

java 初学者实践教程 1—配置环境变量

最近我发现不少初学者，学习 java 的时候，看了好多 java 的历史、优点和应用范围。对于这些知识，并不难理解。我也当然同意 java 是一种优秀的计算机语言。但是对于我们来说要了解的并不是，这些历史等知识。而是掌握 java 这套技术。要想掌握这套技术实践是非常重要的。那么很多初学者，在第一步实践的时候就遇到了困难，就是配置环境变量。以至于，因无法继续实践而苦恼。下面为了帮广大爱好者解决这个问题，“百家拳软件项目研究室”特别写了这个教程来与大家共享。

环境变量这个概念，是我们平时用电脑时不常用的概念，所以大家在[下载完 jdk](#)之后，不知如何配置环境变量。下面我解释一下，环境变量相对于给系统或用户应用程序设置的一些变量。应该怎么理解呢？我们来做一个实验吧！

实践：

鼠标单击 开始——>运行——>cmd，进入了 DOS 的窗口。我们在任意目录下敲 QQ.会出现 “'QQ' 不是内部或外部命令，也不是可运行的程序或批处理文件。”这段话

其实也是啊，在当前的目录根本就没有 QQ 这个文件啊。我的 QQ 程序安装在 D: \Tencent\QQ 下了。你们做的时候找到这个目录。在 dos 下进入这个目录，再敲 QQ.就会发现弹出了 QQ 的登陆窗口。那么怎样能使，我们在任何目录下都可以敲 QQ 就可以执行呢。那就是设置环境变量了。

实践：

我们现在桌面上，右键单击 我的电脑——>属性——>选择“高级”选卡——>环境变量。

显示的结果如图 1—1



图 1-1

环境变量分为两类，一个是上半部分区域用户变量，另一个是下半部分系统变量。用户变量是只适用于当前用户使用，换了用户就不管用了。而系统变量则是任何用户都可以使用。呵呵，这样说可以理解吧。我们现在在用户变量里面按“新建”。在变量名里面输入 path（不区分大小写）。

变量值里面输入你 QQ 的安装路径，我的 QQ 安在了 D: \Tencent\QQ 所以你们按照自己的来做哦。



图 1-2

然后一路按“确定”按钮。接着，新打开一个 DOS 窗口。切记，一定要新打开一个 DOS 窗口，用原来的是不行的。这回在任意的目录下，敲 QQ 回车。就会发现弹出窗口了。大家做出来了吗？

所以现在我来做一下总结性陈词：环境变量相对于给系统或用户应用程序设置的一些变量，具体起什么作用这当然和具体的环境变量相关。象 path，是告诉系统，当要求系统运行一个程序而没有告诉它程序所在的完整路径时，系统除了在当前目录下面寻找此程序外，还应到那些目录下去找。当然还有很多的变量啊！以后我们会慢慢的学到。

说了这么多，我们开始开始正式的配置 jdk 吧！马上就可以敲出 java 代码了。

实践：

- 1、在[sun公司的官方网站下载jdk](#)。或者在百度或google搜索[jdk下载](#)。安装jdk；
- 2、在“用户变量”中，设置 3 项属性，JAVA_HOME，PATH，CLASSPATH（大小写无所谓），若已存在则点击“编辑”，不存在则点击“新建”；
- 3、JAVA_HOME设为JDK的安装路径（如C:\Program Files\Java\jdk1.5.0_11），此路径下包括lib, bin, jre等文件夹（此变量最好设置，因为以后运行tomcat, eclipse等都需要依靠此变量）；

Path 使得系统可以在任何路径下识别 java 命令，设为：%JAVA_HOME%\bin

CLASSPATH 为 java 加载类（class or lib）路径，只有类在 classpath 中，java 命令才能识别，设为：.;%JAVA_HOME%\lib;%JAVA_HOME%\lib（要加。表示当前路径） %JAVA_HOME%就是引用前面指定的 JAVA_HOME.形如图 1—1；

- 4、打开一个 DOS 窗口，输入“java -version”。

看看出现了，一大堆的版本信息就说明已经配置成功了。配置已经成功了，我们如何使用 jdk 呢？

java 初学者实践教程 2—jdk 的使用

通过上一节的学习，相信大家已经能够学会如何配置环境变量了。如果还有问题请联系“百家拳软件项目研究室”或者到我们的论坛交流 bbs.100jq.com。接下来，我们继续进行吧！

这节我们首先体验一下用 java 编写的程序。也让大家过把代码瘾，呵呵。目前世界上大部分的软件教程有一个习惯，最开始时总要输出一个字符串“HelloWorld”。我们也是不能免俗啊，也要输出这么一段话。

实践：

- 1、单击“开始”——>运行——>CMD，进入 DOS 系统。
- 2、用 cd 命令进入一个你容易找到的目录。如图 2—1



图 2—1

- 3、输入命令 notepad Hello.java 用记事本创建一个 java 文件。如图 2—2

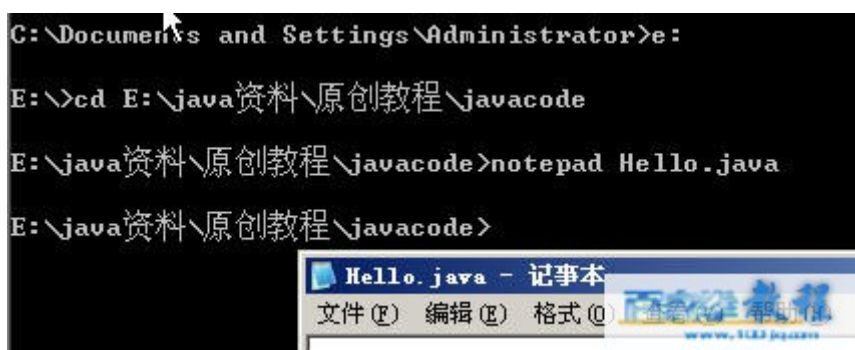


图 2—2

- 4、在里面输入下列代码

```
/*  
简单的 HelloWorld 程序
```

```
*/  
  
public class Hello{  
    //main 方法  
    public static void main (String args[]) {  
        System.out.println ("Hello World! "); //输出字符串“Hello World! ”  
    }  
}
```

5、在 DOS 的界面里，敲 `javac Hello.java` 编译这个文件。会发现文件夹里多了一个 `Hello.class` 的文件。如图 2—3

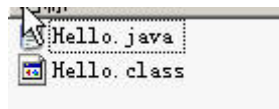


图 2—3

6、最后一步，还是在 DOS 的界面里，敲 `java Hello` 观察结果。

相信你已经看到结果了吧！

总结

通过上述几个步骤我们体验了 java 代码原来是这么编写和运行的啊。那么具体这些东西是什么意思呢？

`javac` 是 jdk 的编译器，刚才我们输入 `javac Hello.java` 的时候意思是把 `Hello.java` 这个源文件编译成了字节码，就是 `Hello.class` 这个文件。

[Java](#) 命令是 java 的解释器 `java Hello` 的意思是将编译后的字节码放在解释器上执行。从中我们也可以看到 java 语言的执行过程，是先编译后解释的。

JDK 里面还有许多命令呢！下面我们来全面了解一下 JDK。JDK 的命令为 4 类。有基本命令，RMI 命令，国际化命令，[安全](#) 控制命令。在这里我只介绍些，有代表性的命令。刚才那两个 `javac` 和 `java` 已经说过了。

他们是基本命令，基本命令里还有 `jar` 命令，也是很常用的。`Jar` 命令是 java 类的归档命令。`Jar` 命令可将多个文件合并为单个 JAR 归档文件。`Jar` 是个多用途的存档及压缩工具，它基于 `zip` 和 `zlib` 压缩格式。说的通俗一点就是它是把 java 的类文件，即 `*.class` 文件打包用的。我们来做个例子，

实践：

1、在刚才那个目录的 DOS 窗口里敲 `jar cvf hello.jar Hello.class`

2、观察结果。如图 2—4

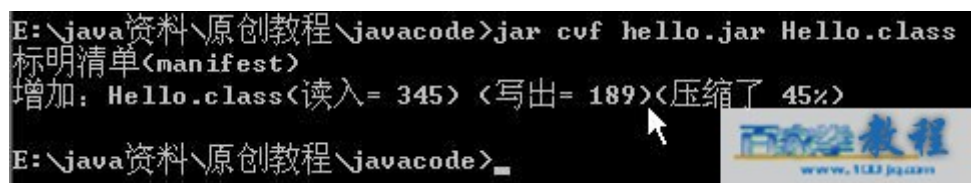


图 2—4

看看有没有生成一个叫做 hello.jar 的文件，用 winrar 打开有没有 Hello.class 这个文件呢？如果有的话就好了，其实 jar 命令还可以打很多格式的包哦。上一节我们配置 JDK 的时候，是不是把 CLASSPATH 里面配置了一个 lib 的目录，那里面也有很多 jar 包。所以说 jar 命令，是大家要掌握的一个命令。

下一个介绍国际化的命令，JDK 里只有一个这样的命令 native2ascii，该命令将本地编码字符（既非 Latin-1，又非 Unicode 字符）的文件，转换为 Unicode 编码字符文件。这是一个处理多国语言字符的命令，都转换为 Unicode 编码了，就容易处理了。这样开发国际化的软件，是非常方便的。

实践：

1、在任意目录里面建立两个文件，一个叫 gb2312.txt，另一个叫 ascii.txt

在 gb2312.txt 里面输入“百家拳软件项目研究室”这段话。之后保存。

2、在 dos 里面进入文件所在的目录。敲 native2ascii -encoding gb2312 gb2312.txt ascii.txt .

