"
   redirectPort="8443"
   maxThreads="500"
   minSpareThreads="20"
   acceptCount="100"
   disableUploadTimeout="true"
   enableLookups="false"
   URIEncoding="UTF-8" />
```

5.2 HTTP 请求的 GET 与 POST 方式的区别

1、GET 请求，请求的数据会附加在 URL 之后，以 **?** 分割 URL 和传输数据，多个参数用 **&** 连接。URL 的编码格式采用的是 ASCII 编码，而不是 unicode，即是说所有的非 ASCII 字符都要编码之后再传输。

POST 请求：POST 请求会把请求的数据放置在 HTTP 请求包的包体中。

因此，GET 请求的数据会暴露在地址栏中，而 POST 请求则不会。

2、传输数据的大小

在 HTTP 规范中，没有对 URL 的长度和传输的数据大小进行限制。但是在实际开发过程中，对于 GET，特定的浏览器和服务器对 URL 的长度有限制。因此，在使用 GET 请求时，传输数据会受到 URL 长度的限制。

对于 POST，由于不是 URL 传值，理论上是不会受限制的，但是实际上各个服务器会规定对 POST 提交数据大小进行限制，Apache、IIS 都有各自的配置。

3、安全性

POST 的安全性比 GET 的高。这里的安全是指真正的安全，而不同于上面 GET 提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，

通过 GET 请求，用户名和密码都会暴露再 URL 上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET 请求提交的数据还可能会造成 Cross-site request frogery 攻击

4、HTTP 中的 GET，POST，SOAP 协议都是在 HTTP 上运行的

5.3 解释一下什么是 servlet

答：Servlet 是一种服务器端的 Java 应用程序，具有独立于平台和协议的特性,可以生成动态的 Web 页面。 它担当客户请求（Web 浏览器或其他 HTTP 客户程序）与服务器响应（HTTP 服务器上的数据库或应用程序）的中间层。 Servlet 是位于 Web 服务器内部的服务器端的 Java 应用程序，与传统的从命令行启动的 Java 应用程序不同，Servlet 由 Web 服务器进行加载，该 Web 服务器必须包含支持 Servlet 的 Java 虚拟机

5.4 说一说 Servlet 的生命周期?

答:servlet 有良好的生存期的定义，包括加载和实例化、初始化、处理请求以及服务结束。这个生存期由 javax.servlet.Servlet 接口的 init、service 和 destroy 方法表达。

Servlet 被服务器实例化后，容器运行其 init 方法，请求到达时运行其 service 方法，service 方法自动派遣运行与请求对应的 doXxx 方法（doGet，doPost）等，当服务器决定将实例销毁的时候调用其 destroy 方法。

web 容器加载 servlet，生命周期开始。通过调用 servlet 的 init()方法进行 servlet 的初始化。通过调用 service()方法实现，根据请求的不同调用不同的 do***()方法。结束服务，web 容器调用 servlet 的 destroy()方法。

5.5 Servlet 的基本架构

```
public class ServletName extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response) throws
   ServletException, IOException {
```

```
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    }
}
```

5.6 SERVLET API 中 forward() 与 redirect()的区别？

答:前者仅是容器中控制权的转向，在客户端浏览器地址栏中不会显示出转向后的地址；后者则是完全的跳转，浏览器将会得到跳转的地址，并重新发送请求链接。这样，从浏览器的地址栏中可以看到跳转后的链接地址。所以，前者更加高效，在前者可以满足需要时，尽量使用 forward()方法，并且，这样也有助于隐藏实际的链接。在有些情况下，比如，需要跳转到一个其它服务器上的资源，则必须使用 sendRedirect()方法。

5.7 什么情况下调用 doGet()和 doPost()？

答：Jsp 页面中的 FORM 标签里的 method 属性为 get 时调用 doGet()，为 post 时调用 doPost()。

5.8 request 对象的主要方法：

setAttribute(String name,Object)：设置名字为 name 的 request 的参数值

getAttribute(String name)：返回由 n ame 指定的属性值

getAttributeNames()：返回 request 对象所有属性的名字集合，结果是一个枚举的实例

getCookies()：返回客户端的所有 Cookie 对象，结果是一个 Cookie 数组

getCharacterEncoding()：返回请求中的字符编码方式

getContentLength()：返回请求的 Body 的长度

getHeader(String name)：获得 HTTP 协议定义的文件头信息

getHeaders(String name)：返回指定名字的 request Header 的所有值，结果是一个枚举的实例

getHeaderNames()：返回所以 request Header 的名字，结果是一个枚举的实例

getInputStream()：返回请求的输入流，用于获得请求中的数据

getMethod()：获得客户端向服务器端传送数据的方法

getParameter(String name)：获得客户端传送给服务器端的有 name 指定的参数值

getParameterNames()：获得客户端传送给服务器端的所有参数的名字，结果是一个枚举的实例

getParametervalues(String name)：获得有 name 指定的参数的所有值

getProtocol()：获取客户端向服务器端传送数据所依据的协议名称

getQueryString()：获得查询字符串

getRequestURI()：获取发出请求字符串的客户端地址

getRemoteAddr()：获取客户端的 IP 地址

getRemoteHost()：获取客户端的名字

getSession([Boolean create])：返回和请求相关 Session

getServerName()：获取服务器的名字

getServletPath()：获取客户端所请求的脚本文件的路径

getServerPort()：获取服务器的端口号

removeAttribute(String name)：删除请求中的一个属性

5.9 request.getAttribute() 和 request.getParameter() 有何区别?

getParameter 是用来接受用 post 个 get 方法传递过来的参数的.

getAttribute 必须先 setAttribute.

（1）request.getParameter()取得是通过容器的实现来取得通过类似 post，get 等方式传入的数据，request.setAttribute()和 getAttribute()只是在 web 容器内部流转，仅仅是请求处理阶段。

（2）request.getParameter()方法传递的数据，会从 Web 客户端传到 Web 服务器端，代表 HTTP 请求数据。request.getParameter()方法返回 String 类型的数据。

request.setAttribute()和 getAttribute()方法传递的数据只会存在于 Web 容器内部

还有一点就是，HttpServletRequest 类有 setAttribute()方法，而没有 setParameter()方法。

5.10 jsp 有哪些内置对象?作用分别是什么？分别有什么方法？

答:JSP 共有以下 9 个内置的对象：

request 用户端请求，此请求会包含来自 GET/POST 请求的参数

response 网页传回用户端的回应

pageContext 网页的属性是在这里管理

session 与请求有关的会话期

application servlet 正在执行的内容

out 用来传送回应的输出

config servlet 的构架部件

page JSP 网页本身

exception 针对错误网页，未捕捉的例外

request 表示 HttpServletRequest 对象。它包含了有关浏览器请求的信息，并且提供了几个用于获取 cookie, header, 和 session 数据的有用的方法。

response 表示 HttpServletResponse 对象，并提供了几个用于设置送回 浏览器的响应的方法（如 cookies,头信息等）

out 对象是 javax.jsp.JspWriter 的一个实例，并提供了几个方法使你能用于向浏览器回送输出结果。

pageContext 表示一个 javax.servlet.jsp.PageContext 对象。它是用于方便存取各种范围的名字空间、servlet 相关的对象的 API，并且包装了通用的 servlet 相关功能的方法。

session 表示一个请求的 javax.servlet.http.HttpSession 对象。Session 可以存贮用户的状态信息

applicaton 表示一个 javax.servle.ServletContext 对象。这有助于查找有关

servlet 引擎和 servlet 环境的信息

config 表示一个 javax.servlet.ServletConfig 对象。该对象用于存取 servlet 实例的初始化参数。

page 表示从该页面产生的一个 servlet 实例

5.11 jsp 有哪些动作?作用分别是什么?

答:JSP 共有以下 6 种基本动作

jsp:include：在页面被请求的时候引入一个文件。

jsp:useBean：寻找或者实例化一个 JavaBean。

jsp:setProperty：设置 JavaBean 的属性。

jsp:getProperty：输出某个 JavaBean 的属性。

jsp:forward：把请求转到一个新的页面。

jsp:plugin：根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记

5.12 JSP 的常用指令

答：常用的指令有三个：page、include、taglib;

1. page 指令

<%@page language="java" contentType="text/html;charset=gb2312" session="true" buffer="64kb" autoFlush="true" isThreadSafe="true" info="text" errorPage="error.jsp" isErrorPage="true" isELIgnored=" true" pageEncoding="gb2312" import="java.sql.*"%>

说明:isErrorPage(是否能使用 Exception 对象)，isELIgnored(是否忽略表达式)

2. include 指令

<%@ include file="filename"%>

3. taglib 指令

<%@ taglib prefix="c"uri="http://……"%>

5.13 JSP 中动态 INCLUDE 与静态 INCLUDE 的区别？

答：动态 INCLUDE 用 jsp:include 动作实现

<jsp:include page=included.jsp flush=true />它总是会检查所含文件中的变化，适合用于包含动态页面，并且可以带参数。

静态 INCLUDE 用 include 伪码实现,定不会检查所含文件的变化，适用于包含静态页面 <%@ include file=included.htm %>

5.14 页面间对象传递的方法

request，session，application，cookie 等

5.15 JSP 和 Servlet 有哪些相同点和不同点，他们之间的联系是什么？

JSP 是 Servlet 技术的扩展，本质上是 Servlet 的简易方式，更强调应用的外表表达。JSP 编译后是"类 servlet"。Servlet 和 JSP 最主要的不同点在于，Servlet 的应用逻辑是在 Java 文件中，并且完全从表示层中的 HTML 里分离开来。而 JSP 的情况是 Java 和 HTML 可以组合成一个扩展名为.jsp 的文件。JSP 侧重于视图，Servlet 主要用于控制逻辑。

5.16 MVC 的各个部分都有那些技术来实现?如何实现?

答:MVC 是 Model－View－Controller 的简写。Model 代表的是应用的业务逻辑（通过 JavaBean，EJB 组件实现）， View 是应用的表示面（由 JSP 页面产生），Controller 是提供应用的处理过程控制（一般是一个 Servlet），通过这种设计模型把应用逻辑，处理过程和显示逻辑分成不同的组件实现。这些组件可以进行交互和重用。

5.17 我们在 web 应用开发过程中经常遇到输出某种编码的字符，如 iso8859-1 等，如何输出一个某种编码的字符串？

```
Public String translate (String str) {
    String tempStr = "";
```

```
    try {
        tempStr = new String(str.getBytes("ISO-8859-1"), "GBK");
        tempStr = tempStr.trim();
    }
    catch (Exception e) {
        System.err.println(e.getMessage());
    }
    return tempStr;
}
```

5.18 现在输入 n 个数字，以逗号，分开；然后可选择升或者降序排序；按提交键就在另一页面显示按什么排序

```
public class Test {
    public static void main(String[] args) {
        String str = "1,3,6,4,2,5";
        String[] s = str.split(",");
        Arrays.sort(s);// 升序
        for (String show : s) {
            System.out.print("升序: "+show + " ");// 1 2 3 4 5 6
        }
        System.out.println();
        swapArray(s);// 降序
        for (String show : s) {
            System.out.print("降序: "+show + " ");// 6 5 4 3 2 1
        }
    }

    private static void swapArray(String[] s) {// 降序方法实现
        int len = s.length;
```

```
        // 折半，两端交换
        for (int i = 0; i < len / 2; i++) {
            String temp = s[i];
            s[i] = s[len - 1 - i];
            s[len - 1 - i] = temp;
        }
    }
    }
```

六、数据库部分

6.1 根据部门号从高到低，工资从低到高列出每个员工的信息。

employee:eid,ename,salary,deptid;

select * from employee order by deptid desc,salary

6.2 列出各个部门中工资高于本部门的平均工资的员工数和部门号，并按部门号排序

创建表：

mysql> create table employee921(id int primary key auto_increment,name varchar(50),salary bigint,deptid int);

插入实验数据：

mysql> insert into employee921 values(null,'zs',1000,1),(null,'ls',1100,1),(null,'ww',1100,1),(null,'zl',900,1) ,(null,'zl',1000,2), (null,'zl',900,2) ,(null,'zl',1000,2) , (null,'zl',1100,2);

编写 sql 语句：

（）select avg(salary) from employee921 group by deptid;

（）mysql> select employee921.id,employee921.name,employee921.salary,employee921.deptid tid from employee921 where salary > (select avg(salary) from employee921 where deptid = tid);

效率低的一个语句，仅供学习参考使用（在 group by 之后不能使用 where，只能使用 having，在 group by 之前可以使用 where，即表示对过滤后的结果分组）：

mysql> select employee921.id,employee921.name,employee921.salary,employee921.deptid tid from employee921 where salary > (select avg(salary) from employee921 group by deptid having deptid = tid);

（）select count(*) ,tid

from (

select employee921.id,employee921.name,employee921.salary,employee921.deptid tid

from employee921

where salary >

(select avg(salary) from employee921 where deptid = tid)

) as t

group by tid ;

另外一种方式：关联查询

```
select a.ename,a.salary,a.deptid
  from emp a,
      (select deptd,avg(salary) avgsal from emp group by deptid ) b
  where a.deptid=b.deptid and a.salary>b.avgsal;
```

6.3 存储过程与触发器必须讲，经常被面试到?

```
create procedure insert_Student (_name varchar(50),_age int ,out _id int)
begin
      insert into student value(null,_name,_age);
      select max(stuId) into _id from student;
end;
call insert_Student('wfz',23,@id);
select @id;
mysql> create trigger update_Student BEFORE update on student FOR EACH ROW
      -> select * from student;
```

触发器不允许返回结果

```
create trigger update_Student BEFORE update on student FOR EACH ROW
insert into    student value(null,'zxx',28);
```

mysql 的触发器目前不能对当前表进行操作

create trigger update_Student BEFORE update on student FOR EACH ROW

delete from articles where id=8;

这个例子不是很好，最好是用删除一个用户时，顺带删除该用户的所有帖子

这里要注意使用 OLD.id

触发器用处还是很多的，比如校内网、开心网、Facebook，你发一个日志，自动通知好友，其实就是在增加日志时做一个后触发，再向通知表中写入条目。因为触发器效率高。而 UCH 没有用触发器，效率和数据处理能力都很低。

存储过程的实验步骤：

```
mysql> delimiter |
mysql> create procedure insertArticle_Procedure (pTitle varchar(50),pBid int,out
  pId int)
      -> begin
      -> insert into article1 value(null,pTitle,pBid);
      -> select max(id) into pId from article1;
      -> end;
      -> |
Query OK, 0 rows affected (0.05 sec)


mysql> call insertArticle_Procedure('某某培训',1,@pid);
      -> |
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter ;
mysql> select @pid;
+------+
| @pid |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

```
mysql> select * from article1;
+----+--------------+------+
| id | title        | bid  |
+----+--------------+------+
| 1  | test         | 1    |
| 2  | chuanzhiboke | 1    |
| 3  | 某某培训     | 1    |
+----+--------------+------+
3 rows in set (0.00 sec)
```

触发器的实验步骤：

```
create table board1(id int primary key auto_increment,name varchar(50),ar
ticleCount int);
create table article1(id int primary key auto_increment,title varchar(50)
,bid int references board1(id));
delimiter |
create trigger insertArticle_Trigger after insert on article1 for each ro
w begin
    -> update board1 set articleCount=articleCount+1 where id= NEW.bid;
    -> end;
    -> |
delimiter ;
insert into board1 value (null,'test',0);
insert into article1 value(null,'test',1);
```

还有，每插入一个帖子，都希望将版面表中的最后发帖时间，帖子总数字段进行同步更新，用触发器做效率就很高。下次课设计这样一个案例，写触发器时，对于最后发帖时间可能需要用 declare 方式声明一个变量，或者是用 NEW.posttime 来生成。

6.4 数据库三范式是什么？

1NF:字段不可分;

2NF:有主键，非主键字段依赖主键;

3NF:非主键字段不能相互依赖;

6.5 说出一些数据库优化方面的经验?

用 PreparedStatement 一般来说比 Statement 性能高：一个 sql 发给服务器去执行，涉及步骤：语法检查、语义分析， 编译，缓存

“inert into user values(1,1,1)”-→二进制

“inert into user values(2,2,2)”-→二进制

“inert into user values(?,?,?)”-→二进制

有外键约束会影响插入和删除性能，如果程序能够保证数据的完整性，那在设计数据库时就去掉外键。（比喻：就好比免检产品，就是为了提高效率，充分相信产品的制造商）

（对于 hibernate 来说，就应该有一个变化：empleyee->Deptment 对象，现在设计时就成了 employee→deptid）

看 mysql 帮助文档子查询章节的最后部分，例如，根据扫描的原理，下面的子查询语句要比第二条关联查询的效率高：

1. select e.name,e.salary where e.managerid=(select id from employee where name='zxx');
2. select e.name,e.salary,m.name,m.salary from employees e,employees m where e.managerid = m.id and m.name='zxx';

表中允许适当冗余，譬如，主题帖的回复数量和最后回复时间等

将姓名和密码单独从用户表中独立出来。这可以是非常好的一对一的案例哟!

sql 语句全部大写，特别是列名和表名都大写。特别是 sql 命令的缓存功能，更加需要统一大小写，sql 语句→发给 oracle 服务器→语法检查和编译成为内部指令→缓存和执行指令。

根据缓存的特点，不要拼凑条件，而是用?和 PreparedStatment

还有索引对查询性能的改进也是值得关注的。

备注：下面是关于性能的讨论举例

4 航班 3 个城市

m*n

select * from flight,city where flight.startcityid=city.cityid and city.name='beijing';

m + n

select * from flight where startcityid = (select cityid from city where cityname='beijing');

select flight.id,'beijing',flight.flightTime from flight where startcityid = (select cityid from city where cityname='beijing')

6.6 union 和 union all 有什么不同?

