

---

## 2. 笔试题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. 公司想招什么样的人 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，很容易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。

### 3. 笔试题之 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，有些题来自传智播客学员面试后的反馈，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、一个".java"源文件中是否可以包括多个类（不是内部类）？有什么限制？

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

#### 2、Java 有没有 goto？

java 中的保留字，现在没有在 java 中使用。

#### 3、说说&和&&的区别。

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

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

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

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

## 4、在 JAVA 中如何跳出当前的多重嵌套循环？

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

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

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

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

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

在 switch (expr1) 中，expr1 只能是一个整数表达式或者枚举常量（更大字体），整数表达式可以是 int 基本类型或 Integer 包

装类型，由于，byte,short,char 都可以隐含转换为 int，所以，这些类型以及这些类型的包装类型也是可以的。显然，long 和 String 类型都不符合 switch 的语法规则，并且不能被隐式转换成 int 类型，所以，它们不能作用于 switch 语句中。

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

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

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

## 7、char 型变量中能不能存贮一个中文汉字?为什么?

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

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

## 8、用最有效率的方法算出 2 乘以 8 等於几?

$2 \ll 3$ ,

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  $2 \ll 3$ 。

## 9、请设计一个一百亿的计算器

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

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

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

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

- （）这个类内部有两个成员变量，一个表示符号，另一个用字节数组表示数值的二进制数
- （）有一个构造方法，把一个包含有多位数值的字符串转换到内部的符号和字节数组中

( ) 提供加减乘除的功能

---

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

    }
    public BigInteger subtract(BigInteger other)
    {

    }
    public BigInteger multiply(BigInteger other)
    {

    }
    public BigInteger divide(BigInteger other)
    {

    }
}
```

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

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

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

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

执行如下语句将报告编译期错误：

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

但是，执行如下语句则可以通过编译：

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

---

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

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

实际上，这是办不到的，在该方法内部仍然可以增加如下代码来修改参数对象：

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

## 11、"=="和 equals 方法究竟有什么区别？

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

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

如果一个变量指向的数据是对象类型的，那么，这时候涉及了两块内存，对象本身占用一块内存（堆内存），变量也占用一块内存，例如 `Object 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，也是在使用==操作符来比较两个变量指向的对象是否是同一对象，这时候使用 `equals` 和使用==会得到同样的结果，如果比较的是两个独立的对象则总返回 `false`。如果你编写的类希望能够比较该类创建的两个实例对象的内容是否相同，那么你必须覆盖 `equals` 方法，由你自己写代码来决定在什么情况即可认为两个对象的内容是相同的。

## 12、静态变量和实例变量的区别？

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

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

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

```
public class VariantTest
```

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

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

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

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

## 14、Integer 与 int 的区别

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

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

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

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

Math 类中提供了三个与取整有关的方法：ceil、floor、round，这些方法的作用与它们的英文名称的含义相对应，例如，ceil 的英文意义是天花板，该方法就表示向上取整，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。

## 15、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;
```

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

这四个作用域的可见范围如下表所示。

说明：如果在修饰的元素上面没有写任何访问修饰符，则表示 friendly。

作用域	当前类	同一 package	子孙类	其他 package
public	√	√	√	√
protected	√	√	√	×
friendly	√	√	×	×
private	√	×	×	×

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

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

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

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

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

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

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

- 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 `private`，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。

### 3、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？

说说对 `javaee` 中的 `session` 的理解，你是怎么用 `session` 的？

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

### 4. 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

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代码，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。分层就是要将界面部分、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号后，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？说

（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。

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

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

## 6.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 oriented 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配两个属性, 分别表示目标和切面对象 (Advisor)。

### 3、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。

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

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

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

## 5、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。

## 6、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，这就是一种抽象，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，即一个引用变量到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()在运行的时候返回，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？

## 7、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。

## 8、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 IOException,ServletException
```

```

{
    记录访问日志
    进行权限判断
    if(具有权限)
    {
        try
        {
            doService(request,response);
        }
        catch(Exception e)
        {
            记录异常信息
        }
    }
}

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

## 9、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，FileOutputStream 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，从来没见过过这种情况，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

## 10、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，我们也可以在外部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 Runnable(){  
        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。

## 11、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。

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

可以继承其他类或实现其他接口。不仅是可能，而是必须!

## 13、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();
```

## 14. jdk 中哪些类是不能继承的?

不能继承的类是那些用 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被 final 的，在 jdk 中 System,String,StringBuffer 等都是基本类型。

## 2、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

## 111、~~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。也就是说，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。

## 41、是否可以继承 String 类？

String 类是 final 类故不可以继承。

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

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

## 5、String 和 StringBuffer 的区别

JAVA 平台提供了两个类: String 和 StringBuffer, 它们可以储存和操作字符串, 即包含多个字符的字符数据。String 类表示内容不可改变的字符串。而 StringBuffer 类表示内容可以被修改的字符串。当你知道字符数据要改变的时候你就可以使用 StringBuffer。典型地, 你可以使用 StringBuffers 来动态构造字符数据。另外, String 实现了 equals 方法, new String("abc").equals(new String("abc"))的结果为 true,而 StringBuffer 没有实现 equals 方法, 所以, new StringBuffer("abc").equals(new StringBuffer("abc"))的结果为 false。

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

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

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

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

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

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

## 8.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。

## 3、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 `StringTokenizer`, 代码为: `StringTokenizer tokenizer = StringTokenizer(orgStr,",");`  
`String [] result = new String[tokenizer.countTokens()];`  
`Int i=0;`  
`while(tokenizer.hasNext()){result[i++]=tokenizer.nextToken();}`

## 38、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 `length()` 这个方法。

## 39、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`。

## 43、`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。

## 7、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

## 12、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

## 5、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。

## 15、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。

## 50、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；



### 33、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`。

### 96、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？

### 99、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 Runnable(){public void run(){} });
}
```

```
Executors.newCachedThreadPool().execute(new Runnable(){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。

## 13、sleep() 和 wait() 有什么区别？

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

`sleep` 就是正在执行的线程主动让出 `cpu`，`cpu` 去执行其他线程，在 `sleep` 指定的时间过后，`cpu` 才会回到这个线程上继续往下执行，如果当前线程进入了同步锁，`sleep` 方法并不会释放锁，即使当前线程使用 `sleep` 方法让出了 `cpu`，但其他被同步锁挡住了的线程也无法得到执行。`wait` 是指在一个已经进入了同步锁的线程内，让自己暂时让出同步锁，以便其他正在等待此锁的线程可以得到同步锁并运行，只有其他线程调用了 `notify` 方法（`notify` 并不释放锁，只是告诉调用过 `wait` 方法的线程可以去参与获得锁的竞争了，但不是马上得到锁，因为锁还在别人手里，别人还没释放。如果 `notify` 方法后面的代码还有很多，需要这些代码执行完后才会释放锁，可以在 `notify` 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!");
```

```
}
```

```
}
```

```
}
```

```
}
```

## 16、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。

## 17. 下面两个方法同步吗？（自己发明）

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

    }

    synchronized void getX() {}
}
```

## 56、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。

## 30、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。

## 47、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**。

## 58、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个执行线索同时执行，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，在这种情况下可以调用 **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。

## 71、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();
                }
            }
        }
    }
}

```

## 28、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++;
}

}
```

---

**28、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：

[illegible]

```

        "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();
```

```
    }
```

```
}
```

}

### 3、介绍 Collection 框架的结构

答：随意发挥题，天南海北谁便谈，只要让别觉得你知识渊博，理解透彻即可。

### 3、Collection 框架中实现比较要实现什么接口

comparable/comparator

### 3、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。

### 4、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 是线程不安全，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

## 5、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。

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

List, Set 是，Map 不是

## 109、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(Object)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, Object)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(Object key, Object 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。

## 7、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。

## 4、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);

## 9、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。



### 39、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。

### 53、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

Properties→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 eclipse 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。

### 45、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。

## 46、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());  
}  
  
}
```

## 112、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 FileWriter 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.struts.action,org.hibernate

常用的接口：Remote List Map Document

NodeList ,Servlet,HttpServletRequest,HttpServletResponse,Transaction(Hibernate)、Session(Hibernate),HttpSession

## 100、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。

## 102、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);
    }
}
```

## 105、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, 如果要是让 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。

## 54、描述一下 JVM 加载 class 文件的原理机制?

JVM 中类的装载是由 ClassLoader 和它的子类来实现的, Java ClassLoader 是一个重要的 Java 运行时系统组件。它负责在运行时查找和装入类文件的类。

## 18、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。

## 24、GC 是什么? 为什么要有 GC?

GC 是垃圾收集的意思 (Garbage 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。

## 51、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。

## 103、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。

## 23、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.huawei.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;
    }
}
```

## 101、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，即对象编程了孤儿的时候，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[200000000];
        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到的，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去，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。

## 8、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/classes目录中的类，然后才委托上级的类加载器加载，如果我们在tomcat的web应用程序中写一个java.lang.String，这时候Servlet程序加载的就是我们自己写的java.lang.String，但是这么干就会出很多潜在的问题，原来所有用了java.lang.String类的都将出现问题。

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

---

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

```
package 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。

## 2. Java 代码查错

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。

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）。

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)。

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。

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 variable (成员变量), 而 o 的 reference 并没有改变。

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 属于 instant variable (实例变量, 或叫成员变量)。instant variable 有 default value。int 的 default value 是 0。

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;"。

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。

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。

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。

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。

## 4. 算法与编程

### 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.Pattern;

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");
        }
    }
}
}
}
}

```

**1、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++];
    }
}

```

## 1、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** 对象就是过滤的策略对象，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 implements 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[])acceptedFilesList.toArray();  
}
```

1、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;
}
```

1、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"aaaabbcc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。

```

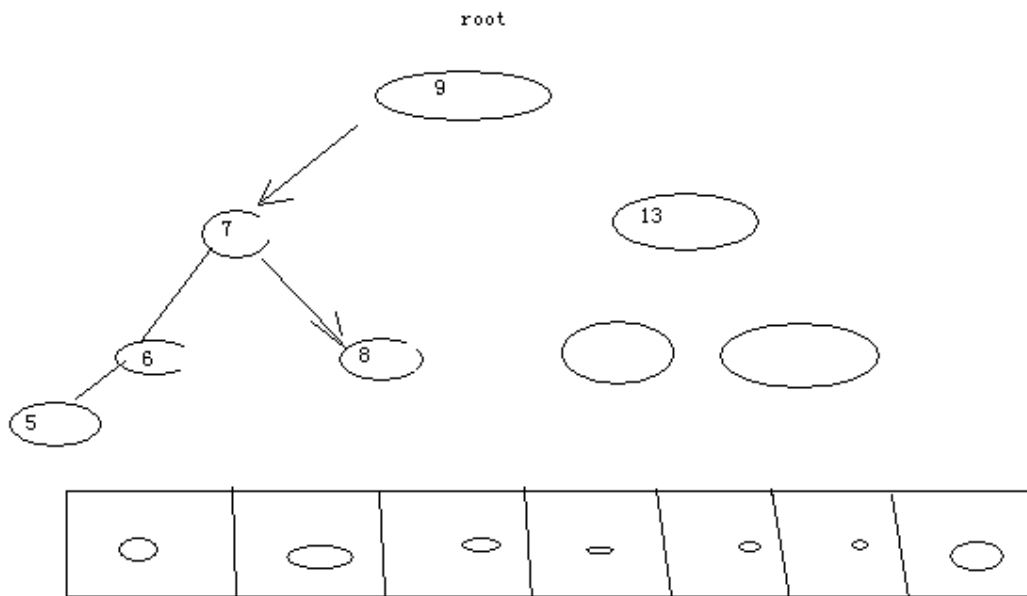
int englishCount;
int chineseCount;
int digitCount;
for(int i=0;i<str.length;i++)
{
    char ch = str.charAt(i);
    if(ch>='0' && ch<='9')
    {
        digitCount++;
    }
    else if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))
    {
        englishCount++;
    }
    else
    {
        chineseCount++;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

```

## 1、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;  
}
```

```
}
```

## 1、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);
    }
}
}
}

```

## 48、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，在类初始化时实例化，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

## 7、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);
```

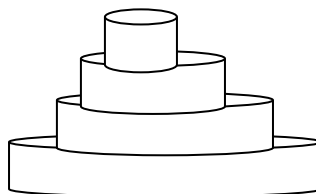
```
    if (n<=5000)
```

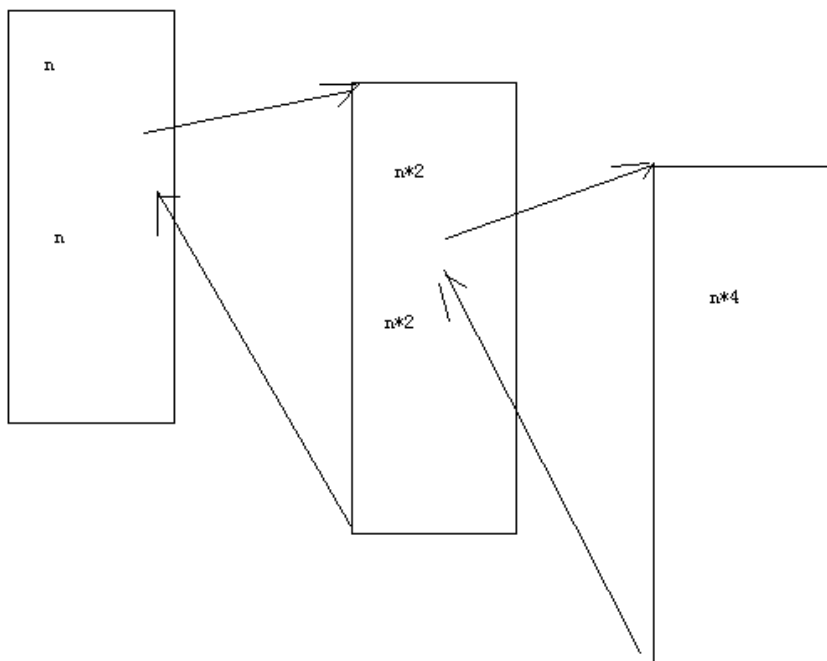
```
        doubleNum(n*2);
```

```
    System.out.println(n);
```

```
}
```

$$\text{Gaibaota}(N) = \text{Gaibaota}(N-1) + n$$





## 7、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);
    }
```

## 94、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]+" ");  
}  
}
```

```
}
```

## 7、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. ~~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();
    }
}
```

## 5. html&JavaScript&ajax 部分

### 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>
```

**1. 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>
```

## 1、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>
```

---

## 2、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。

## 1、说说你用过那些 ajax 技术和框架，说说它们的区别

答: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

## 6. Java web 部分

### 1、Tomcat 的优化经验

答: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

## 1、HTTP 请求的 GET 与 POST 方式的区别

---

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。

## 62、解释一下什么是 servlet;

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。

## 1、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。

## 4、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 {  
    }  
}
```

## 3、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。

## 60、什么情况下调用 doGet()和 doPost()?

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()。

## 66、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 name 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  
getContentTypeLength(): 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

## 19、forward 和 redirect 的区别

forward 是服务器请求资源, 服务器直接访问目标地址的 URL, 把那个 URL 的响应内容读取过来, 然后把  
这些内容再发给浏览器, 浏览器根本不知道服务器发送的内容是从哪儿来的, 所以它的地址栏中还是原  
来的地址。

redirect 就是服务端根据逻辑, 发送一个状态码, 告诉浏览器重新去请求那个地址, 一般来说浏览器会用刚才请  
求的所有参数重新请求, 所以 session,request 参数都可以获取。

## 4、request.getAttribute() 和 request.getParameter() 有何区别?

### 1. 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

## 2. jsp 有哪些动作?作用分别是什么?

（这个问题似乎不重要，不明白为何有此题）

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

## 59、JSP 的常用指令

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

## 3. 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 %>`

## 4、两种跳转方式分别是什么?有什么区别?

(下面的回答严重错误,应该是想问 forward 和 sendRedirect 的区别,毕竟出题的人不是专业搞文字艺术的人,可能表达能力并不见得很强,用词不一定精准,加之其自身的技术面也可能存在一些问题,不一定真正将他的意思表达清楚了,严格意思上来讲,一些题目可能根本就无人能答,所以,答题时要掌握主动,只要把自己知道的表达清楚就够了,而不要去推敲原始题目的具体含义是什么,不要一味想着是在答题)

答:有两种,分别为:

```
<jsp:include page=included.jsp flush=true>
```

```
<jsp:forward page= nextpage.jsp/>
```

前者页面不会转向 include 所指的页面,只是显示该页的结果,主页面还是原来的页面。执行完后还会回来,相当于函数调用。并且可以带参数.后者完全转向新页面,不会再回来。相当于 go to 语句。

## 63、页面间对象传递的方法

request, session, application, cookie 等

## 64、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。

## 1、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。

## 68、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;
```

---

```
}
```

1. 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，结果为，提供 **reset**

## 7. 实际项目开发

### 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。

### 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。

### 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。

## 7. 数据库部分

### 1、用两种方式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
```

## 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 deptd ) b
     where a.deptid=b.deptid and a.salary>b.avgsal;
```

## 1、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('传智播客',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  | 传智播客      | 1   |
```

```
+----+-----+-----+
```

触发器的实验步骤：

```
create table board1(id int primary key auto_increment,name varchar(50),articleCount 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 row  
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。

## 1、数据库三范式是什么？

第一范式（1NF）：字段具有原子性,不可再分。所有关系型数据库系统都满足第一范式）

数据库表中的字段都是单一属性的，不可再分。例如，姓名字段，其中的姓和名必须作为一个整体，无法区分哪部分是姓，哪部分是名，如果要区分出姓和名，必须设计成两个独立的字段。

第二范式（2NF）：

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。要求数据库表中的每个实例或行必须可以被惟一地区分。通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键。

第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，第二范式就是非主属性非部分依赖于主关键字。

第三范式的要求如下：

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。

所以第三范式具有如下特征：

- 1，每一列只有一个值

2, 每一行都能区分。

3, 每一个表都不包含其他表已经包含的非主关键字信息。

---

例如, 帖子表中只能出现发帖人的 id, 而不能出现发帖人的 id, 还同时出现发帖人姓名, 否则, 只要出现同一发帖人 id 的所有记录, 它们中的姓名部分都必须严格保持一致, 这就是数据冗余。

## 1、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

“insert into user values(1,1,1)”->二进制

“insert into user values(2,2,2)”->二进制

“insert 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: employee->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 PreparedStatement

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')
```

## 2、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，

### 3.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.prepareStatement(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));
}
}
catch(){ }
finally
{
    if(rs!=null)try{rs.close();}catch(Exception e){ }
    if(stmt.....
    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.prepareStatement(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));
}
}
catch({})
finally
{
    if(rs!=null)try{rs.close();}catch(Exception e){}
    if(stm.....
    if(cn.....
}
```

### 3. 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 distinct name t1 from score where 80< all (select score from score where name=t1);

## 4.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, 对应四个球队，现在四个球队进行比赛，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