3、打开 ascii.txt 看看是什么结果呢？里面的有很多符号吧

\u767e\u5bb6\u62f3\u8f6f\u4ef6\u9879\u76ee\u7814\u7a76\u5ba4 这些就是

“百家拳软件项目研究室”这段话的 Unicode 编码。

好了这节课介绍了 java 代码的编写和 jdk 的一些命令。相信大家应该有所了解了吧！但是我们要学习一门技术的话，也不能只会编 hello world 啊。下一节我们将介绍 java 语言的基础。

java 初学者实践教程 3—基本语法 1

上回课，我们学习了并实践操作了一个 Hello World 的例子，大家显然是没有解渴。不过，回过头来有些同学问了。“你写了一大堆，是出字了。不过我不明白是什么意思啊！

这个不用着急。下面我为大家解释一下这段程序。

```
1 /*
2 简单的 HelloWorld 程序
3 */
4 public class Hello{
5  //main 方法
6  public static void main (String args[]) {
7  System.out.println ("Hello World! "); //输出字符串“Hello World! ”
8  }
9 }
```

程序中的 1-3 行是注释行

```
/*
简单的 HelloWorld 程序
*/
```

“/*.....*/”是多行注释，而“//”是单行注释的意思。

第 4 行

声明类名为 Hello，保存时要以 Classname.java 保存。类名（Classname）是在源文件中指明的，源文件编译后可在源代码所在的目录里生成一个 classname.class 文件。在本例题中，编译器创建了一个称为 Hello.class 的文件，它包含了公共类 Hello 的编译代码。

```
public class Hello{
```

第 5 行是一个单行注释

第 6 行

是程序执行的起始点。[Java](#) 技术解释器必须发现这一严格定义的点，否则将拒绝运行程序。C和[C++](#)语言，也采用main（）做为程序的起点。但是与java有些不同，以后的课程会介绍的。

第 7 行

声明如何使用类名、对象名和方法调用。它使用由 System 类的 out 成员引用的 PrintStreamout 对象的 println（）方法，将字符串“Hello World！”打印到标准输出上。

```
System.out.println（“Hello World！”）；
```

分号“；”是 java 语言语句的分隔符

第 8—9 行

那两个花括号是分别和第 4 行和第 6 行的花括号配对

数据类型的概述

数据类型对于任何一门计算机语言来说都是重要的，因为变量的数据类型决定了如何将代表这些值的位[存储](#)到计算机的内存中。在 java 语言里，数据类型分为两大类：

一、基本数据类型。

二、复合数据类型（对象数据类型）。

基本数据类型又分 4 类 8 种。如下：

逻辑型：boolean.

文本型：char.

整型：byte, short, int 和 long.

浮点型：double 和 float.

复合数据类型就有很多种了，他们都是从 Object 这个类继承下来的。

下面我想重点的讲一个问题：

文字类型的 char 和 String，这两个都是文本类型。但是不同之处，

1、char 是基本数据类型，而 String 不是，但是 String 是非常有用的；

2、char 是一个 16 位的 unicode（国际码）字符，用单引号引上。例如，

```
char c = ‘100jq’；
```


String 是一个类。字符串在 java 里是对象。在 java SE 5 中有三个类可以表示字符串：

String, StringBuffer 和 StringBuilder.StringBuilder 是 jdk1.5 的特性, 在 jdk1.5 之前的版本中没有。字符串要放在双引号中。字符串中的字符也是 Unicode .String 对象表示的字符串是不能修改的。如果要对字符串修改, 应该使用 StringBuffer 和 StringBuilder 类。

实践：

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

        // 声明整数型变量
        int x,y;

        // 声明并赋值给一个单精度浮点数变量
        float z = 3.414f;

        // 声明并赋值给一个双精度浮点数变量
        double w = 3.1415;

        // 声明并赋值给一个布尔类型的变量
        boolean truth = true;

        // 声明字符型变量
        char c;

        // 声明字符串型变量
        String str;

        //声明并赋值给一个字符串型变量
        String str1 = "bye";

        // 为字符型变量复值
        c = 'A';

        // 给字符串型变量赋值
        str = "Hi out there! ";

        // 给整型变量赋值
        x = 6;
        y = 1000;
    }
}
```

错误的赋值举例：

```
y = 15.454; //y 是个整型数
```

```
w = 456; //w 是个双精度数
```

将这个程序用上节课的方法，编译执行。就可以看到结果了。

这节课，我们基本了解了 java 的数据类型的知识。也知道了基本数据类型的概念。但是对象型的数据呢？java 是面向对象的语言啊，光靠基本数据类型也无法描述客观的世界啊。因为我们不可能说，桌子、椅子是整数，还是字符。所以对象型的数据类型是非常必要的，也是理解面向对象概念的基础之一。请看下节。

java 初学者实践教程 4—基本语法 2

上节课给大家留下一个问题，计算机要描述的是现实世界。光有基本数据类型，是不能满足我们的需要的。在这个大千世界里，任何东西都可以看做成对象，它们当然不能都是整数和文字了。现在学习 java 的同学，一般会看到一些 java 的教程里写的一个日期的例子。这个例子是个典型的例子。你们想啊，日期里面有很多属性啊，像是年，月，日。基本类型的数据就不能很好的描述它。就像是人，是一个类型。人有很多属性，头，躯干，四肢，这样的话，我们就更难用基本类型来描述了。

我们还是看看这个日期的例子吧！

```
public class MyDate {
    private int day; //日
    private int month; //月
    private int year; //年
    public MyDate(int day,int month,int year){
        this.day = day;
        this.month = month;
        this.year = year;
    }

    public MyDate(MyDate date) {
        this.day = date.day;
        this.month = date.month;
        this.year = date.year;
    }

    public int getDay() {
        return day;
    }

    public void setDay(int day) {
        this.day = day;
    }

    public MyDate addDays(int more_days) {
        MyDate new_date = new MyDate(this);
        new_date.day = new_date.day + more_days;
        return new_date;
    }
}
```

```
public void print() {  
    System.out.println("MyDate:  " + day + "-" + month +  "-" + year);  
}  
}
```

在上次课的那个例子里我们已经看到了，`class` 关键字后面是个类名。这个类名就是 `MyDate.MyDate` 这个类里面定义了 3 个属性, 4 个方法, 2 个构造函数。因为还没有讲到这些概念，目前说还是有点超前。我还是先来讲些理论的东西。

理论阐述：

类描述了同一对象都具有的数据和行为。[Java](#) 语言中的类将这些数据和行为进行封装，形成了复合数据类型。创建一个新类，就是创建了一种新的数据类型。在程序中，类只定义一次，而用 `new` 运算符可以实例化同一个类的一个或多个对象。比如人是一个类，每一个人就是一个对象。那当然是人定义一次，对象可以 `new` 出很多对象了。但是这些具体的人，都具有同样的特征（数据和行为）。

[Java](#) 的类里面有 3 个东西，看下面：

```
class 类名 {  
    声明属性;  
    声明构造函数;  
    声明方法;  
}
```

刚才讲了，类不是要描述现实的数据和行为吗？在这里属性就是要描述封装的数据，方法就是描述行为。构造函数嘛，是在 `new` 的运算符后面的，当然是构造对象的了，要不怎么能叫构造函数呢！

顺便说一下，那两个属性的前面有一个 `private` 那是权限修饰符。意思是私有的也就是别人不能用。[Java](#) 语言用这种权限修饰符实现封装。不想 C 语言的结构体，都是公有的属性，那样是不[安全](#)的。就像是人有五脏六腑，那些东西就是私有的。怎么能让谁都碰呢？这就是面向对象的一个重要的概念叫做封装。面向对象一共有三个重要特征（封装，继承，多态）我们以后会学到的。

所以，刚才 `MyDate` 那个类里面，有 3 个属性 `int` 类型的 `day`, `month`, `year` . 4 个方法。

`setDay (int day)` , `getDay ()` , `addDays (int more_days)` , `print ()` 。还有两个构造函数。

看下面的例子如何调用这个类的方法的：

```
public class TestMyDate {
```

```
public static void main (String[] args) {  
    MyDate my_birth = new MyDate (22, 7, 1964); //通过第一个  
    构造函数 new 了一个叫 my_birth 的对象，并在参数里面赋值  
  