假设我们有一个表 Student，包括以下字段与数据：

drop table student;

create table student

(

id int primary key,

name nvarchar2(50) not null,

score number not null

);

insert into student values(1,'Aaron',78);

insert into student values(2,'Bill',76);

insert into student values(3,'Cindy',89);

insert into student values(4,'Damon',90);

insert into student values(5,'Ella',73);

insert into student values(6,'Frado',61);

insert into student values(7,'Gill',99);

insert into student values(8,'Hellen',56);

insert into student values(9,'Ivan',93);

insert into student values(10,'Jay',90);

commit;

Union 和 Union All 的区别。

select *

from student

where id < 4

union

select *

from student

where id > 2 and id < 6

结果将是

1 Aaron 78

2 Bill 76

3 Cindy 89

4 Damon 90

5 Ella 73

如果换成 Union All 连接两个结果集，则返回结果是：

1 Aaron 78

2 Bill 76

3 Cindy 89

3 Cindy 89

4 Damon 90

5 Ella 73

可以看到，Union 和 Union All 的区别之一在于对重复结果的处理。

UNION 在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。实际大部分应用中是不会产生重复的记录，最常见的是过程表与历史表 UNION。如：

select * from gc_dfys

union

select * from ls_jg_dfys

这个 SQL 在运行时先取出两个表的结果，再用排序空间进行排序删除重复的记录，最后返回结果集，如果表数据量大的话可能会导致用磁盘进行排序。

而 UNION ALL 只是简单的将两个结果合并后就返回。这样，如果返回的两个结果集中有重复的数据，那么返回的结果集就会包含重复的数据了。

从效率上说，UNION ALL 要比 UNION 快很多，所以，如果可以确认合并的两个结果集

中不包含重复的数据的话，那么就使用 UNION ALL，

6.7 分页语句

取出 sql 表中第 31 到 40 的记录（以自动增长 ID 为主键）

sql server 方案 1：

```
    select top 10 * from t where id not in (select top 30 id from t order by id )
orde by id
```

sql server 方案 2：

```
    select top 10 * from t where id in (select top 40 id from t order by id) order
by id desc
```

mysql 方案：select * from t order by id limit 30,10

oracle 方案：select * from (select rownum r,* from t where r<=40) where r>30

----------待整理进去的内容----------

pageSize=20;

pageNo = 5;

1.分页技术 1（直接利用 sql 语句进行分页，效率最高和最推荐的）

mysql:sql = "select * from articles limit " + (pageNo-1)*pageSize + "," + pageSize;

```
oracle: sql = "select * from " +
                                        "(select rownum r,* from " +
                                            "(select * from articles order by postime desc)" +
                                        "where rownum<= " + pageNo*pageSize +") tmp " +
                                    "where r>" + (pageNo-1)*pageSize;
```

注释：第 7 行保证 rownum 的顺序是确定的，因为 oracle 的索引会造成 rownum 返回不同的值

简洋提示：没有 order by 时，rownum 按顺序输出，一旦有了 order by，rownum 不按顺序输出了，这说明 rownum 是排序前的编号。如果对 order by 从句中的字段建立了索引，那么，rownum 也是按顺序输出的，因为这时候生成原始的查询结果集时会参照索引表的顺序来构建。

sqlserver:sql = "select top 10 * from id not id(select top " + (pageNo-1)*pageSize + "id from articles)"

```
DataSource ds = new InitialContext().lookup(jndiurl);
Connection cn = ds.getConnection();
//"select * from user where id=?"    --->binary directive
PreparedStatement pstmt = cn.prepareSatement(sql);
ResultSet rs = pstmt.executeQuery()
while(rs.next())
{
      out.println(rs.getString(1));
}
```

2.不可滚动的游标

```
pageSize=20;
pageNo = 5;
cn = null
stmt = null;
rs = null;
try
{
sqlserver:sql = "select    * from articles";

DataSource ds = new InitialContext().lookup(jndiurl);
Connection cn = ds.getConnection();
```

```
//"select * from user where id=?"    --->binary directive
PreparedStatement pstmt = cn.prepareSatement(sql);
ResultSet rs = pstmt.executeQuery()
for(int j=0;j<(pageNo-1)*pageSize;j++)
{
    rs.next();
}

int i=0;

while(rs.next() && i<10)
{
    i++;
    out.println(rs.getString(1));
}
}
cacth(){}
finnaly
{
    if(rs!=null)try{rs.close();}catch(Exception e){}
    if(stm.....
    if(cn.....
}
```

3.可滚动的游标

```
pageSize=20;
pageNo = 5;
cn = null
stmt = null;
rs = null;
```

```
try
{
sqlserver:sql = "select    * from articles";

DataSource ds = new InitialContext().lookup(jndiurl);
Connection cn = ds.getConnection();
//"select * from user where id=?"    --->binary directive
PreparedStatement                                    pstmt                                    =
cn.prepareSatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,...);
//根据上面这行代码的异常 SQLFeatureNotSupportedException，就可判断驱动是否支持可滚动游标

ResultSet rs = pstmt.executeQuery()
rs.absolute((pageNo-1)*pageSize)
int i=0;
while(rs.next() && i<10)
{
      i++;
      out.println(rs.getString(1));
}
}
cacth(){}
finnaly
{
      if(rs!=null)try{rs.close();}catch(Exception e){}
      if(stm.....
      if(cn.....
}
```

6.8 用一条 SQL 语句 查询出每门课都大于 80 分的学生姓名

name kecheng fenshu

张三 语文 81

张三 数学 75

李四 语文 76

李四 数学 90

王五 语文 81

王五 数学 100

王五 英语 90

准备数据的 sql 代码：

```
create table score(id int primary key auto_increment,name varchar(20),subject varchar(20),score int);
insert into score values
(null,'张三','语文',81),
(null,'张三','数学',75),
(null,'李四','语文',76),
(null,'李四','数学',90),
(null,'王五','语文',81),
(null,'王五','数学',100),
(null,'王五 ','英语',90);
```

提示：当百思不得其解时，请理想思维，把小变成大做，把大变成小做，

答案：

A: select distinct name from score where name not in (select distinct name from score where score<=80)

B:select distince name t1 from score where 80< all (select score from score where name=t1);

6.9 所有部门之间的比赛组合

一个叫 department 的表，里面只有一个字段 name,一共有 4 条纪录，分别是 a,b,c,d,对应四个球对，现在四个球对进行比赛，用一条 sql 语句显示所有可能的比赛组合.

答：select a.name, b.name

from team a, team b

where a.name < b.name

6.10 统计每年每月的信息

year month amount

1991 1 1.1

1991 2 1.2

1991 3 1.3

1991 4 1.4

1992 1 2.1

1992 2 2.2

1992 3 2.3

1992 4 2.4

查成这样一个结果

year m1 m2 m3 m4

1991 1.1 1.2 1.3 1.4

1992 2.1 2.2 2.3 2.4

提示：这个与工资条非常类似，与学生的科目成绩也很相似。

准备 sql 语句：

```
drop table if exists sales;
create table sales(id int auto_increment primary key,year varchar(10), month varchar(10), amount float(2,1));
insert into sales values
(null,'1991','1',1.1),
```

(null,'1991','2',1.2),

(null,'1991','3',1.3),

(null,'1991','4',1.4),

(null,'1992','1',2.1),

(null,'1992','2',2.2),

(null,'1992','3',2.3),

(null,'1992','4',2.4);

答案一、

```
select sales.year ,
(select t.amount from sales t where t.month='1' and t.year= sales.year) '1',
(select t.amount from sales t where t.month='1' and t.year= sales.year) '2',
(select t.amount from sales t where t.month='1' and t.year= sales.year) '3',
(select t.amount from sales t where t.month='1' and t.year= sales.year) as '4'
from sales  group by year;
```

6.11 显示文章标题，发帖人、最后回复时间

表：id,title,postuser,postdate,parentid

准备 sql 语句：

```
drop table if exists articles;
create table articles(id int auto_increment primary key,title varchar(50), postuser varchar(10),
postdate datetime,parentid int references articles(id));
insert into articles values
(null,'第一条','张三','1998-10-10 12:32:32',null),
(null,'第二条','张三','1998-10-10 12:34:32',null),
(null,'第一条回复 1','李四','1998-10-10 12:35:32',1),
(null,'第二条回复 1','李四','1998-10-10 12:36:32',2),
(null,'第一条回复 2','王五','1998-10-10 12:37:32',1),
(null,'第一条回复 3','李四','1998-10-10 12:38:32',1),
(null,'第二条回复 2','李四','1998-10-10 12:39:32',2),
```

(null,'第一条回复 4','王五','1998-10-10 12:39:40',1);

答案：

select a.title,a.postuser,(select max(postdate) from articles where parentid=a.id) reply

from articles a where a.parentid is null;

注释：子查询可以用在选择列中，也可用于 where 的比较条件中，还可以用于 from 从句中。

6.12 查出比经理薪水还高的员工信息：

Drop table if not exists employees;

create table employees(id int primary key auto_increment,name varchar(50)

,salary int,managerid int references employees(id));

insert into employees values (null,' lhm',10000,null), (null,' zxx',15000,1

),(null,'flx',9000,1),(null,'tg',10000,2),(null,'wzg',10000,3);

Wzg 大于 flx,lhm 大于 zxx

解题思路：

根据 sql 语句的查询特点，是逐行进行运算，不可能两行同时参与运算。

涉及了员工薪水和经理薪水，所有，一行记录要同时包含两个薪水，所有想到要把这个表自关联组合一下。

首先要组合出一个包含有各个员工及该员工的经理信息的长记录，譬如，左半部分是员工，右半部分是经理。而迪卡尔积会组合出很多垃圾信息，先去除这些垃圾信息。

select e.* from employees e,employees m where e.managerid=m.id and e.sala

ry>m.salary;

6.13 一个用户表中有一个积分字段，假如数据库中有 100 多万个用户，若要在每年第一天凌晨将积分清零，你将考虑什么，你将想什么办法解决?

alter table drop column score;

alter table add colunm score int;

可能会很快，但是需要试验，试验不能拿真实的环境来操刀，并且要注意，

这样的操作时无法回滚的，在我的印象中，只有 inert update delete 等 DML 语句才能回滚，

对于 create table,drop table ,alter table 等 DDL 语句是不能回滚。

解决方案一，update user set score=0;

解决方案二，假设上面的代码要执行好长时间，超出我们的容忍范围，那我就 alter table user drop column score;alter table user add column score int。

下面代码实现每年的那个凌晨时刻进行清零。

```
Runnable runnable =
    new Runnable(){
        public void run(){
            clearDb();
            schedule(this,new Date(new Date().getYear()+1,0,0));
            }
            };
schedule(runnable,
    new Date(new Date().getYear()+1,0,1));
```

6.14 xxx 公司的 sql 面试

Table **EMPLOYEES** Structure:

EMPLOYEE_ID	NUMBER	Primary Key,
FIRST_NAME	VARCHAR2(25),	
LAST_NAME	VARCHAR2(25),	
Salary number(8,2),		
HiredDate DATE,		
Departmentid number(2)		

Table **Departments** Structure:

Departmentid number(2)	Primary Key,
DepartmentName	VARCHAR2(25).

（2）基于上述 EMPLOYEES 表写出查询：写出雇用日期在今年的，或者工资在[1000,2000]之间的，或者员工姓名（last_name）以’Obama’打头的所有员工，列出这些员工的全部个人信息。（4 分）

```
select * from employees
where Year(hiredDate) = Year(date())
    or (salary between 1000 and 200)
    or left(last_name,3)='abc';
```

(3) 基于上述 EMPLOYEES 表写出查询：查出部门平均工资大于 1800 元的部门的所有员工，列出这些员工的全部个人信息。（4 分）

```
mysql> select id,name,salary,deptid did from employee1 where (select avg(salary)
  from employee1 where deptid = did) > 1800;
```

(4) 基于上述 EMPLOYEES 表写出查询：查出个人工资高于其所在部门平均工资的员工，列出这些员工的全部个人信息及该员工工资高出部门平均工资百分比。（5 分）

```
select employee1.*,(employee1.salary-t.avgSalary)*100/employee1.salary
from employee1,
      (select deptid,avg(salary) avgSalary from employee1 group by deptid) as t
where employee1.deptid = t.deptid and employee1.salary>t.avgSalary;
```

6.15 注册 Jdbc 驱动程序的三种方式

第一种：通过 Class.forName("com.mysql.jdbc.Driver");//加载数据库驱动

第二种：通过系统属性 System.setProperty("jdbc.driver","com.mysql.jdbc.Driver");//系统属性指定数据库驱动。

第三种：看起来比较直观的一种方式，注册相应的 db 的 jdbc 驱动，在编译时需要导入对应的 lib。

6.16 用 JDBC 如何调用存储过程

代码如下：

```
package com.hzit.interview.lym;


import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Types;


public class JdbcTest {
```

```
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Connection cn = null;
        CallableStatement cstmt = null;
        try {
            //这里最好不要这么干，因为驱动名写死在程序中了
            Class.forName("com.mysql.jdbc.Driver");
            //实际项目中，这里应用DataSource数据，如果用框架，
            //这个数据源不需要我们编码创建，我们只需Datasource ds = context.lookup()
            //cn = ds.getConnection();
            cn =
DriverManager.getConnection("jdbc:mysql:///test","root","root");
            cstmt = cn.prepareCall("{call insert_Student(?,?,?)}");
            cstmt.registerOutParameter(3,Types.INTEGER);
            cstmt.setString(1, "wangwu");
            cstmt.setInt(2, 25);
            cstmt.execute();
            //get第几个，不同的数据库不一样，建议不写
            System.out.println(cstmt.getString(3));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        finally
        {
```

```
            /*try{cstmt.close();}catch(Exception e){}
            try{cn.close();}catch(Exception e){}*/
            try {
                if(cstmt != null)
                    cstmt.close();
                if(cn != null)
                    cn.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

6.17 JDBC 中的 PreparedStatement 相比 Statement 的好处

答：一个 sql 命令发给服务器去执行的步骤为：语法检查，语义分析，编译成内部指令，缓存指令，执行指令等过程。

select * from student where id =3----缓存--→xxxxx 二进制命令

select * from student where id =3----直接取-→xxxxx 二进制命令

select * from student where id =4--- -→会怎么干？

如果当初是 select * from student where id =?--- -→又会怎么干？

上面说的是性能提高

可以防止 sql 注入。

6.18 写一个用 jdbc 连接并访问 oracle 数据的程序代码

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Statement;


public class JDBCDemo {

    public static void main(String[] args){

        Connection conn = null;

        ResultSet rs = null;

        try {

            //加载驱动

            Class.forName("oracle.jdbc.OracleDriver");

            //获取链接

            conn =

DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",

"scott","tiger" );

            Statement stat = conn.createStatement();

            //sql 语句

            String sql = "SELECT * FROM emp";

            //执行语句获得结果集

            rs = stat.executeQuery(sql);

            //遍历结果集

            while(rs.next()){

                String name = rs.getString("name");

                System.out.println(name);

            }


        } catch (ClassNotFoundException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        } catch (SQLException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();
```

```
        } finally {
        //关闭连接
        try {
            conn.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

6.19 Class.forName 的作用?为什么要用?

答：按参数中指定的字符串形式的类名去搜索并加载相应的类，如果该类字节码已经被加载过，则返回代表该字节码的 Class 实例对象，否则，按类加载器的委托机制去搜索和加载该类，如果所有的类加载器都无法加载到该类，则抛出 ClassNotFoundException。加载完这个 Class 字节码后，接着就可以使用 Class 字节码的 newInstance 方法去创建该类的实例对象了。有时候，我们程序中所有使用的具体类名在设计时（即开发时）无法确定，只有程序运行时才能确定，这时候就需要使用 Class.forName 去动态加载该类，这个类名通常是在配置文件中配置的，例如，spring 的 ioc 中每次依赖注入的具体类就是这样配置的，jdbc 的驱动类名通常也是通过配置文件来配置的，以便在产品交付使用后不用修改源程序就可以更换驱动类名。

6.20 大数据量下的分页解决方法。

答：最好的办法是利用 sql 语句进行分页，这样每次查询出的结果集中就只包含某页的数据内容。再 sql 语句无法实现分页的情况下，可以考虑对大的结果集通过游标定位方式来获取某页的数据。

sql 语句分页，不同的数据库下的分页方案各不一样，下面是主流的三种数据库的分页 sql：

sql server:

String sql =

"select top " + pageSize + " * from students where id not in" +

"(select top " + pageSize * (pageNumber-1) + " id from students order by id)" +

"order by id";

mysql:

String sql =

"select * from students order by id limit " + pageSize*(pageNumber-1) + "," + pageSize;

oracle:

String sql =

"select * from " +

(select *,rownum rid from (select * from students order by postime desc) where rid<=" + pagesize*pagenumber + ") as t" +

"where t>" + pageSize*(pageNumber-1);

6.21 用 JDBC 查询学生成绩单，把主要代码写出来（考试概率极大）.

```
Connection cn = null;
PreparedStatement pstmt =null;
Resultset rs = null;
try
{
    Class.forname(driveClassName);
    cn =  DriverManager.getConnection(url,username,password);
    pstmt = cn.prepareStatement("select  score.* from score ,student " +
        "where score.stuId = student.id and student.name = ?");
    pstmt.setString(1,studentName);
    Resultset rs = pstmt.executeQuery();
```

```
        while(rs.next())
        {
            system.out.println(rs.getInt("subject")  +  "      " + rs.getFloat("score") );
        }
}catch(Exception e){e.printStackTrace();}
finally
{
        if(rs != null) try{ rs.close() }catch(exception e){}
        if(pstmt != null) try{pstmt.close()}catch(exception e){}
        if(cn != null) try{ cn.close() }catch(exception e){}
}
```

6.22 这段代码有什么不足之处?

```
try {
Connection conn = ...;
Statement stmt = ...;

ResultSet rs = stmt.executeQuery("select * from table1");

while(rs.next()) {

}
} catch(Exception ex) {
}
```

答：没有 finally 语句来关闭各个对象，另外，使用 finally 之后，要把变量的定义放在 try 语句块的外面，以便在 try 语句块之外的 finally 块中仍可以访问这些变量。

6.23 说出数据连接池的工作机制是什么？

J2EE 服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量有配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

实现方式，返回的 Connection 是原始 Connection 的代理，代理 Connection 的 close 方法不是真正关连接，而是把它代理的 Connection 对象还回到连接池中。

6.24 为什么要用 ORM？ 和 JDBC 有何不一样？

orm 是一种思想，就是把 object 转变成数据库中的记录，或者把数据库中的记录转变成 objecdt，我们可以用 jdbc 来实现这种思想，其实，如果我们的项目是严格按照 oop 方式编写的话，我们的 jdbc 程序不管是有意还是无意，就已经在实现 orm 的工作了。

现在有许多 orm 工具，它们底层调用 jdbc 来实现了 orm 工作，我们直接使用这些工具，就省去了直接使用 jdbc 的繁琐细节，提高了开发效率，现在用的较多的 orm 工具是 hibernate。也听说一些其他 orm 工具，如 toplink,ojb 等。

七、XML 部分

7.1 xml 有哪些解析技术?区别是什么?

答:有 DOM,SAX,STAX 等

DOM:处理大型文件时其性能下降的非常厉害。这个问题是由 DOM 的树结构所造成的，这种结构占用的内存较多，而且 DOM 必须在解析文件之前把整个文档装入内存，适合对 XML 的随机访问 SAX:不现于 DOM,

SAX 是事件驱动型的 XML 解析方式。它顺序读取 XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理 XML 文件，适合对 XML 的顺序访问

STAX:Streaming API for XML (StAX) 。

7.2 你在项目中用到了 xml 技术的哪些方面?如何实现的?

答:用到了数据存贮，信息配置两方面。

在做数据交换平台时，将不能数据源的数据组装成 XML 文件，然后将 XML 文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同 XML 文件中还原相关信息进行处理。在做软件配置时，利用 XML 可以很方便的进行，软件的各种配置参数都存贮在 XML 文件中。

7.3 XML 文档定义有几种形式？它们之间有何本质区别？解析 XML 文档有哪几种方式？

a: 两种形式 dtd schema，

b: 本质区别:schema 本身是 xml 的,可以被 XML 解析器解析(这也是从 DTD 上发展 schema 的根本目的)，

c:有 DOM,SAX,STAX 等

- DOM:处理大型文件时其性能下降的非常厉害。这个问题是由 DOM 的树结构所造成的，这种结构占用的内存较多，而且 DOM 必须在解析文件之前把整个文档装入内存,适合对 XML 的随机访问
- SAX:不现于 DOM,SAX 是事件驱动型的 XML 解析方式。它顺序读取 XML 文件，不需要

一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理 XML 文件，适合对 XML 的顺序访问

- STAX:Streaming API for XML (StAX)

八、流行的框架与新技术

8.1 谈谈你对 Spring 的理解。

1.Spring 实现了**工厂模式的工厂类**（在这里有必要解释清楚什么是工厂模式），这个类名为 BeanFactory（实际上是一个接口），在程序中通常 BeanFactory 的子类 ApplicationContext。Spring 相当于一个大的工厂类，在其配置文件中通过<bean>元素配置用于创建实例对象的类名和实例对象的属性。

2. Spring 提供了对 **IOC** 良好支持，IOC 是一种编程思想，是一种架构艺术，利用这种思想可以很好地实现模块之间的解耦。IOC 也称为 DI（Depency Injection），什么叫依赖注入呢？譬如，

```
class Programmer {
    Computer computer = null;
    public void code(){
        //Computer computer = new IBMComputer();
        //Computer computer = beanfacotry.getComputer();
        computer.write();
    }
    public void setComputer(Computer computer)
    {
        this.computer = computer;
    }
}
```

另外两种方式都由依赖，第一个直接依赖于目标类，第二个把依赖转移到工厂上，第三个彻底与目标和工厂解耦了。在 spring 的配置文件中配置片段如下：

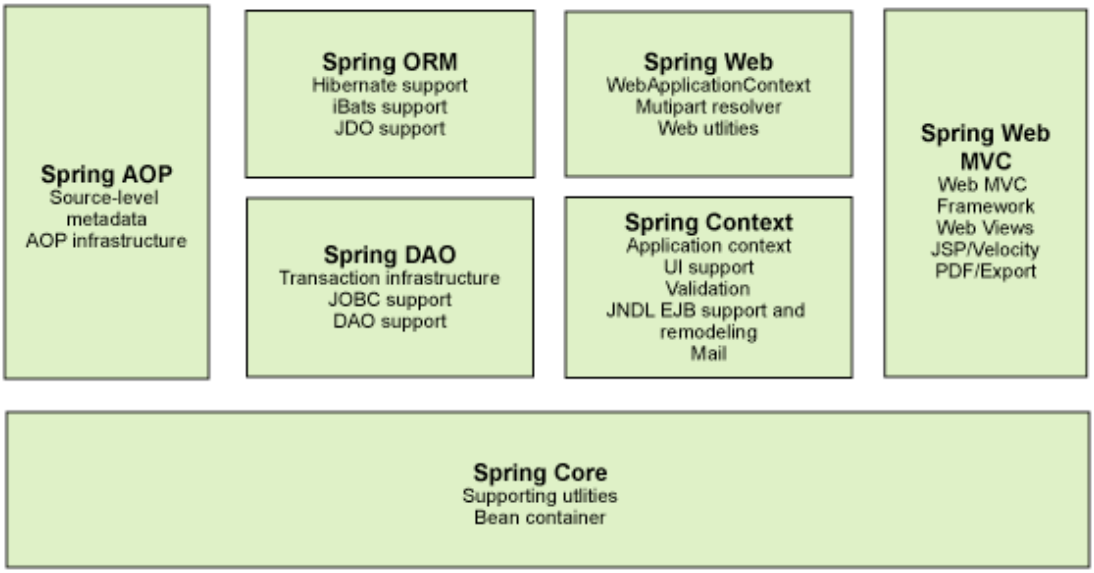
```
<bean id=”computer” class=”cn.itcast.interview.Computer”>
</bean>
<bean id=”programmer” class=”cn.itcast.interview.Programmer”>
    <property name=”computer” ref=”computer”></property>
</bean>
```

3. Spring 提供了对 AOP 技术的良好封装， AOP 称为**面向切面编程**，就是系统中有很多各不相干的类的方法，在这些众多方法中要加入某种系统功能的代码，例如，**加入日志，加入权限判断，加入异常处理**，这种应用称为 AOP。实现 AOP 功能采用的是代理技术，客户端程序不再调用目标，而调用代理类，代理类与目标类对外具有相同的方法声明，有两种方式可以实现相同的方法声明，一是实现相同的接口，二是作为目标的子类在，JDK 中采用 Proxy 类产生动态代理的方式为某个接口生成实现类，如果要为某个类生成子类，则可以用 CGLI B。在生成的代理类的方法中加入系统功能和调用目标类的相应方法，系统功能的代理以 Advice 对象进行提供，显然要创建出代理对象，至少需要目标类和 Advice 类。spring 提供了这种支持，只需要在 spring 配置文件中配置这两个元素即可实现代理和 aop 功能，例如，