## 4.每个月份的发生额都比 101 科目多的科目

请用 SQL 语句实现：从 TestDB 数据表中查询出所有月份的发生额都比 101 科目相应月份的发生额高的科目。  
请注意：TestDB 中有很多科目，都有 1—12 月份的发生额。

AccID: 科目代码，Occmonth: 发生额月份，DebitOccur: 发生额。

数据库名: JcyAudit, 数据集: Select \* from TestDB

准备数据的 sql 代码:

```
drop table if exists TestDB;
```

```
create table TestDB(id int primary key auto_increment,AccID varchar(20), Occmonth date, DebitOccur bigint);
```

```
insert into TestDB values
```

```
(null,'101','1988-1-1',100),
```

```
(null,'101','1988-2-1',110),
```

```
(null,'101','1988-3-1',120),
```

```
(null,'101','1988-4-1',100),
```

```
(null,'101','1988-5-1',100),
```

```
(null,'101','1988-6-1',100),
```

```
(null,'101','1988-7-1',100),
```

```
(null,'101','1988-8-1',100);
```

```
--复制上面的数据，故意把第一个月份的发生额数字改小一点
```

```
insert into TestDB values
```

```
(null,'102','1988-1-1',90),
```

```
(null,'102','1988-2-1',110),
```

```
(null,'102','1988-3-1',120),
```

```
(null,'102','1988-4-1',100),
```

```
(null,'102','1988-5-1',100),
```

```
(null,'102','1988-6-1',100),
```

```
(null,'102','1988-7-1',100),
```

```
(null,'102','1988-8-1',100);
```

```
--复制最上面的数据，故意把所有发生额数字改大一点
```

```
insert into TestDB values
```

```

(null,'103','1988-1-1',150),
(null,'103','1988-2-1',160),
(null,'103','1988-3-1',180),
(null,'103','1988-4-1',120),
(null,'103','1988-5-1',120),
(null,'103','1988-6-1',120),
(null,'103','1988-7-1',120),
(null,'103','1988-8-1',120);
--复制最上面的数据，故意把所有发生额数字改大一点
insert into TestDB values
(null,'104','1988-1-1',130),
(null,'104','1988-2-1',130),
(null,'104','1988-3-1',140),
(null,'104','1988-4-1',150),
(null,'104','1988-5-1',160),
(null,'104','1988-6-1',170),
(null,'104','1988-7-1',180),
(null,'104','1988-8-1',140);
--复制最上面的数据，故意把第二个月份的发生额数字改小一点
insert into TestDB values
(null,'105','1988-1-1',100),
(null,'105','1988-2-1',80),
(null,'105','1988-3-1',120),
(null,'105','1988-4-1',100),
(null,'105','1988-5-1',100),
(null,'105','1988-6-1',100),
(null,'105','1988-7-1',100),
(null,'105','1988-8-1',100);
答案：
select distinct AccID from TestDB
where AccID not in
    (select TestDB.AccID from TestDB,
        (select * from TestDB where AccID='101') as db101
    where TestDB.Occmonth=db101.Occmonth and TestDB.DebitOccur<=db101.DebitOccur
    );

```

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

## 5.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。

### 3.删除除了 id 号不同,其他都相同的学生冗余信息

2.学生表 如下:

id 号 学号 姓名 课程编号 课程名称 分数

1 2005001 张三 0001 数学 69

2 2005002 李四 0001 数学 89

3 2005001 张三 0001 数学 69

A: delete from tablename where id 号 not in(select min(id 号) from tablename group by 学号,姓名,课程编号,课程名称,分数)

实验:

```
create table student2(id int auto_increment primary key,code varchar(20),name varchar(20));
insert into student2 values(null,'2005001','张三'),(null,'2005002','李四'),(null,'2005001','张三');
```

//如下语句，mysql 报告错误，可能删除依赖后面统计语句，而删除又导致统计语句结果不一致。

```
delete from student2 where id not in(select min(id) from student2 group by name);
```

//但是，如下语句没有问题:

```
select * from student2 where id not in(select min(id) from student2 group by name);
```

//于是，我想先把分组的结果做成虚表，然后从虚表中选出结果，最后再将结果作为删除的条件数据。

```
delete from student2 where id not in(select mid from (select min(id) mid
from student2 group by name) as t);
```

或者:

```
delete from student2 where id not in(select min(id) from (select * from s
tudent2) as t group by t.name);
```

### 4.航空网的几个航班查询题:

表结构如下:

flight{flightID,StartCityID ,endCityID,StartTime}

city{cityID, CityName}

实验环境:

```
create table city(cityID int auto_increment primary key,cityName varchar(20));
```

```
create table flight (flightID int auto_increment primary key,
```

```
StartCityID int references city(cityID),
```

```
endCityID int references city(cityID),
```

```
StartTime timestamp);
```

//航班本来应该没有日期部分才好，但是下面的题目当中涉及到了日期

```
insert into city values(null,'北京'),(null,'上海'),(null,'广州');
```

```
insert into flight values
```

```
(null,1,2,'9:37:23'),(null,1,3,'9:37:23'),(null,1,2,'10:37:23'),(null,2,3,'10:37:23');
```

---

## 1、查询起飞城市是北京的所有航班，按到达城市的名字排序

参与运算的列是我起码能够显示出来的那些列，但最终我不一定把它们显示出来。各个表组合出来的中间结果字段中必须包含所有运算的字段。

```
select * from flight f,city c
where f.endcityid = c.cityid and startcityid =
(select c1.cityid from city c1 where c1.cityname = "北京")
order by c.cityname asc;
```

```
mysql> select flight.flightid,'北京' startcity, e.cityname from flight,city e wh
ere flight.endcityid=e.cityid and flight.startcityid=(select cityid from city wh
ere cityname='北京');
```

```
mysql> select flight.flightid,s.cityname,e.cityname from flight,city s,city e wh
ere flight.startcityid=s.cityid and s.cityname='北京' and flight.endCityId=e.cit
yID order by e.cityName desc;
```

## 2、查询北京到上海的所有航班纪录（起飞城市，到达城市，起飞时间，航班号）

```
select c1.CityName,c2.CityName,f.StartTime,f.flightID
from city c1,city c2,flight f
where f.StartCityID=c1.cityID
and f.endCityID=c2.cityID
and c1.cityName='北京'
and c2.cityName='上海'
```

## 3、查询具体某一天（2005-5-8）的北京到上海的航班次数

```
select count(*) from
(select c1.CityName,c2.CityName,f.StartTime,f.flightID
from city c1,city c2,flight f
where f.StartCityID=c1.cityID
and f.endCityID=c2.cityID
and c1.cityName='北京'
and c2.cityName='上海'
and 查帮助获得的某个日期处理函数(startTime) like '2005-5-8%'
```

mysql 中提取日期部分进行比较的示例代码如下：

```
select * from flight where date_format(starttime,'%Y-%m-%d')='1998-01-02'
```

## 5.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);
```

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.salary>m.salary;
```

## 6、求出小于 45 岁的各个老师所带的大于 12 岁的学生人数

数据库中有 3 个表 teacher 表, student 表, tea\_stu 关系表。

teacher 表 teaID name age

student 表 stuID name age

teacher\_student 表 teaID stuID

要求用一条 sql 查询出这样的结果

1.显示的字段要有老师 name, age 每个老师所带的学生人数

2 只列出老师 age 为 40 以下, 学生 age 为 12 以上的记录

预备知识:

1.sql 语句是对每一条记录依次处理,条件为真则执行动作 (select,insert,delete,update)

2.只要是迪卡尔积,就会产生“垃圾”信息,所以,只要迪卡尔积了,我们首先就要想到清除“垃圾”信息

实验准备:

```
drop table if exists tea_stu;
drop table if exists teacher;
drop table if exists student;
create table teacher(teaID int primary key,name varchar(50),age int);
create table student(stuID int primary key,name varchar(50),age int);
create table tea_stu(teaID int references teacher(teaID),stuID int references student(stuID));
insert into teacher values(1,'zxx',45), (2,'lhm',25), (3,'wzg',26), (4,'tg',27);
insert into student values(1,'wy',11), (2,'dh',25), (3,'ysq',26), (4,'mxc',27);
insert into tea_stu values(1,1), (1,2), (1,3);
insert into tea_stu values(2,2), (2,3), (2,4);
insert into tea_stu values(3,3), (3,4), (3,1);
insert into tea_stu values(4,4), (4,1), (4,2), (4,3);
```

结果: 2→3,3→2,4→3

解题思路:(真实面试答题时,也要写出每个分析步骤,如果纸张不够,就找别人要)

1 要会统计分组信息,统计信息放在中间表中:

```
select teaid,count(*) from tea_stu group by teaid;
```

2 接着其实应该是筛除掉小于 12 岁的学生,然后再进行统计,中间表必须与 student 关联才能得到 12 岁以下学生和把该学生记录从中间表中剔除,代码是:

```
select tea_stu.teaid,count(*) total from student,tea_stu
where student.stuid=tea_stu.stuid and student.age>12 group by tea_stu.teaid
```

---

3.接着把上面的结果做成虚表与 teacher 进行关联，并筛选大于 45 的老师

```
select teacher.teaid,teacher.name,total from teacher ,(select tea_stu.tea
id,count(*) total from student,tea_stu where student.stuid=tea_stu.stuid and stu
dent.age>12 group by tea_stu.teaid) as tea_stu2 where teacher.teaid=tea_stu2.tea
id and teacher.age<45;
```

## 7.求出发帖最多的人:

```
select authorid,count(*) total from articles
group by authorid
having total=
(select max(total2) from (select count(*) total2 from articles group by authorid) as t);
```

```
select t.authorid,max(t.total) from
(select authorid,count(*) total from articles ) as t
这条语句不行，因为 max 只有一列，不能与其他列混淆。
```

```
select authorid,count(*) total from articles
group by authorid having total=max(total)也不行。
```

## 10、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 column 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 insert 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));
```

**10、一个用户具有多个角色，请查询出该表中具有该用户的所有角色的其他用户。**

```
select count(*) as num,tb.id  
from  
    tb,  
    (select role from tb where id=xxx) as t1  
where  
    tb.role = t1.role and tb.id != t1.id  
group by tb.id  
having  
    num = select count(role) from tb where id=xxx;
```

## 8. 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 2000)  
    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;
```

## 1、注册 Jdbc 驱动程序的三种方式

---

### 1、用 JDBC 如何调用存储过程

代码如下：

```
package com.huawei.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();
}
}
}

```

## 1、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。

## 1. 写一个用 jdbc 连接并访问 oracle 数据的程序代码

## 2、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。

## 3、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);
```

#### 4、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){ }  
}
```

## 5、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。

## 36、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标记为忙。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标记为空闲，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。

## 4、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 object，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 toptlink,obj 等。

## 8. XML 部分

### 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

讲解这些区别是不需要特别去比较，就像说传智播客与其他培训机构的区别时，我们只需说清楚传智播客有什么特点和优点就行了，这就已经间接回答了彼此的区别。

## 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。

## 3、用 jdom 解析 xml 文件时如何解决中文问题?如何解析?

答:看如下代码,用编码方式加以解决

```
package test;
import java.io.*;
public class DOMTest
{
    private String inFile = "c:\\people.xml"
    private String outFile = "c:\\people.xml"
    public static void main(String args[])
    {
        new DOMTest();
    }
    public DOMTest()
    {
        try
        {
            javax.xml.parsers.DocumentBuilder builder =
            javax.xml.parsers.DocumentBuilderFactory.newInstance().newDocumentBuilder();
            org.w3c.dom.Document doc = builder.newDocument();
            org.w3c.dom.Element root = doc.createElement("老师");
            org.w3c.dom.Element wang = doc.createElement("王");
            org.w3c.dom.Element liu = doc.createElement("刘");
            wang.appendChild(doc.createTextNode("我是王老师"));
            root.appendChild(wang);
            doc.appendChild(root);
            javax.xml.transform.Transformer transformer =
            javax.xml.transform.TransformerFactory.newInstance().newTransformer();
            transformer.setOutputProperty(javax.xml.transform.OutputKeys.ENCODING, "gb2312");
            transformer.setOutputProperty(javax.xml.transform.OutputKeys.INDENT, "yes");
            transformer.transform(new javax.xml.transform.dom.DOMSource(doc),
            new
            javax.xml.transform.stream.StreamResult(outFile));
        }
        catch (Exception e)
```

```
{  
System.out.println (e.getMessage());  
}  
}  
}
```

---

## 4、编程用 JAVA 解析 XML 的方式.

答:用 SAX 方式解析 XML, XML 文件如下:

```
<?xml version=1.0 encoding=gb2312?>  
<person>  
<name>王小明</name>  
<college>信息学院</college>  
<telephone>6258113</telephone>  
<notes>男,1955 年生,博士, 95 年调入海南大学</notes>  
</person>
```

事件回调类 SAXHandler.java

```
import java.io.*;  
import java.util.Hashtable;  
import org.xml.sax.*;  
public class SAXHandler extends HandlerBase  
{  
private Hashtable table = new Hashtable();  
private String currentElement = null;  
private String currentValue = null;  
public void setTable(Hashtable table)  
{  
this.table = table;  
}  
public Hashtable getTable()  
{  
return table;  
}  
public void startElement(String tag, AttributeList attrs)  
throws SAXException  
{  
currentElement = tag;  
}  
public void characters(char[] ch, int start, int length)  
throws SAXException  
{  
currentValue = new String(ch, start, length);  
}  
public void endElement(String name) throws SAXException  
{  
if (currentElement.equals(name))  
table.put(currentElement, currentValue);  
}  
}
```

---

```

}
JSP 内容显示源码,SaxXml.jsp:
<HTML>
<HEAD>
<TITLE>剖析 XML 文件 people.xml</TITLE>
</HEAD>
<BODY>
<% @ page errorPage=ErrPage.jsp
contentType=text/html;charset=GB2312 %>
<% @ page import=java.io.* %>
<% @ page import=java.util.Hashtable %>
<% @ page import=org.w3c.dom.* %>
<% @ page import=org.xml.sax.* %>
<% @ page import=javax.xml.parsers.SAXParserFactory %>
<% @ page import=javax.xml.parsers.SAXParser %>
<% @ page import=SAXHandler %>
<%
File file = new File(c:\people.xml);
FileReader reader = new FileReader(file);
Parser parser;
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
SAXHandler handler = new SAXHandler();
sp.parse(new InputSource(reader), handler);
Hashtable hashTable = handler.getTable();
out.println(<TABLE BORDER=2><CAPTION>教师信息表</CAPTION>);
out.println(<TR><TD>姓名</TD> + <TD> +
(String)hashTable.get(new String(name)) + </TD></TR>);
out.println(<TR><TD>学院</TD> + <TD> +
(String)hashTable.get(new String(college))+</TD></TR>);
out.println(<TR><TD>电话</TD> + <TD> +
(String)hashTable.get(new String(telephone)) + </TD></TR>);
out.println(<TR><TD>备注</TD> + <TD> +
(String)hashTable.get(new String(notes)) + </TD></TR>);
out.println(</TABLE>);
%>
</BODY>
</HTML>

```

## 70、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)

## 9. 流行的框架与新技术

### 1、谈谈你对 Struts 的理解。

答:

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。

### 2、谈谈你对 Hibernate 的理解。

答:

1. 面向对象设计的软件内部运行过程可以理解成就是在不断创建各种新对象、建立对象之间的关系, 调用对象的方法来改变各个对象的状态和对象消亡的过程, 不管程序运行的过程和操作怎么样, 本质上都是要得到一个结果, 程序上一个时刻和下一个时刻的运行结果的差异就表现在内存中的对象状态发生了变化。

2.为了在关机和内存空间不够的状况下, 保持程序的运行状态, 需要将内存中的对象状态保存到持久化设备和从持久化设备中恢复出对象的状态, 通常都是保存到关系数据库来保存大量对象信息。从 Java 程序的运行功能上来讲, 保存对象状态的功能相比系统运行的其他功能来说, 应该是一个很不起眼的附属功能, java 采用 jdbc 来实现这个功能, 这个不起眼的功能却要编写大量的代码, 而做的事情仅仅是保存对象和恢复对象, 并且那些大量的 jdbc 代码并没有什么技术含量, 基本上是采用一套例行公事的标准代码模板来编写, 是一种苦活和重复性的工作。

3.通过数据库保存 java 程序运行时产生的对象和恢复对象, 其实就是实现了 java 对象与关系数据库记录的映射关系, 称为 ORM (即 Object Relation Mapping), 人们可以通过封装 JDBC 代码来实现了这种功能, 封装出来的产品称之为 ORM 框架, Hibernate 就是其中的一种流行 ORM 框架。使用 Hibernate 框架, 不用写 JDBC 代码, 仅仅是调用一个 save 方法, 就可以将对象保存到关系数据库中, 仅仅是调用一个 get 方法, 就可以从数据库中加载出一个对象。

- 4.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。
- 5.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。
- 6.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。