    MyDate the_next_week = my_birth.addDays (7); //这个对象调用了  
    addDays (int more_days) 的方法，赋值给 the_next_week 的变量  
    the_next_week.print (); //调用 print () 方法  
}  
}
```

调用一个类的方法，实际上是进行对象之间或用户与对象之间的消息传递。

实践：

1、编译上述两个类，直接编译 TestMyDate.java 就行。还记得怎么做吗？
进入 DOS 界面，到你存放这两个文件的目录下，敲 javac TestMyDate.java

2、然后敲 java TestMyDate 就 OK 了。

观察结果

总结：

今天我们理解了对象类型的数据类型，也知道了对象和对象之间的调用方式。

java 实践教程 5—基本类型和引用类型变量

上两次课我们知道了，java 语言中的两种数据类型。这节课呢，我们对上两次课做一个补充，也加深一下理论知识的学习。理论的东西是很有用的啊。这节课介绍基本类型变量和引用类型变量。

[Java](#)中数据类型分为两大类，上次课已经讲完了，是基本类型和对象类型。相应的，变量也就有两种类型：基本类型和引用类型。基本类型自然不用说了，它的值就是一个数字，一个字符或一个布尔值。引用类型，可是引用类型呢？它是一个对象类型的啊，值是什么呢？它的值是指向内存空间的引用，就是地址，所指向的内存中保存着变量所表示的一个值或一组值。很好理解吧，因为一个对象，比如说一个人，不可能是一个数字也不可能是个字符啊，所以要想找它的话只能找它的地址了。

罗唆：

我们在学习计算机的过程中，所谓的难的东西，就是以前没有听过的概念，和不常用的思想。像是这个引用类型的概念就是以前不常用的，基本类型当然好理解不讲大家也知道。所以我们对于这样陌生的概念我们只要多琢磨一下就会明白的。

我们废话少说，接下来看看这两种类型变量的不同处理吧。基本类型自然是简单，声明是自然系统就给它空间了。例如，

```
int baijq;
```

```
baijq=250; //声明变量 baijq 的同时，系统给 baijq 分配了空间。
```

引用类型就不是了，只给变量分配了引用空间，数据空间没有分配，因为谁都不知道数据是什么啊，整数，字符？我们看一个错误的例子：

```
MyDate today;
```

```
today.day = 4; //发生错误，因为 today 对象的数据空间未分配。
```

那我们怎么给它赋值啊？引用类型变量在声明后必须通过实例化开辟数据空间，才能对变量所指向的对象进行访问。举个例子：

```
MyDate today; //将变量分配一个保存引用的空间
```

`today = new MyDate ();` //这句话是 2 步，首先执行 `new MyDate ()`，给 `today` 变量开辟数据空间，然后再执行赋值操作。

小总结：

刚才说了一大堆，其实就是一件事。如果是引用类型的变量，必须先得 `new` 一个对象出来。不 `new` 哪来的对象啊，不 `new` 哪有数据空间啊？没有数据空间怎么能访问呢？这回明白了吧！

我们还有个问题没有说，引用类型变量怎么赋值？这个就不是很难了。举个例子：

`MyDate a, b;` //在内存开辟两个引用空间

`A = new MyDate ();` //开辟 `MyDate` 对象的数据空间，并把该空间的首地址赋给 `a`

`B = a;` //将[a](#)存储空间中的地址写到**a**的[存储](#)空间中

如图 5-1

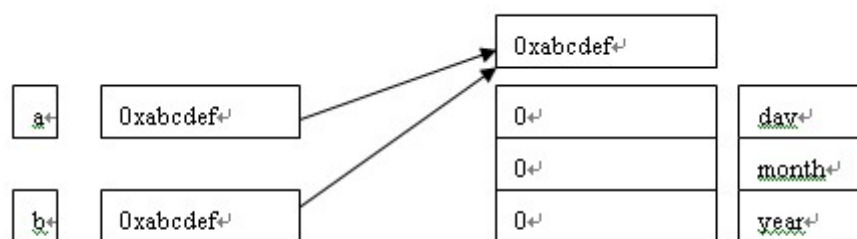


图 5-1

java 初学者实践教程 6—程序流程控制

这节课我们又要讲语法了，这是“百家拳软件项目研究室”这部教程的第 6 节课，我们这个教程侧重的是实践的内容和语言的重点。在 java 语言中还有很多细节的东西，请参考 sun 公司的官方培训教程。我们这里不能一一讲述。这节课我们来给大家提供一些程序流程控制的一些例子供大家学习。计算机怎么做事情，是我们教给他的。我们用它解决实际生活中的问题，所以计算机要描述现实生活中的流程。

[Java](#)语言中提供了 4 类程序控制语句，来描述流程：

- 1.循环语句：while，do-while，for
- 2.分支语句：if-else，switch，
- 3.跳转语句 break，continue，label： 和 return
- 4.异常处理语句：try-catch-finally，throw

实践：

- 1.循环语句

while 语句

```
class While {  
    public static void main (String args[]) {  
        int n = 10;  
        while (n > 0) {  
            System.out.println ("tick " + n) ;  
            n--;  
        }  
    }  
}
```

do...while 语句

```
class DoWhile {  
    public static void main (String args[]) {  
        int n = 10;  
        do {  
            System.out.println ("tick " + n) ;  
        }  
    }  
}
```

```

        n--;
    } while (n > 0) ;
}
}

```

二者区别，do...while 至少循环一次，而 while 的表达式要是为 false 的话可以一次也不循环。再通俗一点，do...while 就算是括号里的是 false，人家最少也能 do 一次。

for 语句

```

class ForTick {
    public static void main (String args[]) {
        int n;
        for (n=10; n>0; n--)
            System.out.println ("tick " + n) ;
        }
    }
}

```

与上面那两个的区别，for 循环执行的次数是可以在执行之前确定的。通俗一点说吧，看这个例子 for (n=10; n>0; n--) 就是在括号里的时候，就已经知道要循环 10 次了。

还有啊，for 循环的部分可以为空的

```

class ForVar {
    public static void main (String args[]) {
        int i;
        boolean done = false;
        i = 0;
        for ( ; ! done; ) {
            System.out.println ("i is " + i) ;
            if (i == 10) done = true;
            i++;
        }
    }
}
}

```

2. 分支语句

if/else 语句

```

class IfElse {

```

```

public static void main (String args[]) {
    int month = 4;    // April
    String season;
    if (month == 12 || month == 1 || month == 2)
        season = "Winter";
    else if (month == 3 || month == 4 || month == 5)
        season = "Spring";
    else if (month == 6 || month == 7 || month == 8)
        season = "Summer";
    else if (month == 9 || month == 10 || month == 11)
        season = "Autumn";
    else
        season = "Bogus Month";
    System.out.println ("April is in the " + season + ".") ;
}
}

```

//这段程序输出:

//April is in the Spring.

// 注意 “||”是或运算

switch 语句

```

class Switch {
    public static void main (String args[]) {
        int month = 4;
        String season;
        switch (month) {
            case 12:
            case 1:
            case 2:
                season = "Winter";
                break;
            case 3:
            case 4:
            case 5:
                season = "Spring";
                break;
            case 6:
            case 7:
            case 8:
                season = "Summer";

```

```
        break;
    case 9:
    case 10:
    case 11:
        season = "Autumn";
        break;
    default:
        season = "Bogus Month";
    }
    System.out.println ("April is in the " + season + ".") ;
}
}
```

switch 语句适合于条件非常多的逻辑

请看上述语句可以混合使用。

java 初学者实践教程 7—跳转语句

上一节我们说有 4 类程序控制语句，但是才讲了 2 个。今天讲跳转语句。异常处理语句我们找一节专题来讲。

循环跳转语句：

`break [label]` //用来从语句、循环语句中跳出。

`continue [label]` //跳过循环体的剩余语句，开始下一次循环。

这两个语句都可以带标签（`label`）使用，也可以不带标签使用。标签是出现在一个语句之前的标识符，标签后面要跟上一个冒号（`:`），标签的定义如下：

`label: statement;`

实践：

1、break 语句

```
class Break {
    public static void main(String args[]) {
        boolean t = true;
        first: {
            second: {
                third: {
                    System.out.println("Before the break.");
                    if(t) break second; // break out of second block
                    System.out.println("This won't execute");
                }
                System.out.println("This won't execute");
            }
            System.out.println("This is after second block.");
        }
    }
}
```

// 跳出循环

```
class BreakLoop {
    public static void main(String args[]) {
        for(int i=0; i<100; i++) {
```

```
if(i == 10) break; // terminate loop if i is 10
System.out.println("i: " + i);
}
System.out.println("Loop complete.");
}
}
```

//跳出 switch

```
class SampleSwitch {
public static void main(String args[]) {
for(int i=0; i<6; i++)
switch(i) {
case 0:
System.out.println("i is zero.");
break;
case 1:
System.out.println("i is one.");
break;
case 2:
System.out.println("i is two.");
break;
case 3:
System.out.println("i is three.");
break;
default:
System.out.println("i is greater than 3.");
}
}
}
```

这个在昨天的分支语句中，我们就已经学到了。

2、continue 语句

```
class Continue {
public static void main(String args[]) {
for(int i=0; i<10; i++) {
System.out.print(i + " ");
if (i%2 == 0) continue;
System.out.println("");
}
}
}
```

```
}
```

//带标签的 continue

```
class ContinueLabel {  
    public static void main(String args[]) {  
        outer: for (int i=0; i<10; i++) {  
            for(int j=0; j<10; j++) {  
                if(j > i) {  
                    System.out.println();  
                    continue outer;  
                }  
                System.out.print(" " + (i * j));  
            }  
        }  
        System.out.println();  
    }  
}
```


java 初学者实践教程 8—jdk5 的拆箱与装箱

前几次课的讲解，我们了解了这样几个问题。[Java](#)的两种数据类型，和一些程序控制语句。今天，我们还是要通过一些例子。对上述东西有一个更深的理解。

我们现在知道了，所有对象型的数据类型的基类是`java.lang.Object`。而写java程序的时候非常多的工作都是在写这些类，和实现里面的方法。而偏偏就有那么8种基本类型和他们不一样。以至于让你来回在这两种之间转换，这是很让人头疼的事情。[Java](#)中`int`, `long`, `char`这样的类型不是对象型。因此java里提供了一种叫做包装类（wrapper）的东西，使基本类型，有着相应的对象类型`Integer`, `Long`, `Character`等。这样就可以，先把基本类型的东西，转成对象来用，然后再转回去。来来回回，千锤百炼。

到了 jdk5.0 的时候呢，就不用了。看下面的例子：

实践：