```
<bean id=”proxy” type=”org.spring.framework.aop.ProxyBeanFactory”>
    <property name=”target” ref=””></property>
    <property name=”advisor” ref=””></property>
</bean>
```

8.2 什么是 Spring 框架？Spring 框架有哪些主要模块？

Spring 框架是一个为 Java 应用程序的开发提供了**综合、广泛**的基础性支持的 Java 平台。Spring 帮助开发者解决了开发中基础性的问题，使得开发人员可以专注于应用程序的开发。Spring 框架本身亦是按照设计模式精心打造，这使得我们可以在开发环境中安心的集成 Spring 框架，不必担心 Spring 是如何在后台进行工作的。



Spring 框架至今已集成了 20 多个模块。这些模块主要被分如下图所示的核心容器、数

据访问/集成、Web、AOP（面向切面编程）、工具、消息和测试模块。

8.3 使用 Spring 框架能带来哪些好处？

下面列举了一些使用 Spring 框架带来的主要好处：

Dependency Injection(DI) 方法使得构造器和 JavaBean properties 文件中的依赖关系一目了然。

与 EJB 容器相比较，IoC 容器更加趋向于轻量级。这样一来 IoC 容器在有限的内存和 CPU 资源的情况下进行应用程序的开发和发布就变得十分有利。

Spring 并没有闭门造车，Spring 利用了已有的技术比如 ORM 框架、logging 框架、J2EE、Quartz 和 JDK Timer，以及其他视图技术。

Spring 框架是按照模块的形式来组织的。由包和类的编号就可以看出其所属的模块，开发者仅仅需要选用他们需要的模块即可。

要测试一项用 Spring 开发的应用程序十分简单，因为测试相关的环境代码都已经囊括在框架中了。更加简单的是，利用 JavaBean 形式的 POJO 类，可以很方便的利用依赖注入来写入测试数据。

Spring 的 Web 框架亦是一个精心设计的 Web MVC 框架，为开发者们在 web 框架的选择上提供了一个除了主流框架比如 Struts、过度设计的、不流行 web 框架的以外的有力选项。

Spring 提供了一个便捷的事务管理接口，适用于小型的本地事物处理（比如在单 DB 的环境下）和复杂的共同事物处理（比如利用 JTA 的复杂 DB 环境）。

8.4 什么是控制反转(IOC)？什么是依赖注入？

控制反转是应用于软件工程领域中的，在运行时被装配器对象来绑定耦合对象的一种编程技巧，对象之间耦合关系在编译时通常是未知的。在传统的编程方式中，业务逻辑的流程是由应用程序中的早已被设定好关联关系的对象来决定的。在使用控制反转的情况下，业务逻辑的流程是由对象关系图来决定的，该对象关系图由装配器负责实例化，这种实现方式还可以将对象之间的关联关系的定义抽象化。而绑定的过程是通过“依赖注入”实现的。

控制反转是一种以给予应用程序中目标组件更多控制为目的设计范式，并在我们的实际工作中起到了有效的作用。

依赖注入是在编译阶段尚未知所需的功能是来自哪个的类的情况下，将其他对象所依赖的功能对象实例化的模式。这就需要一种机制用来激活相应的组件以提供特定的功能，所以依赖注入是控制反转的基础。否则如果在组件不受框架控制的情况下，框架又怎么知道要创

建哪个组件？

在 Java 中依然注入有以下三种实现方式：

- 构造器注入
- Setter 方法注入
- 接口注入

8.5 请解释下 Spring 框架中的 IoC？

Spring 中的 org.springframework.beans 包和 org.springframework.context 包构成了 Spring 框架 IoC 容器的基础。

BeanFactory 接口提供了一个先进的配置机制，使得任何类型的对象的配置成为可能。

ApplicationContex 接口对 BeanFactory（是一个子接口）进行了扩展，在 BeanFactory 的基础上添加了其他功能，比如与 Spring 的 AOP 更容易集成，也提供了处理 message resource 的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对 Web 应用的 WebApplicationContext。

org.springframework.beans.factory.BeanFactory 是 Spring IoC 容器的具体实现，用来包装和管理前面提到的各种 bean。BeanFactory 接口是 Spring IoC 容器的核心接口。

8.6 BeanFactory 和 ApplicationContext 有什么区别？

BeanFactory 可以理解为含有 bean 集合的工厂类。BeanFactory 包含了种 bean 的定义，以便在接收到客户端请求时将对应的 bean 实例化。

BeanFactory 还能在实例化对象的时生成协作类之间的关系。此举将 bean 自身与 bean 客户端的配置中解放出来。BeanFactory 还包含了 bean 生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。

从表面上看，application context 如同 bean factory 一样具有 bean 定义、bean 关联关系的设置，根据请求分发 bean 的功能。但 application context 在此基础上还提供了其他的功能。

1.提供了支持国际化的文本消息

2.统一的资源文件读取方式

3.已在监听器中注册的 bean 的事件

以下是三种较常见的 **ApplicationContext** 实现方式：

1、**ClassPathXmlApplicationContext**：从 classpath 的 XML 配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中取得。

```
1  ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");
```

2、**FileSystemXmlApplicationContext** ：由文件系统中的 XML 配置文件读取上下文。

```
1  ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");
```

3、**XmlWebApplicationContext**：由 Web 应用的 XML 文件读取上下文。

8.7 谈谈你对 Struts1 的理解。

1. struts 是一个按 MVC 模式设计的 Web 层框架，其实它就是一个大大的 servlet，这个 Servlet 名为 ActionServlet，或是 ActionServlet 的子类。我们可以在 web.xml 文件中将符合某种特征的所有请求交给这个 Servlet 处理，这个 Servlet 再参照一个配置文件（通常为 /WEB-INF/struts-config.xml）将各个请求分别分配给不同的 action 去处理。

一个扩展知识点：struts 的配置文件可以有多个，可以按模块配置各自的配置文件，这样可以防止配置文件的过度膨胀；

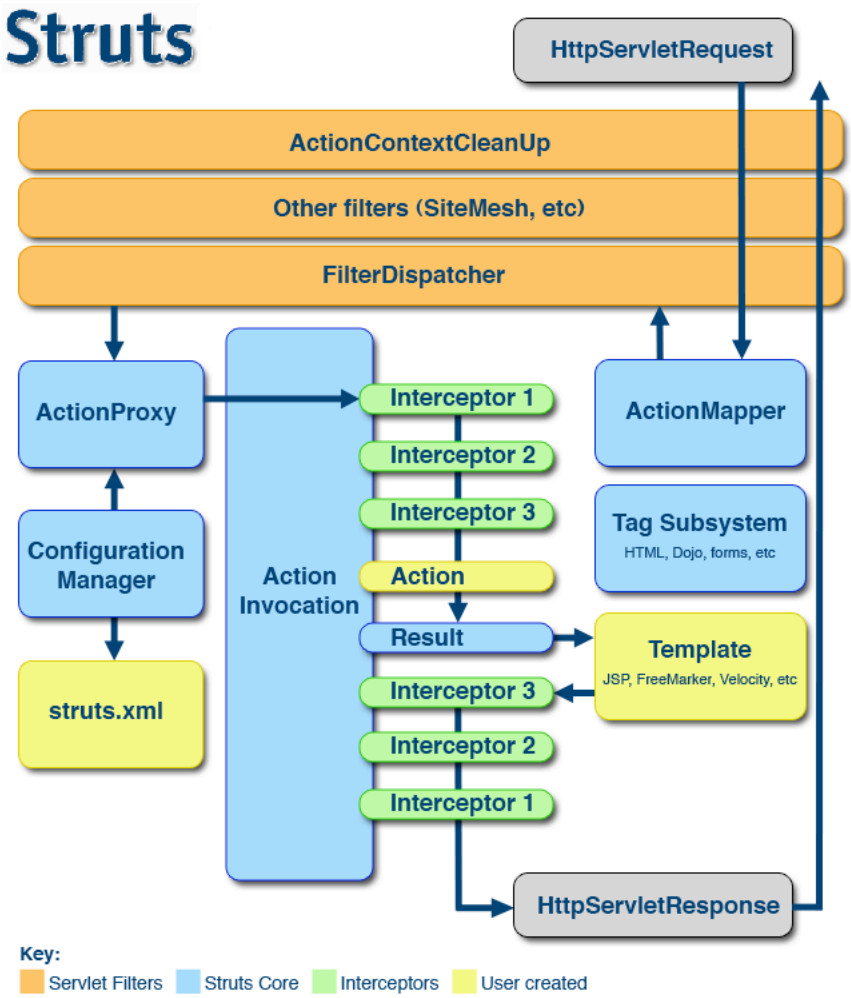
2. ActionServlet 把请求交给 action 去处理之前，会将请求参数封装成一个 formbean 对象（就是一个 java 类，这个类中的每个属性对应一个请求参数），封装成一个什么样的 formbean 对象呢？看配置文件。

3.要说明的是， ActionServlet 把 formbean 对象传递给 action 的 execute 方法之前，可能会调用 formbean 的 validate 方法进行校验，只有校验通过后才将这个 formbean 对象传递给 action 的 execute 方法，否则，它将返回一个错误页面，这个错误页面由 input 属性指定，（看配置文件）作者为什么将这里命名为 input 属性，而不是 error 属性，我们后面结合实际的运行效果进行分析。

4.action 执行完后要返回显示的结果视图，这个结果视图是用一个 ActionForward 对象来表示的，actionforward 对象通过 struts-config.xml 配置文件中的配置关联到某个 jsp 页面，因为程序中使用的是在 struts-config.xml 配置文件为 jsp 页面设置的逻辑名，这样可以实现 action 程序代码与返回的 jsp 页面名称的解耦。

你对 struts 可能还有自己的应用方面的经验，那也要一并说出来。

8.8 Struts2 工作流程概述。



一个请求在 Struts2 框架中的处理分为以下几个步骤：

1. 客户端发出一个指向 servlet 容器的请求(tomcat).

2. 这个请求会经过图中的几个过滤器，最后会到达 FilterDispatcher 过滤器。

3. 过滤器 FilterDispatcher 是 struts2 框架的心脏，在处理用户请求时，它和请求一起相互配合访问 struts2 的底层框架结构。**在 web 容器启动时，struts2 框架会自动加载配置文件里相关参数，并转换成相应的类。** 如：ConfigurationManager、ActionMapper 和 ObjectFactory。ConfigurationManager 存有配置文件的一些基本信息，ActionMapper 存有 action 的配置信息。在请求过程中所有的对象（Action，Results， Interceptors，等）都是通过 ObjectFactory 来创建的。过滤器会通过询问 ActionMapper 类来查找请求中需要用到的 Action。

4. 如果找到需要调用的 Action，过滤器会把请求的处理交给 ActionProxy。ActionProxy 为 Action 的代理对象。ActionProxy 通过 ConfigurationManager 询问框架的配置文件，找

到需要调用的 Action 类。

5. ActionProxy 创建一个 ActionInvocation 的实例。ActionInvocation 在 ActionProxy 层之下，它表示了 Action 的执行状态,或者说它控制的 Action 的执行步骤。它持有 Action 实例和所有的 Interceptor。

6. ActionInvocation 实例使用命名模式来调用，1. ActionInvocation 初始化时，根据配置，加载 Action 相关的所有 Interceptor。2. 通过 ActionInvocation.invoke 方法调用 Action 实现时，执行 Interceptor。在调用 Action 的过程前后，涉及到相关拦截器(intercepetor)的调用。

7. 一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。返回结果通常是（但不总是，也可能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模版。在表示的过程中可以使用 Struts2 框架中继承的标签。

Struts 优缺点

优点：

1.实现 MVC 模式，结构清晰,使开发者只关注业务逻辑的实现;

2.有丰富的 tag 可以用 ,Struts 的标记库(Taglib)，如能灵活动用，则能大大提高开发效率;

3.页面导航 使系统的脉络更加清晰。通过一个配置文件，即可把握整个系统各部分之间的联系，这对于后期的维护有着莫大的好处。尤其是当另一批开发者接手这个项目时，这种优势体现得更加明显。

4. 提供 Exception 处理机制

5. 数据库链接池管理

6. 支持 I18N(国际化)

缺点：

1.转到展示层时，需要配置 forward，如果有十个展示层的 jsp，需要配置十次 struts，而且还不包括有时候目录、文件变更，需要重新修改 forward，注意，每次修改配置之后，要求重新部署整个项目，而 tomcate 这样的服务器，还必须重新启动服务器。

2.Struts 的 Action 必需是 thread－safe(线程)方式，它仅仅允许一个实例去处理所有的请求。所以 action 用到的所有的资源都必需统一同步，这个就引起了线程安全的问题。

3.测试不方便. Struts 的每个 Action 都同 Web 层耦合在一起，这样它的测试依赖于 Web 容器，单元测试也很难实现。不过有一个 Junit 的扩展工具 Struts TestCase 可以实现它的单元测试。

4.类型的转换. Struts 的 FormBean 把所有的数据都作为 String 类型，它可以使用工具 Commons-Beanutils 进行类型转化。但它的转化都是在 Class 级别，而且转化的类型是不可配置的。类型转化时的错误信息返回给用户也是非常困难的。

5.对 Servlet 的依赖性过强. Struts 处理 Action 时必需要依赖 ServletRequest 和 ServletResponse，所有它摆脱不了 Servlet 容器。

6.前端表达式语言方面.Struts 集成了 JSTL，所以它主要使用 JSTL 的表达式语言来获取数据。可是 JSTL 的表达式语言在 Collection 和索引属性方面处理显得很弱。

7.对 Action 执行的控制困难. Struts 创建一个 Action，如果想控制它的执行顺序将会非常困难。甚至你要重新去写 Servlet 来实现你的这个功能需求。

8.对 Action 执行前和后的处理. Struts 处理 Action 的时候是基于 class 的 hierarchies，很难在 action 处理前和后进行操作。

9.对事件支持不够. 在 struts 中，实际是一个表单 Form 对应一个 Action 类(或 DispatchAction)，换一句话说：在 Struts 中实际是一个表单只能 对应一个事件，struts 这种事件方式称为 application event，application event 和 component event 相比是一种粗粒度的事件。

8.9 Struts 的应用(如 Struts 架构)

Struts 是采用 Java Servlet/JavaServer Pages 技术，开发 Web 应用程序的开放源码的 framework。 采用 Struts 能开发出**基于 MVC(Model-View-Controller)设计模式**的应用构架。

Struts 有如下的主要功能： 一.包含一个 controller servlet，能将用户的请求发送到相应的 Action 对象。 二.JSP 自由 tag 库，并且在 controller servlet 中提供关联支持，帮助开发员创建交互式表单应用。 三.提供了一系列实用对象：XML 处理、通过 Java reflection APIs 自动处理 JavaBeans 属性、国际化的提示和消息。

8.10 说说 struts1 与 struts2 的区别。

1. 都是 MVC 的 WEB 框架。

2. Struts1 的老牌框架，应用很广泛，有很好的群众基础，使用它开发风险很小，成本更低！Struts2 虽然基于这个框架，但是应用群众并多，相对不成熟，未知的风险和变化很多，开

发人员相对不好招，使用它开发项目的风险系数更大，用人成本更高！

3.Struts2 毕竟是站在前辈的基础设计出来，它会改善和完善 Struts1 中的一些缺陷，Struts1 中一些悬而未决问题在 Struts2 得到了解决。

4.Struts1 的前端控制器是一个 Servlet，名称为 ActionServlet，Struts2 的前端控制器是一个 filter，在 Struts2.0 中叫 FilterDispatcher，在 Struts2.1 中叫 StrutsPrepareAndExecuteFilter。

5.Struts1 的 action 需要继承 Action 类，Struts2 的 action 可以不继承任何类；Struts1 对同一个路径的所有请求共享一个 Action 实例，Struts2 对同一个路径的每个请求分别使用一个独立 Action 实例对象，所有对于 struts2 的 Action 不用考虑线程安全问题。

6.在 Struts1 中使用 formbean 封装请求参数，在 Struts2 中直接使用 action 的属性来封装请求参数。

7.Struts1 中的多个业务方法放在一个 Action 中时（即继承 DispatchAction 时），要么都校验，要么都不校验；对于 Struts2，可以指定只对某个方法进行校验，当一个 Action 继承了 ActionSupport 且在这个类中只编写了 validateXxx()方法，那么则只对 Xxx()方法进行校验。

（一个请求来了的执行流程进行分析，Struts2 是自动支持分模块开发，并可以不同模块设置不同的 url 前缀，这是通过 package 的 namespace 来实现的；struts2 是支持多种类型的视图；struts2 的视图地址可以是动态的，即视图的名称是支持变量方式的，举例，论坛发帖失败后回来还要传递 boardid。视图内容显示方面：它的标签用 ognl，要 el 强大很多，在国际化方面支持分模块管理，两个模块用到同样的 key，对应不同的消息；）

与 Struts1 不同，Struts2 对用户的每一次请求都会创建一个 Action，所以 Struts2 中的 Action 是线程安全的。

给我印象最深刻的是：struts 配置文件中的 redirect 视图的 url 不能接受参数，而 struts2 配置文件中的 redirect 视图可以接受参数。

8.11 谈谈你对 Hibernate 的理解。

hibernate 是一个 ORM 框架，是对 JDBC 的封装。目的就是简化对数据库表访问的操作。

ORM 的意思是对象关系的映射，通过实体类与数据表建立映射，就可以通过持久层操

作来代替 sql 语句操作数据库。

hibernate 的核心原理就是对象关系映射（ORM），就是通过 java 的反射机制来实现。

java 反射机制：在 java 运行状态中，对于任何一个类，都能够知道这个类的所有属性和方法，就是对于任意一个对象都能够调用他的任意属性和方法，这种动态获取信息以及动态调用的方法就是功能就是 java 的反射机制。

以上这些就是 hibernate 的核心原理。

1. 使用 Hibernate 的基本流程是：配置 Configuration 对象、产生 SessionFactory、创建 session 对象，启动事务，完成 CRUD 操作，提交事务，关闭 session。

2. 使用 Hibernate 时，先要配置 hibernate.cfg.xml 文件，其中配置数据库连接信息和方言等，还要为每个实体配置相应的 hbm.xml 文件，hibernate.cfg.xml 文件中需要登记每个 hbm.xml 文件。

3. 在应用 Hibernate 时，重点要了解 Session 的缓存原理，级联，延迟加载和 hql 查询。

8.12 hibernate 中的 update()和 saveOrUpdate()的区别，session 的 load()和 get()的区别。

saveorupdate()如果传入的对象在数据库中有就做 update 操作，如果没有就做 save 操作。

save()在数据库中生成一条记录，如果数据库中有，会报错说有重复的记录。

hibernate 中 **get** 方法和 **load** 方法的根本区别:

如果你使用 **load 方法**，hibernate 认为该 id 对应的对象（数据库记录）在数据库中是一定存在的，所以它可以放心的使用，它可以放心的使用代理来延迟加载该对象。在用到对象中的其他属性数据时才查询数据库，但是万一数据库中不存在该记录，那没办法，只能抛异常 ObjectNotFoundException，所说的 load 方法抛异常是指在使用该对象的数据时，数据库中不存在该数据时抛异常，而不是在创建这个对象时。由于 session 中的缓存对于 hibernate 来说是个相当廉价的资源，所以在 load 时会先查一下 session 缓存看看该 id 对应的对象是否存在，不存在则创建代理。所以如果你知道该 id 在数据库中一定有对应记录存
在就可以使用 load 方法来实现延迟加载。

对于 **get 方法**，hibernate 会确认一下该 id 对应的数据是否存在，首先在 session 缓存中查找，然后在二级缓存中查找，还没有就查数据库，数据库中没有就返回 null。

虽然好多书中都这么说：“get()永远只返回实体类”，但实际上这是不正确的，get 方法如果在 session 缓存中找到了该 id 对应的对象，如果刚好该对象前面是被代理过的，如被 load 方法使用过，或者被其他关联对象延迟加载过，那么返回的还是原先的代理对象，而不是实

体类对象，如果该代理对象还没有加载实体数据（就是 id 以外的其他属性数据），那么它会查询二级缓存或者数据库来加载数据，但是返回的还是代理对象，只不过已经加载了实体数据。

前面已经讲了，get 方法首先查询 session 缓存，没有的话查询二级缓存，最后查询数据库；反而 load 方法创建时首先查询 session 缓存，没有就创建代理，实际使用数据时才查询二级缓存和数据库。

总之对于 get 和 load 的根本区别，一句话，hibernate 对于 load 方法认为该数据在数据库中一定存在，可以放心的使用代理来延迟加载，如果在使用过程中发现了问题，就抛异常；而对于 get 方法，hibernate 一定要获取到真实的数据，否则返回 null。

8.13 简述 Hibernate 和 JDBC 的优缺点？如何书写一个 one to many 配置文件。

JDBC 与 **hibernate** 在性能上相比，JDBC 灵活性有优势。而 Hibernate 在易学性，易用性上有些优势。当用到很多复杂的多表联查和复杂的**数据库**操作时，JDBC 有优势。

相同点：

◆两者都是 **Java** 的数据库操作中间件。

◆两者对于数据库进行直接操作的对象都不是线程安全的，都需要及时关闭。

◆两者都可以对数据库的更新操作进行显式的事务处理。

不同点：

◆使用的 SQL 语言不同：JDBC 使用的是基于关系型数据库的标准 SQL 语言，Hibernate 使用的是 HQL(Hibernate query language)语言

◆操作的对象不同：JDBC 操作的是数据，将数据通过 SQL 语句直接传送到数据库中执行，Hibernate 操作的是持久化对象，由底层持久化对象的数据更新到数据库中。

◆数据状态不同：JDBC 操作的数据是“瞬时”的，变量的值无法与数据库中的值保持一致，而 Hibernate 操作的数据是可持久的，即持久化对象的数据属性的值是可以跟数据库中的值保持一致的。

JDBC 与 Hibernate 读取性能

1、JDBC 仍然是最快的访问方式，不论是 Create 还是 Read 操作，都是 JDBC 快。

2、Hibernate 使用 uuid.hex 构造主键，性能稍微有点损失，但是不大。

3、Create 操作，JDBC 在使用批处理的方式下速度比 Hibernate 快，使用批处理方式耗用 JVM 内存比不使用批处理方式要多得多。

4、读取数据，Hibernate 的 Iterator 速度非常缓慢，因为他是每次 next 的时候才去数据库取数据，这一点从观察任务管理器的 java 进程占用内存的变化也可以看得很清楚，内存是几十 K 几十 K 的增加。

5、读取数据，Hibernate 的 List 速度很快，因为他是一次性把数据取完，这一点从观察任务管理器的 java 进程占用内存的变化也可以看得很清楚，内存几乎是 10M 的 10M 的增加。

6、JDBC 读取数据的方式和 Hibernate 的 List 方式是一样的（这跟 JDBC 驱动有很大关系，不同的 JDBC 驱动，结果会很不一样），这从观察 java 进程内存变化可以判断出来，由于 JDBC 不需要像 Hibernate 那样构造一堆 Cat 对象实例，所以占用 JVM 内存要比 Hibernate 的 List 方式大概少一半左右。

7、Hibernate 的 Iterator 方式并非一无是处，它适合于从大的结果集中选取少量的数据，即不需要占用很多内存，又可以迅速得到结果。另外 Iterator 适合于使用 JCS 缓冲。

8.14 写 Hibernate 的一对多和多对一双向关联的 orm 配置?

one to many 比如 Class(班级)和 Student(学生)，就是一个班级对应多个学生

在 Class 类中追加集合属性 Set<Student> students;

在 Class 的配置文件中追加（Class.hbm.xml）