### 3、AOP 的作用。

### 3、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（Dependency 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 CGLIB。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.springframework.aop.ProxyBeanFactory">
    <property name="target" ref=""></property>
```

<property name="advisor" ref=""></property>

</bean>

## 11、谈谈 Struts 中的 Action servlet。

## 12、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

缺点

- 一、 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
- 二、 Struts 的 Action 必需是 thread-safe 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。
- 三、 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。
- 四、 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。
- 五、 对 Servlet 的依赖性过强. Struts 处理 Action 时必须需要依赖 ServletRequest 和 ServletResponse，所有它摆脱不了 Servlet 容器。
- 六、 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。
- 七、 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。
- 八、 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。
- 九、 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

## 119、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。

## 110、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,但是应用群众并不多,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。

~~110、hibernate 中的 update()和 saveOrUpdate()的区别，session 的 load()和 get()的区别。~~

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

## 7、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。

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

## 9、hibernate 的 inverse 属性的作用？

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 field1,type2 field2)，然后在 hql 里面就可以直接生成这个 bean 了。

## 10、在 DAO 中如何体现 DAO 设计模式？

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 field1,type2 field2)，然后在 hql 里面就可以直接生成这个 bean 了。

## 10、spring+Hibernate 中委托方案怎么配置？

---

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。

## 10、spring+Hibernate 中委托方案怎么配置？

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. 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.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題。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 application 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。

**111、Spring 的依赖注入是什么意思？给一个 Bean 的 message 属性，字符串类型，注入值为 "Hello" 的 XML 配置文件该怎么写？**

**120、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 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。

**什么是 spring 的 IOC AOP**

**STRUTS 的工作流程！**

**spring 与 EJB 的区别！！**

**10. 软件工程与设计模式**

**111、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。

**112. 软件开发的**

**92、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 Responsibility（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。

## 113、开发中都用到那些设计模式?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。

## 11. j2ee 部分

### 117、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。

## 2、应用服务器与 WEB SERVER 的区别？

应用服务器：Weblogic、Tomcat、Jboss

WEB SERVER：IIS、 Apache

## 32、应用服务器有那些？

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

## 3、J2EE 是什么？

答:Je22 是 Sun 公司提出的多层(multi-tiered),分布式(distributed),基于组件(component-base)的企业级应用模型(enterprise 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(client 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。

## 67、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。

## 95、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,WEBSPPHERE 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。

## 14. Linux

### 118、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。

## 65、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

## 69、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

- 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, 城市有外键引用省份, 这个规律就是下层的要有一外键去引用上层。

## Android 部分

# 面试题目及其答案

### 1、 Android dvm 的进程和 Linux 的进程, 应用程序的进程是否为同一个概念

DVM 指 dalvik 的虚拟机。每一个 Android 应用程序都在它自己的进程中运行, 都拥有一个独立的 Dalvik 虚拟机实例。而每一个 DVM 都是在 Linux 中的一个进程, 所以说可以认为是同一个概念。

### 2、 sim 卡的 EF 文件有何作用

sim 卡的文件系统有自己规范, 主要是为了和手机通讯, sim 本身可以有操作系统, EF 就是作存储并和手机通讯用的

### 3、嵌入式操作系统内存管理有哪几种, 各有何特性

页式, 段式, 段页, 用到了 MMU, 虚拟空间等技术

### 4、 什么是嵌入式实时操作系统, Android 操作系统属于实时操作系统吗?

嵌入式实时操作系统是指当外界事件或数据产生时, 能够接受并以足够快的速度予以处理, 其处理的结果又能在规定的时间之内来控制生产过程或对处理系统作出快速响应, 并控制所有实时任务协调一致运行的嵌入式操作系统。主要用于工业控制、 军事设备、 航空航天等领域对系统的响应时间有苛刻的要求, 这就需要使用实时系统。又可分为软实时和硬实时两种, 而 android 是基于 linux 内核的, 因此属于软实时。

### 5、一条最长的短信息约占多少 byte?

中文 70(包括标点), 英文 160, 160 个字节。

### 6、 android 中的动画有哪几类, 它们的特点和区别是什么?

两种, 一种是 Tween 动画、还有一种是 Frame 动画。Tween 动画, 这种实现方式可以使视图组件移动、放大、缩小以及产生透明度的变化;另一种 Frame 动画, 传统的动画方法, 通过顺序的播放排列好的图片来实现, 类似电影。

## 7、handler 机制的原理

---

android 提供了 Handler 和 Looper 来满足线程间的通信。Handler 先进先出原则。Looper 类用来管理特定线程内对象之间的消息交换(Message Exchange)。

1)Looper: 一个线程可以产生一个 Looper 对象, 由它来管理此线程里的 Message Queue(消息队列)。

2)Handler: 你可以构造 Handler 对象来与 Looper 沟通, 以便 push 新消息到 Message Queue 里;或者接收 Looper 从 Message Queue 取出)所送来的消息。

3) Message Queue(消息队列):用来存放线程放入的消息。

4)线程: UI thread 通常就是 main thread, 而 Android 启动程序时会替它建立一个 Message Queue。

## 8、说说 mvc 模式的原理, 它在 android 中的运用

MVC(Model\_view\_contraller)“模型\_视图\_控制器”。MVC 应用程序总是由这三个部分组成。Event(事件)导致 Controller 改变 Model 或 View, 或者同时改变两者。只要 Controller 改变了 Models 的数据或者属性, 所有依赖的 View 都会自动更新。类似的, 只要 Contro

## 9、Activity 的生命周期

和其他手机 平台 的应用 程序 一样, Android 的应用程序 的生命周期是被统一掌控 的, 也 就是说我们写的应用程序命运掌握在别人(系统)的手里, 我们不能改变它, 只能学习 并 适应它。

简单地说一下为什么是这样: 我们手机在运行 一个应用程序的时候, 有可能打进来电话 发进来短信 , 或者没有电了, 这时候程序都会被中断, 优先去服务电话的基本功能 , 另 外系统也不允许你占用太多资源 , 至少要保证电话功能吧,所以资源不足的时候也就有可 能被干掉。

言归正传, Activity 的基本生命周期如下代码 所示:

你自己写的 Activity 会按需要 重载这些方法, onCreate 是免不了的, 在一个 Activity 正常启动的过程中, 他们被调用的顺序是 onCreate -> onStart -> onResume, 在 Activity 被干掉的时候顺序是 onPause -> onStop -> onDestroy , 这样就是一个完整的生命周期, 但是有人问了 , 程序正运行着呢来电话了, 这个程序咋办? 中止了呗, 如果中止的时候新出的一个 Activity 是全屏的那么: onPause->onStop , 恢复的时候 onStart->onResume , 如果打断 这个应用程序的是一个 Theme 为 Translucent 或者 Dialog 的 Activity 那么只是 onPause ,恢复 的时候 onResume 。

详细介绍一下这几个方法中系统在做什么以及我们应该做什么:

onCreate: 在这里创建界面 , 做一些数据 的初始化工作

onStart: 到这一步变成用户可见不可交互 的

onResume: 变成和用户可交互 的, (在 activity 栈系统通过栈的方式管理这些个

Activity 的最上面, 运行完弹出栈, 则回到上一个 Activity)

onPause: 到这一步是可见但不可交互 的, 系统会停止动画 等消耗 CPU 的事情 从上文的描述已经知道, 应该在这里保存你的一些数据,因为这个时候 你的程序的优先级降低, 有可能被系统收回。在这里保

存的数据，应该在 `onResume` 里读出来，注意：这个方法里做的事情时间要短，因为下一个 `activity` 不会等到这个方法完成才启动

---

`onstop`: 变得不可见，被下一个 `activity` 覆盖了

`onDestroy`: 这是 `activity` 被干掉前最后一个被调用方法了，可能是外面类调用 `finish` 方法或者是系统为了节省空间将它暂时性的干掉，可以用 `isFinishing()` 来判断它，如果你有一个 `Progress Dialog` 在线程中转动，请在 `onDestroy` 里把他 `cancel` 掉，不然等线程结束的时候，调用 `Dialog` 的 `cancel` 方法会抛异常的。

`onPause`, `onstop`, `onDestroy`, 三种状态下 `activity` 都有可能被系统干掉

为了保证程序的正确性，你要在 `onPause()` 里写上持久层操作的代码，将用户编辑的内容都保存到存储介质上(一般都是数据库)。实际工作中因为生命周期的变化而带来的问题也很多，比如你的应用程序起了新的线程在跑，这时候中断了，你还要去维护那个线程，是暂停还是杀掉还是数据回滚，是吧？因为 `Activity` 可能被杀掉，所以线程中使用的变量和一些界面元素就千万要注意了，一般我都是采用 `Android` 的消息机制 [`Handler`, `Message`] 来处理多线程和界面交互的问题。这个我后面会讲一些，最近因为这些东西头已经很大了，等我理清思绪再跟大家分享。 `ller` 改变了 `View`, `View` 会从潜在的 `Model` 中获取数据来刷新自己。

## 10、让 `Activity` 变成一个窗口： `Activity` 属性设定

讲点轻松的吧,可能有人希望做出来的应用程序是一个漂浮在手机主界面的东西，那么很简单你只需要设置一下 `Activity` 的主题就可以了在 `AndroidManifest.xml` 中定义 `Activity` 的地方一句话：

Xml 代码

1. `android:theme="@android:style/Theme.Dialog"`

2.

3. `android:theme="@android:style/Theme.Dialog"`

`android:theme="@android:style/Theme.Dialog"`

`android:theme="@android:style/Theme.Dialog"`

这就使你的应用程序变成对话框的形式弹出来了，或者

Xml 代码

1. `android:theme="@android:style/Theme.Translucent"`

2.

3. `android:theme="@android:style/Theme.Translucent"`

`android:theme="@android:style/Theme.Translucent"`

`android:theme="@android:style/Theme.Translucent"`

就变成半透明的，[友情提示--]类似的这种 `activity` 的属性可以在 `android.R.styleable` 类的 `AndroidManifestActivity` 方法中看到，`AndroidManifest.xml` 中所有元素的属性的介绍都可以参考这个类 `android.R.styleable`

上面说的是属性名称，具体有什么值是在 `android.R.style` 中 可以看到，比如这个 `"@android:style/Theme.Dialog"` 就对应于 `android.R.style.Theme_Dialog` ,('换成'.' < --注意：这个是文章内容不是笑脸)就可以用在描述文件 中了,找找类定义和描述文件中的对应关系就都明白了。

## 11、 你后台的 **Activity** 被系统回收怎么办: `onSaveInstanceState`

当你的程序中某一个 **Activity A** 在运行时中，主动或被动地运行另一个新的 **Activity B** 这个时候 **A** 会执行

Java 代码

```
1. public
2. void onSaveInstanceState(Bundle outState) {
3. super.onSaveInstanceState(outState);
4. outState.putLong("id", 1234567890);
5. }
```

public

```
void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putLong("id", 1234567890);
}
```

**B** 完成以后又会来找 **A**, 这个时候就有两种情况，一种是 **A** 被回收，一种是没有被回收，被回收的 **A** 就要重新调用 `onCreate()`方法，不同于直接启动的是这回 `onCreate()`里是带上参数 `savedInstanceState`，没被收回的就还是 `onResume` 就好了。

`savedInstanceState` 是一个 `Bundle` 对象，你基本上可以把他理解为系统帮你维护的一个 `Map` 对象。在 `onCreate()`里你可能会用到它，如果正常启动 `onCreate` 就不会有它，所以用的时候要判断一下是否为空。

Java 代码

```
1. if(savedInstanceState != null){
2. long id = savedInstanceState.getLong("id");
3. }
```

---

```
if(savedInstanceState != null){

    long id = savedInstanceState.getLong("id");

}
```

就像官方的 **Notepad** 教程 里的情况，你正在编辑某一个 **note**，突然被中断，那么就把这个 **note** 的 **id** 记住，再起来的时候就可以根据这个 **id** 去把那个 **note** 取出来，程序就完整一些。这也是看你的应用需不需要保存什么，比如你的界面就是读取一个列表，那就不需要特殊记住什么，哦， 没准你需要记住滚动条的位置...

## 12、 调用与被调用：我们的通信使者 **Intent**

要说 **Intent** 了，**Intent** 就是这个这个意图 ，应用程序间 **Intent** 进行交流，打个电话啦，来个 电话啦都会发 **Intent**, 这个是 **Android** 架构的松耦合的精髓部分，大大提高了组件的复用性，比如你要在你的应用程序中点击按钮，给某人打电话，很简单啊，看下代码先：

Java 代码

```
1. Intent intent = new Intent();

2. intent.setAction(Intent.ACTION_CALL);

3. intent.setData(Uri.parse("tel:" + number));

4. startActivity(intent);

Intent intent = new Intent();

intent.setAction(Intent.ACTION_CALL);

intent.setData(Uri.parse("tel:" + number));

startActivity(intent);
```

扔出这样一个意图，系统看到了你的意图就唤醒了电话拨号程序，打出来电话。什么读联系人，发短信啊，邮件啊，统统只需要扔出 **intent** 就好了，这个部分设计 地确实很好啊。

那 **Intent** 通过什么来告诉系统需要谁来接受他呢？

通常使用 **Intent** 有两种方法，第一种是直接说明需要哪一个类来接收代码如下：

Java 代码

```
1. Intent intent = new Intent(this, MyActivity.class);

2. intent.putExtra().putString("id", "1");

3. tartActivity(intent);
```



---

```
Intent intent = new Intent(this, MyActivity.class);
```

```
intent.getExtras().putString("id", "1");
```

```
startActivity(intent);
```

第一种方式很明显，直接指定了 **MyActivity** 为接受者,并且传了一些数据给 **MyActivity**，在 **MyActivity** 里可以用 **getIntent()**来的到这个 **intent** 和数据。

第二种就需要先看一下 **AndroidManifest** 中的 **intentfilter** 的配置了

Xml 代码

```
< intent-filter>
```

```
< action
```

```
android:name="android.intent.action.VIEW"
```

```
/>
```

```
< action
```

```
android:value="android.intent.action.EDIT"
```

```
/>
```

```
< action
```

```
android:value="android.intent.action.PICK"
```

```
/>
```

```
< category
```

```
android:name="android.intent.category.DEFAULT"
```

```
/>
```

```
< data
```

```
android:mimeType="vnd.android.cursor.dir/vnd.google.note"
```

```
/>
```

```
< /intent-filter>
```

这里面配置用到了 **action**, **data**, **category** 这些东西，那么聪明的你一定想到 **intent** 里也会有这些东西，然

后一匹配不就找到接收者了吗?

---

**action** 其实就是一个意图的字符串名称。

上面这段 **intent-filter** 的配置文件说明了这个 **Activity** 可以接受不同的 **Action**, 当然相应的程序逻辑也不一样咯, 提一下那个  **mimeType**, 他是在 **ContentProvider** 里定义的, 你要是自己实现一个 **ContentProvider** 就知道了, 必须指定 **mimeType** 才能让数据被别人使用。

不知道原理说明白没, 总结一句, 就是你调用别的界面不是直接 **new** 那个界面, 而是通过扔出一个 **intent**, 让系统帮你去调用那个界面, 这样就多么松耦合啊, 而且符合了生命周期被系统管理的原则。

想知道 **category** 都有啥, **Android** 为你预先定制好的 **action** 都有啥等等, 请亲自访问官方链接 **Intent**

**ps:** 想知道怎么调用系统应用程序的同学, 可以仔细看一下你的 **logcat**, 每次运行一个程序的时候是不是有一些信息比如:

```
Starting activity: Intent
{ action=android.intent.action.MAIN categories={android.intent.category.LAUNCHER}
flags=0x10200000 comp={com.android.camera/com.android.camera.GalleryPicker} }
```

再对照一下 **Intent** 的一些 **set** 方法, 就知道怎么调用咯, 希望你喜欢: )

开发 75 条 (写的不错) 选择自 **churujianghu** 的 Blog

开发 75 条:

1. 你们的项目组使用源代码管理工具了么?

应该用。VSS、CVS、PVCS、ClearCase、CCC/Harvest、FireFly 都可以。我的选择是 VSS。

2. 你们的项目组使用缺陷管理系统了么?

应该用。ClearQuest 太复杂, 我的推荐是 BugZilla。

3. 你们的测试组还在用 Word 写测试用例么?

不要用 Word 写测试用例 (Test Case)。应该用一个专门的系统, 可以是 Test Manager, 也可以是自己开发一个 ASP.NET 的小网站。主要目的是 Track 和 Browse。

4. 你们的项目组有没有建立一个门户网站?

要有一个门户网站, 用来放 Contact Info、Baselined Schedule、News 等等。推荐 Sharepoint Portal Server 2003 来实现, 15 分钟就搞定。买不起 SPS 2003 可以用 WSS (Windows Sharepoint Service)。

5. 你们的项目组用了你能买到最好的工具么?

应该用尽量好的工具来工作。比如, 应该用 VS.NET 而不是 Notepad 来写 C#。用 Notepad 写程序多半只是一种炫耀。但也要考虑到经费, 所以说是“你能买到最好的”。

6. 你们的程序员工作在安静的环境里么?

需要安静环境。这点极端重要, 而且要保证每个人的空间大于一定面积。

7. 你们的员工每个人都有一部电话么? 需要每人一部电话。而且电话最好是带留言功能的。当然,

上这么一套带留言电话系统开销不小。不过至少每人一部电话要有, 千万别搞得经常有人站起来喊: “某某某电话”。《人件》里面就强烈谴责这种做法。

8. 你们每个人都知道出了问题应该找谁么?

应该知道。任何一个 **Feature** 至少都应该有一个 **Owner**，当然，**Owner** 可以继续 **Dispatch** 给其他人。

---

9. 你遇到过有人说“我以为...”么？

要消灭“我以为”。**Never assume anything**。

10. 你们的项目组中所有的人都坐在一起么？

需要。我反对 **Virtual Team**，也反对 **Dev** 在美国、**Test** 在中国这种开发方式。能坐在一起就最好坐在一起，好处多得不得了。

11. 你们的进度表是否反映最新开发进展情况？

应该反映。但是，应该用 **Baseline** 的方法来管理进度表：维护一份稳定的 **Schedule**，再维护一份最新更改。**Baseline** 的方法也应该用于其它的 **Spec**。**Baseline** 是变更管理里面的一个重要手段。

12. 你们的工作量是先由每个人自己估算的么？

应该让每个人自己估算。要从下而上估算工作量，而不是从上往下分派。除非有其他原因，比如政治任务工期固定等。

13. 你们的开发人员从项目一开始就加班么？

不要这样。不要一开始就搞疲劳战。从项目一开始就加班，只能说明项目进度不合理。当然，一些对日软件外包必须天天加班，那属于剥削的范畴。

14. 你们的项目计划中 **Buffer Time** 是加在每个小任务后面的么？

不要。**Buffer Time** 加在每个小任务后面，很容易轻易的就被消耗掉。**Buffer Time** 要整段的加在一个 **Milestone** 或者 **checkpoint** 前面。

15. 值得再多花一些时间，从 **95%**做到 **100%**好值得，非常值得。

尤其当项目后期人困马乏的时候，要坚持。这会给产品带来质的区别。

16. 登记新缺陷时，是否写清了重现步骤？

要。这属于 **Dev** 和 **Test** 之间的沟通手段。面对面沟通需要，详细填写 **Repro Steps** 也需要。

17. 写新代码前会把已知缺陷解决么？要。每个人的缺陷不能超过 **10** 个或 **15** 个，否则必须先解决老的 **bug** 才能继续写新代码。

18. 你们对缺陷的轻重缓急有事先的约定么？

必须有定义。**Severity** 要分 **1**、**2**、**3**，约定好：蓝屏和 **Data Lost** 算 **Sev 1**，**Function Error** 算 **Sev 2**，界面上的算 **Sev 3**。但这种约定可以根据产品质量现状适当进行调整。

19. 你们对意见不一的缺陷有三国会议么？必须要有。要有一个明确的决策过程。这类似于 **CCB** (**Change Control Board**)的概念。

20. 所有的缺陷都是由登记的人最后关闭的么？

**Bug** 应该由 **Opener** 关闭。**Dev** 不能私自关闭 **Bug**。

21. 你们的程序员厌恶修改老的代码么？

厌恶是正常的。解决方法是组织 **Code Review**，单独留出时间来。**XP** 也是一个方法。

22. 你们项目组有 **Team Morale Activity** 么？

每个月都要搞一次，吃饭、唱歌、**Outing**、打球、开卡丁车等等，一定要有。不要剩这些钱。

23. 你们项目组有自己的 Logo 么？

要有自己的 Logo。至少应该有自己的 Codename。

---

24. 你们的员工有印有公司 Logo 的 T-Shirt 么？

要有。能增强归属感。当然，T-Shirt 要做的好看一些，最好用 80 支的棉来做。别没穿几次就破破烂烂的。

25. 总经理至少每月参加次项目组会议要的。

要让 team member 觉得高层关注这个项目。

26. 你们是给每个 Dev 开一个分支么？

反对。Branch 的管理以及 Merge 的工作量太大，而且容易出错。

27. 有人长期不 Check-In 代码么？

不可以。对大部分项目来说，最多两三天就应该 Check-In。

28. 在 Check-In 代码时都填写注释了么？

要写的，至少一两句话，比如“解决了 Bug No.225”。如果往高处拔，这也算做“配置审计”的一部分。

29. 有没有设定每天 Check-In 的最后期限？

要的，要明确 Check-In Deadline。否则会 Build Break。

30. 你们能把所有源码一下子编译成安装文件吗？

要的。这是每日编译（Daily Build）的基础。而且必须要能够做成自动的。

31. 你们的项目组做每日编译么？

当然要做。有三样东西是软件项目/产品开发必备的：1. bug management; 2. source control; 3. daily build。

32. 你们公司有没有积累一个项目风险列表？

要。Risk Inventory。否则，下个项目开始的时候，又只能拍脑袋分析 Risk 了。

33. 设计越简单越好越简单越好。

设计时候多一句话，将来可能就带来无穷无尽的烦恼。应该从一开始就勇敢的砍。这叫 scope management。

34. 尽量利用现有的产品、技术、代码千万别什么东西都自己 Coding。BizTalk 和 Sharepoint 就是最好的例子，有这两个作为基础，可以把起点提高很多。或者可以尽量多用现成的 Control 之类的。或者尽量用 XML，而不是自己去 Parse 一个文本文件；尽量用 RegExp，而不是自己从头操作字符串，等等等等。这就是“软件复用”的体现。

35. 你们会隔一段时间就停下来夯实代码么？

要。最好一个月左右一次。传言去年年初 Windows 组在 Stevb 的命令下停过一个月增强安全。Btw，“夯”这个字念“hang”，第一声。

36. 你们的项目组每个人都写 Daily Report 么？

要写。五分钟就够了，写 10 句话左右，告诉自己小组的人今天我干了什么。一则为了沟通，二则鞭策自己（要是游手好闲一天，自己都会不好意思写的）。

37. 你们的项目经理会发出 Weekly Report 么？

要。也是为了沟通。内容包括目前进度，可能的风险，质量状况，各种工作的进展等。

---

38. 你们项目组是否至少每周全体开会一次？

要。一定要开会。程序员讨厌开会，但每个礼拜开会时间加起来至少应该有 4 小时。包括 **team meeting**, **spec review meeting**, **bug triage meeting**。千万别大家闷头写 **code**。

39. 你们项目组的会议、讨论都有记录么？

会前发 **meeting request** 和 **agenda**，会中有人负责主持和记录，会后有人负责发 **meeting minutes**，这都是 **effective meeting** 的要点。而且，每个会议都要形成 **agreements** 和 **action items**。

40. 其他部门知道你们项目组在干什么么？

要发一些 **Newsflash** 给整个大组织。Show your team's value。否则，当你坐在电梯里面，其他部门的人问：“你们在干嘛”，你回答“ABC 项目”的时候，别人全然不知，那种感觉不太好。

41. 通过 Email 进行所有正式沟通

Email 的好处是免得抵赖。但也要避免矫枉过正，最好的方法是先用电话和当面说，然后 Email 来确认。

42. 为项目组建立多个 Mailing Group

如果在 AD+Exchange 里面，就建 Distribution List。比如，我会建 **ABC Project Core Team**, **ABC Project Dev Team**, **ABC Project All Testers**, **ABC Project Extended Team** 等等。这样发起 Email 来方便，而且能让该收到 email 的人都收到、不该收到不被骚扰。

43. 每个人都知道哪里可以找到全部的文档么？

应该每个人都知道。这叫做知识管理（**Knowledge Management**）。最方便的就是把文档放在一个集中的 **File Share**，更好的方法是用 **Sharepoint**。

44. 你做决定、做变化时，告诉大家原因了么？

要告诉大家原因。**Empower team member** 的手段之一是提供足够的 **information**，这是 **MSF** 一开篇的几个原则之一。的确如此，**tell me why** 是人之常情，**tell me why** 了才能有 **understanding**。中国人做事喜欢搞限制，限制信息，似乎能够看到某一份文件的人就是有身份的人。大错特错。权威、权力，不在于是不是能 **access information/data**，而在于是不是掌握资源。

45. **Stay agile and expect change** 要这样。

需求一定会变的，已经写好的代码一定会被要求修改的。做好心理准备，对 **change** 不要抗拒，而是 **expect change**。

46. 你们有没有专职的软件测试人员？

要有专职测试。如果人手不够，可以 **peer test**，交换了测试。千万别自己测试自己的。

47. 你们的测试有一份总的计划来规定做什么和怎么做么？这就是 **Test Plan**。要不要做性能测试？

要不要做 **Usability** 测试？什么时候开始测试性能？测试通过的标准是什么？用什么手段，自动的还是手动的？这些问题需要用 **Test Plan** 来回答。

48. 你是先写 **Test Case** 然后再测试的么？

应该如此。应该先设计再编程、先 **test case** 再测试。当然，事情是灵活的。我有时候在做第一遍测试的同时补上 **test case**。至于先 **test case** 再开发，我不喜欢，因为不习惯，太麻烦，至于别人推荐，那试试看也无妨。

---

49. 你是否会为各种输入组合创建测试用例？

不要，不要搞边界条件组合。当心组合爆炸。有很多 **test case** 工具能够自动生成各种边界条件的组合——但要想清楚，你是否有时间去运行那么多 **test case**。

50. 你们的程序员能看到测试用例么？

要。让 **Dev** 看到 **Test Case** 吧。我们都是为了同一个目的走到一起来的：提高质量。

51. 你们是否随便抓一些人来做易用性测试？

要这么做。自己看自己写的程序界面，怎么看都是顺眼的。这叫做审美疲劳——臭的看久了也就不臭了，不方便的永久了也就习惯了。

52. 你对自动测试的期望正确么？

别期望太高。依我看，除了性能测试以外，还是暂时先忘掉“自动测试”吧，忘掉 **WinRunner** 和 **LoadRunner** 吧。对于国内的软件测试的现状来说，只能“矫枉必须过正”了。

53. 你们的性能测试是等所有功能都开发完才做的么？

不能这样。性能测试不能被归到所谓的“系统测试”阶段。早测早改正，早死早升天。

54. 你注意到测试中的杀虫剂效应了么？

虫子有抗药性，**Bug** 也有。发现的新 **Bug** 越来越少是正常的。这时候，最好大家交换一下测试的 **area**，或者用用看其他工具和手法，就又会发现一些新 **bug** 了。

55. 你们项目组中有人能说出产品的当前整体质量情况么？

要有。当老板问起这个产品目前质量如何，**Test Lead/Manager** 应该负责回答。

56. 你们有单元测试么？

单元测试要有的。不过没有单元测试也不是不可以，我做过没有单元测试的项目，也做成功了——可能是侥幸，可能是大家都是熟手的关系。还是那句话，软件工程是非常实践、非常工程、非常灵活的一套方法，某些方法在某些情况下会比另一些方法好，反之亦然。

57. 你们的程序员是写完代码就扔过墙的么？

大忌。写好一块程序以后，即便不做单元测试，也应该自己先跑一跑。虽然有了专门的测试人员，做开发的人也不可以一点测试都不做。微软还有 **Test Release Document** 的说法，程序太烂的话，测试有权踢回去。

58. 你们的程序中所有的函数都有输入检查么？

不要。虽然说做输入检查是 **write secure code** 的要点，但不要做太多的输入检查，有些内部函数之间的参数传递就不必检查输入了，省点功夫。同样的道理，未必要给所有的函数都写注释。写一部分主要的就够了。

59. 产品有统一的错误处理机制和报错界面么？

要有。最好能有统一的 **error message**，然后每个 **error message** 都带一个 **error number**。这样，用户可以自己根据 **error number** 到 **user manual** 里面去看看错误的具体描述和可能原因，就像 **SQL Server** 的错误那样。同样，**ASP.NET** 也要有统一的 **Exception** 处理。可以参考有关的 **Application Block**。

60. 你们有统一的代码书写规范么？

要有。**Code Convention** 很多，搞一份来发给大家就可以了。当然，要是有了 **FxCop** 这种工具来检查代码就更好了。

---

61. 你们的每个人都了解项目的商业意义么？

要。这是 **Vision** 的意思。别把项目只当成工作。有时候要想着自己是在为中国某某行业的信息化作先驱者，或者时不时的告诉 **team member**，这个项目能够为某某某国家部门每年节省多少多少百万的纳税人的钱，这样就有动力了。平凡的事情也是可以有个崇高的目标的。

62. 产品各部分的界面和操作习惯一致么？

要这样。要让用户觉得整个程序好像是一个人写出来的那样。

63. 有可以作为宣传亮点的 **Cool Feature** 么？

要。这是增强团队凝聚力、信心的。而且，“一俊遮百丑”，有亮点就可以掩盖一些问题。这样，对于客户来说，会感觉产品从质量角度来说还是 **acceptable** 的。或者说，**cool feature** 或者说亮点可以作为质量问题的一个事后弥补措施。

64. 尽可能缩短产品的启动时间要这样。

软件启动时间（**Start-Up time**）是客户对性能好坏的第一印象。

65. 不要过于注重内在品质而忽视了第一眼的外在印象程序员容易犯这个错误：太看重性能、稳定性、存储效率，但忽视了外在感受。而高层经理、客户正相反。这两方面要兼顾，协调这些是 **PM** 的工作。

66. 你们根据详细产品功能说明书做开发么？

要这样。要有设计才能开发，这是必须的。设计文档，应该说清楚这个产品会怎么运行，应该采取一些讲故事的方法。设计的时候千万别钻细节，别钻到数据库、代码等具体实现里面去，那些是后面的事情，一步步来不能着急。

67. 开始开发和测试之前每个人都仔细审阅功能设计么？

要做。**Function Spec review** 是用来统一思想的。而且，**review** 过以后形成了一致意见，将来再也没有人可以说“你看，当初我就是反对这么设计的，现在吃苦头了吧”

68. 所有人都始终想着 **The Whole Image** 么？要这样。项目里面每个人虽然都只是在制造一片叶子，但每个人都应该知道自己在制造的那片叶子所在的树是什么样子的。我反对软件蓝领，反对过分的把软件制造看成流水线、车间。参见第 61 条。

69. **Dev** 工作的划分是单纯纵向或横向的么？

不能单纯的根据功能模块分，或者单纯根据表现层、中间层、数据库层分。我推荐这么做：首先根据功能模块分，然后每个“层”都有一个 **Owner** 来 **Review** 所有人的设计和代码，保证 **consistency**。

70. 你们的程序员写程序设计说明文档么？

要。不过我听说微软的程序员 1999 年以前也不写。所以说，写不写也不是绝对的，偷懒有时候也是可以的。参见第 56 条。

71. 你在招人面试时让他写一段程序么？

要的。我最喜欢让人做字符串和链表一类的题目。这种题目有很多循环、判断、指针、递归等，既不偏向过于考算法，也不偏向过于考特定的 **API**。

72. 你们有没有技术交流讲座？

要的。每一两个礼拜搞一次内部的 **Tech Talk** 或者 **Chalk Talk** 吧。让组员之间分享技术心得，这笔花钱送到外面去培训划算。

---

### 73. 你们的程序员都能专注于一件事情么？

要让程序员专注一件事。例如说，一个部门有两个项目和 10 个人，一种方法是让 10 个人同时参加两个项目，每个项目上每个人都花 50% 时间；另一种方法是 5 个人去项目 A，5 个人去项目 B，每个人都 100%

在某一个项目上。我一定选后面一种。这个道理很多人都懂，但很多领导实践起来就把属下当成可以任意拆分的资源了。

### 74. 你们的程序员会夸大完成某项工作所需要的时间么？

会的，这是常见的，尤其会在项目后期夸大做某个 change 所需要的时间，以此来抵制 change。解决的方法是坐下来慢慢磨，磨掉程序员的逆反心理，一起分析，并把估算时间的颗粒度变小。

### 75..尽量不要用 Virtual Heads 最好不要用 Virtual Heads。

Virtual heads 意味着 resource is not secure, shared resource 会降低 resource 的工作效率，容易增加出错的机会，会让一心二用的人没有太多时间去 review spec、review design。一个 dedicated 的人，要强过两个只能投入 50% 时间和精力的人。我是吃过亏的：7 个 part time 的 tester，发现的 Bug 和干的活，加起来还不如两个 full-time 的。参见第 73 条。73 条是针对程序员的，75 条是针对 Resource Manager 的。

我现在做的项目是采用如下方法管理的：

**BD：**基础设计。在这个文档里，我们把程序的界面全部画出来；界面上的功能全部描述完整。如：一个查询界面的条件是什么，查询出来的结果如何显示等等。

**FD：**功能设计。在这个文档里，对 BD 阶段的各个页面里包含的数据逻辑处理做说明。如：查询时调用的数据处理函数该如何设计，入口参数，返回参数，关联的表等等各方面的说明。

因为程序的界面已经定了，数据处理逻辑也定了。所以，就开始编码阶段。当编码过程中发生什么问题，程序的整个功能还是必须满足 BD 和 FD 设计文档中的要求。程序中的各种疑问，都以 BD 和 FD 文档中的说明为准。

基本上我们一个小项目的开发周期为 2 个月，BD 为 2-3 周，FD 1 周，PG（编程）2-3 周，CT(测试)2 周。测试完毕后交出去的就是成品，基本上不会再有系统要求变更的问题了。如果有变更，且不在 BD 设计范围内，那就是新增需求。就是一个新项目了。

## Android 面试题

1. 请描述下 Activity 的生命周期。
2. 如果后台的 Activity 由于某原因被系统回收了，如何在被系统回收之前保存当前状态？
3. 如何将一个 Activity 设置成窗口的样式。(Edited by Sodino)
4. 如何退出 Activity?如何安全退出已调用多个 Activity 的 Application?
5. 请介绍下 Android 中常用的五种布局。
6. 请介绍下 Android 的数据存储方式。(Edited by Sodino)
7. 请介绍下 ContentProvider 是如何实现数据共享的。(Edited by Sodino)
8. 如何启用 Service，如何停用 Service。(Edited by Sodino)
9. 注册广播有几种方式，这些方式有何优缺点?请谈谈 Android 引入广播机制的用意。
10. 请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系。
11. AIDL 的全称是什么?如何工作?能处理哪些类型的数据?
12. 请解释下 Android 程序运行时权限与文件系统权限的区别。(Edited by Sodino)
13. 系统上安装了多种浏览器，能否指定某浏览器访问指定页面?请说明原由。
14. 有一个一维整型数组 int[]data 保存的是一张宽为 width，高为 height 的图片像素值信息。请写一个算



法, 将该图片所有的白色不透明(0xffffffff)像素点的透明度调整为 50%。

---

## 15. 你如何评价 Android 系统?优缺点。

### 1、什么是 ANR 如何避免它?

答: ANR: Application Not Responding, 五秒

在 Android 中, 活动管理器和窗口管理器这两个系统服务负责监视应用程序的响应。当出现下列情况时, Android 就会显示 ANR 对话框了:

对输入事件(如按键、触摸屏事件)的响应超过 5 秒

意向接受器(intentReceiver)超过 10 秒钟仍未执行完毕

Android 应用程序完全运行在一个独立的线程中(例如 main)。这就意味着, 任何在主线程中运行的, 需要消耗大量时间的操作都会引发 ANR。因为此时, 你的应用程序已经没有任何机会去响应输入事件和意向广播(Intent broadcast)。

因此, 任何运行在主线程中的方法, 都要尽可能的只做少量的工作。特别是活动生命周期中的重要方法如 onCreate()和 onResume()等更应如此。潜在的比较耗时的操作, 如访问网络和数据库;或者是开销很大的计算, 比如改变位图的大小, 需要在一个单独的子线程中完成(或者是使用异步请求, 如数据库操作)。但这并不意味着你的主线程需要进入阻塞状态已等待子线程结束 -- 也不需要调用 Thread.wait()或者 Thread.sleep()方法。取而代之的是, 主线程为子线程提供一个句柄(Handler), 让子线程在即将结束的时候调用它(xing:可以参看 Snake 的例子, 这种方法与以前我们所接触的有所不同)。使用这种方法涉及你的应用程序, 能够保证你的程序对输入保持良好的响应, 从而避免因输入事件超过 5 秒钟不被处理而产生的 ANR。这种实践需要应用到所有显示用户界面的线程, 因为他们都面临着同样的超时问题。

### 2、什么情况会导致 Force Close ?如何避免?能否捕获导致其的异常?

答: 一般像空指针啊, 可以看起 logcat, 然后对应到程序中 来解决错误

3、Android 本身的 api 并未声明会抛出异常, 则其在运行时有可能抛出 runtime 异常, 你遇到过吗?诺有的话会导致什么问题?如何解决?

### 4、简要解释一下 activity、 intent 、intent filter、service、Broadcastcase、BroadcastReceiver

答: 一个 activity 呈现了一个用户可以操作的可视化用户界面

一个 service 不包含可见的用户界面, 而是在后台无限地运行

可以连接到一个正在运行的服务中, 连接后, 可以通过服务中暴露出来的借口与其进行通信

一个 broadcast receiver 是一个接收广播消息并作出回应的 component, broadcast receiver 没有界面

intent:content provider 在接收到 ContentResolver 的请求时被激活。

activity, service 和 broadcast receiver 是被称为 intents 的异步消息激活的。

一个 intent 是一个 Intent 对象, 它保存了消息的内容。对于 activity 和 service 来说, 它指定了请求的操作名称和待操作数据的 URI

Intent 对象可以显式的指定一个目标 component。如果这样的话, android 会找到这个 component(基于 manifest 文件中的声明)并激活它。但如果一个目标不是显式指定的, android 必须找到响应 intent 的最佳 component。

它是通过将 Intent 对象和目标的 intent filter 相比较来完成这一工作的。一个 component 的 intent filter 告诉 android 该 component 能处理的 intent。intent filter 也是在 manifest 文件中声明的。

### 5、IntentService 有何优点?

答: IntentService 的好处

\* Activity 的进程, 当处理 Intent 的时候, 会产生一个对应的 Service

\* Android 的进程处理器现在会尽可能的不 kill 掉你

\* 非常容易使用

### 6、横竖屏切换时候 activity 的生命周期?

1、不设置 Activity 的 android:configChanges 时, 切屏会重新调用各个生命周期, 切横屏时会执行一次, 切竖屏时会执行两次

2、设置 Activity 的 android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次

3、设置 Activity 的 android:configChanges="orientation|keyboardHidden"时，切屏不会重新调用各个生命周期，只会执行 onConfigurationChanged 方法

### 1. 如何将 SQLite 数据库(dictionary.db 文件)与 apk 文件一起发布?

解答：可以将 dictionary.db 文件复制到 Eclipse Android 工程中的 res aw 目录中。所有在 res aw 目录中的文件不会被压缩，这样可以直接提取该目录中的文件。可以将 dictionary.db 文件复制到 res aw 目录中

### 2. 如何将打开 res aw 目录中的数据库文件?

解答：在 Android 中不能直接打开 res aw 目录中的数据库文件，而需要在程序第一次启动时将该文件复制到手机内存或 SD 卡的某个目录中，然后再打开该数据库文件。复制的基本方法是使用 getResources().openRawResource 方法获得 res aw 目录中资源的 InputStream 对象，然后将该 InputStream 对象中的数据写入其他的目录中相应文件中。在 Android SDK 中可以使用 SQLiteDatabase.openOrCreateDatabase 方法来打开任意目录中的 SQLite 数据库文件。

### 3. Android 引入广播机制的用意?

答：a:从 MVC 的角度考虑(应用程序内)

其实回答这个问题的时候还可以这样问，android 为什么要有那 4 大组件，现在的移动开发模型基本上也是照搬的 web 那一套 MVC 架构，只不过是改了点嫁妆而已。android 的四大组件本质上就是为了实现移动或者说嵌入式设备上的 MVC 架构，它们之间有时候是一种相互依存的关系，有时候又是一种补充关系，引入广播机制可以方便几大组件的信息和数据交互。

b: 程序间互通消息(例如在自己的应用程序内监听系统来电)

c: 效率上(参考 UDP 的广播协议在局域网的方便性)

d: 设计模式上(反转控制的一种应用，类似监听者模式)

1. 请描述下 Activity 的生命周期。
2. 如果后台的 Activity 由于某原因被系统回收了，如何在被系统回收之前保存当前状态?
3. 如何将一个 Activity 设置成窗口的样式。(Edited by Sodino)
4. 如何退出 Activity? 如何安全退出已调用多个 Activity 的 Application?
5. 请介绍下 Android 中常用的五种布局。
6. 请介绍下 Android 的数据存储方式。(Edited by Sodino)
7. 请介绍下 ContentProvider 是如何实现数据共享的。(Edited by Sodino)
8. 如何启用 Service，如何停用 Service。(Edited by Sodino)
9. 注册广播有几种方式，这些方式有何优缺点? 请谈谈 Android 引入广播机制的用意。
10. 请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系。
11. AIDL 的全称是什么? 如何工作? 能处理哪些类型的数据?
12. 请解释下 Android 程序运行时权限与文件系统权限的区别。(Edited by Sodino)
13. 系统上安装了多种浏览器，能否指定某浏览器访问指定页面? 请说明原由。
14. 有一个一维整型数组 int[]data 保存的是一张宽为 width，高为 height 的图片像素值信息。请写一个算法，将该图片所有的白色不透明(0xffffffff)像素点的透明度调整为 50%。
15. 你如何评价 Android 系统? 优缺点。

1. Android dvm 的进程和 Linux 的进程，应用程序的进程是否为同一个概念

DVM 执行时，在 linux 看来就是一应用程序进程，所以说是同一概念

2. sim 卡的 EF 文件有何作用

sim 卡的文件系统有自己规范，主要是为了和手机通讯，sim 本身可以有自己的操作系统，EF 就是作存储并和手机通讯用的

4.嵌入式操作系统内存管理有哪几种， 各有何特性

页式，段式，段页，用到了 MMU,虚拟空间等技术

5. 什么是嵌入式实时操作系统,Android 操作系统属于实时操作系统吗?

分 硬实时和软实时，android 属于 linux 内核，linux 在用户空间可抢占，内核空间在 2.4 以后可局部抢占，

6. 一条最长的短信息约占多少 byte?

1、 Android dvm 的进程和 Linux 的进程, 应用程序的进程是否为同一个概念

DVM 指 dalvik 的虚拟机。每一个 Android 应用程序都在它自己的进程中运行, 都拥有一个独立的 Dalvik 虚拟机实例。而每一个 DVM 都是在 Linux 中的一个进程, 所以说可以认为是同一个概念。

2、 sim 卡的 EF 文件有何作用

sim 卡的文件系统有自己规范, 主要是为了和手机通讯, sim 本身可以有自己的操作系统, EF 就是作存储并和手机通讯用的

3、 嵌入式操作系统内存管理有哪几种, 各有何特性

页式, 段式, 段页, 用到了 MMU, 虚拟空间等技术

4、 什么是嵌入式实时操作系统, Android 操作系统属于实时操作系统吗?

嵌入式实时操作系统是指当外界事件或数据产生时, 能够接受并以足够快的速度予以处理, 其处理的结果又能在规定的时间之内来控制生产过程或对处理系统作出快速响应, 并控制所有实时任务协调一致运行的嵌入式操作系统。主要用于工业控制、军事设备、航空航天等领域对系统的响应时间有苛刻的要求, 这就需要使用实时系统。又可分为软实时和硬实时两种, 而 android 是基于 linux 内核的, 因此属于软实时。

5、 一条最长的短信息约占多少 byte?

中文 70(包括标点), 英文 160, 160 个字节

6、 android 中的动画有哪几类, 它们的特点和区别是什么?

两种, 一种是 Tween 动画、还有一种是 Frame 动画。Tween 动画, 这种实现方式可以使视图组件移动、放大、缩小以及产生透明度的变化; 另一种 Frame 动画, 传统的动画方法, 通过顺序的播放排列好的图片来实现, 类似电影。

7、 handler 机制的原理

android 提供了 Handler 和 Looper 来满足线程间的通信。Handler 先进先出原则。Looper 类用来管理特定线程内对象之间的消息交换(Message Exchange)。

1) Looper: 一个线程可以产生一个 Looper 对象, 由它来管理此线程里的 Message Queue(消息队列)。

2) Handler: 你可以构造 Handler 对象来与 Looper 沟通, 以便 push 新消息到 Message Queue 里; 或者接收 Looper 从 Message Queue 取出)所送来的消息。

3) Message Queue(消息队列): 用来存放线程放入的消息。

4) 线程: UI thread 通常就是 main thread, 而 Android 启动程序时会替它建立一个 Message Queue。

8、 说说 mvc 模式的原理, 它在 android 中的运用

MVC(Model\_view\_contraller)" 模型\_视图\_控制器"。 MVC 应用程序总是由这三个部分组成。Event(事件)导致 Controller 改变 Model 或 View, 或者同时改变两者。只要 Controller 改变了 Models 的数据或者属性, 所有依赖的 View 都会自动更新。类似的, 只要 Controller 改变了 View, View 会从潜在的 Model 中获取数据来刷新自己。

一, listview 你是怎么优化的。

二, view 的刷新, 之前说过

三, IPC 及原理

四, Android 多线程 6 n;

五, Android 为什么要设计 4 大组件, 他们之间的联系, 不设计行不行(主要是为了实现 MVC 模式, 然而 java 中最难的模式也是这个, 很少有产品能将这个模式做得很好【Technicolor 的面试官问的这个】)

六, service 的周期, activity 的周期, 谈下你对 Android 内部应用的了解, 比如他做电话, 以及联系人等等应用。框架层有很多东西还是多看看, 熟悉 Android 怎么做的, 不管你做应用程开发还是应用框架层开发很有好处的。

在就是你项目经验, 突出你遇到什么难点, 然后是怎么解决的! 尽量将每个技术点凸显出来, 当然面试官有时候会为了体现你是否真正做过, 他会问你, 你在这个应用中做那个模块, 用了多少个类之类的问题。

偶尔有的面试官会问你, 你用过 Android 自带的单元测试了没, 怎么用的?

当然我面试过很多家单位, 有的是做平板, 手机, 数字电视, 有的是做出个 erp 之类的客户端等等, 出于前面的三个, 基本上都是将 Android 的全部改掉, 如果真正要做 Android 的话, 大家要学的还很多。

总之, 一句话, 什么样的面试官都有, 去面试的时候要做好一切心理准备, 不管是技术还是基础都得扎实。

一个人的交谈能力也很重要，总之不是非常标准的普通话，最起码你说的得让别人听得懂，而且得把面试官讲得非常彻底，这样你获得 offer 的机会更大，谈工资也有优势~~当然曾经一家公司的面试官跟我说过，技术是不惜钱的，只要你有能力，多少钱他都请。

确实，来北京求职期间，牛人真的很多，而且有的面试官也非常好，给了很多忠肯的意见。并不是每个面试官都特想为难你的~最主要的还是想知道你的技术，因为他们也是吃公司饭，得为这个负责。

View 重绘和内存泄露的好像是面试经常问的问题

#### 1. View 的刷新:

在需要刷新的地方,使用 `handle.sendMessage` 发送信息,然后在 `handle` 的 `getMessage` 里面执行 `invalidate` 或者 `postinvalidate`.

#### 2. GC 内存泄露

出现情况:

1.数据库的 `cursor` 没有关闭

2.构造 `adapter` 时,没有使用缓存 `contentview`

衍生 `listview` 的优化问题-----减少创建 `view` 的对象,充分使用 `contentview`,可以使用一静态类来优化处理 `getView` 的过程/

3.`Bitmap` 对象不使用时采用 `recycle()` 释放内存

4.`activity` 中的对象的生命周期大于 `activity`

调试方法: `DDMS`==> `HEAPSIZE`==>`dataobject`==>[Total Size]

还有其他问题,大家欢迎提出阿,可以是整体架构的,还有这个 `Hal` 层.

这篇文章会涉及到以下几个内容

一 `Activity` 的生命周期

二 让 `Activity` 变成一个窗口: `Activity` 属性设定

三 你后台的 `Activity` 被系统 回收怎么办: `onSaveInstanceState`

四 调用与被调用: 我们的通信使者 - `Intent`

一 `Activity` 的生命周期

和其他手机 平台 的应用 程序 一样, `Android` 的应用程序 的生命周期是被统一掌控 的, 也就是说我们写的应用程序命运掌握在别人(系统)的手里, 我们不能改变它, 只能学习 并适应它。

简单地说一下为什么是这样: 我们手机在运行 一个应用程序的时候, 有可能打进来电话

发进来短信 , 或者没有电了, 这时候程序都会被中断, 优先去服务电话的基本功能 , 另外系统也不允许你占用太多资源 , 至少要保证电话功能吧,所以资源不足的时候也就有可能被干掉。

如何优化 `Android` 中 `ListView` 滚动速度的方法

他是让你滚动的快点还是慢点?

慢的话:

有个延迟加载数据的方法, 在 `API` 的 `demo` 中。

要是让 `ListView` 滚动速度快的话, 你可以这样告诉他:

让他的手指在屏幕上迅速的上下滑动, 当他的手指移动速度达到 `100px/0.1` 秒后, 就可以看到快速滑动的效果了! ~

`Android` 的国际化与本地化

`internationalization` (国际化) 简称 `i18n`,因为在 `i` 和 `n` 之间还有 18 个字符, `localization` (本地化), 简称 `L10n`。一般说明一个地区的语言时, 用 语言\_地区的形式, 如 `zh_CN`, `zh_TW`。

各国语言缩写 [http://www.loc.gov/standards/iso639-2/php/code\\_list.php](http://www.loc.gov/standards/iso639-2/php/code_list.php)

国家和地区简写 <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html>

`android` 对 `i18n` 和 `L10n` 提供了非常好的支持。`android` 没有专门的 `API` 来提供国际化, 而是通过对不同

resource 的命名来达到国际化，同时这种命名方法还可用于对硬件的区分，如不同的新视屏用不同的图片。

---

在 eclipse 的工程中，res 目录有默认几项 resource，如 drawable， layout， menu， values

其余还有

res/anim/ 用来放置动画

res/xml/ 用来放置 style theme 等 xml 定义。

res/raw/ 用来放置 data 数据

我们引用这些 resource 时候，在 java 代码中是通过 R.resource\_type.resource\_name 的方式来使用，如  
setTitle(R.string.main\_title);

还有一种是在 xml 中直接引用，如