```
public class Test1 {
    public static void main(String[] args) {
        // 装箱
        int i = 0;
        Integer integer = i; // i 这么一个基本类型的数，可以赋值给 Integer
        // 简单的拆箱
        int j = integer; // integer 这种原始类型的数，也能赋值给 j 这个原始类型的变量

        Integer counter = 1;           // 装箱
        int counter2 = counter;        // 拆箱

        while (counter < 100) {
            System.out.println("计数 "+counter++); // 看啊，counter 这个对象型的数，还能自动增加
        }
    }
}
```

在幕后 JVM 已经自动执行了转换，同理 `Boolean` 和 `boolean` 之间也可以，自动拆箱装箱。但是，`Integer` 和 `int` 毕竟还是有着不同的。

看下面例子：

```
public class Test2 {  
    public static void main(String[] args) {  
        Integer i1 = 256;  
        Integer i2 = 256;  
        if (i1 == i2)  
            System.out.println("相等!");  
        else  
            System.out.println("不相等!");  
    }  
}
```

结果输出的是“不相等！”，两个对象比较，它们在内存中开辟的是两个地址怎么能相等呢？

警告：你可千万不能依赖这个结果，请把 i1 和 i2 的值，改成 100.（请看 Test3.java）看看什么结果，令人惊讶的是改了个数，居然输出了“相等！”。

这是因为 JVM 可以选择要尝试这段代码的最佳优化，并对两个 Integer 对象使用一个实例，这样的话“==”就会返回 true 了。在自动装箱时对于值从-128 到 127 之间的值，使用一个实例。

这种装箱与拆箱机制对，程序流程控制语句，也有很大影响：

```
public class Test4 {  
    public static void main(String[] args) {  
        Boolean arriving = true;  
        Integer peopleInRoom = 0;  
        int maxCapacity = 100;  
  
        while (peopleInRoom < maxCapacity) {  
            if (arriving) {  
                System.out.printf("很高兴见到你.%d 号先生\n", peopleInRoom);  
                peopleInRoom++;  
            }  
            else {  
                peopleInRoom--;  
            }  
        }  
    }  
}
```

另外一个从 unboxing 获得好处的语句是 switch.在 jdk5.0 之前的 JVM, switch 接受 int、short、character 或者 byte 值，而在 unboxing 的操作中，你现在也可以

为它赋予新引入的 `enum` 之外的 `Integer`, `Short`, `Char` 以及 `Byte` 值。`Enum` 的值, 我们在后面的教程会详细讲述。

java 初学者实践教程 9—数组

今天我们讲个不一样的概念——数组，数组也没什么不好理解的，就是一组数。不过这组数有点特性。今天我们的任务就是，了解这个有特性的这组数。下面我们具体讲一下它有哪些特性：

1、数组中的元素是同一类型。数组的长度在创建时确定，并且在创建后不变。解释一下 声明一个数组 `int i[5]`；这就是 `int` 类型，名字叫 `i` 的数组。里面的数都必须是 `int` 类型。并且长度在创建时确定了是 5。

2、在 `java` 语言中，数组就是一个对象，所以创建数组与创建对象一样也是用 `new` 关键字来创建。举个例子，`s = new char[20]`；`p = new Point[50]`。

3、数组在被创建后，其元素被系统自动初始化了。字符元素被初始化为 `'\u0000'`，而对于对象数组都被初始化为 `null`。如果你不初始化的话，在内存是找不到它的位置的。

4、数组中的第一元素记做第 0 个，`i[0]`是数组 `i` 的第一个元素。

说了这么些，我们还是得练练啊

实践：

```
public class TestArrays {
    public static void main(String[] args) {
        // 第 1, 2 步: 声明并初始化数组变量
        int[] array1 = { 2, 3, 5, 7, 11, 13, 17, 19 };
        int[] array2;

        // 第 3 步: 显示数组初始化值
        System.out.print("array1 is ");
        printArray(array1);
        System.out.println();
        // 第 4 步: array2 引用 array1
        array2 = array1;
        // 更改 array2
        array2[0] = 0;
        array2[2] = 2;
        array2[4] = 4;
        array2[6] = 6;
        // 打印 array1
        System.out.print("array1 is ");
```

```

printArray(array1);
System.out.println();
// 第 5 步: 声明一个整数类型的二维数组
int[][] matrix = new int[5][];
// 第 6 步: 将这个矩阵构成三角形
for ( int i = 0; i < matrix.length; i++ ) {
    matrix[i] = new int[i];
    for ( int j = 0; j < i; j++ ) {
matrix[i][j] = i * j;
    }
}
// 第 7 步打印矩阵
for ( int i = 0; i < matrix.length; i++ ) {
    System.out.print("matrix[" + i + "] is ");
    printArray(matrix[i]);
    System.out.println();
}
}
public static void printArray(int[] array) {
    System.out.print('<');
    for ( int i = 0; i < array.length; i++ ) {
        // 打印一个元素
        System.out.print(array[i]);
        // 输出最后一个元素的时候不输出逗号
        if ( (i + 1) < array.length ) {
            System.out.print(", ");
        }
    }
    System.out.print('>');
}
}

```

在 jdk5.0 中, 我们发现了一些更简单的方法, 打印一维数组时, 用 `toString(array)` 方法, 打印二维数组时, 用 `deepToString(array)` 方法。这样的话就剩了我们又是循环又是判断的。我们看个例子吧:

实践:

```

import java.util.Arrays;
public class ArraysTester {
    private int[] ar;

    public ArraysTester(int numValues) {
        ar = new int[numValues];
    }
}

```

```

        for (int i=0; i < ar.length; i++) {
            ar[i] = (1000 - (300 + i));
        }
    }
    public int[] get() {
        return ar;
    }
    public static void main(String[] args) {
        ArraysTester tester = new ArraysTester(50);
        int[] myArray = tester.get();
        // 比较两个数组
        int[] myOtherArray = tester.get().clone();
        if (Arrays.equals(myArray, myOtherArray)) {
            System.out.println("这两个数组是相等的!");
        } else {
            System.out.println("这两个数组是不相等的!");
        }
        // 填上一些值
        Arrays.fill(myOtherArray, 2, 10, new Double(Math.PI).intValue());
        myArray[30] = 98;
        // 打印数组
        System.out.println("这是一个未排序的数组...");
        System.out.println(Arrays.toString(myArray));
        System.out.println();
        // 数组排序
        Arrays.sort(myArray);
        // 打印被排序的数组 用 toString()
        System.out.println("这是一个被排序的数组...");
        System.out.println(Arrays.toString(myArray));
        System.out.println();

        // 得到特殊值的索引
        int index = Arrays.binarySearch(myArray, 98);
        System.out.println("98 被定位在第 " + index + "个位置上");

        String[][] ticTacToe = { {"X", "O", "O"},
                                   {"O", "X", "X"},
                                   {"X", "O", "X"} };

        //打印二维数组用 deepToString()
        System.out.println(Arrays.deepToString(ticTacToe));
        String[][] ticTacToe2 = { {"O", "O", "X"},
                                   {"O", "X", "X"},
                                   {"X", "O", "X"} };

        String[][] ticTacToe3 = { {"X", "O", "O"},

```

```
                {"O", "X", "X"},  
                {"X", "O", "X"}  
            };  
            if (Arrays.deepEquals(ticTacToe, ticTacToe2)) {  
                System.out.println("Boards 1 和 2 相等.");  
            } else {  
                System.out.println("Boards 1 和 2 不相等.");  
            }  
            if (Arrays.deepEquals(ticTacToe, ticTacToe3)) {  
                System.out.println("Boards 1 和 3 are 相等.");  
            } else {  
                System.out.println("Boards 1 和 3 are 不相等.");  
            }  
        }  
    }  
}
```


java 初学者实践教程 10—集合类

上次课我们学过了数组，知道它只是一组数（或是对象），但是有些自己的特性。在 java 里还有一类东西与数组类似，也是有着特性的一组数（或是对象），叫做集合类。我们上节课讲到了，数组的长度在创建时已经确定了，但是有时候我们事先根本不知道长度是多少啊，比如我们做电子商务网站时，有个购物车程序。你总不能用数组规定，人家只能买 5 样东西吧。你就是把长度定为 10000 也不行，万一遇上个特别有钱的呢！呵呵，这只是开玩笑的。我们会使用集合类解决这个问题。

集合类是放在 `java.util.*`；这个包里。集合类存放的都是对象的引用，而非对象本身，为了说起来方便些，我们称集合中的对象就是指集合中对象的引用（reference）。引用的概念大家不会忘了吧，在前边我们讲数据类型时讲的。

集合类型主要有 3 种：`set`（集）、`list`（列表）、`map`（映射）和 `Queue`（队列）。//队列为 jdk5 中的加上的

（1）Set

集（set）是最简单的一种集合，它的对象不按特定方式排序，只是简单的把对象加入集合中，就像往口袋里放东西。对集中成员的访问和操作是通过集中对象的引用进行的，所以集中不能有重复对象。我们知道数学上的集合也是 Set 这个，集合里面一定是没有重复的元素的。

（2）List

列表（List）的主要特征是其对象以线性方式[存储](#)，没有特定顺序，只有一个开头和一个结尾，当然，它与根本没有顺序的 Set 是不同的。它是链表嘛，一条链肯定有顺序这个顺序就不一定了。

（3）Map

映射（Map），这个在 java 里不是地图的意思，其实地图也是映射哈。它里面的东西是键—值对（key-value）出现的，键值对是什么呢？举个例子，比如我们查字典，用部首查字法。目录那个字就是键，这个字的解释就是值。键和值成对出现。这样说可以理解吧。这也是很常用的数据结构哦。

（4）Queue

在 jdk5.0 以前，通常的实现方式是使用 `java.util.List` 集合来模仿 `Queue`。`Queue` 的概念通过把对象添加（称为 `enqueueing` 的操作）到 List 的尾部（即 `Queue` 的后部）并通过从 List 的头部（即 `Queue` 的前部）提取对象而从 List 中移除（称为

dequeuing 的操作) 来模拟。你需要执行先进先出的动作时可以直接使用 Queue 接口就可以了。

这 4 个东西, 有时候功能还不太完善, 需要有些子类继承它的特性。Set 的子接口有 TreeSet, SortedSet, List 的有 ArrayList 等, Map 里有 HashMap, Hashtable 等, Queue 里面有 BlockingQueue 等。我们来看看例子吧:

实践:

Set 举例