```
<!-- 追加集合属性的配置 -->
<!-- 设置集合属性 -->
<set name="students" lazy="false" fetch="join" cascade="all" inverse="true">
        <!-- 设置关联字段 -->
        <key column="classId" />
        <!-- 设置关联关系 -->
        <one-to-many class="Student" />
</set>
```

将 Student 中的 classId 属性去掉换成 Class class;

在 Student 的配置文件中（Student.hbm.xml）

```
<many-to-one name="class" column="classId" lazy="false" fetch="join"
                                    class="Class">
</many-to-one>
```

8.15 hibernate 的 inverse 属性的作用?

inverse 属性，是在维护关联关系的时候起作用的。表示控制权是否转移。(在一的一方起作用)

inverse , 控制反转。inverse = false 不反转，当前方有控制权；true 控制反转，当前方没有控制权。

维护关联关系中，是否设置 inverse 属性：

1.保存数据 有影响。

如果设置控制反转,即 inverse=true, 然后通过部门方维护关联关系。在保存部门的时候，同时保存员工，数据会保存，但关联关系不会维护。即外键字段为 NULL。

2.获取数据 无。

3.解除关联关系？有影响。

inverse = false， 可以解除关联

inverse = true， 当前方(部门)没有控制权，不能解除关联关系(不会生成 update 语句,也不会报错)

4.删除数据对关联关系的影响？有影响。

inverse=false，有控制权。可以删除。先清空外键引用，再删除数据。

inverse=true，没有控制权。如果删除的记录有被外键引用，会报错，违反主外键引用约束！ 如果删除的记录没有被引用，可以直接删除。

8.16 hibernate 进行多表查询每个表中各取几个字段，也就是说查询出来的结果集没有一个实体类与之对应如何解决；

解决方案一，按照 Object[]数据取出数据，然后自己组 bean。

解决方案二，对每个表的 bean 写构造函数，比如表一要查出 field1,field2 两个字段，那么有一个构造函数就是 Bean(type1 filed1,type2 field2) ，然后在 hql 里面就可以直接生成这个 bean 了。

8.17 介绍一下 Hibernate 的二级缓存

按照以下思路来回答：（1）首先说清楚什么是缓存，（2）再说有了 hibernate 的 Session 就是一级缓存，即有了一级缓存，为什么还要有二级缓存，（3）最后再说如何配置 Hibernate 的二级缓存。

（1）缓存就是把以前从数据库中查询出来和使用过的对象保存在内存中（一个数据结构中），这个数据结构通常是或类似 Hashmap，当以后要使用某个对象时，先查询缓存中是否有这个对象，如果有则使用缓存中的对象，如果没有则去查询数据库，并将查询出来的对象保存在缓存中，以便下次使用。下面是缓存的伪代码：

引出 hibernate 的第二级缓存，用下面的伪代码分析了 Cache 的实现原理

```
Dao{
    hashmap map = new map();
    User getUser(integer id)   {
        User user = map.get(id)
        if(user == null)
        {
            user = session.get(id);
            map.put(id,user);
        }
        return user;
    }
}
Dao
{
    Cache cache = null
    setCache(Cache cache)
    {
        this.cache = cache
    }

    User getUser(int id)
```

```
    {
        if(cache!=null)
        {
            User user = cache.get(id);
            if(user ==null)
            {
                user = session.get(id);
                cache.put(id,user);
            }
            return user;
        }

        return session.get(id);
    }
}
```

（2）Hibernate 的 Session 就是一种缓存，我们通常将之称为 Hibernate 的一级缓存，当想使用 session 从数据库中查询出一个对象时，Session 也是先从自己内部查看是否存在这个对象，存在则直接返回，不存在才去访问数据库，并将查询的结果保存在自己内部。由于 Session 代表一次会话过程，一个 Session 与一个数据库连接相关连，所以 Session 最好不要长时间保持打开，通常仅用于一个事务当中，在事务结束时就应关闭。并且 Session 是线程不安全的，被多个线程共享时容易出现问题。通常只有那种全局意义上的缓存才是真正的缓存应用，才有较大的缓存价值，因此，Hibernate 的 Session 这一级缓存的缓存作用并不明显，应用价值不大。Hibernate 的二级缓存就是要为 Hibernate 配置一种全局缓存，让多个线程和多个事务都可以共享这个缓存。我们希望的是一个人使用过，其他人也可以使用，session 没有这种效果。

（3）二级**缓存是独立于 Hibernate 的软件部件，属于第三方的产品**，多个厂商和组织都提供有缓存产品，例如，EHCache 和 OSCache 等等。在 Hibernate 中使用二级缓存，首先就要在 hibernate.cfg.xml 配置文件中配置使用哪个厂家的缓存产品，接着需要配置该缓存产品自己的配置文件，最后要配置 Hibernate 中的哪些实体对象要纳入到二级缓存的管理中。明白了二级缓存原理和有了这个思路后，很容易配置起 Hibernate 的二级缓存。扩展知识：

一个 SessionFactory 可以关联一个二级缓存，也即一个二级缓存只能负责缓存一个数据库中的数据，当使用 Hibernate 的二级缓存后，注意不要有其他的应用或 SessionFactory 来更改当前数据库中的数据，这样缓存的数据就会与数据库中的实际数据不一致。

8.18 iBatis 与 Hibernate 有什么不同?

相同点：屏蔽 jdbc api 的底层访问细节，使用我们不用与 jdbc api 打交道，就可以访问数据。

jdbc api 编程流程固定，还将 sql 语句与 java 代码混杂在了一起，经常需要拼凑 sql 语句，细节很繁琐。

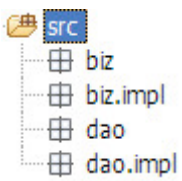
ibatis 的好处：屏蔽 jdbc api 的底层访问细节；将 sql 语句与 java 代码进行分离；提供了将结果集自动封装称为实体对象和对象的集合的功能，queryForList 返回对象集合，用 queryForObject 返回单个对象；提供了自动将实体对象的属性传递给 sql 语句的参数。

Hibernate 是一个全自动的 orm 映射工具，它可以自动生成 sql 语句，ibatis 需要我们自己在 xml 配置文件中写 sql 语句，hibernate 要比 ibatis 功能负责和强大很多。因为 hibernate 自动生成 sql 语句，我们无法控制该语句，我们就无法去写特定的高效率的 sql。对于一些不太复杂的 sql 查询，hibernate 可以很好帮我们完成，但是，对于特别复杂的查询，hibernate 就很难适应了，这时候用 ibatis 就是不错的选择，因为 ibatis 还是由我们自己写 sql 语句。

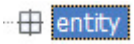
8.19 在 DAO 中如何体现 DAO 设计模式?

DAO(Data Access Object)模式实际上是两个模式的组合，即 Data Accessor 模式和 Active Domain Object 模式，其中 Data Accessor 模式实现了**数据访问**和**业务逻辑**的分离，而 Active Domain Object 模式，其中 Data Accessor 模式实现了**数据访问**和**业务逻辑**的分离，而 Active Domain Object 模式实现了业务数据的对象化封装，一般我们将这两个模式组合使用。

下面我就来分析一下 dao 模式的组成：



先说最基本的，4 个包，一个 dao 包，一个 dao 实现类包，一个 biz 包，一个 biz 实现类包。dao 以及 dao.impl 属于数据访问层。biz 以及 biz.impl 属于业务逻辑层。下面是实体类包 也就是 JavaBean 不知道的朋友可以先看看 javabean 这里面的说明（左键点击）。



然后我们一步一步走，

1.先创建实体类 Users 用户类

2.创建 dao 包相应的 dao 接口 UsersDao,并编写相应的数据访问接口方法

3.创建数据库操作基本类 BaseDao （dao 包下）

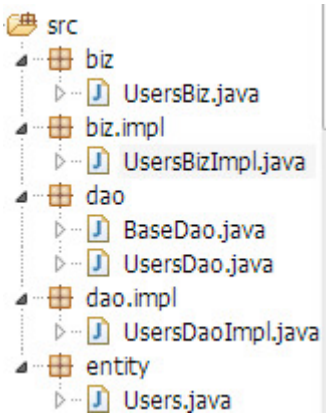
4.创建 biz 包业务接口 UsersBiz，并编写相应的业务逻辑方法

5.创建 UsersDao 实现类 UsersDaoImpl （继承于 BaseDao 实现 UsersDao）

6.创建 UsersBiz 实现类 UsersBizImpl (实现接口 UsersBiz)

7.在 UsersDaoImpl 中写出方法的具体实现。

8.在业务逻辑层 UsersBizImpl 中调用数据访问层中的方法（将 UsersDao 作为成员变量）



8.20 Spring+Hibernate 中委托方案怎么配置?

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <property name="dataSource" ref="oracleDataSource"/>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">
          ${hibernate.dialect}
        </prop>
        <prop key="hibernate.connection.release_mode">
          after_transaction
        </prop>
        <prop key="hibernate.show_sql">true</prop>
          <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
          <prop key="hibernate.temp.use_jdbc_metadata_defaults">false</prop>
      </props>
    </property>
    <!-- Must references all OR mapping files. -->
    <property name="defineMappingResources">
      <list>
        <value>classpath:*-*ql.xml</value>
      </list>
    </property>
  </bean>
```

8.21 Struts2、spring2、hibernate3 在 SSH 中各起什么作用

简单的说：**struts 控制用的；hibernate 操作数据库的；spring 用解耦的**。

详细的说：STRUTS 在 SSH 框架中起控制的作用，其核心是 Controller，即 ActionServlet，而 ActionServlet 的核心就是 Struts-config.xml，主要控制逻辑关系的处理。hibernate 是数据持久化层，是一种新的对象、关系的映射工具，提供了从 Java 类到数据表的映射，也提供了数据查询和恢复等机制，大大减少数据访问的复杂度。把对数据库的直接操作，转换为对持久对象的操作。SPRING 是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架，面向接口的编程，由容器控制程序之间的（依赖）关系，而非传统实现中，由程序代码直接操控。这也就是所谓“控制反转”的概念所在：（依赖）控制权由应用代码中转到了外部容器，控制权的转移，是所谓反转。依赖注入，即组件之间的依赖关系由容器在运行期决定，形象的来说，即 由容器动态的将某种依赖关系注入

到组件之中起到的主要作用是解耦。

Struts 、 spring 、 Hibernate 在各层的作用

1.struts 负责 web 层 ActionFormBean 接收网页中表单提交的数据，然后通过 Action 进行处理，再 Forward 到对应的网页。在 struts-config.xml 中定义 <action-mapping>, ActionServlet 会加载。

2.spring 负责业务层管理，即 Service （或 Manager)。

a.service 为 action 提供统计的调用接口，封装持久层的 DAO。

b.可以写一些自己的业务方法。

c.统一的 javabean 管理方法；

d.声明式事务管理

e.集成 Hiberante

3.Hiberante 负责持久化层，完成数据库的 crud 操作。hibernate 为持久层，提供 OR/Mapping。它有一组 .hbm.xml 文件和 POJO, 是跟数据库中的表相对应的。然后定义 DAO ，这些是跟数据库打交道的类，它们会使用 PO 。

在 struts+spring+hibernate 的系统中，对象的调用流程是： jsp-> Action - > Service ->DAO ->Hibernate。数据的流向是 ActionFormBean 接受用户的数据， Action 将数据从 ActionFromBean 中取出，封装成 VO 或 PO,再调用业务层的 Bean 类，完成各种业务处理后再 forward 。而业务层 Bean 收到这个 PO 对象之后，会调用 DAO 接口方法，进行持久化操作。

8.22 什么是 Spring 的 IOC AOP?

什么是 DI 机制？

依赖注入（Dependecy Injection）和控制反转（Inversion of Control）是同一个概念，具体的讲：当某个角色需要另外一个角色协助的时候，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。但在 spring 中创建被调用者的工作不再由调用者来完成，因此称为控制反转。创建被调用者的工作由 spring 来完成，然后注入调用者因此也称为依赖注入。

spring 以动态灵活的方式来管理对象 ， 注入的两种方式，**设置注入**和**构造注入**。

设置注入的优点：直观，自然

构造注入的优点：可以在构造器中决定依赖关系的顺序。

什么是 AOP？

面向切面编程（AOP）完善 spring 的依赖注入（DI），面向切面编程在 spring 中主要表现为两个方面：

1. 面向切面编程提供声明式事务管理

2. spring 支持用户自定义的切面

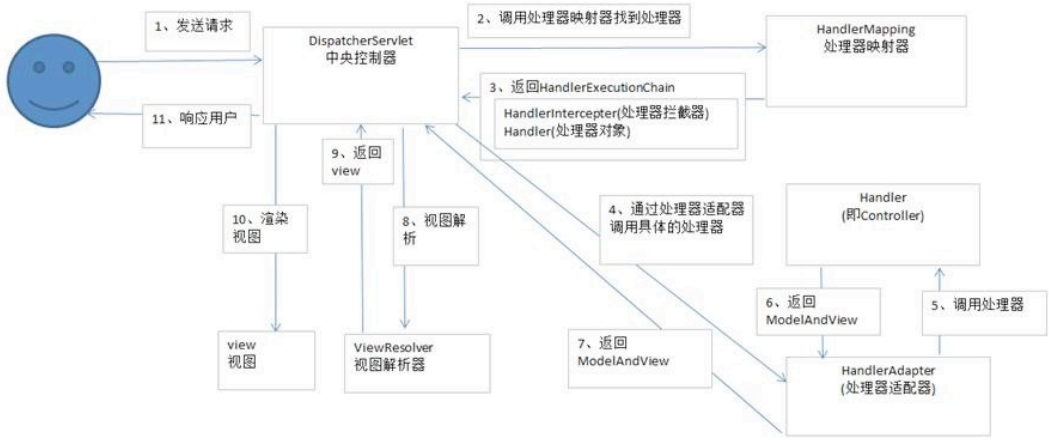
面向切面编程（aop）是对面向对象编程（oop）的补充，面向对象编程将程序分解成各个层次的对象，面向切面编程将程序运行过程分解成各个切面。AOP 从程序运行角度考虑程序的结构，提取业务处理过程的切面，oop 是静态的抽象，aop 是动态的抽象，是对应用执行过程中的步骤进行抽象，从而获得步骤之间的逻辑划分。

aop 框架具有的两个特征：

1. 各个步骤之间的良好隔离性

2. 源代码无关性

8.23 简单的谈一下 Spring MVC 的工作流程？



流程

1、用户发送请求至前端控制器 DispatcherServlet

2、DispatcherServlet 收到请求调用 HandlerMapping 处理器映射器。

3、处理器映射器找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)

一并返回给 DispatcherServlet。

4、DispatcherServlet 调用 HandlerAdapter 处理器适配器

5、HandlerAdapter 经过适配调用具体的处理器(Controller，也叫后端控制器)。

6、Controller 执行完成返回 ModelAndView

7、HandlerAdapter 将 controller 执行结果 ModelAndView 返回给 DispatcherServlet

8、DispatcherServlet 将 ModelAndView 传给 ViewReslover 视图解析器

9、ViewReslover 解析后返回具体 View

10、DispatcherServlet 根据 View 进行渲染视图（即将模型数据填充至视图中）。

11、DispatcherServlet 响应用户

8.24 如何解决 POST 请求中文乱码问题，GET 的又如何处理呢？

在 web.xml 中加入字符过滤器：

```
<filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
                <param-name>encoding</param-name>
                <param-value>utf-8</param-value>
        </init-param>
</filter>
<filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

以上可以解决 post 请求乱码问题。对于 get 请求中文参数出现乱码解决方法有两个：

一种方法是：修改 tomcat 配置文件添加编码与工程编码一致，如下：