```
<TextView
    android:id="@+id/hello_view"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
/>
```

引用了名字为 hello 的字符串。当程序运行时，通过 Resource 类会装载 strings.xml 中名字为 hello 的字符串。但 Resource 类装载 strings.xml 时会根据当前手机的设置来选择装载哪一个 xml 文件。这些手机设置有

MCC and MNC， Language and region， Screen dimensions， Wider/taller screens， Screen orientation， Screen pixel density， Touchscreen type， SDK version，也就是说通过手机当前语言区域，屏幕的大小，屏幕像素，以及当前 android sdk 的版本号来选择。通过把 strings.xml 放置在以这些选项命名的文件夹下，Resource 类就能正确的装载所需的字符串。

我们以不同的 local 和 region 来做一次国际化，首先 values 表示默认的字符串，也即当 Resource 找不到匹配的资源时，默认使用 values 文件夹下的资源，其余 drawable 等资源也是同样的。

当命名这些文件夹的名字时，不同的选项用-分开，而且次序必须和 andorid 文档中 table <http://developer.android.com/guide/topics/resources/resources-i18n.html#AlternateResources>

文件夹的命名必须都是小写字符，否则在一些大小敏感的文件系统中可能会出错，如果你用大写，eclipse 的 adt 都会自动报错。小写字母 r 表示 region 的意思。上图命名了中文简体和繁体以及默认选项，在 strings.xml 中除了字符串本身不一样，xml 中定义该字符串的名字，id 都是一样的。所以在代码或者 xml 中引用该资源时，只要引用名字即可或者 id 即可，程序启动时候 Resource 类回到相应的目录下去寻找正确的字符串（资源）

通过在 Settings 中设置 locale& Text 我们可以让 Resource 类来选择相应文件夹下的内容，

选择英语时候结果如下，也即选择了 values 下的 strings.xml

选择 chinese(china)时，也即选了 values-zh-rcn 目录下的 strings.xml

选择 chinese(taiwan)时，也即选了 values-zh-rtw 目录下的 strings.xml

其余手机的选项，像屏幕的像素等都可以建立相应得目录。

Android 常用控件的信息！

单选框(RadioButton 与 RadioGroup):

RadioGroup 用于对单选框进行分组，相同组内的单选框只有一个单选框被选中。

事件：setOnCheckedChangeListener(), 处理单选框被选择事件。把 RadioGroup.OnCheckedChangeListener 实例作为参数传入。

多选框(CheckBox):

每个多选框都是独立的，可以通过迭代所有的多选框，然后根据其状态是否被选中在获取其值。

事件：setOnCheckedChangeListener()处理多选框被选择事件。把 CompoundButton.OnCheckedChangeListener 实例作为参数传入

下拉列表框(Spinner):

Spinner.getItemAtPosition(Spinner.getSelectedItemPosition());获取下拉列表框的值。

事件：setOnItemSelectedListener(),处理下拉列表框被选择事件把 AdapterView.OnItemSelectedListener 实例作为参数传入；

拖动条(SeekBar):

SeekBar.getProgress()获取拖动条当前值

事件:setOnSeekBarChangeListener(), 处理拖动条值变化事件, 把 SeekBar.OnSeekBarChangeListener 实例作为参数传入。

菜单(Menu):

重写 Activity 的 onCreateOptionsMenu(Menu menu)方法, 该方法用于创建选项菜单, 当用户按下手机的"Menu"按钮时就会显示创建好的菜单, 在 onCreateOptionsMenu(Menu menu)方法内部可以调用 Menu.add()方法实现菜单的添加。

重写 Activity 的 onOptionsItemSelected()方法, 该方法用于处理菜单被选择事件。

进度对话框(ProgressDialog):

创建并显示一个进度对话框: ProgressDialog.show(ProgressDialogActivity.this,"请稍等", "数据正在加载中....", true);

设置对话框的风格: setProgressStyle()

ProgressDialog.STYLE\_SPINNER 旋转进度条风格(为默认风格)

ProgressDialog.STYLE\_HORIZONTAL 横向进度条风格

Android 系统的架构!

android 的系统架构和其操作系统一样, 采用了分层的架构。从架构图看, android 分为四个层, 从高层到低层分别是应用程序层、应用程序框架层、系统运行库层和 linux 核心层。

### 1.应用程序

Android 会同一系列核心应用程序包一起发布, 该应用程序包包括 email 客户端, SMS 短消息程序, 日历, 地图, 浏览器, 联系人管理程序等。所有的应用程序都是使用 JAVA 语言编写的。

### 2.应用程序框架

开发人员也可以完全访问核心应用程序所使用的 API 框架。该应用程序的架构设计简化了组件的重用; 任何一个应用程序都可以发布它的功能块并且任何其它的应用程序都可以使用其所发布的功能块(不过得遵循框架的安全性限制)。同样, 该应用程序重用机制也使用户可以方便的替换程序组件。

隐藏在每个应用后面的是一系列的服务和系统, 其中包括;

- \* 丰富而又可扩展的视图(Views), 可以用来构建应用程序, 它包括列表(lists), 网格(grids), 文本框(text boxes), 按钮(buttons), 甚至可嵌入的 web 浏览器。

- \* 内容提供者(Content Providers)使得应用程序可以访问另一个应用程序的数据(如联系人数据库), 或者共享它们自己的数据

- \* 资源管理器(Resource Manager)提供 非代码资源的访问, 如本地字符串, 图形, 和布局文件(layout files)。

- \* 通知管理器 (Notification Manager) 使得应用程序可以在状态栏中显示自定义的提示信息。

- \* 活动管理器(Activity Manager) 用来管理应用程序生命周期并提供常用的导航回退功能。

有关更多的细节和怎样从头写一个应用程序, 请参考 如何编写一个 Android 应用程序。

### 3.系统运行库

#### 1)程序库

Android 包含一些 C/C++库, 这些库能被 Android 系统中不同的组件使用。它们通过 Android 应用程序框架为开发者提供服务。以下是一些核心库:

- \* 系统 C 库 - 一个从 BSD 继承来的标准 C 系统函数库( libc ), 它是专门为基于 embedded linux 的设备定制的。

- \* 媒体库 - 基于 PacketVideo OpenCORE;该库支持多种常用的音频、视频格式回放和录制, 同时支持静态图像文件。编码格式包括 MPEG4, H.264, MP3, AAC, AMR, JPG, PNG 。

- \* Surface Manager - 对显示子系统的管理, 并且为多个应用程序提供了 2D 和 3D 图层的无缝融合。

- \* LibWebCore - 一个最新的 web 浏览器引擎用, 支持 Android 浏览器和一个可嵌入的 web 视图。

- \* SGL - 底层的 2D 图形引擎

- \* 3D libraries - 基于 OpenGL ES 1.0 APIs 实现;该库可以使用硬件 3D 加速(如果可用)或者使用高度优化的 3D 软加速。

- \* FreeType -位图(bitmap)和矢量(vector)字体显示。

\* SQLite - 一个对于所有应用程序可用，功能强劲的轻型关系型数据库引擎。

---

## 2)Android 运行库

Android 包括了一个核心库，该核心库提供了 JAVA 编程语言核心库的大多数功能。

每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 被设计成一个设备可以同时高效地运行多个虚拟系统。Dalvik 虚拟机执行(.dex)的 Dalvik 可执行文件，该格式文件针对小内存使用做了优化。同时虚拟机是基于寄存器的，所有的类都经由 JAVA 编译器编译，然后通过 SDK 中的“dx”工具转化成.dex 格式由虚拟机执行。

Dalvik 虚拟机依赖于 linux 内核的一些功能，比如线程机制和底层内存管理机制。

## 4.Linux 内核

Android 的核心系统服务依赖于 Linux 2.6 内核，如安全性，内存管理，进程管理，网络协议栈和驱动模型。Linux 内核也同时作为硬件和软件栈之间的抽象层。

开机自动运行程序！

Android 实现开机自动运行程序 2009-12-31 15:01:17.0 开机 自动 运行 程序 有些时候，应用需要在开机时就自动运行，例如某个自动从网上更新内容的后台 service。怎样实现开机自动运行的应用？在撰写本文时，联想到高焕堂先生以“Don't call me, I'll call you back!”总结 Android 框架，真是说到点子上了。理解这句话的含义，许多有关 Android 平台上实现某种功能的问题，都能迎刃而解。

使用场景：手机开机后，自动运行程序，在屏幕上显示"Hello. I started!"字样。

背景知识：当 Android 启动时，会发出一个系统广播，内容为 ACTION\_BOOT\_COMPLETED，它的字符串常量表示为 android.intent.action.BOOT\_COMPLETED。只要在程序中“捕捉”到这个消息，再启动之即可。记住，Android 框架说：Don't call me, I'll call you back。我们要做的是做好接收这个消息的准备，而实现的手段就是实现一个 BroadcastReceiver。

代码解析：

### 1、界面 Activity: SayHello.java