```
import java.util.*;

public class SetExample {
    public static void main(String[] args) {
        Set set = new HashSet(); //HashSet 是 Set 的子接口
        set.add("one");
        set.add("second");
        set.add("3rd");
        set.add(new Integer(4));
        set.add(new Float(5.0F));
        set.add("second");
        set.add(new Integer(4));
        System.out.println(set);
    }
}
```

List 举例:

```
import java.util.*;

public class ListExample {
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add("one");
        list.add("second");
        list.add("3rd");
        list.add(new Integer(4));
        list.add(new Float(5.0F));
        list.add("second");
        list.add(new Integer(4));
        System.out.println(list);
    }
}
```

Map 举例

```
import java.util.Map;
```

```

import java.util.HashMap;
import java.util.Iterator;
import java.io.FileReader;

public class MapExample {
    public static void main(String[] args) throws
java.io.FileNotFoundException {
        Map word_count_map = new HashMap();
        FileReader reader = new FileReader(args[0]);
        Iterator words = new WordStreamIterator(reader);

        while ( words.hasNext() ) {
            String word = (String) words.next();
            String word_lowercase = word.toLowerCase();
            Integer frequency =
(Integer)word_count_map.get(word_lowercase);
            if ( frequency == null ) {
                frequency = new Integer(1);
            } else {
                int value = frequency.intValue();
                frequency = new Integer(value + 1);}
            word_count_map.put(word_lowercase,
frequency);
        }
        System.out.println(word_count_map);
    }
}

```

Queue 举例:

```

import java.io.IOException;
import java.io.PrintStream;
import java.util.LinkedList;
import java.util.Queue;

public class QueueTester {
    public Queue<String> q; //发现了一个奇怪的语法, 这个尖
括号是泛型声明
    public QueueTester() {q = new LinkedList<String>();}
    public void testFIFO(PrintStream out) throws IOException {
        q.add("First");
        q.add("Second");
        q.add("Third");
        Object o;
    }
}

```

```
        while ((o = q.poll()) != null) {  
            out.println(o);}  
    public static void main(String[] args) {  
        QueueTester tester = new QueueTester();  
        try {    tester.testFIFO(System.out);  
            } catch (IOException e) {  
                e.printStackTrace(); } } }
```

总结：

刚才我们看了上述例子了，对集合类有了一个初步的认识，它们跟数组的区别不只是长度问题，在集合类里面放进去的类型可以是任意的。不像数组，数组里面的类型是一样的。这样的话，对于集合类来说即使好事，也是坏事。因为你不考虑类型可以随意的放，但是你放进去什么就不知道了不容易找。

还有啊，什么叫泛型声明啊？我们下次课就告诉你。这可是 jdk5 的酷炫之处哦。

java 初学者实践教程 11—泛型声明

上节课我们留下了一个泛型声明的概念，这个概念乍一听起来是很陌生的，不过不要紧，听我细细道来。泛型声明就是泛泛的声明类型。我们用其它的语言做一个比较：

[JavaScript](#)声明变量： `var i=1;var c='char'`

VB 声明变量： `dim i=1;dim c='char'`

Perl 声明变量： `$i = 1; $c = 'char'`

这些脚本语言，在声明变量的时候，根本就不想 java 那样还得声明类型。他们的类型已经自己声明完了，[是泛泛的声明的，这些语言本身就是泛型](#)。因为数据类型可能会改变，所以用不着像 java 定的那么死。但是数据类型可不是说变就变的，java 的强类型机制，保证了逻辑的严谨性，而且确保着程序员犯错误，这是 java 的优点。同时使得它开发起来没有上述语言那样简单，一点小事上做起来都很笨拙。这是其中一个方面，另一个方面如我们上节讲的[java 的集合类里面的类型是不确定的](#)，放什么都行啊。这样的话你明明自己放进去的类型，也就是说你自己已经知道类型了，拿出来时候，还得不断的转换。我们在介绍拆箱与装箱机制的时候已经说过这个类型的问题了。拆箱与装箱确实能解决不少问题但那是不够的。

所以接着上节课学集合类的劲头，趁热打铁。我们讲一下刚才说的第二个方面，关于集合类的问题。我们刚才说，java 这种类型[安全](#)之下的破绽，我们要用泛型的方式来弥补。我们来实践一个例子。

[实践：](#)

```
import java.io.IOException;
import java.io.PrintStream;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
public class GenericsTester {
    public void testTypeSafeMaps(PrintStream out) throws IOException {
        Map<Integer, Integer> squares = new HashMap<Integer, Integer>();

        for (int i=0; i<100; i++) {
            squares.put(i, i*i);
        }
    }
}
```

```

    }
    for (int i=0; i<10; i++) {
        int n = i*3;
        out.println(n + "的平方是" + squares.get(n));
    }
}

//测试安全的链表
public void testTypeSafeLists(PrintStream out) throws IOException {
    List listOfStrings = getListOfStrings();
    for (Iterator i = listOfStrings.iterator(); i.hasNext(); ) {
        String item = (String)i.next();
    }
    List<String> onlyStrings = new LinkedList<String>();
    onlyStrings.add("Legal addition");
    /**
     * Uncomment these two lines for an error
     onlyStrings.add(new StringBuilder("Illegal Addition"));
     onlyStrings.add(25);
    */
}

public void testTypeSafeIterators(PrintStream out) throws IOException
{
    //初始化迭代
    List<String> listOfStrings = new LinkedList<String>();
    listOfStrings.add("Happy");
    listOfStrings.add("Birthday");
    listOfStrings.add("To");
    listOfStrings.add("You");

    for (Iterator<String> i = listOfStrings.iterator(); i.hasNext(); ) {
        String s = i.next();
        out.println(s);
    }
    printListOfStrings(getListOfStrings(), out);
}

//得到普通链表
private List getList() {
    List list = new LinkedList();
    list.add(3);
    list.add("Blind");
    list.add("Mice");
    return list;
}

//得到安全的链表
private List<String> getListOfStrings() {

```

```

    List<String> list = new LinkedList<String>();
    list.add("Hello");
    list.add("World");
    list.add("How");
    list.add("Are");
    list.add("You?");
    return list;
}

public void testTypeSafeReturnValues(PrintStream out) throws
IOException {
    List<String> strings = getListOfStrings();
    for (String s : strings) {
        out.println(s);
    }
} //接受参数化类型的链表

private void printListOfStrings(List<String> list, PrintStream out)
throws IOException {
    for (Iterator<String> i = list.iterator(); i.hasNext(); ) {
        out.println(i.next());
    }
}

public void printList(List<?> list, PrintStream out) throws IOException
{
    for (Iterator<?> i = list.iterator(); i.hasNext(); ) {
        out.println(i.next().toString());
    }
}

public static void main(String[] args) {
    GenericsTester tester = new GenericsTester();
    try {
        tester.testTypeSafeLists(System.out);
        tester.testTypeSafeMaps(System.out);
        tester.testTypeSafeIterators(System.out);
        tester.testTypeSafeReturnValues(System.out);

        List<Integer> ints = new LinkedList<Integer>();
        ints.add(1);
        ints.add(2);
        ints.add(3);
        tester.printList(ints, System.out);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```


在 `List` 和 `Map` 类的后面有个 `<>` 的参数，这个参数表示集合类里面的元素类型。`List<String>` 就是表示，`List` 里面的元素都是 `String` 类型的。这样我们就可以在事先确定 `List` 的类型了，省着大家一起做项目的时候。只知道 `List` 里面有什么还得问写的那个人，你同事写完了还得给你讲那里面是什么。这回你看着方法名就全知道了。关于泛型的概念还有很多我们不能一下子讲完，以后的教程我们会详细讲解。

java 初学者实践教程 12—面向对象

在[第4节课](#)中我们初步的了解到了一些面向对象的概念，和一些特性。例如封装。这节课我们来进一步认识一下，这种思想。对了，在这里我说这是一种思想，是想问题的方法。即使没有java, [C++](#), smalltalk这样的面向对象语言也有这种思想。其实这种思想在上世纪 60 年代就已经存在了。还有一个常见的误区，不光是初学者，就是有些干了几年的程序员也是这样，以为用java写东西就是面向对象，这都是错误的。