```
<Connector
URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>
```

另外一种方法：对参数进行重新编码：

String userName = new String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8");

ISO8859-1 是 tomcat 默认编码，需要将 tomcat 编码后的内容按 utf-8 编码

8.25 SpringMVC 与 Struts2 的主要区别？

1.SpringMVC 的入口是一个 servlet 即前端控制器，而 struts2 入口是一个 filter 过虑器。

2.SpringMVC 是基于方法开发，传递参数是通过方法形参，可以设计为单例或多例(建议单例)，struts2 是基于类开发，传递参数是通过类的属性，只能设计为多例。

3.Struts 采用值栈存储请求和响应的数据，通过 OGNL 存取数据， springmvc 通过参数解析器是将 request 对象内容进行解析成方法形参，将响应数据和页面封装成 ModelAndView 对象，最后又将模型数据通过 request 对象传输到页面。 Jsp 视图解析器默认使用 jstl。

8.26 Spring MVC Framework 有这样一些特点:

1.它是基于组件技术的全部的应用对象,无论控制器和视图,还是业务对象之类的都是 java 组件，并且和 Spring 提供的其他基础结构紧密集成。

2.不依赖于 Servlet API(目标虽是如此,但是在实现的时候确实是依赖于 Servlet 的)。

3.可以任意使用各种视图技术,而不仅仅局限于 JSP。

4.支持各种请求资源的映射策略。

5.它应是易于扩展的。

8.27 SSM 优缺点、使用场景？

1.Mybatis 和 hibernate 不同，它不完全是一个 ORM 框架，因为 MyBatis 需要程序员自己编写 Sql 语句，不过 mybatis 可以通过 XML 或注解方式灵活配置要运行的 sql 语句，并将 java 对象和 sql 语句映射生成最终执行的 sql，最后将 sql 执行的结果再映射生成 java 对象。

2.Mybatis 学习门槛低，简单易学，程序员直接编写原生态 sql，可严格控制 sql 执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是 mybatis 无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套 sql 映射文件，工作量大。

3.Hibernate 对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例

如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的学习门槛高，要精通门槛更高，而且怎么设计 O/R 映射，在性能和对象模型之间如何权衡，以及怎样用好 Hibernate 需要具有很强的经验和能力才行。

总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。

8.28 简单介绍下你对 mybatis 的理解？

1. mybatis 配置
2. SqlMapConfig.xml，此文件作为 mybatis 的全局配置文件，配置了 mybatis 的运行环境等信息。
3. mapper.xml 文件即 sql 映射文件，文件中配置了操作数据库的 sql 语句。此文件需要在 SqlMapConfig.xml 中加载。
4. 通过 mybatis 环境等配置信息构造 SqlSessionFactory 即会话工厂
5. 由会话工厂创建 sqlSession 即会话，操作数据库需要通过 sqlSession 进行。
6. mybatis 底层自定义了 Executor 执行器接口操作数据库，Executor 接口有两个实现，一个是基本执行器、一个是缓存执行器。
7. Mapped Statement 也是 mybatis 一个底层封装对象，它包装了 mybatis 配置信息及 sql 映射信息等。mapper.xml 文件中一个 sql 对应一个 Mapped Statement 对象，sql 的 id 即是 Mapped statement 的 id。
8. Mapped Statement 对 sql 执行输入参数进行定义，包括 HashMap、基本类型、pojo，Executor 通过 Mapped Statement 在执行 sql 前将输入的 java 对象映射至 sql 中，输入参数映射就是 jdbc 编程中对 preparedStatement 设置参数。
9. Mapped Statement 对 sql 执行输出结果进行定义，包括 HashMap、基本类型、pojo，Executor 通过 Mapped Statement 在执行 sql 后将输出结果映射至 java 对象中，输出结果映射过程相当于 jdbc 编程中对结果的解析处理过程。

8.29 Mybatis 比 IBatis 比较大的几个改进是什么？

a.有接口绑定,包括注解绑定 sql 和 xml 绑定 Sql ,

b.动态 sql 由原来的节点配置变成 OGNL 表达式,

c. 在一对一,一对多的时候引进了 association,在一对多的时候引入了 collection 节点,不过都是在 resultMap 里面配置

8.30 什么是 MyBatis 的接口绑定，有什么好处？

接口映射就是在 IBatis 中任意定义接口,然后把接口里面的方法和 SQL 语句绑定，我们直接调用接口方法就可以,这样比起原来了 SqlSession 提供的方法我们可以有更加灵活的选择和设置.

8.31 接口绑定有几种实现方式,分别是怎么实现的?

接口绑定有两种实现方式,一种是通过注解绑定,就是在接口的方法上面加上@Select@Update 等注解里面包含 Sql 语句来绑定,另外一种就是通过 xml 里面写 SQL 来绑定,在这种情况下,要指定 xml 映射文件里面的 namespace 必须为接口的全路径名.

8.32 MyBatis 什么情况下用注解绑定，什么情况下用 xml 绑定？

当 Sql 语句比较简单时候,用注解绑定，当 SQL 语句比较复杂时候用 xml 绑定,一般用 xml 绑定的比较多。

8.33 MyBatis 实现一对一有几种方式，具体怎么操作的？

有联合查询和嵌套查询，联合查询是几个表联合查询，只查询一次，通过在 resultMap 里面配置 association 节点配置一对一的类就可以完成；

嵌套查询是先查一个表,根据这个表里面的结果的外键 id,去再另外一个表里面查询数据,也是通过 association 配置,但另外一个表的查询通过 select 属性配置。

8.34 MyBatis 实现一对多有几种方式，怎么操作的？

有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次,通过在 resultMap 里面配置 collection 节点配置一对多的类就可以完成;

嵌套查询是先查一个表,根据这个表里面的结果的外键 id,去再另外一个表里面查询数据,也是通过配置 collection,但另外一个表的查询通过 select 节点配置;

8.35 MyBatis 里面的动态 Sql 是怎么设定的?用什么语法?

MyBatis 里面的动态 Sql 一般是通过 if 节点来实现,通过 OGNL 语法来实现,但是如果要写的完整,必须配合 where,trim 节点,where 节点是判断包含节点有内容就插入 where,否则不插入,trim 节点是用来判断如果动态语句是以 and 或 or 开始,那么会自动把这个 and 或者 or 取掉 ;

8.36 iBatis 和 MyBatis 在核心处理类分别叫什么

iBatis 里面的核心处理类交 SqlMapClient，MyBatis 里面的核心处理类叫做 SqlSession。

8.37 iBatis 和 MyBatis 在细节上的不同有哪些

在 sql 里面变量命名有原来的#变量#变成了#{变量}原来的 $变量$；变成了${变量}，原来在 sql 节点里面的 class 都换名字交 type 原来的 queryForObject queryForList 变成了 selectOne selectList 原来的别名设置在映射文件里面放在了核心配置文件里。

8.38 讲下 MyBatis 的缓存

MyBatis 的缓存分为**一级缓存**和**二级缓存**, 一级缓存放在 session 里面，默认就有；二级缓存放在它的命名空间里，默认是打开的，使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态)，可在它的映射文件中配置<cache/>;

8.39 MyBatis(IBatis)的好处是什么

ibatis 把 sql 语句从 Java 源程序中独立出来，放在单独的 XML 文件中编写，给程序的维护带来了很大便利。ibatis 封装了底层 JDBC API 的调用细节，并能自动将结果集转换成 Java Bean 对象，大大简化了 Java 数据库编程的重复工作。因为 Ibatis 需要程序员自己去编写 sql 语句，程序员可以结合数据库自身的特点灵活控制 sql 语句， 因此能够实现比 hibernate 等全自动 orm 框架更高的查询效率，能够完成复杂查询。

8.40 JDO 是什么?

JDO 是 Java 对象**持久化**的新的规范，为 java data object 的简称,也是一个用于存取某种数据仓库中的对象的标准化 API。JDO 提供了透明的对象存储，因此对开发人员来说，存储数据对象完全不需要额外的代码（如 JDBC API 的使用）。这些繁琐的例行工作已经转移到 JDO 产品提供商身上，使开发人员解脱出来，从而集中时间和精力在业务逻辑上。另外，JDO 很灵活，因为它可以在任何数据底层上运行。JDBC 只是面向关系数据库（RDBMS）JDO 更通用，提供到任何数据底层的存储功能，比如关系数据库、文件、XML 以及对象数据库（ODBMS）等等，使得应用可移植性更强。

应用程序的开发人员通过访问 JDO Instance ， 达到访问 JDO Instance 所代表的数据对象,包括:ERP,数据库系统等.使数据的存储介质对于应用的开发人员完全透明.

JDO 最早是由 Sun 召集众多的 O/R Mapping 开发团队集中起来共同提出的，首先是通过会议确定了 JDO 需要包括的内容，然后正式提出一个 Java 规范请求（JSR-12），正式开始了 JDO 规范的制定。

九、软件工程与设计模式

9.1 UML 方面

标准建模语言 UML。用例图，静态图(包括类图、对象图和包图)，行为图，交互图(顺序图、合作图)，实现图。

9.2 J2ee 常用的设计模式？说明工厂模式。

总共 23 种，分为三大类：创建型，结构型，行为型。我只记得其中常用的 6、7 种，分别是：创建型（工厂、工厂方法、抽象工厂、单例），结构型（包装、适配器，组合，代理），行为（观察者，模版，策略），然后再针对你熟悉的模式谈谈你的理解即可。

Java 中的 23 种设计模式：

Factory（工厂模式），Builder（建造模式），Factory Method（工厂方法模式），

Prototype（原始模型模式），Singleton（单例模式），Facade（门面模式），

Adapter（适配器模式），Bridge（桥梁模式），Composite（合成模式），

Decorator（装饰模式），Flyweight（享元模式），Proxy（代理模式），

Command（命令模式），Interpreter（解释器模式），Visitor（访问者模式），

Iterator（迭代子模式），Mediator（调停者模式），Memento（备忘录模式），

Observer（观察者模式），State（状态模式），Strategy（策略模式），

Template Method（模板方法模式），Chain Of Responsibleity（责任链模式）

工厂模式：工厂模式是一种经常被使用到的模式，根据工厂模式实现的类可以根据提供的数据生成一组类中某一个类的实例，通常这一组类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，该类的子类通过不同的方法实现了基类中的方法。然后需要定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

9.3 开发中都用到了那些设计模式?用在什么场合?

每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案

的核心。通过这种方式，你可以无数次地使用那些已有的解决方案，无需在重复相同的工作。主要用到了 MVC 的设计模式。用来开发 JSP/Servlet 或者 J2EE 的相关应用。简单工厂模式等。

十、J2EE 部分

10.1 BS 与 CS 的联系与区别。

C/S 是 Client/Server 的缩写。服务器通常采用高性能的 PC、工作站或小型机，并采用大型数据库系统，如 Oracle、Sybase、InFORMix 或 SQL Server。客户端需要安装专用的客户端软件。

B/S 是 Brower/Server 的缩写，客户机上只要安装一个浏览器（Browser），如 Netscape Navigator 或 Internet Explorer，服务器安装 Oracle、Sybase、InFORMix 或 SQL Server 等数据库。在这种结构下，用户界面完全通过 WWW 浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现。浏览器通过Web Server 同数据库进行数据交互。

C/S 与 B/S 区别：

1．硬件环境不同:

C/S 一般建立在专用的网络上, 小范围里的网络环境, 局域网之间再通过专门服务器提供连接和数据交换服务.

B/S 建立在广域网之上的, 不必是专门的网络硬件环境,例与电话上网, 租用设备. 信息自己管理. 有比 C/S 更强的适应范围, 一般只要有操作系统和浏览器就行

2．对安全要求不同

C/S 一般面向相对固定的用户群, 对信息安全的控制能力很强. 一般高度机密的信息系统采用 C/S 结构适宜. 可以通过 B/S 发布部分可公开信息.

B/S 建立在广域网之上, 对安全的控制能力相对弱, 可能面向不可知的用户。

3．对程序架构不同

C/S 程序可以更加注重流程, 可以对权限多层次校验, 对系统运行速度可以较少考虑.

B/S 对安全以及访问速度的多重的考虑, 建立在需要更加优化的基础之上. 比 C/S 有更高的要求 B/S 结构的程序架构是发展的趋势, 从 MS 的.Net 系列的 BizTalk 2000 Exchange 2000 等, 全面支持网络的构件搭建的系统. SUN 和 IBM 推的 JavaBean 构件技术等,使 B/S 更加成熟.

4．软件重用不同

C/S 程序可以不可避免的整体性考虑, 构件的重用性不如在 B/S 要求下的构件的重用性好.

B/S 对的多重结构,要求构件相对独立的功能. 能够相对较好的重用.就入买来的餐桌可以再利用,而不是做在墙上的石头桌子

5．系统维护不同

C/S 程序由于整体性, 必须整体考察, 处理出现的问题以及系统升级. 升级难. 可能是再做一个全新的系统

B/S 构件组成,方面构件个别的更换,实现系统的无缝升级. 系统维护开销减到最小. 用户从网上自己下载安装就可以实现升级.

6．处理问题不同

C/S 程序可以处理用户面固定, 并且在相同区域, 安全要求高需求, 与操作系统相关. 应该都是相同的系统

B/S 建立在广域网上, 面向不同的用户群, 分散地域, 这是 C/S 无法作到的. 与操作系统平台关系最小.

7．用户接口不同

C/S 多是建立的 Window 平台上,表现方法有限,对程序员普遍要求较高

B/S 建立在浏览器上, 有更加丰富和生动的表现方式与用户交流. 并且大部分难度减低,减低开发成本.

8．信息流不同

C/S 程序一般是典型的中央集权的机械式处理, 交互性相对低

B/S 信息流向可变化, B-B B-C B-G 等信息、流向的变化, 更像交易中心。

10.2 应用服务器与 WEB SERVER 的区别？

应用服务器处理业务逻辑，web 服务器主要是让客户可以通过浏览器进行访问。其中应用服务器处理业务逻辑，web 服务器是用于处理 html 文件的。那么下面就让我们详细介绍一下两种的区别：

一、web 服务器：

Web 服务器可以解析 http 协议。当服务器接收到一个 http 请求（request），会返回一个 http 响应（response）。例如送回一个 HTML 页面。为了处理一个请求(request)，Web

服务器可以响应(response)一个静态页面或图片，进行页面跳转(redirect)，或者把动态响应(dynamic response)的产生委托(delegate)给一些其它的程序例如 CGI 脚本，JSP(JavaServer Pages)脚本，servlets，ASP(Active Server Pages)脚本，服务器端(server-side)JavaScript，或者一些其它的服务器端(server-side)技术。无论它们(译者注：脚本)的目的如何，这些服务器端(server-side)的程序通常产生一个 HTML 的响应(response)来让浏览器可以浏览。

要知道，Web 服务器的代理模型(delegation model)非常简单。当一个请求(request)被送到 Web 服务器里来时，它只单纯的把请求(request)传递给可以很好的处理请求(request)的程序(译者注：服务器端脚本)。Web 服务器仅仅提供一个可以执行服务器端(server-side)程序和返回(程序所产生的)响应(response)的环境，而不会超出职能范围。服务器端(server-side)程序通常具有事务处理(transaction processing)，数据库连接(database connectivity)和消息(messaging)等功能。

虽然 Web 服务器不支持事务处理或数据库连接池，但它可以配置(employ)各种策略(strategies)来实现容错性(fault tolerance)和可扩展性(scalability)，例如负载平衡(load balancing)，缓冲(caching)。集群特征(clustering—features)经常被误认为仅仅是应用程序服务器专有的特征。

二、应用服务器（The Application Server）：

根据我们的定义，作为应用服务器，它通过各种协议，包括 http，把商业逻辑暴露给客户端应用程序。Web 服务器主要是处理向浏览器发送 html 以供浏览，而应用程序服务器提供访问商业逻辑的途径以供客户端应用程序使用。应用程序使用此商业逻辑就像你调用对象的一个方法（或过程语言中的一个函数）一样。

应用程序服务器的客户端（包括有图形用户界面（GUI）的）可能会运行在一台 PC、一个 web 服务器或者甚至是其它的应用程序服务器上。在应用程序服务器与其客户端之间来回穿梭（traveling）的信息不仅仅局限于简单的显示标记。相反，这种信息就是程序逻辑（program logic）。正是由于这种逻辑取得了（takes）数据和方法调用（calls）的形式而不是静态 html，所以客户端才可以随心所欲的使用这种暴露的商业逻辑。

在大多数情形下，应用程序服务器是通过组件（component）的应用程序接口（API）把商业逻辑暴露（expose）（给客户端应用程序）的，例如基于 J2EE 应用程序服务器的 EJB

组件模型。此外，应用程序服务器可以管理自己的资源。例如，看大门的工作（gate-keeping duties）包括安全（security）,事务处理（transaction processing),资源池（resource pooling）,和消息（messaging）。就像 web 服务器一样，应用服务器配置了可扩展（scalability）和容错（fault tolerance）技术。

Web 服务器通常比应用服务器简单，如 apache 就是 web 服务器，jboss 就是 ejb 应用服务器。应用服务器：BEA weblogic Server，IBM WebSphere Application Server，Oracle9i Application Server，jBoss，Tomcat.

应用服务器：Weblogic、Tomcat、Jboss

WEB SERVER：IIS、 Apache

10.3 应用服务器有那些？

答：BEA WebLogic Server，IBM WebSphere Application Server，Oracle9i Application Server，jBoss，Tomcat

10.4 J2EE 是什么？

J2EE 是 Sun 公司提出的多层(multi-diered)，分布式(distributed)，基于组件(component-base)的企业级应用模型(enterpriese application model)。在这样的一个应用系统中，可按照功能划分为不同的组件，这些组件又可在不同计算机上，并且处于相应的层次(tier)中。所属层次包括客户层(clietn tier)组件，web 层和组件，Business 层和组件，企业信息系统(EIS)层。

一个另类的回答：j2ee 就是增删改查。

10.5 J2EE 是技术还是平台还是框架？ 什么是 J2EE

J2EE 本身是一个标准，一个为企业分布式应用的开发提供的标准平台。

J2EE 也是一个框架，包括 JDBC、JNDI、RMI、JMS、EJB、JTA 等技术。

10.6 请对以下在 J2EE 中常用的名词进行解释(或简单描述)

web 容器：给处于其中的应用程序组件（JSP，SERVLET）提供一个环境，使 JSP,SERVLET 直

接更容器中的环境变量接口交互，不必关注其它系统问题。主要有 WEB 服务器来实现。例如：TOMCAT,WEBLOGIC,WEBSPHERE 等。该容器提供的接口严格遵守 J2EE 规范中的 WEB APPLICATION 标准。我们把遵守以上标准的 WEB 服务器就叫做 J2EE 中的 WEB 容器。

EJB 容器：Enterprise java bean 容器。更具有行业领域特色。他提供给运行在其中的组件 EJB 各种管理功能。只要满足 J2EE 规范的 EJB 放入该容器，马上就会被容器进行高效率的管理。并且可以通过现成的接口来获得系统级别的服务。例如邮件服务、事务管理。

JNDI：（Java Naming & Directory Interface）JAVA 命名目录服务。主要提供的功能是：提供一个目录系统，让其它各地的应用程序在其上面留下自己的索引，从而满足快速查找和定位分布式应用程序的功能。

JMS：（Java Message Service）JAVA 消息服务。主要实现各个应用程序之间的通讯。包括点对点和广播。

JTA：（Java Transaction API）JAVA 事务服务。提供各种分布式事务服务。应用程序只需调用其提供的接口即可。

JAF：（Java Action FrameWork）JAVA 安全认证框架。提供一些安全控制方面的框架。让开发者通过各种部署和自定义实现自己的个性安全控制策略。

RMI/IIOP:（Remote Method Invocation /internet 对象请求中介协议）他们主要用于通过远程调用服务。例如，远程有一台计算机上运行一个程序，它提供股票分析服务，我们可以在本地计算机上实现对其直接调用。当然这是要通过一定的规范才能在异构的系统之间进行通信。RMI 是 JAVA 特有的。

10.7 如何给 weblogic 指定大小的内存?

在启动 Weblogic 的脚本中（位于所在 Domian 对应服务器目录下的 startServerName），增加 set MEM_ARGS=-Xms32m -Xmx200m，可以调整最小内存为 32M，最大 200M。

10.8 如何设定的 weblogic 的热启动模式(开发模式)与产品发布模式?

可以在管理控制台中修改对应服务器的启动模式为开发或产品模式之一。或者修改服务的启动文件或者 commenv 文件，增加 set PRODUCTION_MODE=true。

10.9 如何启动时不需输入用户名与密码?

修改服务启动文件，增加 WLS_USER 和 WLS_PW 项。也可以在 boot.properties 文件中增加加密过的用户名和密码。

10.10 在 weblogic 管理制台中对一个应用域(或者说是一个网站,Domain)进行 jms 及 ejb 或连接池等相关信息进行配置后,实际保存在什么文件中?

保存在此 Domain 的 config.xml 文件中，它是服务器的核心配置文件。

10.11 说说 weblogic 中一个 Domain 的缺省目录结构?比如要将一个简单的 helloWorld.jsp 放入何目录下,然的在浏览器上就可打入 http://主机:端口号//helloword.jsp 就可以看到运行结果了？又比如这其中用到了一个自己写的 javaBean 该如何办?

Domain 目录服务器目录 applications,将应用目录放在此目录下将可以作为应用访问，如果是 Web 应用,应用目录需要满足 Web 应用目录要求，jsp 文件可以直接放在应用目录中，Javabean 需要放在应用目录的 WEB-INF 目录的 classes 目录中，设置服务器的缺省应用将可以实现在浏览器上无需输入应用名。

10.12 在 weblogic 中发布 ejb 需涉及到哪些配置文件

不同类型的 EJB 涉及的配置文件不同，都涉及到的配置文件包括 ejb-jar.xml,weblogic-ejb-jar.xmlCMP 实体 Bean 一般还需要 weblogic-cmp-rdbms-jar.xml。

10.13 如何在 weblogic 中进行 ssl 配置与客户端的认证配置或说说 j2ee(标准)进行 ssl 的配置?

缺省安装中使用 DemoIdentity.jks 和 DemoTrust.jks KeyStore 实现 SSL，需要配置服务器使用 Enable SSL，配置其端口，在产品模式下需要从 CA 获取私有密钥和数字证书，创建 identity 和 trust keystore，装载获得的密钥和数字证书。可以配置此 SSL 连接是单向还是双向的。

10.14 如何查看在 weblogic 中已经发布的 EJB?

可以使用管理控制台，在它的 Deployment 中可以查看所有已发布的 EJB。

十一、Web Service 部分

11.1 WEB SERVICE 名词解释。JSWDL 开发包的介绍。JAXP、JAXM 的解释。SOAP、UDDI,WSDL 解释。

Web ServiceWeb Service 是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得 Web Service 能与其他兼容的组件进行互操作。

JAXP(Java API for XML Parsing) 定义了在 Java 中使用 DOM, SAX, XSLT 的通用的接口。这样在你的程序中你只要使用这些通用的接口，当你需要改变具体的实现时候也不需要修改代码。

JAXM(Java API for XML Messaging) 是为 SOAP 通信提供访问方法和传输机制的 API。

WSDL 是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。这种格式首先对操作和消息进行抽象描述，然后将其绑定到具体的网络协议和消息格式上以定义端点。相关的具体端点即组合成为抽象端点（服务）。

SOAP 即简单对象访问协议(Simple Object Access Protocol)，它是用于交换 XML 编码信息的轻量级协议。

UDDI 的目的是为电子商务建立标准;UDDI 是一套基于 Web 的、分布式的、为 Web Service 提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的 Web Service 注册，以使别的企业能够发现的访问协议的实现标准。

11.2 CORBA 是什么?用途是什么?

CORBA 标准是公共对象请求代理结构(Common Object Request Broker Architecture)，由对象管理组织 (Object Management Group，缩写为 OMG)标准化。它的组成是接口定义语言(IDL)，语言绑定(binding:也译为联编)和允许应用程序间互操作的协议。 其目的为：用不同的程序设计语言书写在不同的进程中运行，为不同的操作系统开发。

十二、Linux 部分

12.1 LINUX 下线程，GDI 类的解释。

LINUX 实现的就是基于核心轻量级进程的"一对一"线程模型，一个线程实体对应一个核心轻量级进程，而线程之间的管理在核外函数库中实现。

GDI 类为图像设备编程接口类库。

12.2 绝对路径用什么符号表示？当前目录、上层目录用什么表示？主目录用什么表示？切换目录用什么命令？

绝对路径： 如/etc/init.d

当前目录和上层目录： ./ ../

主目录： ~/

切换目录： cd

12.3 怎么查看当前进程？怎么执行退出？怎么查看当前路径？

查看当前进程： ps

执行退出： exit

查看当前路径： pwd

12.4 怎么清屏？怎么退出当前命令？怎么执行睡眠？怎么查看当前用户 id？查看指定帮助用什么命令？

清屏： clear

退出当前命令： ctrl+c 彻底退出

执行睡眠 ： ctrl+z 挂起当前进程 fg 恢复后台

查看当前用户 id： ”id“：查看显示目前登陆账户的 uid 和 gid 及所属分组及用户名

查看指定帮助： 如 man adduser 这个很全 而且有例子； adduser --help 这个告诉你一些常用参数； info adduesr；

12.5 Ls 命令执行什么功能？ 可以带哪些参数，有什么区别？

ls 执行的功能： 列出指定目录中的目录，以及文件

哪些参数以及区别： a 所有文件 l 详细信息，包括大小字节数，可读可写可执行的权限等

12.6 建立软链接(快捷方式)，以及硬链接的命令。

软链接： ln -s slink source

硬链接： ln link source

12.7 目录创建用什么命令？创建文件用什么命令？复制文件用什么命令？

创建目录： mkdir

创建文件：典型的如 touch，vi 也可以创建文件，其实只要向一个不存在的文件输出，都会创建文件

复制文件： cp 7. 文件权限修改用什么命令？格式是怎么样的？

文件权限修改： chmod

格式如下：

$ chmod u+x file 给 file 的属主增加执行权限

$ chmod 751 file 给 file 的属主分配读、写、执行(7)的

权限，给 file 的所在组分配读、执行(5)的权限，给其他用户分配执行(1)的权限

$ chmod u=rwx,g=rx,o=x file 上例的另一种形式

$ chmod =r file 为所有用户分配读权限

$ chmod 444 file 同上例

$ chmod a-wx,a+r file 同上例

$ chmod -R u+r directory 递归地给 directory 目录下所有文件和子目录的属主分配读的权限

12.8 查看文件内容有哪些命令可以使用？

vi 文件名 #编辑方式查看，可修改

cat 文件名 #显示全部文件内容

more 文件名 #分页显示文件内容

less 文件名 #与 more 相似，更好的是可以往前翻页

tail 文件名 #仅查看尾部，还可以指定行数

head 文件名 #仅查看头部,还可以指定行数

12.9 随意写文件命令？怎么向屏幕输出带空格的字符串，比如”hello world”？

写文件命令：vi

12.10 终端是哪个文件夹下的哪个文件？黑洞文件是哪个文件夹下的哪个命令？

/dev/tty /dev/null

12.11 移动文件用哪个命令？改名用哪个命令？

mv mv

12.12 复制文件用哪个命令？如果需要连同文件夹一块复制呢？如果需要有提示功能呢？

cp cp -r ？？？？

12.13 删除文件用哪个命令？如果需要连目录及目录下文件一块删除呢？删除空文件夹用什么命令？

rm rm -r rmdir

12.14 Linux 下命令有哪几种可使用的通配符？分别代表什么含义?

“？”可替代单个字符。 ·“*”可替代任意多个字符。 ·方括号“[charset]”可替代 charset 集中的任何单个字符，如[a-z]，[abABC] 15. 用什么命令对一个文件的内容进行统计？(行号、单词数、字节数)

wc 命令 - c 统计字节数。 - l 统计行数。 - w 统计字数。

12.15 Grep 命令有什么用？ 如何忽略大小写？ 如何查找不含该串的行?

是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。

grep [stringSTRING] filename grep [^string] filename

12.16 Linux 中进程有哪几种状态？在 ps 显示出来的信息中，分别用什么符号表示的？

（1）、不可中断状态：进程处于睡眠状态，但是此刻进程是不可中断的。不可中断， 指进程不响应异步信号。

（2）、暂停状态/跟踪状态：向进程发送一个 SIGSTOP 信号，它就会因响应该信号 而进入 TASK_STOPPED 状态;当进程正在被跟踪时，它处于 TASK_TRACED 这个特殊的状态。

“正在被跟踪”指的是进程暂停下来，等待跟踪它的进程对它进行操作。（3）、就绪状态：在 run_queue 队列里的状态（4）、运行状态：在 run_queue 队列里的状态

（5）、可中断睡眠状态：处于这个状态的进程因为等待某某事件的发生（比如等待 socket 连接、等待信号量），而被挂起

（6）、zombie 状态（僵尸）：父亲没有通过 wait 系列的系统调用会顺便将子进程的尸体（task_struct）也释放掉

（7）、退出状态

D 不可中断 Uninterruptible（usually IO）

R 正在运行，或在队列中的进程

S 处于休眠状态

T 停止或被追踪

Z 僵尸进程

W 进入内存交换（从内核 2.6 开始无效）

X 死掉的进程

12.17 怎么使一个命令在后台运行?

&

12.18 利用 ps 怎么显示所有的进程? 怎么利用 ps 查看指定进程的信息？

ps -ef (system v 输出) ps -aux bsd 格式输出 ps -ef | grep pid

12.19 哪个命令专门用来查看后台任务?

job -l

12.20 把后台任务调到前台执行使用什么命令?把停下的后台任务在后台执行起来用什么命令?

fg

例如：#jobs

[1]+ Running /root/bin/rsync.sh &

#fg 1 bg 类似

12.21 终止进程用什么命令? 带什么参数?

kill -9 pid

12.22 怎么查看系统支持的所有信号?

kill -l

12.23 搜索文件用什么命令? 格式是怎么样的?

find dir -name "string*"

12.24 查看当前谁在使用该主机用什么命令? 查找自己所在的终端信息用什么命令?

w 用户名称;用户的机器名称或 tty 号;远程主机地址;用户登录系统的时间;空闲时间（作用不大）;附加到 tty（终端）的进程所用的时间（JCPU 时间）;当前进程所用时间（PCPU 时间）;用户当前正在使用的命令.

who 用户名、tty 号、时间日期、主机地址

whoami,id -un 命令用于显示登入的用户名

last 命令可用于显示特定用户登录系统的历史记录(last jason):用户名称;tty 设备号;历史登录时间日期;登出时间日期;总工作时间.

查找自己所在终端信息：who am i

12.25 使用什么命令查看用过的命令列表?

history

12.26 使用什么命令查看磁盘使用空间； 空闲空间呢?

df -hl

文件系统 容量 已用 可用 已用% 挂载点

Filesystem Size Used Avail Use% Mounted on /dev/hda2 45G 19G 24G 44% /

/dev/hda1 494M 19M 450M 4% /boot

12.27 使用什么命令查看网络是否连通?

Netstat

12.28 使用什么命令查看 ip 地址及接口信息？

ifconfig

12.29 查看各类环境变量用什么命令?

查看所有 env

查看某个，如 home： env $HOME

12.30 通过什么命令指定命令提示符?

\u 显示当前用户账号 \h 显示当前主机名

\W 只显示当前路径最后一个目录 \w 显示当前绝对路径（当前用户目录会以 ~代替）

$PWD 显示当前全路径 \$ 显示命令行'$'或者'#'符号 \# ：下达的第几个命令

\d ：代表日期，格式为 weekday month date，例如："Mon Aug 1"

\t ：显示时间为 24 小时格式，如：HH：MM：SS \T ：显示时间为 12 小时格式 \A ：显示时间为 24 小时格式：HH：MM \v ：BASH 的版本信息

如 export PS1='[\u@\h \w\#]\$ '

12.31 查找命令的可执行文件是去哪查找的？怎么对其进行设置及添加?

whereis [-bfmsu][-B <目录>...][-M <目录>...][-S <目录>...][文件...]

补充说明：whereis 指令会在特定目录中查找符合条件的文件。这些文件的烈性应属于原始代码，二进制文件，或是帮助文件。

-b 只查找二进制文件。

-B<目录> 只在设置的目录下查找二进制文件。 -f 不显示文件名前的路径名称。

-m 只查找说明文件。

-M<目录> 只在设置的目录下查找说明文件。 -s 只查找原始代码文件。

-S<目录> 只在设置的目录下查找原始代码文件。 -u 查找不包含指定类型的文件。

which 指令会在 PATH 变量指定的路径中，搜索某个系统命令的位置，并且返回第一个搜索结果。

-n 指定文件名长度，指定的长度必须大于或等于所有文件中最长的文件名。

-p 与-n 参数相同，但此处的包括了文件的路径。 -w 指定输出时栏位的宽度。

-V 显示版本信息

12.32 通过什么命令查找执行命令?

which 只能查可执行文件 whereis 只能查二进制文件、说明文档，源文件等

12.33 怎么对命令进行取别名？

alias la='ls -a'

12.34 du 和 df 的定义，以及区别？ du 显示目录或文件的大小

df 显示每个<文件>所在的文件系统的信息，默认是显示所有文件系统。

（文件系统分配其中的一些磁盘块用来记录它自身的一些数据，如 i 节点，磁盘分布图，间接块，超级块等。

这些数据对大多数用户级的程序来说是不可见的，通常称为 Meta Data。） du 命令是用户级的程序，它不考虑 Meta Data，而 df 命令则查看文件系统的磁盘分配

图并考虑 Meta Data。

df 命令获得真正的文件系统数据，而 du 命令只查看文件系统的部分情况。

12.35 awk 详解

awk '{pattern + action}' {filenames}

#cat /etc/passwd |awk -F ':' '{print $1"\t"$7}' //-F 的意思是以':'分隔 root /bin/bash

daemon /bin/sh 搜索/etc/passwd 有 root 关键字的所有行

#awk -F: '/root/' /etc/passwd root:x:0:0:root:/root:/bin/bash

十三、其他（问得稀里糊涂的题）

13.1 四种会话跟踪技术

会话作用域 ServletsJSP 页面描述。

page 否是代表与一个页面相关的对象和属性。一个页面由一个编译好的 Java servlet 类（可以带有任何的 include 指令，但是没有 include 动作）表示。这既包括 servlet 又包括被编译成 servlet 的 JSP 页面。

request 是是代表与 Web 客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个 Web 组件（由于 forward 指令和 include 动作的关系）。

session 是是代表与用于某个 Web 客户机的一个用户体验相关的对象和属性。一个 Web 会话可以也经常会跨越多个客户机请求。

application 是是代表与整个 Web 应用程序相关的对象和属性。这实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域 。

13.2 简述逻辑操作(&,|,^)与条件操作(&&,||)的区别。

区别主要答两点：

a.条件操作只能操作布尔型的,而逻辑操作不仅可以操作布尔型,而且可以操作数值型

b.逻辑操作不会产生短路。

十四、实际项目开发

14.1 在 eclipse 中调试时，怎样查看一个变量的值？

在要查看的变量前先设置断点，然后选中变量，右键选 debug as-->Java Application，打开 debug 透视图，这时在 Variables 窗口中可以看到变量当前的值。

如果是局部变量，也可以在局部变量窗口中查看。

要知道一个方法被调用的方法调用链，可以在方法栈中查看。

14.2 你们公司使用的代码配置管理工具是什么？

除了说以前使用 cvs，现在新项目使用 svn 了，还简要说一下使用的过程，如果有可能，还说说仓库的概念和如何使用锁之类的细节。

14.3 你们的项目总金额多少，多少人开发，总共花了多少个月？

像巴巴运动网这种规模的项目，可以说是 4、5 个人、开发了 4、5 个月，费用则是 4、50 万。按每人每月两万收入去计算，就差不多了。

十五、笔试答题技巧与若干问题

说明，为了节省大家的时间和提高学习效率，一些过时知识点和被笔试概率极低的题目不再被收录和分析。

回答问题的思路：先正面叙述一些**基本的核心知识**，然后描述一些**特殊的东西**，最后再来一些**锦上添花的东西**。要注意有些不是锦上添花，而是画蛇添足的东西，不要随便写上。把答题像写书一样写。我要回答一个新技术的问题大概思路和步骤是：我们想干什么，怎么干，干的过程中遇到了什么问题，现在用什么方式来解决。其实我们讲课也是这样一个思路。

例如，讲 ajax 时，我们希望不改变原来的整个网页，而只是改变网页中的局部内容，例如，用户名校验，级联下拉列表，下拉树状菜单。用传统方式，就是浏览器自己直接向服务器发请求，服务器返回新页面会盖掉老页面，这样就不流畅了。

对本面试宝典中的题目有信心吗？本来有信心的，结果听你讲完后，就没信心了！我非常理解。因为他觉得我的太深，他想记住我的些东西，可是记不住，所以没信心了。我又问：听懂了吗？他说听懂了。你到现在只要把你的理解尽量清晰地、有条理地表达出来，就很棒了。

这套面试题主要目的是帮助那些还没有 java 软件开发实际工作经验，而正在努力寻找 java 软件开发工作的朋友在笔试时更好地赢得笔试和面试。由于这套面试题涉及的范围很泛，很广，很杂，大家不可能一天两天就看完和学完这套面试宝典，即使你已经学过了有关的技术，那么至少也需要一个月的时间才能消化和掌握这套面试宝典，所以，大家应该早作准备，从拿到这套面试宝典之日起，就要坚持在每天闲暇之余学习其中几道题目，日积月累，等到出去面试时，一切都水到渠成，面试时就自然会游刃有余了。

答题时，先答是什么，再答有什么作用和要注意什么（这部分最重要，展现自己的心得）

答案的段落分别，层次分明，条理清晰都非常重要，从这些表面的东西也可以看出一个人的习惯、办事风格、条理等。

要讲你做出答案的思路过程，或者说你记住答案的思想都写下来。把答题想着是辩论赛。答题就是给别人讲道理、摆事实。答题不局限于什么格式和形式，就是要将自己的学识展现出来！

别因为人家题目本来就模棱两可，你就心里胆怯和没底气了，不敢回答了。你要大胆地指出对方题目很模糊和你的观点，不要把面试官想得有多高，其实他和你就是差不多的，你

想想，如果他把你招进去了，你们以后就是同事了，可不是差不多的吗？

关于就业薪水，如果你是应届生，那不能要高工资，好比大饼的故事，没有文凭还想拿高工资，就去中关村缺什么补什么吧！少数人基础确实很好，在校期间确实又做过一些项目，那仍然是可以要到相对高的工资的。

公司招聘程序员更看重的要用到的编码技术、而不是那些业务不太相关的所谓项目经历：

1.公司想招什么样的;2.公司面试会问什么;3.简历怎么写;4 怎样达到简历上的标准（培训中心教项目的目的）

对于一些公司接到了一些项目，想招聘一些初中级的程序员过来帮助写代码，完成这个项目，你更看重的是他的专业技术功底，还是以前做过几个项目的经历呢？我们先排除掉那些编码技术功底好，又正好做过相似项目的情况，实际上，这种鱼和熊掌兼得的情况并不常见。其实公司很清楚，只要招聘进来的人技术真的很明白，那他什么项目都可以做出来，公司招人不是让你去重复做你以前的项目，而是做一个新项目，业务方面，你只要进了项目团队，自然就能掌握。所以，大多数招聘单位在招聘那些编码级别的程序员时也没指望能招聘到做过类似项目的人，也不会刻意去找做过类似项目的人，用人单位也不是想把你招进，然后把你以前做过的项目重做一遍，所以，用人单位更看重招进来的人对要用到的编码技术的功底到底怎样，技术扎实不扎实，项目则只要跟着开发团队走，自然就没问题。除非是一些非常专业的行业，要招聘特别高级的开发人员和系统分析师，招聘单位才特别注重他的项目经验和行业经验，要去找到行业高手，公司才关心项目和与你聊项目的细节，这样的人通常都不是通过常规招聘渠道去招聘进来的，而是通过各种手段挖过来的，这情况不再我今天要讨论的范围中。

技术学得明白不明白，人家几个问题就把你的深浅问出来了，只要问一些具体的技术点，就很容易看出你是真懂还是假懂，很容看出你的技术深度和实力，所以，技术是来不得半点虚假的，必须扎扎实实。

由于项目的种类繁多，涉及到现实生活中的各行各业，什么五花八门的业务都有，例如，酒店房间预定管理，公司车辆调度管理，学校课程教室管理，超市进销存管理，知识内容管理，等等……成千上万等等，但是，不管是什么项目，采用的无非都是我们学习的那些目前流行和常用的技术。技术好、经验丰富，则项目做出来的效率高些，程序更稳定和更容易维护些；技术差点，碰碰磕磕最后也能把项目做出来，无非是做的周期长点、返工的次数多点，程序代码写得差些，用的技术笨拙点。如果一个人不是完完全全做过某个项目，他是不太关

心该项目的业务的，对其中的一些具体细节更是一窍不知，(如果我招你来做图书管理，你项目经历说你做过汽车调度，那我能问你汽车调度具体怎么回事吗？不会，所以，你很容易蒙混过去的)而一个程序员的整个职业生涯中能实实在在和完完整整做出来的项目没几个，更别说在多个不同行业的项目了，有的程序员更是一辈子都只是在做某一个行业的项目，结果他就成了这个行业的专家(专门干一件事的家伙)。所以，技术面试官通常没正好亲身经历过你简历写的那些项目，他不可能去问你写的那些项目的具体细节，而是只能泛泛地问你这个项目是多少人做的，做了多长时间，开发的过程，你在做项目的过程中有什么心得和收获，用的什么技术等面上的问题，所以，简历上的项目经历可以含有很多水分，很容易作假，技术面试官也无法在项目上甄别你的真伪。

简历该怎么写：**精通那些技术，有一些什么项目经历**

教项目是为了巩固和灵活整合运用技术，增强学习的趣味性，熟悉做项目的流程，或得一些专业课程中无法获得的特有项目经验，增强自己面试的信心。讲的项目应该真实可靠才有价值，否则，表面上是项目，实际上还是知识点的整合，对巩固技术点和增强学习的趣味性，但无法获得实际的项目经验。（项目主要是增加你经验的可信度，获得更多面试机会，真正能不能找到工作，找到好工作，主要看你键盘上的功夫了），好的面试官几下就能面出你是否真有工作经验，他们问技术以外的公司的人和事，并且问开始、过程、结果，看你怎么编。

建议大家尽量开自己的 blog，坚持每天写技术 blog。在简历上写上自己的 blog 地址，可以多转载一些技术文章。

十六、其他情况

16.1 请用英文简单介绍一下自己?

答案忽略，根据自身情况而定，标准不一。

16.2 请把 http://tomcat.apache.org/ 首页的这一段话用中文翻译一下?

Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process.

Apache Tomcat is developed in an open and participatory environment and released under the Apache Software License. Apache Tomcat is intended to be a collaboration of the best-of-breed developers from around the world. We invite you to participate in this open development project. To learn more about getting involved, click here.

Apache Tomcat powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations. Some of these users and their stories are listed on the PoweredBy wiki page.

16.3 美资软件公司 JAVA 工程师电话面试题目

1. Talk about overriding, overloading.
2. Talk about JAVA design patterns you known.
3. Talk about the difference between LinkList, ArrayList and Victor.
4. Talk about the difference between an Abstract class and an Interface.
5. Class a = new Class(); Class b = new Class();
   if(a == b) returns true or false, why?
6. Why we use StringBuffer when concatenating strings?
7. Try to explain Singleton to us? Is it thread safe? If no, how to make it thread safe?

8. Try to explain Ioc?

9. How to set many-to-many relationship in Hibernate?

10. Talk about the difference between INNER JOIN and LFET JOIN.

11. Why we use index in database? How many indexes is the maximum in one table as your suggestion?

12. When 'Final' is used in class, method and property, what dose it mean?

13. Do you have any experience on XML? Talk about any XML tool you used ,e.g. JAXB, JAXG.

14. Do you have any experience on Linux?

15. In OOD what is the reason when you create a Sequence diagram?

Administrator 10:34:20

1，堆和栈的区别，有一个64k的字符串，是放到堆上，还是放到栈上，为什么？

2，什么时候用到接口，什么时候用到抽象类，二者区别

3，有一个100万的数组，里边有两个市重复的，如何设计算法找到。

4，设计数据库时，n维，如何设计。

例如[省份][城市][网吧]，这是三维关系，它的表也应该有三个，网吧有外键引用城市，城市有外键应用省份，这个规律就是下层的要有一外键去引用上层。

十七、附件（公司实际面试题）

17.1 上海汉得

1.springmvc 及 springmvc 自带登陆验证。

2.Mybatis 和 hibernate 区别

Hibernate 是全自动化，及 Hibernate 中 sql 语句是封装好的

sql 语句由 xml 映射文件完成，完全的面向对象。

Mybatis 是半自动化，及 Mybatis 中 sql 语句是自己编写的。

操作：

Hibernate 可移植性好，不依赖底层的数据库，不同的数据库之间可以应用自如，不需要修改代码。

Mybatis 的可移植性较差，依赖底层数据库，用于不同数据库时，需要在配置文件中修改数据库方言等问题。

Hibernate 开发速度快，sql 语句封装，所以大大减少代码的书写。Mybatis 使用速度快，sql 语句灵活，大大提高速度；比如：修改，Hibernate 需要全部修改，而 Mybatis 则可以具体到某个属性的修改。

Tomcat 在 linux 部署

3.数据库的优化

1.SQL 语句大写，在数据库内部执行 sql 语句之前，会将 sql 语句转换为大写然后执行。所以写成小写可以节约这部分时间。

2.在 sql 语句中，查询的列表需要调整；比如：当查询新闻信息时，需要查询标题和内容，可以将标题放在内容之前，可以

提高查询效率。

3.在设计数据库列表时，可以将经常需要的列表添加索引，加快速度

4.在 Oracle 中，可以分区。

数据库死锁

项目的过程中遇到的困难及如何解决

4. 使用 Hibernate 框架的时候，有乐观锁和悲观锁，这两区别

在使用 Hibernate 框架的时候，有乐观锁和悲观锁；一般设置为乐观锁。

乐观锁的话，会在数据库中有一个字段标识，当有人修改的时候会自动加 1，所以当我们进行修改的时候，会有比较，

若两个不同，则会将 数据回滚 ，保证数据的一致性。

悲观锁，

5. Linux 中 Tomcat 的部署

在 Linux 下部署 Tomcat 以及发布 web 项目：

需要用到的软件：

xShell：远程登录访问 Linux 系统

xftp：上传文件（或者用 wget 命令）

注意：Tomcat 依赖 jdk，所以在部署之前要先安装和配置好 jdk

部署 Tomcat：

步骤一：在 Apache 官网下载 Tomcat （Linux：后缀名 .tar Windows：后缀名 .zip ）

步骤二：使用 xftp 上传 tar 包 到 Linux 的 /opt 目录

步骤三：到/opt 目录下解压：tar

步骤四：可以修改端口号

解压之后的文件中，进入 conf 目录，修改 server.xml 文件

cd conf

vi server.xml --例如：将端口号 8080 该为：8888

启动：

方式一：在安装目录下的 bin 目录下输入命令： ./startup.sh

方式二：在任意地方输入：startup.sh

在此方法之前，需要配置环境变量

vi /etc/profile

1.添加 TOMCAT_HOME = 目录（tomcat 的安装目录 返回到解压文件： pwd 查看路径）

2.将 TOMCAT_HOME 添加到 PATH 目录中 $Tomcat_HOME/bin:$

关闭： shutdown.sh

发布 web 项目：

步骤一：启动 Tomcat 服务器

步骤二：在 Windows 中将开发好的项目 导出为 wvar 包

步骤三：将 var 包用 xftp 上传到 Tomcat 的 webapp 目录下

步骤四：在浏览器中访问

注意：如果访问不到，需要开放端口号 8888；具体步骤如下：