```
package com.ghstudio.BootStartDemo;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class SayHello extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello. I started!");
        setContentView(tv);
    }
}
```

这段代码很简单，当 Activity 启动时，创建一个 TextView，用它显示"Hello. I started!"字样。

### 2、接收广播消息: BootBroadcastReceiver.java

```
package com.ghstudio.BootStartDemo;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class BootBroadcastReceiver extends BroadcastReceiver {
    static final String ACTION = "android.intent.action.BOOT_COMPLETED";
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(ACTION)){
            Intent sayHelloIntent=new Intent(context,SayHello.class);
```

```
sayHelloIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

```
context.startActivity(sayHelloIntent);
```

```
}
```

```
}
```

```
}
```

该类派生自 `BroadcastReceiver`，覆载方法 `onReceive` 中，检测接收到的 `Intent` 是否符合 `BOOT_COMPLETED`，如果符合，则启动 `SayHello` 那个 `Activity`。

3、配置文件：AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
package="com.ghstudio.BootStartDemo"
```

```
android:versionCode="1"
```

```
android:versionName="1.0">
```

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
```

```
<activity android:name=".SayHello"
```

```
android:label="@string/app_name">
```

```
<intent-filter>
```

```
<action android:name="android.intent.action.MAIN" />
```

```
<category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>
```

```
</activity>
```

```
<receiver android:name=".BootBroadcastReceiver">
```

```
<intent-filter>
```

```
<action android:name="android.intent.action.BOOT_COMPLETED" />
```

```
</intent-filter>
```

```
</receiver>
```

```
</application>
```

```
<uses-sdk android:minSdkVersion="3" />
```

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses-permission>
```

```
</manifest>
```

注意其中粗体字那一部分，该节点向系统注册了一个 `receiver`，子节点 `intent-filter` 表示接收 `android.intent.action.BOOT_COMPLETED` 消息。不要忘记配置 `android.permission.RECEIVE_BOOT_COMPLETED` 权限。

完成后，编译出 `apk` 包，安装到模拟器或手机中。关机，重新开机。

如果是系统进程的话，需要加入 `init` 配置文件来启动，随着 `linux` 启动而启动

注：还有一些关于 `Activity` 之间的数据传递与 `Intent` 的具体应用、`Activity` 的生命周期等老师的 `PPT` 上已经有了。具体的笔试题目前只有华为有笔试题，并且也只是一般的选择题，其余公司都是一些 `JAVA` 的基础题，只会在面试的时候问一些关于 `Android` 的题，并且有的公司经常问一些关于底层 `C` 库的调用与 `Android` 隐式 `API` 的题。

1. 请描述下 `Activity` 的生命周期。 2. 如果后台的 `Activity` 由于某原因被系统回收了，如何在被系统回收之前保存当前状态？ 3. 如何将一个 `Activity` 设置成窗口的样式。(Edited by Sodino) 4. 如何退出 `Activity`？如何安全退出已调用多个 `Activity` 的 `Application`？ 5. 请介绍下 `Android` 中常用的五种布局。 6. 请介绍下 `Android` 的数据存储方式。(Edited by Sodino) 7. 请介绍下 `ContentProvider` 是如何实现数据共享的。(Edited by Sodino) 8. 如何启用 `Service`，如何停用 `Service`。(Edited by Sodino) 9. 注册广播有几种方式，这些方式有何优缺点？请谈谈 `Android` 引入广播机制的用意。 10. 请解释下在单线程模型中 `Message`、



Handler、Message Queue、Looper 之间的关系。 11. AIDL 的全称是什么？如何工作？能处理哪些类型的数据？ 12. 请解释下 Android 程序运行时权限与文件系统权限的区别。(Edited by Sodino) 13. 系统上安装了多种浏览器，能否指定某浏览器访问指定页面？请说明原由。 14. 有一个一维整型数组 `int[] data` 保存的是一张宽为 `width`，高为 `height` 的图片像素值信息。请写一个算法，将该图片所有的白色不透明 (0xffffffff) 像素点的透明度调整为 50%。 15. 你如何评价 Android 系统？优缺点。

## 1. 请描述下 Activity 的生命周期

<http://weizhulin.blog.51cto.com/1556324/311495>

详细介绍一下这几个方法中系统在做什么以及我们应该做什么：

<code>onCreate:</code>	在这里创建界面，做一些数据的初始化工作
<code>onStart:</code>	到这一步变成用户可见不可交互的
<code>onResume:</code>	变成和用户可交互的，（在 activity 栈系统通过栈的方式管理这些个 Activity 的最上面，运行完弹出栈，则回到上一个 Activity）
<code>onPause:</code>	到这一步是可见但不可交互的，系统会停止动画等消耗 CPU 的事情 从上文的描述已经知道，应该在这里保存你的一些数据，因为这个时候你的程序的优先级降低，有可能被系统收回。在这里保存的数据，应该在 <code>onResume</code> 里读出来，注意：这个方法里做的事情时间要短，因为下一个 activity 不会等到这个方法完成才启动
<code>onstop:</code>	变得不可见，被下一个 activity 覆盖了
<code>onDestroy:</code>	这是 activity 被干掉前最后一个被调用方法了，可能是外面类调用 <code>finish</code> 方法或者是系统为了节省空间将它暂时性的干掉，可以用 <code>isFinishing()</code> 来判断它，如果你有一个 <code>Progress Dialog</code> 在线程中转动，请在 <code>onDestroy</code> 里把他 <code>cancel</code> 掉，不然等线程结束的时候，调用 <code>Dialog</code> 的 <code>cancel</code> 方法会抛异常的。

`onPause`，`onstop`，`onDestroy`，三种状态下 activity 都有可能被系统干掉。为了保证程序的正确性，你要在 `onPause()` 里写上持久层操作的代码，将用户编辑的内容都保存到存储介质上（一般都是数据库）。实际工作中因为生命周期的变化而带来的问题也很多，比如你的应用程序起了新的线程在跑，这时候中断了，你还要去维护那个线程，是暂停还是杀掉还是数据回滚，是吧？因为 Activity 可能被杀掉，所以线程中使用的变量和一些界面元素就千万要注意了，一般我都是采用 Android 的消息机制 [Handler, Message] 来处理多线程和界面交互的问题。这个我后面会讲一些，最近因为这些东西头已经很大了，等我理清思绪再跟大家分享。

## 2. 如果后台的 Activity 由于某原因被系统回收了，如何在被系统回收之前保存当前状态？

当你的程序中某一个 Activity A 在运行时中，主动或被动地运行另一个新的 Activity B 这个时候 A 会执行

Java 代码

```

1.    public
2.    void onSaveInstanceState(Bundle outState) {
3.        super.onSaveInstanceState(outState);
4.        outState.putLong("id", 1234567890);
5.    }
6.
7.    public void onSaveInstanceState(Bundle outState) {        super.onSaveInstanceState(
    outState);        outState.putLong("id", 1234567890);}
8.

```

B 完成以后又会来找 A，这个时候就有两种情况，一种是 A 被回收，一种是没有被回收，被回收的 A 就要重新调用 onCreate() 方法，不同于直接启动的是这回 onCreate() 里是带上参数 savedInstanceState，没被收回的就还是 onResume 就好了。

savedInstanceState 是一个 Bundle 对象，你基本上可以把他理解为系统帮你维护的一个 Map 对象。在 onCreate() 里你可能会用到它，如果正常启动 onCreate 就不会有它，所以用的时候要判断一下是否为空。

Java 代码

```

if(savedInstanceState != null){
    long id = savedInstanceState.getLong("id");
}

if(savedInstanceState != null){        long id = savedInstanceState.getLong("id");}

```

就像官方的 Notepad 教程 里的情况，你正在编辑某一个 note，突然被中断，那么就把这个 note 的 id 记住，再起来的时候就可以根据这个 id 去把那个 note 取出来，程序就完整 一些。这也是看你的应用需不需要保存什么，比如你的界面就是读取一个列表，那就不需要特殊记住什么，哦， 没准你需要记住滚动条的位置...

### 3. 如何将一个 Activity 设置成窗口的样式

简单你只需要设置 一下 Activity 的主题就可以了在 AndroidManifest.xml 中定义 Activity 的地方一句话：

Xml 代码

```

1.    android :theme="@android:style/Theme.Dialog"
2.
3.    android:theme="@android:style/Theme.Dialog"
4.

```

这就使你的应用程序变成对话框的形式弹出来了，或者

Xml 代码

```

1.    android:theme="@android:style/Theme.Translucent"
2.
3.    android:theme="@android:style/Theme.Translucent"

```

就变成半透明的，[友情提示-.-]类似的这种 activity 的属性可以在 `android.R.styleable` 类的 `AndroidManifestActivity` 方法中看到，`AndroidManifest.xml` 中所有元素的属性的介绍都可以参考这个类 `android.R.styleable`

上面说的是属性名称，具体有什么值是在 `android.R.style` 中 可以看到，比如这个 `"@android:style/Theme.Dialog"` 就对应于 `android.R.style.Theme_Dialog`，（'\_' 换成 '.'）<--注意：这个是文章内容不是笑脸）就可以用在描述文件 中了，找找类定义和描述文件中的对应关系就都明白了。

#### 4. 如何退出 Activity

对于单一 Activity 的应用来说，退出很简单，直接 `finish()` 即可。

当然，也可以用 `killProcess()` 和 `System.exit()` 这样的方法。

现提供几个方法，供参考：

##### 1、抛异常强制退出：

该方法通过抛异常，使程序 Force Close。

验证可以，但是，需要解决的问题是，如何使程序结束掉，而不弹出 Force Close 的窗口。

##### 2、记录打开的 Activity：

每打开一个 Activity，就记录下来。在需要退出时，关闭每一个 Activity 即可。

##### 3、发送特定广播：

在需要结束应用时，发送一个特定的广播，每个 Activity 收到广播后，关闭即可。

##### 4、递归退出

在打开新的 Activity 时使用 `startActivityForResult`，然后自己加标志，在 `onActivityResult` 中处理，递归关闭。

除了第一个，都是想办法把每一个 Activity 都结束掉，间接达到目的。

但是这样做同样不完美。

你会发现，如果自己的应用程序对每一个 Activity 都设置了 `nosensor`，在两个 Activity 结束的间隙，`sensor` 可能有效了。

但至少，我们的目的达到了，而且没有影响用户使用。

为了编程方便，最好定义一个 Activity 基类，处理这些共通问题。

#### 5. 请介绍下 Android 中常用的五种布局

Android 布局是应用界面开发的重要一环，在 Android 中，共有五种布局方式，分别是：`FrameLayout`（框架布局），`LinearLayout`（线性布局），`AbsoluteLayout`（绝对布局），`RelativeLayout`（相对布局），`TableLayout`（表格布局）。

##### 一、FrameLayout

这个布局可以看成是墙脚堆东西，有一个四方的矩形的左上角墙脚，我们放了第一个东西，要再放一个，那就在放在原来放的位置的上面，这样依次放，会盖住原来的东西。这个布

局比较简单，也只能放一点比较简单的东西。

---

## 二、LinearLayout

线性布局，这个东西，从外框上可以理解为一个 div，他首先是一个一个从上往下罗列在屏幕上。每一个 LinearLayout 里面又可分为垂直布局

（`android:orientation="vertical"`）和水平布局（`android:orientation="horizontal"`）。当垂直布局时，每一行就只有一个元素，多个元素依次垂直往下；水平布局时，只有一行，每一个元素依次向右排列。

LinearLayout 中有一个重要的属性 `android:layout_weight="1"`，这个 weight 在垂直布局时，代表行距；水平的时候代表列宽；weight 值越大就越大。

## 三、AbsoluteLayout

绝对布局犹如 div 指定了 absolute 属性，用 X,Y 坐标来指定元素的位置 `android:layout_x="20px"` `android:layout_y="12px"` 这种布局方式也比较简单，但是在垂直随便切换时，往往会出问题，而且多个元素的时候，计算比较麻烦。

## 四、RelativeLayout

相对布局可以理解为一个元素为参照物，来定位的布局方式。主要属性有：  
相对于某一个元素

`android:layout_below="@id/aaa"` 该元素在 id 为 aaa 的下面

`android:layout_toLeftOf="@id/bbb"` 改元素的左边是 bbb

相对于父元素的地方

`android:layout_alignParentLeft="true"` 在父元素左对齐

`android:layout_alignParentRight="true"` 在父元素右对齐

还可以指定边距等，具体详见 API

## 五。TableLayout

表格布局类似 Html 里面的 Table。每一个 TableLayout 里面有表格行 TableRow, TableRow 里面可以具体定义每一个元素，设定他的对齐方式 `android:gravity=""`。

每一个布局都有自己适合的方式，另外，这五个布局元素可以相互嵌套应用，做出美观的界面。

## 6. 请介绍下 Android 的数据存储方式

Android 提供了 5 种方式存储数据：

- 使用 SharedPreferences 存储数据；
- 文件存储数据；
- SQLite 数据库存储数据；
- 使用 ContentProvider 存储数据；
- 网络存储数据；

先 说下, Preference, File, DataBase 这三种方式分别对应的目录是/data/data/Package Name/Shared\_Pref, /data/data/Package Name/files, /data/data/Package Name/database 。

在 Android 中通常使用 File 存储方式是用 Context.openFileOutput(String fileName, int mode)和 Context.openFileInput(String fileName)。

Context.openFileOutput(String fileName, int mode)生成的文件自动存储在/data/data/Package Name/files 目录下, 其全路径是/data/data/Package Name/files/fileName 。注意下, 这里的参数 fileName 不可以包含路径分割符(如"/")。

通常来说, 这种方式生成的文件只能在这个 apk 内访问。但这个结论是指使用 Context.openFileInput(String fileName)的方式。使用这种方式, 每个 apk 只可以访问自己的/data/data/Package Name/files 目录下的文件, 原因很简单, 参数 fileName 中不可以包含路径分割符, Android 会自动在/data/data /Package Name/files 目录下寻找文件名为 fileName 的文件。

#### 一: 使用 SharedPreferences 存储数据

首先说明 SharedPreferences 存储方式, 它是 Android 提供的用来存储一些简单配置信息的一种机制, 例如: 登录用户的用户名与密码。其采用了 Map 数据结构来存储数据, 以键值的方式存储, 可以简单的读取与写入, 具体实例如下:

```
void ReadSharedPreferences() {
String strName, strPassword;
SharedPreferences      user = getSharedPreferences( "user_info", 0);
strName = user.getString( "NAME", " " );
strPassword = user.getString( "PASSWORD", " " );
}

void WriteSharedPreferences(String strName, String strPassword) {
SharedPreferences      user = getSharedPreferences( "user_info", 0);
uer.edit();
user.putString( "NAME", strName);
user.putString( "PASSWORD", strPassword);
user.commit();
}
```

数据读取与写入的方法都非常简单, 只是在写入的时候有些区别: 先调用 edit() 使其处于编辑状态, 然后才能修改数据, 最后使用 commit() 提交修改 的数据。实际上 SharedPreferences 是采用了 XML 格式将数据存储到设备中, 在 DDMS 中的 File Explorer 中的/data/data/<package name>/shares\_prefs 下。以上面的数据存储结果为例, 打开后可以看到一个 user\_info.xml 的文件, 打开后可以看到:

```
<?xml version=" 1.0" encoding=" UTF-8" ?>
<map>
<string name=" NAME" >moandroid</string>
<string name="  PASSWORD" >SharedPreferences</string>
</map>
```

使用 SharedPreferences 是有些限制的: 只能在同一个包内使用, 不能在不同的包之间使用。

#### 二: 文件存储数据

文件存储方式是一种较常用的方法, 在 Android 中读取/写入文件的方法, 与 Java 中实现 I/O 的程序是完全一样的, 提供了 openFileInput() 和 openFileOutput() 方法来读取设备上的文件。 FilterInputStream, FilterOutputStream 等可以到 Java io package 说明中去详细学习, 不再此详细说明, 具体实例如下:

```
String fn = "moandroid.log";
```

```
FileInputStream fis = openFileInput(fn);
```

```
FileOutputStream fos = openFileOutput(fn, Context.MODE_PRIVATE);
```

除此之外，Android 还提供了其他函数来操作文件，详细说明请阅读 Android SDK。

### 三：网络存储数据

网络存储方式，需要与 Android 网络数据包打交道，关于 Android 网络数据包的详细说明，请阅读 Android SDK 引用了 Java SDK 的哪些 package？。

### 四：ContentProvider

#### 1、ContentProvider 简介

当应用继承 ContentProvider 类，并重写该类用于提供数据和存储数据的方法，就可以向其他应用共享其数据。虽然使用其他方法也可以对外共享数据，但数据访问方式会因数据存储的方式而不同，如：采用文件方式对外共享数据，需要进行文件操作读写数据；采用 sharedpreferences 共享数据，需要使用 sharedpreferences API 读写数据。而使用 ContentProvider 共享数据的好处是统一了数据访问方式。？

#### 2、Uri 类简介

Uri 代表了要操作的数据，Uri 主要包含了两部分信息：1. 需要操作的 ContentProvider，2. 对 ContentProvider 中的什么数据进行操作，一个 Uri 由以下几部分组成：

1. scheme：ContentProvider（内容提供者）的 scheme 已经由 Android 所规定为：content://。

2. 主机名（或 Authority）：用于唯一标识这个 ContentProvider，外部调用者可以根据这个标识来找到它。

3. 路径（path）：可以用来表示我们要操作的数据，路径的构建应根据业务而定，如下：

？要操作 contact 表中 id 为 10 的记录，可以构建这样的路径：/contact/10

？要操作 contact 表中 id 为 10 的记录的 name 字段，contact/10/name

？要操作 contact 表中的所有记录，可以构建这样的路径：/contact？

要操作的数据不一定来自数据库，也可以是文件等其他存储方式，如下：

要操作 xml 文件中 contact 节点下的 name 节点，可以构建这样的路径：/contact/name

如果要把一个字符串转换成 Uri，可以使用 Uri 类中的 parse() 方法，如下：