那么面向对象到底是什么东西呢？能够明确的给出概念非常少，但是我们可以分析一下。它与面向过程的思想做一个比较，面向过程是指，我们考虑问题时，以一个具体的流程（事务过程）为单位，考虑它的实现过程；面向对象是指，我们考虑问题时，把任何东西看做是对象，以对象为单位，考虑它的属性及方法。

好比一个木匠在做一把凳子，如果他是面向过程的木匠，他会想到制作凳子的过程。“先做什么呢？凳子腿？凳子板？用什么工具呢？”。如果他是一个面向对象的木匠，他会把所有的东西看做成对象，“凳子腿，凳子板两个对象。凳子腿有属性，长方体的，长度，宽度是多少厘米，有方法钉钉子。凳子板的属性，正方形，边长是多少厘米等等问题。”这样的话，面向对象的木匠会依据这些条件。将一个凳子组装在一起。最终目的是做成一个凳子，用什么思想方法去做，是值得研究的。

通过刚才的例子，我们会有一种感觉，面向对象的木匠会对事务量化的分析，用“数学”的方法处理问题似的。好像他更具有进步意义。面向对象的思想也确实有着他的先进之处，它把世界上的所有事务看做成为对象，这样的话更贴近于现实世界，这样的话使得逻辑清晰，谁看报告的时候也喜欢看条理清晰的报告啊。这样使得面向对象的软件开发，成为上世纪 90 年代直到现在的主流开发技术。

传统开发方法存在问题

1.软件重用性差

重用性是指同一事物不经修改或稍加修改就可多次重复使用的性质。软件重用性是软件工程追求的目标之一。谁愿意来来回回的写一件事情呢。

2.软件可维护性差

软件工程强调软件的可维护性，强调文档资料的重要性，规定最终的软件产品应该由完整、一致的配置成分组成。在软件开发过程中，始终强调软件的可读性、可修改性和可测试性是软件的重要的质量指标。实践证明，用传统方法开发出来的软件，维护时其费用和成本仍然很高，其原因是可修改性差，维护困难，导致可维护性差。

3.开发出的软件不能满足用户需要

用传统的结构化方法开发大型软件系统涉及各种不同领域的知识,在开发需求模糊或需求动态变化的系统时,所开发出的软件系统往往不能真正满足用户的需要。

现在的面向对象的思想已经扩展到很多方面,如数据库系统、交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD 技术、人工智能等领域。而且他指的是面向对象分析(OOA),面向对象设计(OOD),面向对象编程(OOP),这一套过程了。

下面我们来看一下,经常用到的重要概念,也就是java语言的的OOP特性,这是对于OOP而言的,不含前面的OOA和OOD的。因为是初学嘛,还没有学到怎么分析和设计呢。Java的OOP有三大特性:封装、继承、多态。

封装的概念已经在第4节课说过了,我们讲的是,它用权限修饰符private使得属性不能被外界访问,像是人的五脏六腑怎么能让人随意的碰呢?人的这个属性也是要封装的。如有再不明白,请访问我们的[技术论坛](#)。

说一下容易理解的继承:

当一个类是另一个类的特例时,这两个类之间具有父子类的关系。子类继承了父类的方法和属性,就是说子类可以重用父类中的这部分代码。比如:轿车是车的一个特例。轿车是车的子类。就是说,轿车继承了车的一切特性。继承用关键字 extends 表示。

实践:

```
//这是基类
public class Che {
    private int wheel = 4;
    public int getWheel() {
        return wheel;
    }
}

public class Jiaochex extends Che {
    private String pinpai = "桑塔纳";
    public String getPinpai() {
        return pinpai;
    }
}

public class Testche {
    public static void main(String[] args) {
        Jiaochex car = new Jiaochex();
        int wheels = car.getWheel(); //调用基类的方法
        String Pinpai = car.getPinpai(); //调用本身的方法
        System.out.println("车有 "+wheels+" 个轮子");
    }
}
```

```
        System.out.println("轿车的品牌是 "+Pinpal);  
    } }
```

注意：java语言与C++不同只能从一个父类继承哦（单继承）。

还有就是最难理解的多态了，我们下次课讲面向对象的多态性。

java 初学者实践教程 13—面向对象之多态

上节课我们了解了比较重要的概念面向对象，和 java 的 OOP 有封装、继承、多态的特征。但是什么叫做多态，是很多初学者不容易理解的问题。对于继承来说，很容易理解因为你就看字面的意思就知道它是继承着父类的特性。多态字面不容易理解了。下面我们具体讲一下吧！

类之间的继承关系使子类具有父类的所有变量和方法，=> 父类所具有的方法也可以在它所有子类中使用，发给父类的消息也可以发送给子类 => 子类的对象也是父类的对象=>子类的对象既可以做本身的类型，也可以做父类的类型。呵呵，上述推导公式好像绕口令似的。我们举个例子理解上述概念。举例：

```
public class 动物 //动物是父类
```

```
public class 猫 extends 动物 //猫是子类
```

动物的所有特性在猫中可以使用，发给动物的信息猫也能收到=>猫的对象 `new 猫()`；既可以作为本身的类型 `猫 a=new 猫()`；也可以作为父类的类型 `动物 b=new 猫()`；这样说理解了吗？如有疑问请访问我们的技术论坛。

如果大家明白了的话，我们就可以从上述公式推导出结论，所有的子类都可以作为父类的类型（同一种类型）来对待。像刚才那个动物有很多子类啊，可以有很多对象。`动物 a=new 猫()`；`动物 b=new 狗()`；`动物 c=new 猪()`；。这样的将子类型的对象引用转换成父类型的对象引用，叫做上溯造型（upcasting）。

我们再来引伸一下，我们在数组那节课里讲了，数组存放的元素是相同类型的数据，但是上溯造型使得 java 允许创建不同类型对象的数组。例如：

```
Employee[] staff = new Employee[3];

staff[0] = new Manager ();

staff[1] = new Secretary ();

staff[2] = new Employee ();
```

夷？这是怎么回事啊，数组里面不是相同类型吗？对啊，因为 `Secretary` 和 `Manager` 是 `Employee` 的子类，所以也可以通过上溯造型变成 `Employee` 啊。以前我们还学到了所有对象都是从 `java.lang.Object` 继承下来的。如果数组要是

Object 型的话 `Object[] obj=new Object[]`; 那就是里面放什么对象都行了。因为什么对象都可以是 Object 型的。

实践：

```
// java 中的多态
class Shape {
    void draw() {}
    void erase() {}
}

//圆形
class Circle extends Shape {
    void draw() {
        System.out.println("Circle.draw()");
    }
    void erase() {
        System.out.println("Circle.erase()");
    }
}

//正方形
class Square extends Shape {
    void draw() {
        System.out.println("Square.draw()");
    }
    void erase() {
        System.out.println("Square.erase()");
    }
}

//三角形
class Triangle extends Shape {
    void draw() {
        System.out.println("Triangle.draw()");
    }
    void erase() {
        System.out.println("Triangle.erase()");
    }
}

public class Shapes {
    public static Shape randShape() {
        switch((int)(Math.random() * 3)) {
            default:
            case 0: return new Circle();
            case 1: return new Square();
            case 2: return new Triangle();
        }
    }
}
```

```
    }  
}  
public static void main(String[] args) {  
    Shape[] s = new Shape[9];  
    // 向数组里添加类型  
    for(int i = 0; i < s.length; i++)  
        s[i] = randShape();  
    // 用多态的方法调用  
    for(int i = 0; i < s.length; i++)  
        s[i].draw();  
}  
}
```

[Java](#)的多态性，有什么意义呢？它的突出优点是使程序具有良好的扩展性。它通过继承，可以派生出任意多个新类型，或向基类增加更多方法时，无须修改原有对基础类进行处理的相关程序。就是扩展性好。

我们返回再看面向对象（专指 OOP）的这三个特性封装、继承、多态三者的关系。没有封装就没有继承，没有继承就没有多态。

java 初学者实践教程 14—垃圾收集器

用过C++编程的人知道，编的时候总是要跟踪所创建的对象，并且需要显示地删除不用的对象。这种方式太麻烦了，容易出错。写了那么多代码，能记住吗，要是把有用的给删了怎么办，要是没用的忘删了怎么办？这些问题是很严重的。在java语言中采用的垃圾收集器这种方式管理内存，就很方便也很安全了。垃圾收集器，可以自动确定哪个对象不再被利用，它可以自动将它删除。这也是java语言的一大优势。

我们要想显示的删除一个对象的引用也很简单，将该引用的变量赋值为 `null` 不就行了吗？对于垃圾收集器来说，当程序员创建对象时，垃圾收集器就开始监控这个对象的地址、大小以及使用情况。通常，垃圾收集器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“可用的”，哪些对象是“不可用的”。当垃圾收集器确定一些对象为“不可用”时，垃圾收集器就回收这些内存空间。

可是垃圾收集器却以较低的优先级在系统空闲周期中执行，通俗一点说就是它级别低，别人不运行时候才轮到它，因此垃圾收集器的速度比较慢。有些时候我们会使用 `System.gc()`。手动回收。这样来提高性能。