```
iptables -I INPUT -p tcp --dport 8888  -j accept
sevice iptables save
```

17.2 汉得第二轮面试

1.抽象和多态

2.hashmap

3.==和 equals

4.数据库的左连接，和右连接

6.问项目中的一个问题，我的是综合查询是如何实现的

7.你觉得你有什么优势

17.3 中软国际面试题

1.JSP 的内置对象

2.hibernate 缓存

3.对还有数据库的优化经验

4.struts2 的理解，如何使用

5.json 对象在后台怎么生成

6.什么时候用到多线程

7.内存泄露

8.hibernate 的两大核心机制

17.4 软通面试题

1、数据库连接池

2、手写一个单例模式

3、给定一个英文字符串，统计大小写字母的个数

4、手写数据库中如何删除一张表中的重复记录

5、说一下集合

6、手写数据库分页(任意一种数据库)

7、说一下 java 的内存溢出，堆和栈

8、mybatis 中 sql 语句中$符合#的区别

9、给定一个程序题，求输出结果？(++n 还有 n++)

10、项目中的代码量

11、后台到前台效率低你会怎么解决？

12、多线程实现 thread 和 runable 接口的区别？两者启动线程的方式有什么不同

13、线程中 sleep()和 wait()的区别

14、ArrayList 和 linkedlist 比较？为什么？

15、什么时候用到抽象类

16、数据库中如何连接两张表？内链接和外连接的效率问题

17、说一下 java 中值传递和地址传递

18、项目中使用了 springMVC，是怎么把数据从前台传给后台，后台怎么读取。

19、怎么读取一个 txt 文件，按行输出。

20、谈谈你的项目中对高并发的优化是怎样考虑的。

21、谈谈你对 jvm 的理解

22、谈谈你对垃圾回收机制，底层算法的理解。

23、谈谈你对线程池的理解。

24、说一下 springMVC+Mybatis 工作流程。

25、谈谈你的项目中的权限是如何实现的。

26、如何在 Linux 系统配置上如何配置 eclipse

27、说说你在项目中遇到的问题，如何解决？

28、谈谈你在项目中的代码量，以及你是如何保证代码质量。

29、谈谈集合框架。

30、谈谈你对面向切面的理解

31、谈谈你对单点登录的理解，你是如何保证登录安全的。

32、谈谈 sqlserver 与 oracle 数据库的区别

33、你所知道的数据库服务器都有哪些？

17.5 上海今日信息科技有限公司

1.请说明什么是 RESTful ，对这个了解多少？到底 REST 和 SOAP、RPC 有何区别？

RESTful 是一种 http 架构风格，而不是具体的协议。

RPC 是远程方法调用，服务端和客户端可以是异构的系统，可跨平台。

SOAP 是通过 xml 来传输消息来调用 web 服务的一种协议，消息可以通过 http、smtp 等网络协议，算是 RPC 的一种 XML 封装形式。

2.数据库三大范式

3.集合

17.6 其他公司涉及数据库的题

1．请说出数据库优化的方法

1、调整数据结构的设计：

考虑是否使用 ORACLE 数据库的分区功能,对于经常访问的数据库表是否需要建立索引等。

2、调整应用程序结构设计：

考虑应用程序使用什么样的体系结构，是使用传统的 Client/Server 两层体系结构，还是使用

Browser/Web/Database 的三层体系结构。不同的应用程序体系结构要求的数据库资源是不同的

3、调整数据库 SQL 语句。

4、调整服务器内存分配：

内存分配是在信息系统运行过程中优化配置的，数据库管理员可以根据数据库运行状况调整数据库

系统 全局区（SGA 区）的数据缓冲区、日志缓冲区和共享池的大小，也可调整程序全局区（PGA 区）

的大小。

需要注意的是，SGA 区不是越大越好，SGA 区过大会占用操作系统使用的内存而引起虚拟内存的页面

交换，这样反而会降低系统。

5、调整硬盘 I/O：

数据库管理员可以将组成同一个表空间的数据文件放在不同的硬盘上，做到硬盘之间 I/O 负载均衡。

6、调整操作系统参数：

例如：运行在 UNIX 操作系统上的 ORACLE 数据库，可以调整 UNIX 数据缓冲池的大小，每个进程所

能使用的内存大小等参数。

2．如何将数据库（实例）备份和还原

对于 sql server 数据库：

一、数据备份：

BACKUP DATABASE 要备份的数据库名 TO DISK ='路径' with name = “备份的数据库起的名称 ”

二、数据还原

RESTORE DATABASE 要还原的数据库名 FROM DISK='路径' with file = 次数（第几次备份就是几）

对于 MySql 数据库：

一、数据备份

***备份一个数据库

mysqldump 基本语法：

mysqldump -u 那个用户 -p 数据库名称 table1 table2 ... > BackupName.sql

其中：

table1 和 table2 参数表示需要备份的表的名称，为空则整个数据库备份；

BackupName.sql 参数表设计备份文件的名称，文件名前面可以加上一个绝对路径。通常将数据库被分

成一个后缀名为 sql 的文件；

例如：mysqldump -u root -p test person > D:\backup.sql

***备份多个数据库

mysqldump 基本语法：

mysqldump -u 那个用户 -p --databases 数据库名 ， 数据库名 > Backup.sql

例如：mysqldump -u root -p --databases test mysql > D:\backup.sql

***备份所有数据库

mysqldump 命令备份所有数据库的语法如下：

mysqldump -u username -p -all-databases > BackupName.sql

例如:mysqldump -u -root -p -all-databases > D:\all.sql

数据还原

使用 mysqldump 命令备份的数据库的语法如下：

mysql -u root -p [dbname] < backup.sq

例如：mysql -u root -p < C:\backup.sql

3．三个数据库之间区别，为什么选用这个？

1、这三个数据库不是同一个公司的产品；

2、其所对应的使用对象也不一样，oracle 是主流的大型数据库，大多数电信项目都是使用的 oracle，

而 sqlserver 与 mysql 主要是个人以及小型公司使用的的数据库，但是 sqlserver 需要收费，mysql

不用；

3、如果按功能上来说，oracle 最为强大，oracle 支持递归查询，二后两者不支持；

4、三个数据库中，只有 sqlserver 有完整的图形化操作界面，而 oracle 与 mysql 都要借助于其他的第

三方数据库图形操作界面，比如 oracle 用的大多都是 plsql；

三个数据库之间的优缺点：

一、sqlserver

优点：

易用性、适合分布式组织的可伸缩性、用于决策支持的数据仓库功能、与许多其他服务器软件紧密关联的集成性、良好的性价比等；

为数据管理与分析带来了灵活性，允许单位在快速变化的环境中从容响应，从而获得竞争优势。从数据管理和分析角度看，将原始数据转化为商业智能和充分利用 Web 带来的机会非常重要。作为一个完备的数据库和数据分析包，SQLServer 为快速开发新一代企业级商业应用程序、为企业赢得核心竞争优势打开了胜利之门。作为重要的基准测试可伸缩性和速度奖的记录保持者，SQLServer 是一个具备完全 Web 支持的数据库产品，提供了对可扩展标记语言 (XML)的核心支持以及在 Internet 上和防火墙外进行查询的能力；

缺点：

开放性 ：SQL Server 只能 windows 上运行没有丝毫开放性操作系统系统稳定对数据库十分重要 Windows9X 系列产品偏重于桌面应用 NT server 只适合小型企业而且 windows 平台靠性安全性和伸缩性非常有限象 unix 样久经考验尤其处理大数据库；

伸缩性并行性 ：SQL server 并行实施和共存模型并成熟难处理日益增多用户数和数据卷伸缩性有限；

安全性：没有获得任何安全证书。

性能 ： SQL Server 多用户时性能佳 ；

客户端支持及应用模式： 客户端支持及应用模式。只支持 C/S 模式，SQL Server C/S 结构只支持 windows 客户用 ADO、DAO、OLEDB、ODBC 连接；

使用风险：SQL server 完全重写代码经历了长期测试断延迟许多功能需要时间来证明并十分兼容；

二、Oracle

优点：

开放性：Oracle 能所有主流平台上运行（包括 windows）完全支持所有工业标准采用完全开放策略使客户选择适合解决方案对开发商全力支持；

可伸缩性,并行性：Oracle 并行服务器通过使组结点共享同簇工作来扩展 windownt 能力提供高用性和高伸缩性簇解决方案 windowsNT 能满足需要用户把数据库移 UNIXOracle 并

行服务器对各种 UNIX 平台集群机制都有着相当高集成度；

安全性：获得最高认证级别的 ISO 标准认证。

性能：Oracle 性能高 保持开放平台下 TPC-D 和 TPC-C 世界记录；

客户端支持及应用模式：Oracle 多层次网络计算支持多种工业标准用 ODBC、JDBC、OCI 等网络客户连接

使用风险：Oracle 长时间开发经验完全向下兼容得广泛应用地风险低

缺点：

对硬件的要求很高；

价格比较昂贵；

管理维护麻烦一些；

操作比较复杂，需要技术含量较高；

三、MySql

优点：

体积小、速度快、总体拥有成本低，开源；

支持多种操作系统；

是开源数据库，提供的接口支持多种语言连接操作

MySql 的核心程序采用完全的多线程编程。线程是轻量级的进程，它可以灵活地为用户提供服务，而不过多的系统资源。用多线程和 C 语言实现的 MySql 能很容易充分利用 CPU；MySql 有一个非常灵活而且安全的权限和口令系统。当客户与 MySql 服务器连接时，他们之间所有的口令传送被加密，而且 MySql 支持主机认证；

支持 ODBC for Windows， 支持所有的 ODBC 2.5 函数和其他许多函数， 可以用 Access 连接 MySql 服务器， 使得应用被扩展；

支持大型的数据库， 可以方便地支持上千万条记录的数据库。作为一个开放源代码的数据库，可以针对不同的应用进行相应的修改。

拥有一个非常快速而且稳定的基于线程的内存分配系统,可以持续使用面不必担心其稳定性；

MySQL 同时提供高度多样性，能够提供很多不同的使用者介面，包括命令行客户端操作，网页浏览器，以及各式各样的程序语言介面，例如 C+，Perl，Java，PHP，以及 Python。你可以使用事先包装好的客户端，或者干脆自己写一个合适的应用程序。MySQL 可用于

Unix，Windows，以及 OS/2 等平台，因此它可以用在个人电脑或者是服务器上；

缺点：

不支持热备份；

MySQL 最大的缺点是其安全系统，主要是复杂而非标准，另外只有到调用 mysqladmin 来重读用户权限时才发生改变；

没有一种存储过程(Stored Procedure)语言，这是对习惯于企业级数据库的程序员的最大限制；

MySQL 的价格随平台和安装方式变化。Linux 的 MySQL 如果由用户自己或系统管理员而不是第三方安装则是免费的，第三方案则必须付许可费。Unix 或 Linux 自行安装 免费 、Unix 或 Linux 第三方安装 收费；

4．左右链接

左链接的左表不管是否符合 on 的链接条件都会出现在数据集中，右表结果集中只会在符合 on 条件时

出现，就像内链接似的

5．有十万条数据截取后十条

select * from (select top 10 * from tableName order by 字段 desc) order by 字段 asc

6．三种数据库分页用关键字

7．A 表数据插入 B

第一种：INSERT INTO 表 2（要插入值的字段 1 ，要插入值的字段 2 ， ...）

SELECT （获取值的表中的字段 1 ，获取值的表中的字段 1 ，.... ）

FROM 表 1 （注：表 2 事先要存在）

举例：