```
Uri uri = Uri.parse("content://com.changcheng.provider.contactprovider/contact")
```

#### 3、UriMatcher、ContentUrist 和 ContentResolver 简介

因为 Uri 代表了要操作的数据，所以我们很经常需要解析 Uri，并从 Uri 中获取数据。Android 系统提供了两个用于操作 Uri 的工具类，分别为 UriMatcher 和 ContentUris。掌握它们的使用，会便于我们的开发工作。

？UriMatcher：用于匹配 Uri，它的用法如下：

1. 首先把你需要匹配 Uri 路径全部给注册上，如下：

```
//常量 UriMatcher.NO_MATCH 表示不匹配任何路径的返回码(-1)。
```

```
UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
```

//如果 match() 方法匹配 content://com.changcheng.sqlite.provider.contactprovider/contact 路径, 返回匹配码为 1

uriMatcher.addURI(“com.changcheng.sqlite.provider.contactprovider”, “contact”, 1); //添加需要匹配 uri, 如果匹配就会返回匹配码

//如果 match() 方法匹配 content://com.changcheng.sqlite.provider.contactprovider/contact/230 路径, 返回匹配码为 2

uriMatcher.addURI(“com.changcheng.sqlite.provider.contactprovider”, “contact/#”, 2); // #号为通配符

2. 注册完需要匹配的 Uri 后, 就可以使用 uriMatcher.match(uri) 方法对输入的 Uri 进行匹配, 如果匹配就返回匹配码, 匹配码是调用 addURI() 方法传入的第三个参数, 假设匹配

content://com.changcheng.sqlite.provider.contactprovider/contact 路径, 返回的匹配码为 1。

?

ContentUri: 用于获取 Uri 路径后面的 ID 部分, 它有两个比较实用的方法:

? withAppendedId(uri, id) 用于为路径加上 ID 部分

? parseId(uri) 方法用于从路径中获取 ID 部分

? ContentResolver: 当外部应用需要对 ContentProvider 中的数据进行添加、删除、修改和查询操作时, 可以使用 ContentResolver 类来完成, 要获取 ContentResolver 对象, 可以使用 Activity 提供的 getContentResolver() 方法。ContentResolver 使用 insert、delete、update、query 方法, 来操作数据。

#### 五: 总结说明

以上 5 中存储方式, 在以后的开发过程中, 根据设计目标、性能需求、空间需求等找到合适的数据存储方式。Android 中的数据存储都是私有的, 其他应用程序都是无法访问的, 除非通过 ContentResolver 获取其他程序共享的数据。采用文件方式对外共享数据, 需要进行文件操作读写数据; 采用 sharedPreferences 共享数据, 需要使用 sharedPreferences API 读写数据。而使用 ContentProvider 共享数据的好处是统一了数据访问方式。

#### 8. 如何启用 Service, 如何停用 Service

Android 中的服务和 windows 中的服务是类似的东西, 服务一般没有用户操作界面, 它运行于系统中不容易被用户发觉, 可以使用它开发如监控之类的程序。服务的开发比较简单, 如下:

第一步: 继承 Service 类

```
public class SMSService extends Service {  
}
```

第二步: 在 AndroidManifest.xml 文件中的<application>节点里对服务进行配置:

```
<service android:name=".SMSService" />
```

服务不能自己运行, 需要通过调用 Context.startService() 或 Context.bindService() 方法启动服务。这两个方法都可以启动 Service, 但是它们的使用场合有所不同。使用 startService() 方法启用服务, 调用者与服务之间没有关连, 即使调用者退出了, 服务仍然运行。使用 bindService() 方法启用服务, 调用者与服务绑定在了一起, 调用者一旦退出, 服务也就终止, 大有“不求同时生, 必须同时死”的特点。

如果打算采用 Context.startService() 方法启动服务, 在服务未被创建时, 系统会先调用服务的 onCreate() 方法, 接着调用 onStart() 方法。如果调用 startService() 方法前服务已经被创建, 多次调用 startService() 方法并不会导致多次创建服务, 但会导致多次调用 onStart() 方法。采用 startService() 方法启动的服务, 只能调用 Context.stopService() 方法结束服务, 服务结束时会调用 onDestroy() 方法。

---

如果打算采用 `Context.bindService()` 方法启动服务,在服务未被创建时,系统会先调用服务的 `onCreate()` 方法,接着调用 `onBind()` 方法。这个时候调用者和服务绑定在一起,调用者退出了,系统就会先调用服务的 `onUnbind()` 方法,接着调用 `onDestroy()` 方法。如果调用 `bindService()` 方法前服务已经被绑定,多次调用 `bindService()` 方法并不会导致多次创建服务及绑定(也就是说 `onCreate()` 和 `onBind()` 方法并不会被多次调用)。如果调用者希望与正在绑定的服务解除绑定,可以调用 `unbindService()` 方法,调用该方法也会导致系统调用服务的 `onUnbind()` → `onDestroy()` 方法。

服务常用生命周期回调方法如下:

`onCreate()` 该方法在服务被创建时调用,该方法只会被调用一次,无论调用多少次 `startService()` 或 `bindService()` 方法,服务也只被创建一次。

`onDestroy()` 该方法在服务被终止时调用。

与采用 `Context.startService()` 方法启动服务有关的生命周期方法

`onStart()` 只有采用 `Context.startService()` 方法启动服务时才会回调该方法。该方法在服务开始运行时被调用。多次调用 `startService()` 方法尽管不会多次创建服务,但 `onStart()` 方法会被多次调用。

与采用 `Context.bindService()` 方法启动服务有关的生命周期方法

`onBind()` 只有采用 `Context.bindService()` 方法启动服务时才会回调该方法。该方法在调用者与服务绑定时被调用,当调用者与服务已经绑定,多次调用 `Context.bindService()` 方法并不会导致该方法被多次调用。

`onUnbind()` 只有采用 `Context.bindService()` 方法启动服务时才会回调该方法。该方法在调用者与服务解除绑定时被调用

采用 `Context.startService()` 方法启动服务的代码如下:

```
public class HelloActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        .....
        Button button = (Button) this.findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent(HelloActivity.this,
                SMSService.class);
                startService(intent);
            }
        });
    }
}
```

采用 `Context.bindService()` 方法启动服务的代码如下:



---

```

public class HelloActivity extends Activity {

    ServiceConnection conn = new ServiceConnection() {

        public void onServiceConnected(ComponentName name, IBinder
service) {

            }

        public void onServiceDisconnected(ComponentName name) {

            }

    };

    @Override public void onCreate(Bundle savedInstanceState) {

        Button button =(Button) this.findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {

                Intent intent = new Intent(HelloActivity.this,
SMSService.class);

                bindService(intent, conn, Context.BIND_AUTO_CREATE);
                //unbindService(conn); //解除绑定

            }

        });

    }

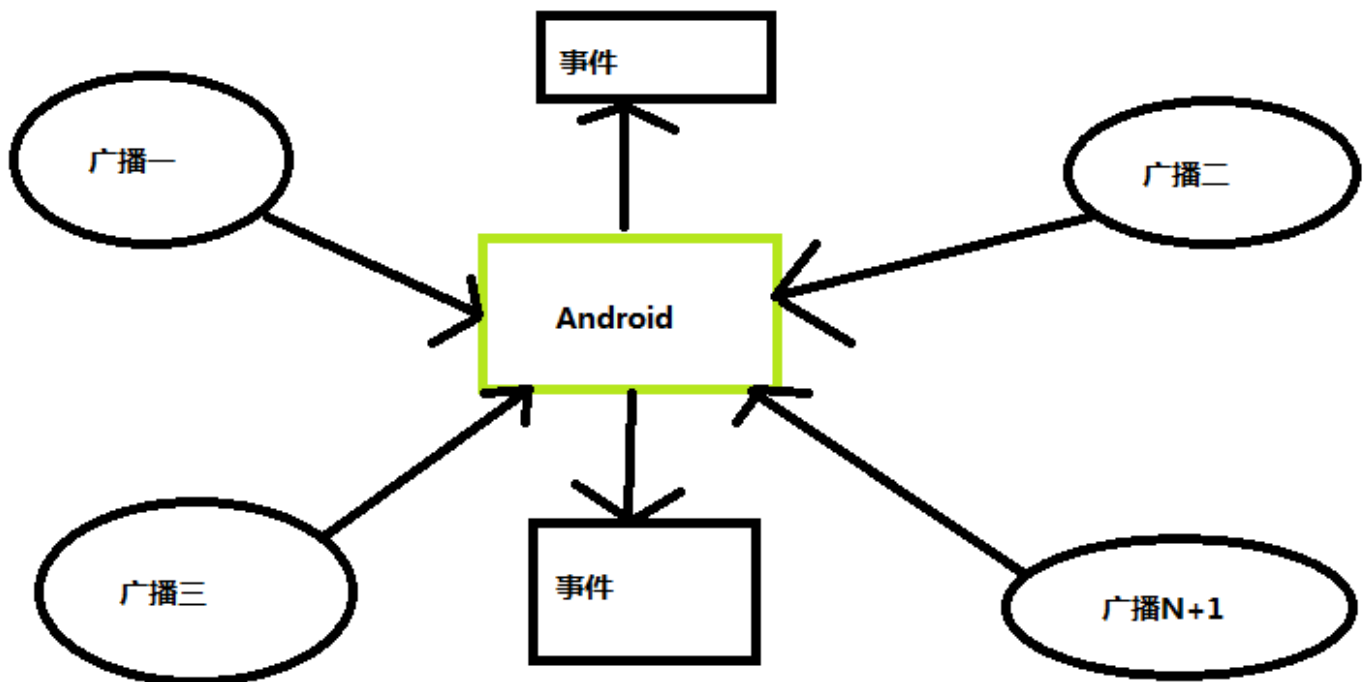
}

```

9. 注册广播有几种方式，这些方式有何优缺点？请谈谈 Android 引入广播机制的用意。

### Android 的广播机制

在 **Android** 里面有各种各样的广播，比如电池的使用状态，电话的接收和短信的接收都会产生一个广播，应用程序开发者也可以监听这些广播并做出程序逻辑的处理。下面我画一张粗略的图来帮助大家理解广播的运行机制。



Android 中有各式各样的广播，各种广播在 Android 系统中运行，当系统/应用程序运行时便会向 Android 注册各种广播，Android 接收到广播后会判断哪种广播需要哪种事件，然后向不同需要事件的应用程序注册事件，不同的广播可能处理不同的事件也可能处理相同的广播事件，这时就需要 Android 系统为我们做筛选。

案例分析：一个经典的电话黑名单，首先通过将黑名单号码保存在数据库里面，当来电时，我们接收到来电广播并将黑名单号码与数据库中的某个数据做匹配，如果匹配的话则做出相应的处理，比如挂掉电话、比如静音等等。。。

Demo 分析：

下面通过一个小 DEMO 来讲解一下广播在 Android 中如何编写，在 Demo 中我们设置了一个按钮为按钮设置点击监听通过点击发送广播，在后台接收到广播并打印 LOG 信息。代码如下：

BroadcastActivity 页面代码

```
public class BroadcastActivity extends Activity {
    public static final String ACTION_INTENT_TEST = "com.terry.broadcast.test";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button btn = (Button) findViewById(R.id.Button01);
        btn.setOnClickListener(new OnClickListener() {
```

```

        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            Intent intent = new Intent(ACTION_INTENT_TEST);
            sendBroadcast(intent);
        }
    });
}
}

```

接收器代码如下：

```

public class myBroadCast extends BroadcastReceiver {

    public myBroadCast() {
        Log.v("BROADCAST_TAG", "myBroadCast");
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
        Log.v("BROADCAST_TAG", "onReceive");
    }

}

```

## Android 广播的生命周期

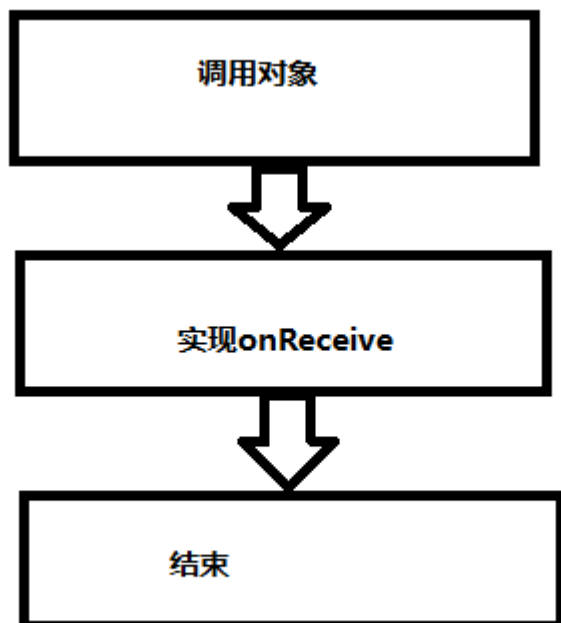
在上面的接收器中，继承了 `BroadcastReceiver` 并重写了它的 `onReceive` 并构造了一个函数，下面通过图片来一步一步认识 Android 广播的生命周期。当我点击一下按钮，它向 Android 发送了一个广播，如下图：

08-16 13:40...	V	235	BROADCAST_TAG	myBroadCast
08-16 13:40...	V	235	BROADCAST_TAG	onReceive

这时我们再点击一下按钮，它还是会再向 Android 系统发送广播，此时日志信息如下：

08-16 13:39...	V	235	BROADCAST_TAG	myBroadCast
08-16 13:39...	V	235	BROADCAST_TAG	onReceive
08-16 13:39...	D	60	dalvikvm	threadid=17: bogus mon 1+0>0; adjusting
08-16 13:40...	V	235	BROADCAST_TAG	myBroadCast
08-16 13:40...	V	235	BROADCAST_TAG	onReceive

下面本人画一张图像，描述了 Android 中广播的生命周期，其次它并不像 Activity 一样复杂，运行原理很简单如下图：



下面来看一下 SDK 给出的解释：

### Receiver Lifecycle

A BroadcastReceiver object is only valid for the duration of the call to [onReceive\(Context, Intent\)](#). Once your code returns from this function, the system considers the object to be finished and no longer active.

大意为：如果一个广播处理完 onReceive 那么系统将认定此对象将不再是一个活动的对象，也就 finished 掉它。

至此，大家应该能明白 Android 的广播生命周期的原理，代码也不用多介绍，很简单的一个发送广播并处理广播的 Demo。

### Android 如何判断并筛选广播？

前面说过 Android 的广播有各式各样，那么 Android 系统是如何帮我们处理我们需要哪种广播并为我们提供相应的广播服务呢？这里有一点需要大家注意，每实现一个广播接收类必须在我们应用程序中的 manifest 中显式的注明哪一个类需要广播，并为其设置过滤器，如下图：

```
package="com.terry"
android:versionCode="1"
android:versionName="1.0">
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".BroadcastActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <receiver android:name=".myBroadcast">
        <intent-filter>
            <action android:name="com.terry.broadcast.test"></action>
        </intent-filter>
    </receiver>
</application>
```

对应的广播接收类

在该节点下的action都是可以通过的动作

表示接受广播注册的广播动作，这里是自定义的一个动作

Tip:action 代表一个要执行的动作,在 Android 中有很多 action 比如 ACTION\_VIEW, ACTION\_EDIT

那么有些人会问了,如果我在一个广播接收器中要处理多个动作呢?那要如何去处理?

在 Android 的接收器中 onReceive 已经为我们想到的,同样的你必须在 Intent-filter 里面注册该动作,可以是系统的广播动作也可以是自己需要的广播,之后你之需要在 onReceive 方法中,通过 intent.getAction() 判断传进来的动作即可做出不同的处理,不同的动作。具体大家可以去尝试测试一下。

小结:

- 在 Android 中如果要发送一个广播必须使用 sendBroadcast 向系统发送对其感兴趣的广播接收器中。
- 使用广播必须要有一个 intent 对象必设置其 action 动作对象
- 使用广播必须在配置文件中显式的指明该广播对象
- 每次接收广播都会重新生成一个接收广播的对象
- 在 Broadcast 中尽量不要处理太多逻辑问题,建议复杂的逻辑交给 Activity 或者 Service 去处理

## Android 广播机制 (两种注册方法)

在 android 下,要想接受广播信息,那么这个广播接收器就得我们自己来实现了,我们可以继承 [BroadcastReceiver](#),就可以有一个广播接受器了。有个接受器还不够,我们还得重写 [BroadcastReceiver](#) 里面的 [onReceive](#) 方法,当来广播的时候我们要干什么,这就要我们自己来实现,不过我们可以搞一个信息防火墙。具体的代码:

```
public class SmsBroadCastReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = intent.getExtras();
        Object[] object = (Object[])bundle.get("pdu");
        SmsMessage sms[]=new SmsMessage[object.length];
        for(int i=0;i<object.length;i++)
        {
            sms[0] = SmsMessage.createFromPdu((byte[])object[i]);
            Toast.makeText(context, "来自"
            "+sms[i].getDisplayOriginatingAddress()+" 的消息是:
            "+sms[i].getDisplayMessageBody(), Toast.LENGTH_SHORT).show();
        }
        //终止广播,在这里我们可以稍微处理,根据用户输入的号码可以实现短信防火墙。
        abortBroadcast();
    }
}
```

当实现了广播接收器,还要设置广播接收器接收广播信息的类型,这里是信息:

[android.provider.Telephony.SMS\\_RECEIVED](#)

我们就可以把广播接收器注册到系统里面,可以让系统知道我们有个广播接收器。这里有两种,一种是代码动态注册:

//生成广播处理

```
smsBroadCastReceiver = new SmsBroadCastReceiver();
```

//实例化过滤器并设置要过滤的广播

```
IntentFilter intentFilter = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");
```

//注册广播

```
BroadcastReceiverActivity.this.registerReceiver(smsBroadCastReceiver, intentFilter);
```

一种是在 `AndroidManifest.xml` 中配置广播

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="spl.broadCastReceiver"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".BroadcastReceiverActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!--广播注册-->
        <receiver android:name=".SmsBroadCastReceiver">
            <intent-filter android:priority="20">
                <action android:name="android.provider.Telephony.SMS_RECEIVED"
            />

            </intent-filter>
        </receiver>

    </application>

    <uses-sdk android:minSdkVersion="7" />

    <!-- 权限申请 -->
    <uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>

</manifest>
```

两种注册类型的区别是：

- 1) 第一种不是常驻型广播，也就是说广播跟随程序的生命周期。
- 2) 第二种是常驻型，也就是说当应用程序关闭后，如果有信息广播来，程序也会被系统调用自动运行。

---

BroadcastReceiver 用于监听被广播的事件

必须被注册，有两种方法：

### 1、在应用程序的代码中注册

注册 BroadcastReceiver：

```
registerReceiver (receiver, filter) ;
```

取消注册 BroadcastReceiver：

```
unregisterReceiver (receiver) ;
```

当 BroadcastReceiver 更新 UI，通常会使用这样的方法注册。启动 Activity 时候注册 BroadcastReceiver，Activity 不可见时候，取消注册。

### 2、在 androidmanifest.xml 当中注册

```
<receiver>

    <intent-filter>

        <action android:name = "android.intent.action.PICK"/>

    </intent-filter>

</receiver>
```