对于垃圾收集器来说还有一个值得一提的是 `finalize()` 这个方法，每一个对象都有一个 `finalize()` 方法，这个方法是从 `Object` 类继承来的。它用来回收内存以外的系统资源，就像是文件处理器和网络连接器。该方法的调用顺序和用来调用该方法的对象的创建顺序是无关的。换句话说，书写程序时该方法的顺序和方法的实际调用顺序是不相干的。这只是 `finalize()` 方法的特点。还有，每个对象只能调用 `finalize()` 方法一次。如果在 `finalize()` 方法执行时产生异常（exception），则该对象仍可以被垃圾收集器收集。那是一定了，不能说用到 `finalize()` 了。垃圾收集器就什么也不做了啊。`finalize()` 的工作量是很大的哦

总结：

Java用了垃圾收集器的内存管理方式，并不是说它完全的好。有的时候会影响它的性能，我们还是要手动来收集的。但是要是像C++那样完全手动来收集的话，那也实在是太麻烦了而且不是很安全。

根据垃圾收集器的工作原理，我们可以通过一些技巧和方式，让垃圾收集器运行更加有效率。

1.最基本的建议就是尽早释放无用对象的引用。

大多数程序员在使用临时变量的时候，都是让引用变量在退出活动域（scope）后，自动设置为 `null`。

2.尽量少用finalize函数。finalize函数是[Java](#)提供给程序员一个释放对象或资源的机会。但是，它会加大垃圾收集器的工作量，因此尽量少采用finalize方式回收资源。

3.当程序有一定的等待时间，程序员可以手动执行 `System.gc()`，通知垃圾收集器运行，但是 **Java** 语言规范并不保证垃圾收集器一定会执行。

java 初学者实践教程 15—方法的重载与重写

[Java](#)语言中的概念就是多，这回又有两个概念，重载和重写。这是两个新概念，也是两个令初学者容易混淆的概念。他们的概念截然不同，只不过都有个“重”字，就以为是很像的。

下面解释一下这两个概念：

方法重载(overloading method) 是在一个类里面，方法名字相同，而参数不同。返回类型呢？可以相同也可以不同。

方法重写(overriding method) 子类不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。方法重写又称方法覆盖。如果还是搞混的话，就把“重写覆盖”，这个词多念几遍吧。知道是覆盖的话，就知道是子类覆盖父类的方法了。

实践： 重载的例子

```
public class MethodOverloading {
    void recieve(int i) {
        System.out.println("接收一个 int 数据");
        System.out.println("i="+i);
    }
    void recieve(float f) {
        System.out.println("接受一个 float 型的数据");
        System.out.println("f="+f);
    }
    void recieve(String s) {
        System.out.println("接受一个 String 型数据");
        System.out.println("s="+s);
    }
    public static void main(String[] args){
        MethodOverloading m = new
        MethodOverloading();
        m.recieve(3456);
        m.recieve(34.56);
        m.recieve("百家拳软件项目研究室");
    }
}
```

大家看到了上面的例子方法 `receive()` 有三个，名字相同参数不同。这样的话，在 `main()` 调用的时候，参数用起来就很方便了。重写的例子似乎不用举了，记不住的话，就和“覆盖”。一起念。

有时候，重载和重写的方式有些复杂，在 `jdk5` 里面。有一些方式能简化一些。我们来看看，`jdk5` 的可变参数。如果把相同参数类型的方法重载好几遍真的是很烦。就一个方法，`pri(String args)`, `pri(String arg0 ,String arg1)`, `pri(String arg0,String arg1,String arg2)`, `pri(String arg0,String arg1,String arg2,String arg3)`。这样的话会写很多烦琐的代码。现在 `jdk5` 可以，用“...”来代替这些参数。

实践：

```
public class overload {
    //若干个相同类型的参数，用“...”代替
    public void pri(String... strings ){
        for (String str : strings)    //for 这个循环语句也有迭代的意思
            System.out.print(str);
    }
    public static void main(String[] args){
        new overload().pri("100jq"," 百家拳软件项目研究室","
www.100jq.com");
    }
}
```

`jdk5` 的方法重写，比以前多了一个叫做协变返回的概念。在以往 `jdk` 的版本中，还有一个比较让人讨厌的地方。方法重写确实是比较不错的机制，如果想用父类的方法，写个 `super` 就可以了，如果不想用父类的方法就重写覆盖。但是重写覆盖的返回类型不能覆盖，父类的类型不够用怎么办，我们想在子类重写它的类型可以吗？现在可以了。

看下面的例子：

```
class Point2D { //定义二维的点
    protected int x, y;
    public Point2D() {
        this.x=0;
        this.y=0;}
    public Point2D(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
//定义三维的点，继承二维
class Point3D extends Point2D {
    protected int z;
```

```

    public Point3D(int x, int y) {
        this(x, y, 0);
    }
    public Point3D(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}
//定义二维的位置
class Position2D {
    Point2D location;
    public Position2D() {
        this.location = new Point2D();
    }
    public Position2D(int x, int y) {
        this.location = new Point2D(x, y);
    }
    public Point2D getLocation() {
        return location;
    }
}
//定义三维的位置，继承二维的位置
class Position3D extends Position2D {
    Point3D location; //在这里已经变成 Point3D 的类型了
    public Position3D(int x, int y, int z) {
        this.location = new Point3D(x, y, z);
    }
    @Override //注释是重写方法
    public Point3D getLocation() {
        return location; //返回是子类的类型而不是原来的类型了
    }
}
}

```

java 初学者实践教程 16—static 关键字

这已经是本系列教程的第 16 次课了，在我们前几次课的基础上，我们继续学习一下 java 的语法。这回我们，讲 static 关键字。

static 关键字可以用来修饰类的变量，方法和内部类。static 是静态的意思，也是全局的意思它定义的东西，属于全局与类相关，不与具体实例相关。就是说它调用的时候，只是 `ClassName.method()`，而不是 `new ClassName().method()`。`new ClassName()` 不就是一个对象了吗？static 的变量和方法不可以这样调用的。它不与具体的实例有关。

实践：

```
class Count {
    private int serialNumber;
    public static int counter = 0; //一个静态变量 counter
    public Count() {
        counter++; //创建 Counter 的时候递增
        serialNumber = counter;}
    public int getSerialNumber(){
        return serialNumber;
    }
}
class OtherClass {
    public int increment(){
        return Count.counter++;//静态的变量不属于任何实例只能直接用类调用
    }
}
public class TestStaticVar {
    public static void main(String[] args){
        Count[] cc = new Count[10];
        OtherClass o = new OtherClass();
        for (int i=0;i<cc.length;i++){
            cc[i] = new Count();
            System.out.println("cc["+i+"].serialNumber = "+cc[i].getSerialNumber());
            System.out.println(o.increment());
        }
    }
}
```

查看结果

类的方法中带有 **static** 关键字，这个方法就是静态方法。静态方法也是要通过类名，而不是实例访问。

实践：

```
class GenerealFunction {  
    public static int add(int x,int y){  
        return x+y;  
    }  
}  
public class UseGeneral {  
    public static void main(String[] args){  
        //调用时还是用类直接调用  
        int c = GenerealFunction.add(19,18);  
        System.out.println("结果是"+c);  
    }  
}
```

注意：子类不能重写父类的静态方法哦，也不能把父类不是静态的重写成静态的方法。想隐藏父类的静态方法的话，在子类中声明和父类相同的方法就行了。

前一阵子有同学问了，main()是什么意思啊？main()的前面不是也有一个static吗，它也是静态方法。它是程序的入口点，就是说java的程序是由java虚拟机执行的，java语言和虚拟机的入口就是main()。因为它是static的，这可以使JVM不创建实例对象就可以运行该方法。因此我们在main()中调用别的类的非静态方法，就要创建实例。像上面的例子：OtherClass o = new OtherClass();

System.out.println(o.increment()); 不用实例去调用是不行的。

前面我们已经见到很多这样的例子了。

大家看一个错误的例子：

```
int x;
public static void x() {
    x = 15; //这个是错误的，x 是非静态变量
}
static 还可以修饰程序块 用{}括起来，用法与上述两种方法相同
public class StaticInit {
    public static int count = 1;
    static {
        count = Integer.getInteger("myApplication.counter").intValue();
    }
}
```

java 初学者实践教程 17—final 关键字

上一节学了 `static` 关键字，这一节接着学习 `final` 关键字。`final` 关键字有三个东西可以修饰的。修饰类，方法，变量。

详细解释一下：

1、在类的声明中使用 `final`

使用了 `final` 的类不能再派生子类，就是说不可以被继承了。有些 java 的面试题里面，问 `String` 可不可以被继承。答案是不可以，因为 `java.lang.String` 是一个 `final` 类。这可以保证 `String` 对象方法的调用确实运行的是 `String` 类的方法，而不是经其子类重写后的方法。

2、在方法声明中使用 `final`

被定义为 `final` 的方法不能被重写了，如果定义类为 `final` 的话，是所有的都不能重写。而我们只需要类中的某几个方法，不可以被重写，就在方法前加 `final` 了。而且定义为 `final` 的方法执行效率要高的啊。

3、在变量声明中使用 `final`

这样的变量就是常量了，在程序中这样的变量不可以被修改的。修改的话编译器会抱错的。而且执行效率也是比普通的变量要高。`final` 的变量如果没有赋予初值的话，其他方法就必需给他赋值，但只能赋值一次。