```
CREATE  TABLE  student(
  s_no  INT,
  s_name  VARCHAR(20)
)
CREATE  TABLE  student1(   ---- 事先要存在
  s_no  INT,
```

```
    s_name  VARCHAR(20),
    s_age  INT ,
    s_address  VARCHAR(20)
  )

  INSERT  INTO  student1(s_no , s_name)
  SELECT s_no , s_name  FROM  student
```

第二种：select * from 目标表 from 资源表（注：目标表事先不存在）

举例：

```
SELECT s_no , s_name , s_age  INTO  student2
FROM  student1(s_no , s_name , s_age);   ----  student2 事先不存在
```

8．你使用过 Powerdesigner？在设计数据库时怎么避免数据冗余的情况？

设计尽量符合数据库范式，保证数据的完整性和唯一性。

9．数据库的存储过程

一、什么是存储过程：

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的 SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。

二、存储过程的优点：

1.执行速度更快 - 在数据库中保存的存储过程语句都是编译过的

2.允许模块化程序设计 - 类似方法的复用

3.提高系统安全性 - 防止 SQL 注入

4.减少网络流通量 - 只要传输存储过程的名称

系统存储过程一般以 sp 开头，用户自定义的存储过程一般以 usp 开头

三、定义存储过程语法，"[" 里面的内容表示可选项