使用这样的方法注册弊端：它会始终处于活动状态，毕竟是手机开发，cpu 和电源资源比较少，一直处于活动耗费大，不利。

10. 请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系。

## 1. Android 进程

在了解 Android 线程之前得先了解一下 Android 的进程。当一个程序第一次启动的时候，Android 会启动一个 LINUX 进程和一个主线程。默认的情况下，所有该程序的组件都将在该进程和线程中运行。

同时，Android 会为每个应用程序分配一个单独的 LINUX 用户。Android 会尽量保留一个正在运行进程，只在内存资源出现不足时，Android 会尝试停止一些进程从而释放足够的资源给其他新的进程使用，也能保证用户正在访问的当前进程有足够的资源去及时地响应用户的事件。Android 会根据进程中运行的组件类别以及组件的状态来判断该进程的重要性，Android 会首先停止那些不重要的进程。按照重要性从高到低一共有五个级别：

### 前台进程

前台进程是用户当前正在使用的进程。只有一些前台进程可以在任何时候都存在。他们是最后一个被结束的，当内存低到根本连他们都不能运行的时候。一般来说，在这种情况下，设备会进行内存调度，中止一些前台进程来保持对用户交互的响应。

### 可见进程

可见进程不包含前台的组件但是会在屏幕上显示一个可见的进程是的重要程度很高，除非前台进程需要获取它的资源，不然不会被中止。

### 服务进程

运行着一个通过 startService() 方法启动的 service，这个 service 不属于上面提到的 2 种更高重要性的。service 所在的进程虽然对用户不是直接可见的，但是他们执行了用户非常关注的任务（比如播放 mp3，从网络下载数据）。只要前台进程和可见进程有足够的内存，系统不会回收他们。

### 后台进程

运行着一个对用户不可见的 activity（调用过 onStop() 方法）。这些进程对用户体验没有直接的影响，可以在服务进程、可见进程、前台进程需要内存的时候回收。通常，系统中会有很多不可见进程在运行，他们被保存在 LRU（least recently used）列表中，以便内存不足的时候被第一时间回收。如果一个 activity 正 确

的执行了它的生命周期，关闭这个进程对于用户体验没有太大的影响。

## 空进程

未运行任何程序组件。运行这些进程的唯一原因是作为一个缓存，缩短下次程序需要重新使用的启动时间。系统经常中止这些进程，这样可以调程序缓存和系统缓存的平衡。

Android 对进程的重要性评级的时候，选取它最高的级别。另外，当被另外的一个进程依赖的时候，某个进程的级别可能会增高。一个为其他进程服务的进程永远不会比被服务的进程重要级低。因为服务进程比后台 activity 进程重要级高，因此一个要进行耗时工作的 activity 最好启动一个 service 来做这个工作，而不是开启一个子进程——特别是这个操作需要的时间比 activity 存在的时间还要长的时候。例如，在后台播放音乐，向网上上传摄像头拍到的图片，使用 service 可以使进程最少获取到“服务进程”级别的重要级，而不用考虑 activity 目前是什么状态。broadcast receivers 做费时的工作的时候，也应该启用一个服务而不是开一个线程。

## 2. 单线程模型

当一个程序第一次启动时，Android 会同时启动一个对应的主线程（Main Thread），主线程主要负责处理与 UI 相关的事件，如用户的按键事件，用户接触屏幕的事件以及屏幕绘图事件，并把相关的事件分发到对应的组件进行处理。所以**主线程通常又被叫做 UI 线程**。在开发 Android 应用时必须遵守单线程模型的原则：Android UI 操作并不是线程安全的并且这些操作必须在 UI 线程中执行。

### 2.1 子线程更新 UI

Android 的 UI 是单线程 (Single-threaded) 的。为了避免拖住 GUI，一些较费时的对象应该交给独立的线程去执行。如果幕后的线程来执行 UI 对象，Android 就会发出错误讯息

**CalledFromWrongThreadException**。以后遇到这样的异常抛出时就要知道怎么回事了！

### 2.2 Message Queue

在单线程模型下，为了解决类似的问题，Android 设计了一个 Message Queue (消息队列)，线程间可以通过该 Message Queue 并结合 Handler 和 Looper 组件进行信息交换。下面将对它们进行分别介绍：

#### 1. Message

Message 消息，理解为线程间交流的信息，处理数据后台线程需要更新 UI，则发送 Message 内含一些数据给 UI 线程。

#### 2. Handler

Handler 处理器，是 Message 的主要处理器，负责 Message 的发送，Message 内容的执行处理。后台线程就是通过传进来的 Handler 对象引用来 sendMessage(Message)。而使用 Handler，需要 implement 该类的 handleMessage(Message)

方法，它是处理这些 Message 的操作内容，例如 Update UI。通常需要子类化 Handler 来实现 handleMessage 方法。

### 3. Message Queue

Message Queue 消息队列，用来存放通过 Handler 发布的消息，按照先进先出执行。

每个 message queue 都会有一个对应的 Handler。Handler 会向 message queue 通过两种方法发送消息：sendMessage 或 post。这两种消息都会插在 message queue 队尾并按先进先出执行。但通过这两种方法发送的消息执行的方式略有不同：通过 sendMessage 发送的是一个 message 对象，会被 Handler 的 handleMessage() 函数处理；而通过 post 方法发送的是一个 runnable 对象，则会自己执行。

### 4. Looper

Looper 是每条线程里的 Message Queue 的管家。Android 没有 Global 的 Message Queue，而 Android 会自动替主线程 (UI 线程) 建立 Message Queue，但在子线程里并没有建立 Message Queue。所以调用 Looper.getMainLooper() 得到的主线程的 Looper 不为 NULL，但调用 Looper.myLooper() 得到当前线程的 Looper 就有可能为 NULL。

对于子线程使用 Looper，API Doc 提供了正确的使用方法：

这个 Message 机制的大概流程：

1. 在 Looper.loop() 方法运行开始后，循环地按照接收顺序取出 Message Queue 里面的非 NULL 的 Message。



2. 一开始 Message Queue 里面的 Message 都是 NULL 的。当 Handler.sendMessage(Message) 到 Message Queue, 该函数里面设置了那个 Message 对象的 target 属性是当前的 Handler 对象。随后 Looper 取出了那个 Message, 则调用 该 Message 的 target 指向的 Handler 的 dispatchMessage 函数对 Message 进行处理。

在 dispatchMessage 方法里, 如何处理 Message 则由用户指定, 三个判断, 优先级从高到低:

1) Message 里面的 Callback, 一个实现了 Runnable 接口的对象, 其中 run 函数做处理工作;

2) Handler 里面的 mCallback 指向的一个实现了 Callback 接口的对象, 由其 handleMessage 进行处理;

3) 处理消息 Handler 对象对应的类继承并实现了其中 handleMessage 函数, 通过这个实现的 handleMessage 函数处理消息。

由此可见, 我们实现的 handleMessage 方法是优先级最低的!

3. Handler 处理完该 Message (update UI) 后, Looper 则设置该 Message 为 NULL, 以便回收!

在网上有很多文章讲述主线程和其他子线程如何交互, 传送信息, 最终谁来执行处理信息之类的, 个人理解是最简单的方法——判断 Handler 对象里面的 Looper 对象是属于哪条线程的, 则由该线程来执行!

1. 当 Handler 对象的构造函数的参数为空, 则为当前所在线程的 Looper;

2. Looper.getMainLooper() 得到的是主线程的 Looper 对象, Looper.myLooper() 得到的是当前线程的 Looper 对象。

现在来看一个例子, 模拟从网络获取数据, 加载到 ListView 的过程:

这个例子, 我自己写完后觉得还是有点乱, 要稍微整理才能看明白线程间交互的过程以及数据的前后变化。随后了解到 AsyncTask 类, 相应修改后就很容易明白了!

## 11. AIDL 的全称是什么? 如何工作? 能处理哪些类型的数据

详情请参看: <http://buaadallas.blog.51cto.com/399160/372090>

### 部分概念:

在 Android 中, 每个应用程序都可以有自己的进程。在写 UI 应用的时候, 经常要用到 Service。在不同的进程中, 怎样传递对象呢? 显然, Java 中不允许跨进程内存共享。因此传递对象, 只能把对象拆分成操作系统能理解的简单形式, 以达到跨界对象访问的目的。在 J2EE 中, 采用 RMI 的方式, 可以通过序列化传递对象。在 Android 中, 则采用 AIDL 的方式。理论上 AIDL 可以传递 Bundle, 实际上做起来却比较麻烦。

Android (AndRoid 接口描述语言) 是一种借口描述语言; 编译器可以通过 aidl 文件生成一段代码, 通过预先定义的接口达到两个进程内部通信的目的。如果需要在 Activity 中, 访问另一个 Service 中的某个对象, 需要先将对象转化成 AIDL 可识别的参数(可能是多个参数), 然后使用 AIDL 来传递这些参数, 在消息的接收端, 使用这些参数组装成自己需要的对象。

AIDL 的 IPC 的机制和 COM 或 CORBA 类似, 是基于接口的, 但它是轻量级的。它使用代理类在客户端和实现层间传递值。如果要使用 AIDL, 需要完成 2 件事情: 1. 引入 AIDL 的相关类; 2. 调用 aidl 产生的 class。

AIDL 的创建方法:

AIDL 语法很简单, 可以用来声明一个带一个或多个方法的接口, 也可以传递参数和返回值。由于远程调用的需要, 这些参数和返回值并不是任何类型。下面是些 AIDL 支持的数据类型:

1. 不需要 import 声明的简单 Java 编程语言类型(int, boolean 等)

2. String, CharSequence 不需要特殊声明

3. List, Map 和 Parcelables 类型, 这些类型内所包含的数据成员也只能是简单数据类型, String 等其他比支持的类型。

(另外: 我没尝试 Parcelables, 在 Eclipse+ADT 下编译不过, 或许以后会有所支持)。

## 12. 请解释下 Android 程序运行时权限与文件系统权限的区别

要区分 apk 运行时的拥有的权限与在文件系统上被访问(读写执行)的权限两个概念。

apk 程序是运行在虚拟机上的, 对应的是 Android 独特的权限机制, 只有体现到文件系统上时才使用 linux 的权限设置。

### (一) linux 文件系统上的权限

```
-rwxr-x--x system      system      4156 2010-04-30 16:13 test.apk
```

代表的是相应的用户/用户组及其他人对此文件的访问权限, 与此文件运行起来具有的权限完全不相关。

比如上面的例子只能说明 system 用户拥有对此文件的读写执行权限; system 组的用户对此文件拥有读、执行权限; 其他人对此文件只具有执行权限。

而 test.apk 运行起来后可以干哪些事情, 跟这个就不相关了。

千万不要看 apk 文件系统上属于 system/system 用户及用户组, 或者 root/root 用户及用户组, 就认为 apk 具有 system 或 root 权限

### (二) Android 的权限规则

#### (1) Android 中的 apk 必须签名

这种签名不是基于权威证书的, 不会决定某个应用允不允许安装, 而是一种自签名证书。

重要的是, android 系统有的权限是基于签名的。比如: system 等级的权限有专门对应的签名, 签名不对, 权限也就获取不到。

默认生成的 APK 文件是 debug 签名的。

获取 system 权限时用到的签名, 见: [如何使 Android 应用程序获取系统权限](#)

#### (2) 基于 UserID 的进程级别的安全机制

大家都知道, 进程有独立的地址空间, 进程与进程间默认是不能互相访问的, 是一种很可靠的保护机制。

Android 通过为每一个安装在设备上的包 (apk) 分配唯一的 linux userID 来实现, 名称为 "app\_" 加一个数字, 比如 app\_43

不同的 UserID, 运行在不同的进程, 所以 apk 之间默认便不能相互访问。

Android 提供了如下的一种机制, 可以使两个 apk 打破前面讲的这种壁垒。

在 AndroidManifest.xml 中利用 sharedUserId 属性给不同的 package 分配相同的 userID, 通过这样做, 两个 package 可以被当做同一个程序,

系统会分配给两个程序相同的 UserID。当然, 基于安全考虑, 两个 package 需要有相同的签名, 否则没有验证也就没有意义了。

(这里补充一点: 并不是说分配了同样的 UserID, 两程序就运行在同一进程, 下面为 PS 指令摘取的, 显然, system、app\_2 分别对应的两个进程的 PID 都不同, 不知 Android 到底是怎样实现它的机制的)

```
User      PID PPID
system     953   883    187340 55052 ffffffff afe0cbcc S system_server
app_2      1072 883    100264 19564 ffffffff afe0dcc4 S com.android.inputmethod.
system     1083 883    111808 23192 ffffffff afe0dcc4 S android.process.omsservi
app_2      1088 883    156464 45720 ffffffff afe0dcc4 S android.process.acore
```

#### (3) 默认 apk 生成的数据对外是不可见的

实现方法是: Android 会为程序存储的数据分配该程序的 UserID。

借助于 Linux 严格的文件系统访问权限, 便实现了 apk 之间不能相互访问似有数据的机制。

例: 我的应用创建的一个文件, 默认权限如下, 可以看到只有 UserID 为 app\_21 的程序才能读写该文件。

```
-rw----- app_21      app_21      87650 2000-01-01 09:48 test.txt
```

如何对外开放?

<1> 使用 MODE\_WORLD\_READABLE and/or MODE\_WORLD\_WRITEABLE 标记。

When creating a new file with `getSharedPreferences(String, int)`, `openFileOutput(String, int)`, or `openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory)`, you can use the

MODE\_WORLD\_READABLE and/or MODE\_WORLD\_WRITEABLE flags to allow any other package to read/write the file. When setting these flags, the file is still owned by your application, but its global read and/or write permissions have been set appropriately so any other application can see it.

#### (4) AndroidManifest.xml 中的显式权限声明

Android 默认应用是没有任何权限去操作其他应用或系统相关特性的，应用在进行某些操作时都需要显式地去申请相应的权限。

一般以下动作时都需要申请相应的权限：

A particular permission may be enforced at a number of places during your program's operation:

- At the time of a call into the system, to prevent an application from executing certain functions.
- When starting an activity, to prevent applications from launching activities of other applications.
- Both sending and receiving broadcasts, to control who can receive your broadcast or who can send a broadcast to you.
- When accessing and operating on a content provider.
- Binding or starting a service.

在应用安装的时候，package installer 会检测该应用请求的权限，根据该应用的签名或者提示用户来分配相应的权限。

在程序运行期间是不检测权限的。如果安装时权限获取失败，那执行就会出错，不会提示用户权限不够。

大多数情况下，权限不足导致的失败会引发一个 SecurityException，会在系统 log (system log) 中有相关记录。

#### (5) 权限继承/UserID 继承

当我们遇到 apk 权限不足时，我们有时会考虑写一个 linux 程序，然后由 apk 调用它去完成某个它没有权限完成的事情，很遗憾，这种方法是行不通的。

前面讲过，android 权限是经营在进程层面的，也就是说一个 apk 应用启动的子进程的权限不可能超越其父进程的权限（即 apk 的权限），

即使单独运行某个应用有权限做某事，但如果它是由一个 apk 调用的，那权限就会被限制。

实际上，android 是通过给予进程分配父进程的 UserID 实现这一机制的。

#### (三) 常见权限不足问题分析

首先要知道，普通 apk 程序是运行在非 root、非 system 层级的，也就是说看要访问的文件的权限时，看的是最后三位。

另外，通过 system/app 安装的 apk 的权限一般比直接安装或 adb install 安装的 apk 的权限要高一些。

言归正传，运行一个 android 应用程序过程中遇到权限不足，一般分为两种情况：

(1) Log 中可明显看到权限不足的提示。

此种情况一般是 AndroidManifest.xml 中缺少相应的权限设置，好好查找一番权限列表，应该就可解决，是最易处理的情况。

有时权限都加上了，但还是报权限不足，是什么情况呢？

Android 系统有一些 API 及权限是需要 apk 具有一定的等级才能运行的。

比如 SystemClock.setCurrentTimeMillis() 修改系统时间，WRITE\_SECURE\_SETTINGS 权限 好像都是需要有 system 级的权限才行。

也就是说 UserID 是 system.

(2) Log 里没有报权限不足，而是一些其他 Exception 的提示，这也有可能是权限不足造成的。

比如：我们常会想读/写一个配置文件或其他一些不是自己创建的文件，常会报

java.io.FileNotFoundException 错误。

系统认为比较重要的文件一般权限设置的也会比较严格，特别是一些很重要的(配置)文件或目录。

如

```
-r--r----- bluetooth bluetooth          935 2010-07-09 20:21 dbus.conf
drwxrwx---x system      system            2010-07-07 02:05 data
```

dbus.conf 好像是蓝牙的配置文件，从权限上来看，根本就不可能改动，非 bluetooth 用户连读的权利都没有。

/data 目录下存的是所有程序的私有数据，默认情况下 android 是不允许普通 apk 访问/data 目录下内容的，通过 data 目录的权限设置可知，其他用户没有读的权限。

所以 adb 普通权限下在 data 目录下敲 ls 命令，会得到 opendir failed, Permission denied 的错误，通过代码 file.listFiles() 也无法获得 data 目录下的内容。

上面两种情况，一般都需要提升 apk 的权限，目前我所知的 apk 能提升到的权限就是 system（具体方法见：[如何使 Android 应用程序获取系统权限](#)），

至于是否有 root 级的，如何提升至 root 级不得而知，知道的朋友劳烦告知，感激不尽。

### 13. 系统上安装了多种浏览器，能否指定某浏览器访问指定页面？请说明原由。

指定浏览器打开网页

如果在你的 android 系统上安装了多种浏览器，能否指定某浏览器访问指定页面？答案当然是：肯定的。

具体方法如下：

```
Intent intent = new Intent();
intent.setAction("android.intent.action.VIEW");
Uri content_uri_browsers = Uri.parse("http://isomobile.com");
intent.setData(content_uri_browsers);
intent.setClassName("com.android.browser", "com.android.browser.BrowserActivity");

startActivity(intent);
```

问题的关键在于我们设置了 class name，也就是我们想要跳转的 package 的 activity。如果你想要跳转到其它的浏览器，只需要修改一下这个函数就 OK 了。

好，我们现在来让刚刚的思路来指导我们的实践。假如我们现在要直接启动 UC 浏览器，那么我们该怎么做呢？让我们 step by step 吧。

1. 下载 UC apk: <http://i-uc.net/read.php?2>

2. 用 7zip 解压 apk 文件，得到 classes.dex 文件

3. 下载反编译 dex 文件工具: <http://nchc.dl.sourceforge.net/project/dedexer/dedexer/1.5/ddx1.5.jar>  
(Dedexer 项目主页: <http://dedexer.sourceforge.net/>)

4. 执行命令: java -jar ddx1.5.jar -o -D -d c:\ c:\classes.dex

5. 得到 package name 是: com.uc.browser, 启动的 activity 是: com.uc.browser.ActivityUpdate (补充: 当我在这里 选择采用 ActivityBrowser 的时候发觉权限不够, 报 permiss denied 异常, 而且也不是我们要的

那个 activity，幸运的是在第二次尝试用 ActivityUpdate，刚好能满足要求)

---

6. 修改上面的代码为 `intent.setClassName("com.uc.browser","com.uc.browser.ActivityUpdate");`