```
create proc 存储过程名
  @参数 1                数据类型 [=默认值] [output],
  @参数 2                数据类型 [=默认值] [output]
  ...
  as
```

SQL 语句

10．数据库的左外连接、右外连接、内连接、外连接以及如何运用

内连接：inner　join　---》 两表存在主外键关系的时候常用

外连接：1、左外连接：left　outer　join

左外连接的结果集包括 left outer　join 子句中指定的左表的所有行，若左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有的选择列均为空值

2、右外连接：right　outer　join

右外连接的结果集包括 right outer　join 子句中指定的右表的所有行，若右表的某行在左表中没有匹配行，则在相关联的结果集行中左表的所有的选择列均为空值

3、完全外连接：full　outer　join

交叉连接：cross　join　---》 笛卡尔积

等值连接：　----》 适用于主外键关系的表

表 1　，　表 2　where　表 1 ． 字段 = 表 2 ． 字段

11．主键生成策略有哪些？在数据库转移时会产生哪些常见的问题？

主键生成方式主要有以下几种：

一、自增长

当我们想向表中插入一行数据时，不必考虑主键字段的取值，记录插入后，数据库系统会自动为其分配一个值，这个值是成自然数字增长的，确保不出现主键重复。

在 MySql 中,把主键设为 auto_increment 类型,数据库会自动为其赋值。ep:create table user{

id int auto_increment primary key not null,name varchar(15)}。使用 Navicat 工具，要设置主键为 int 型时，设计表结构里面才有 Auto increment。

二、手动增长

手动设置字段作为主键，即主键的值需要自己维护。通常情况下需要单独的表存储当前主键值。

三、GUID 值

GUID 是全局唯一标示符的简称，也叫 UUID。GUID 是一种算法生成的二进制长度为 128 位的数字标识符。

四、COMB 类型

COMB 类型可以说是 GUID 的改进类型，它组合 GUID 和系统时间，使其在索引和检索上有更有的性能。

项目中的问题

1. 如何保证你的代码在运行中的质量？
2. 你的项目中大概写了多少行代码？
3. 常用设计模式
4. 详细描述一下项目中设计的技术原理，例如 AJax 原理、jQuery
5. 有没有在项目中用到多线程，什么情况下用的；
6. 有没有用到单例，在什么情况下用到单例，项目中你写到的单例说说；
7. 开发中常用的版本控制工具
8. 开发中遇到什么难题 怎么解决的
9. 保持项目编码一致的方法
10. 看你项目有前端技术，那你怎么看待 HTML5？它和 HTML 有什么技术上的差异？
11. 项目中你使用的分页技术有哪些？是你们自己实施的，还是。。。。？
12. 你在项目中时如何实现权限管理的？是动态加载的目录,还是动态调用的数据库？
13. 你在项目中使用过多线程吗？怎么实现多线程？怎么避免多线程访问数据库时产生的高并发问题？
14. 项目中你只是用过版本控制工具 SVN 吗？那你还了解哪些版本控制工具？
15. 详细谈谈你的项目所使用的技术以及你是如何整合的？
16. 你在项目中哪些用了原型模式\单例\工程模式
17. 你的项目中什么数据用到了存储过程和游标,请赘述一下
18. 你在项目中涉及到了什么设计模式。
19. 挑一个你熟悉的项目描述一下。
20. 你在项目中怎么实现表单验证
21. 你在 OA 系统中怎么实现的签到签退功能
22. 你的电子商务系统里怎么实现的商品购买与商品库同步,是使用简单的同步数据还是其他的？
23. 你项目中有没有使用过抽象类或者接口？
24. 介绍一下你做的项目

25．说说你负责的模块（权限管理是怎么实现的）

26．项目中出现了什么问题，怎么解决的？

27．项目简单的介绍；

28．前台用到了什么技术；

29．后台用到了什么技术；

30．项目周期，项目分工；

31．什么是 hibernate，hibernate 在你的项目中是怎么用的？

32．项目测试人员，反馈的测试问题数量，印象最深刻的一个；

33．你的项目用的是什么框架，是怎么用的？

34．form 表单前台如何传值？后台如何获取值？

35．项目中有没有用到 Ajax;

下表列出四种主键生成方式优缺点的比较：

主键生成策略	优点	缺点
自动增长字段	1. 使用简单	1. 不同数据库获取当前值方式不同； 2. 难以应用在多个数据库间进行数据迁移的情况。
手动增长型字段	1. 可以获得最新键值 2. 可以确保数据合并过程中不会出现键值冲突	1. 通常情况下需要建立一张单独的表存储当前主键键值； 2. 增加一次数据库访问来获取当前主键键值； 3. 考虑并发冲突等，增加系统的复杂程度。
使用GUID	1. 直接生成GUID，获得最新键值以填充主键，使用方便； 2. 可以确保数据合并过程中不会出现键值冲突； 3. 避免了前两种方式获取当前键值所增加的开销。	1. 占用较多存储空间； 2. 索引耗时； 3. 在多表链接查询时效率不如int型
使用"COMB"类型	1. 保留GUID的已有优点； 2. 利用时间信息与GUID组合起来，增加有序性以提高索引效率。	1. 需要设计COMB的生成算法； 2. 和GUID一样占用较多存储空间； 3. 在多表链接查询时效率不如int型，但优于GUID。

数据库转移时会常见的问题： ----没找到

12. **oracle 的事物级别（事务隔离级别）**

隔离级别（isolation level），是指事务与事务之间的隔离程度。

ANSI/ISO SQL92 标准中定义了 4 种事务隔离级别：

1.序列化（serializable）

最高隔离级别。系统中所有的事务都是一个接一个执行的。因此也就不会发生任何事务之间的冲突问题。

2.可重复读（repeatable read）

一个事务所读取的数据记录不允许被其他事务所修改。

3.读已提交（read committed）

该级别允许其他事务修改当前事务所读取的数据记录，并且那个事务提交之后，当前事务可以看到修改后的数据。

4.读未提交（read uncommitted）

该级别允许其他事务修改当前事务所读取的数据记录，并且那个事务尚未提交时，当前事务就可以看到修改后的数据。即允许脏读。

事务隔离级别不同，执行一条数据库查询，得到结果的总结：

1. 脏读

读取了其他事务还没有提交的数据。

2.不可重复读

当前事务已经读取的数据记录，被其他事务修改或删除。

3.幻影读

其他事务插入了新的数据，当前事务以相同的查询条件，在那个事务插入数据之前和之后查询数据，得到的数据记录的条数不一样。

Oracle 共支持 3 种事务隔离级别：

1.serializable

2.read committed

3.read only （Oracle 自己独有的事务隔离级别）

Oracle 默认的隔离级别是 read committed。

查看数据库隔离级别的方法：

```
1. SELECT * FROM dual FOR UPDATE;
2. SELECT s.sid, s.serial#,
   CASE BITAND(t.flag, POWER(2, 28))
     WHEN 0 THEN 'READ COMMITTED'
     ELSE 'SERIALIZABLE'
     END AS isolation_level
     FROM v$transaction t
   JOIN v$session s ON t.addr = s.taddr AND s.sid = sys_context('USERENV', 'SID');
```

设置隔离级别使用

```
SET TRANSACTION ISOLATION LEVEL [READ UNCOMMITTED|READ COMMITTED|REPEATABLE READ|SERIALIZABLE]
```

13．Oracle 索引有哪些

按照逻辑分类：

Single column 单行索引

Concatenated 多行索引

Unique 唯一索引

NonUnique 非唯一索引

Function-based 函数索引

Domain 域索引

按照物理分类：

Partitioned 分区索引

NonPartitioned 非分区索引

B-tree：

Normal 正常型 B 树

Rever Key 反转型 B 树

Bitmap 位图索引

14．将数据库中的某一张表中的重复字段去掉。

自己写

15．一张员工表，一张部门表，手写 sql 查询各部门的人数。

自己写

16．两表联查的 sql 语句

内连接：

select 字段 1，字段 2 ... from 表 1 join 表 2 on 表 1.字段名 = 表 2.字段名;

左外连接：

select * from 表 1 left outer join 表 2 on 表 1.字段名 = 表 2.字段名;

右外连接：

select * from 表 1 right outer join 表 2 on 表 1.字段名 = 表 2.字段名;

全连接：

select * from 表 1 full outer join 表 2 on 表 1.字段名 = 表 2.字段名;

等值连接：

select * from 表 1 ，表 2 where 表 1.字段名 = 表 2.字段名;

交叉连接：

select * from 表1 cross join 表2 on 表1.字段名 = 表2.字段名;

17．数据库三大范式

第一范式：对于表中的每一行，必须且仅仅有唯一的行值.在一行中的每一列仅有唯一的值并且具有原子性.

（第一范式是通过把重复的组放到每个独立的表中,把这些表通过一对多关联联系起来这种方式来消除重复组的）

第二范式：第二范式要求非主键列是主键的子集，非主键列活动必须完全依赖整个主键。主键必须有唯一性的元素,一个主键可以由一个或更多的组成唯一值的列组成。一旦创建，主键无法改变，外键关联一个表的主键。主外键关联意味着一对多的关系.（第二范式处理冗余数据的删除问题。当某张表中的信息依赖于该表中其它的不是主键部分的列的时候，通常会违反第二范式）

第三范式：第三范式要求非主键列互不依赖.（第三范式规则查找以消除没有直接依赖于第一范式和第二范式形成的表的主键的属性。我们为没有与表的主键关联的所有信息建立了一张新表。每张新表保存了来自源表的信息和它们所依赖的主键）

第四范式：第四范式禁止主键列和非主键列一对多关系不受约束

第五范式：第五范式将表分割成尽可能小的块，为了排除在表中所有的冗余

18．在数据库中如何保证字段唯一确定一个值？

有两种方法：

1. 将字段设置为主码（一个表只能定义一个主码）：

2. 设置字段为唯一值约束(一个表可以定义多个唯一值约束)：

19．oracle 和 MySql 的区别，sql 语句有什么区别；

Sql 语句的区别：

1、子查询的规则

在 Oracle 中写子查询，只需要将子查询用括号括起即可（select A, B,?C from TABLE?where CONDITION）

在 MySql 中写子查询，需要在括号后面写上关键字 as Alias (假名)

2、行号 rownum 的书写规范

在 Oracle 中查询行号，直接写 rownum 就可以，还能起别名 select?ROWNUM row from TABLE where CONDITION

在 MySql 中查寻行号,需要在 rownum 前加@,且不能起别名 select @ROWNUM from TABLE where CONDITION

3、分页

分页查询时 oracle 用的伪列（rownum）

mysql 用的是 limit

4、其他

oracle 中有 number 型，有大数据类型，mysql 没得；

oracle 不能设置列自动增长,而 mysql 是可以的,oracle 可以用序列加触发器来解决自动增长问题达到与 mysql 一样的效果。

Mysql 与 Oracle 的区别：

1.组函数用法规则

mysql 中组函数在 select 语句中可以随意使用

oracle 中如果查询语句中有组函数,那其他列名必须是有组函数处理过的,或者是 group by 子句中的列，否则报错

2.自动增长的数据类型处理

MYSQL 有自动增长的数据类型，插入记录时不用操作此字段，会自动获得数据值。

ORACLE 没有自动增长的数据类型，需要建立一个自动增长的序列号，插入记录时要把序列号的下一个值赋于此字段。

3.单引号的处理

MYSQL 里可以用双引号包起字符串,ORACLE 里只可以用单引号包起字符串。在插入和修改字符串前必须做单引号的替换：把所有出现的一个单引号替换成两个单引号。

4.翻页的 SQL 语句的处理

MYSQL 处理翻页的 SQL 语句比较简单，用 LIMIT 开始位置，记录个数；

ORACLE 处理翻页的 SQL 语句就比较繁琐了。每个结果集只有一个 ROWNUM 字段标明它的位置，并且只能用 ROWNUM<100，不能用 ROWNUM>80。

以下是经过分析后较好的两种 ORACLE 翻页 SQL 语句(ID 是唯一关键字的字段名)：

语句一：

SELECT ID, [FIELD_NAME,...] FROM TABLE_NAME WHERE ID IN (SELECT ID FROM (SELECT ROWNUM AS NUMROW, ID FROM TABLE_NAME WHERE 条件 1 ORDER BY 条件 2) WHERE NUMROW > 80 AND NUMROW < 100) ORDER BY 条件 3;

语句二：

SELECT * FROM ((SELECT ROWNUM AS NUMROW, c.* from (select [FIELD_NAME,...] FROM TABLE_NAME WHERE 条件 1 ORDER BY 条件 2) c) WHERE NUMROW > 80 AND NUMROW < 100) ORDER BY 条件 3;

5.长字符串的处理

长字符串的处理 ORACLE 也有它特殊的地方。INSERT 和 UPDATE 时最大可操作的字符串长度小于等于 4000 个单字节，如果要插入更长的字 符串，请考虑字段用 CLOB 类型，方法借用 ORACLE 里自带的 DBMS_LOB 程序包。插入修改记录前一定要做进行非空和长度判断，不能为空的字段值和 超出长度字段值都应该提出警告，返回上次操作。

6.日期字段的处理

MYSQL 日期字段分 DATE 和 TIME 两种

ORACLE 日期字段只有 DATE，包含年月日时分秒信息，用当前数据库的系统时间为 SYSDATE，精确到秒

字符串转换成日期型函数 TO_DATE(‘2001-08-01’,’ YYYY-MM-DD’)

日期型字段转换成字符串函数 TO_CHAR(‘2001-08-01’,’ YYYY-MM-DD HH24:MI:SS’)

日期字段的数学运算公式有很大的不同

MYSQL 找到离当前时间 7 天用 DATE_FIELD_NAME > SUBDATE(NOW(), INTERVAL 7 DAY)

ORACLE 找到离当前时间 7 天用 DATE_FIELD_NAME >SYSDATE - 7;

MYSQL 中插入当前时间的几个函数是：

NOW()函数以``YYYY-MM-DD HH:MM:SS’返回当前的日期时间，可以直接存到 DATETIME 字段中。CURDATE()以’YYYY-MM-DD’的格式返回今天的日期，可以 直接存到 DATE 字段中。CURTIME()以’ HH:MM:SS’的格式返回当前的时间，可以直接存到 TIME 字段中。

oracle 中当前时间是 sysdate

7.空字符的处理

MYSQL 的非空字段也有空的内容，ORACLE 里定义了非空字段就不容许有空的内容。按 MYSQL 的 NOT NULL 来定义 ORACLE 表结构，导数据的时候会产生错误。因此导数据时要对空字符进行判断，如果为 NULL 或空字符，需要把它改成一个空格的字符串。

8.字符串的模糊比较

MYSQL 里用字段名 like%‘字符串%’

ORACLE 里也可以用字段名 like%‘字符串%’但这种方法不能使用索引，速度不快，用字符串比较函数 instr(字段名，‘字符串’)>0 会得到更精确的查找结果。

9.程序和函数里，操作数据库的工作完成后请注意结果集和指针的释放

20．oracle 中左右链接怎么实现

Oracle 表之间的连接分为三种：

1. 内连接(自然连接)

2. 外连接

（1）左外连接 (左边的表不加限制)

（2）右外连接(右边的表不加限制)

（3）全外连接(左右两表都不加限制)

3. 自连接（同一张表内的连接）

具体语法：

如果使用 from 子句指定内、外连接，则必须要使用 on 子句指定连接条件；

如果使用（+）操作符指定外连接，则必须使用 where 子句指定连接条件。

一． 内连接（InnerJoin/Join）

Inner join 逻辑运算符返回满足第一个（顶端）输入与第二个（底端）输入联接的每一行。这个和用 select 查询多表是一样的效果，所以内连接用的很少。

还有一点要说明的就是 Join 默认就是 inner join。 所以我们在写内连接的时候可以省略 inner 这个关键字。

举例：

创建 2 张测试表并插入数据：

```
SQL> select * from dave;

ID                  NAME

---------- ----------

1                       dave

2                       bl

1                       bl

2                       dave

SQL> select * from bl;
```

ID NAME

---------- ----------

1 dave

2 bl

用内链接进行查询：

Select a.id, a.name, b.name from davea inner join bl b on a.id = b.id; -- 标准写法

Select a.id,a.name,b.name from davea join bl b on a.id=b.id; -- 这里省略了 inner 关键字

Select a.id,a.name,b.name from davea,bl b where a.id=b.id; -- select 多表查询

二、 自然连接(Natural join)

自然连接是在两张表中寻找那些数据类型和列名都相同的字段，然后自动地将他们连接起来，并返回所有符合条件按的结果。

举例：

SQL> Select id,name from dave a natural join bl b;

这里我们并没有指定连接的条件，实际上 oracle 为我们自作主张的将，dave 表中的 id 和 name 字段与 bl 表中的 id 和 name 字段进行了连接。也就是实际上相当于：

Select dave.id,bl.name From davejoin bl on dave.id = bl.id and dave.name=bl.name;

因此，我们也可以将自然连接理解为内连接的一种。

有关自然连接的一些注意事项：

（1）如果做自然连接的两个表的有多个字段都满足有相同名称个类型，那么他们会被作为自然连接的条件。

（2）如果自然连接的两个表仅是字段名称相同，但数据类型不同，那么将会返回一个错误。

三、 外连接(OuterJoin)

outer join 则会返回每个满足第一个（顶端）输入与第二个（底端）输入的联接的行。它还返回任何在第二个输入中没有匹配行的第一个输入中的行。

外连接分为三 种：左外连接，右外连接，全外连接。 对应 SQL：LEFT/RIGHT/FULLOUTER JOIN。通常我们省略 outer 这个关键字。 写成：LEFT/RIGHT/FULL JOIN。

在左外连接和右外连接时都会以一张表为基表，该表的内容会全部显示，然后加上两张表匹配的内容。如果基表的数据在另一张表没有记录。 那么在相关联的结果集行中列显示为空值（NULL）。

对于外连接，也可以使用“(+) ”来表示。关于使用（+）的一些注意事项：

1.（+）操作符只能出现在 where 子句中，并且不能与 outer join 语法同时使用。

2. 当使用（+）操作符执行外连接时，如果在 where 子句中包含有多个条件，则必须在所有条件中都包含（+）操作符

3.（+）操作符只适用于列，而不能用在表达式上。

4.（+）操作符不能与 or 和 in 操作符一起使用。

5.（+）操作符只能用于实现左外连接和右外连接，而不能用于实现完全外连接。

举例：

左外连接（Left outer join/ left join）

left join 是以左表的记录为基础的,示例中 Dave 可以看成左表,BL 可以看成右表,它的结果集是 Dave 表中的数据，在加上 Dave 表和 BL 表匹配的数 据。换句话说,左表(Dave)的记录将会全部表示出来,而右表(BL)只会显示符合搜索条件的记录。BL 表记录不足的地方均为 NULL.

示例：

```
select * from dave a left join bl b on a.id = b.id;   --- 无 outer
select * from dave a left outer join bl b on a.id = b.id;  --- 有 outer
Select *from dave a,bl b where a.id=b.id(+);      ---- +号
```

注：

用（+）就要用关键字 where，

用（+）来实现，这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。所以加号写在右表，左表就是全部显示，故是左连接。

右外连接（rightouter join/ right join）

和 left join 的结果刚好相反,是以右表(BL)为基础的， 显示 BL 表的所以记录,在加上 Dave 和 BL 匹配的结果。 Dave 表不足的地方用 NULL 填充.

示例：

select * from dave a right join bl bon a.id = b.id;

select * from dave a right outerjoin bl b on a.id = b.id;

Select * from dave a,bl b wherea.id(+)=b.id;

注：

用（+）来实现，这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。所以加号写在左表，右表就是全部显示，故是右连接。

全外连接（full outer join/ full join）

左表和右表都不做限制，所有的记录都显示，两表不足的地方用 null 填充。全外连接不支持（+）写法。

示例：

select * from dave a full join bl bon a.id = b.id;

select * from dave a full outer joinbl b on a.id = b.id;

四、自连接

自连接(self join)是 SQL 语句中经常要用的连接方式，使用自连接可以将自身表的一个镜像当作另一个表来对待，从而能够得到一些特殊的数据。自连接的本意就是将一张表看成多张表来做连接。

21．oracle 中数据由谁管理

22．简单介绍 oracle 中用户和权限之间的关系

23．请说明数据库主键、外键的作用，以及建立索引的好处与坏处？

1.主键的作用：

1、唯一的标识一行

2、作为一个可以被外键有效引用的对象

2.外键的作用：

1、数据库通过外键保证数据的一致性和完整性

2、增加 E-R 图的可读性

3.建立索引的好处：

提高查询速度，加速表与表之间的连接，减低查询中分组与排序的时间

4.建立索引的坏处：

存储索引占用磁盘时间，执行数据修改操作（insert ， update ， delete）产生索引维护

24．如何写出高性能的 SQL 语句？

4.选择最优效率的表名顺序

5.Select 子句中避免使用“*”

6.减少访问数据库的次数

7.整合简单，无关联的数据库访问

8.删除重复记录

9.用 truncate 语句代替 delete

10.用 where 子句代替 having 子句

11.减少对表的查询

12.用内部函数和索引提高效率

25．请简述什么是事务？事务有哪些特性？

事务(Transaction)是访问并可能更新数据库中各种数据项的一个程序执行单元(unit)。事务由事务开始(begin transaction)和事务结束(end transaction)之间执行的全体操作组成。

事务的特性：事务是恢复和并发控制的基本单位。

事务具有 4 个属性：原子性、一致性、隔离性、持续性。这四个属性通常称为 ACID 特性。

1.原子性（atomicity）:一个事务是一个不可分割的工作单位，事务中的操作要么都做，要么都不做。

2.一致性(consistency):事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原

子性是密切相关的。

3.隔离性（isolation）。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对

并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

4.持久性（durability）。持续性也称永久性（permanence），指一个事务一旦提交，它对数据库中数

据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

26．设计一个数据库（有西安市，宝鸡市，碑林区，雁塔区，...，以及凤仪路，高新路等对应的人数），要求设计一个数据库表，可快速求出某个市对应某个区的总人数，或者某个市的总人数（因我设计时提到了主键自动生成，所以他问了一下：自动生成的数据类型可以是 varchar()吗？）