总结：

这个关键字并不是很难理解，`final` 的英文意思是“最终的”。他修饰了什么东西都是最终的。不可以改变的。效率也比较高。通常在 java 的优化编程中往往会提及到这一点。

java 初学者实践教程 18—抽象类和接口

[Java](#)语言中允许有一种叫做抽象方法的东西，他只是一个名字没有具体的实现。像是这样：`public abstract void abc ()`； 使用了`abstract`关键字，结尾用“；”结束。与前几节我们用的方法都是具体方法，是有实现的。哪怕方法体中什么也不写`public void abc () {}`也是具体方法。概念：包含一个或多个抽象方法的类称为抽象类。抽象类也必须声明`abstract`关键字。抽象类的使用有着一些限制，不能创建抽象类的实例。如果子类实现了抽象方法，则可以创建该子类的实例对象。要是子类也不实现的话，这个子类也是抽象类，也不能创建实例。

接口是什么东西呢？接口是比抽象类更抽象的类。举例：`public interface Name {}`接口里面的方法全都是抽象的，里面的变量全都是 `final` 的常量，而且实现接口的类必须将所有的抽象方法全部实现。抽象类里也可以有具体的方法。所以说，接口是最抽象的，其次是抽象类，而具体类本身就是对现实世界的抽象。软件开发本身就是将现实世界抽象成计算机世界。

因为抽象类和接口比具体类抽象，所以使用时他们总是被继承而被实现的。不过继承他们的类不只是一个，有很多类实现他们的抽象方法。一个方法有多种实现方式，这里用到了 OOP 中的多态性。这使得设计变得非常清晰。因为基类是抽象类或是接口做一个描述，底下继承的类有若干个，我们只需要对接口或抽象类操作，也用不着管有多少个实现。如果是多人共同开发的项目的话，是非常有意义的。你自己写个东西，怎么实现的也不用告诉别人，别人看个接口就够了。

接口的实现用关键字 `implement` 而不是 `extends`.如果用了 `extends` 的那就是继承这个接口。那么那个子类也是接口，是原来的子接口。举个接口的例子吧：

实践：

```
//声明一个接口
public interface Say {
    public void sayMessage();
}
//两个实现类
public class SayHello implements Say {
    public void sayMessage() {
        System.out.println("hello");
    }
}
public class SayHi implements Say {
    public void sayMessage() {
        System.out.println("Hi");
    }
}
```

```
    }}  
    //这是一个测试类  
    public class TestSay {  
        public static void main(String[] args) {  
            //同样都是用 Say 这个接口类型实例，却可以输出两个结果  
            Say say = new SayHello();  
            say.sayMessage();  
            Say say1 = new SayHi();  
            say1.sayMessage();  
        }  
    }}
```

接口还有一个重要的作用，我们在[面向对象](#)那节课里提过一个概念，java语言中只有单继承，就是说只能从一个父类继承。单继承的好处是，一旦继承的太多了，改了一个类子类就都变了。牵一发，而动全身。那么如果我们想继承多个父类的特性怎么办呢？就用接口吧，这个类可以先继承一个类，再去实现其它的接口，接口里面都是抽象方法，不会造成，牵一发，而动全身的效应。改变多继承的特性，也是对[C++](#)语言的一项改进

业界有一种说法说，与其说java是面向对象编程，还不如说它是面向接口编程。强调的方面是接口的抽象描述性。它也是对[C++](#)的一种改进，C++里面没有接口。所以说java语言适合多人团队合作的大项目，看一个接口就可以了，后面怎么实现的可以不管。

java 初学者实践教程 19—访问控制和内部类

今天我们来学习一下 java 语言的基本语法，这节课我们讲访问控制和内部类。

访问控制这种语法在前面的学习中，已经经常见过了。像是 `public`（公有的），`private`（私有的）。大家按照字面理解就能知道了，公有的就是谁都可以用，私有的就是只有自己的类内部可以用。不过访问控制一共有 4 个，`public`（公有的），`protected`（受保护的），`default`（默认的，就是没有修饰符），`private`（私有的）。我们用一个图表示他们的权限：

高级访问控制				
修饰符	同类	同包	子类	通用性
公共	是	是	是	是
受保护	是	是	是	
缺省	是	是		
私有	是			

权限修饰符，可以修饰类，方法和属性。而表达的意思与上表一致。有些注意的地方：类的权限不可以用 `private`，你们想如果写一个类是 `private`，谁也不让用写它干嘛；抽象方法的权限不可以是 `private` 和 `default`。因为抽象方法一定要子类来实现的，子类都不可以用，抽象方法当然也没有意义了。

内部类是我们以前没有提过的概念，就是在类中又写了一个类。注意内部类是可以使用 `private` 权限的，而且还可以是 `static` 的呢。内部类可以正常调用其它类的方法，属性。别人也正常调用它。它的使用和普通的方法，属性一致，我们就把它看做一个普通的方法就行了。不过它可是可以创建对象的哦。

我们把[测试](#)内部类的源码给大家，不过有一个地方大家要注意。

```
public class TestInnerStatic {  
    /*只有声明成 static 的内部类，才可以是里面的成员声明成 static。否则  
    错误  
        如果，声明成 static 的类，不就是全局的了吗？它就相当放在外面了  
        已经不再是内部类了，并且它的对象中将不包含指向外包类对象的  
        指针，所以不能再引用外包类了*/  
    public static class InnerClass {  
        public static int classVar = 0;  
    }  
}
```

```
        public static void doSomething() {  
  
    System.out.println("TestInnerStatic.InnerClass.doSomething");  
        }  
    }  
  
    public static void main(String[] args) {  
        InnerClass.doSomething();  
    }  
}
```

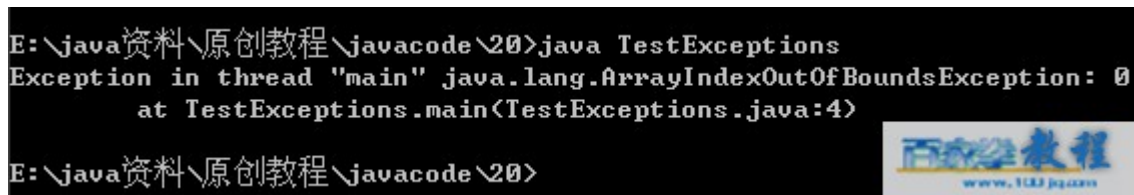
java 初学者实践教程 20—异常处理

异常处理是个很重要的概念，很多语言中都对异常处理下了很大的功夫。如果你的语法没有写错，编译器是不会报错，而且编译成功。如果编译成功后，运行时发生了错误该怎么处理呢？例如我要加载一个类，而这个类被删了。这种情况就是异常。我们采用 try.....catch.....finally 语句作为处理方式。举个异常处理的例子吧。

实践：

```
public class TestExceptions {  
    public static void main(String[] args) {  
        for ( int i = 0; true; i++ ) {  
            System.out.println("args[" + i + "] is '" + args[i] + "'");  
        }  
    }  
}
```

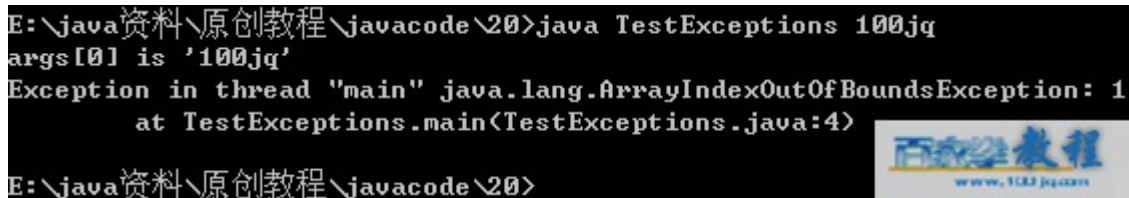
在这里面 main 方法的参数 args 是个字符串型的数组，在执行的时候要输入 java TestExceptions 100jq 后面的就是参数 args[0]就是第一个参数。我们输入 java TestException 是出现了错误。如图 20-1 所示，



```
E:\java资料\原创教程\javacode\20>java TestExceptions  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
    at TestExceptions.main<TestExceptions.java:4>  
E:\java资料\原创教程\javacode\20>
```

这上说的是数组边界溢出异常，第 0 个产生错误，因为根本就没有 args[0]，这个元素。

我们再敲一下 java TestExceptions 100jq 如图 20—2 所示，



```
E:\java资料\原创教程\javacode\20>java TestExceptions 100jq  
args[0] is '100jq'  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1  
    at TestExceptions.main<TestExceptions.java:4>  
E:\java资料\原创教程\javacode\20>
```

输出了args[0]没有异常了，并且输出了。而循环到i=1 时，又发生异常。我们再输入两个参数java TestExceptions chinaitlab www.chinaitlab.com 这回两个参数了。同样的道理，args[2]发生异常。

那么我们如何来捕捉这个异常呢，我们对上述代码做一下简单的修改。

实践：

```
public class TestExceptions1 {  
    public static void main(String[] args) {  
        try {  
            for ( int i = 0; true; i++ ) {  
                System.out.println("args[" + i + "] is " + args[i] + "");  
            }  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("异常捕捉: " + e);  
            System.out.println("退出...");  
        }  
    }  
}
```

这回输入刚才那两个参数的话，就不会出现那一堆难懂的英文了。异常已经在我们的掌控之中。否则，有很多异常是足够使内存导毁的。

这里面我们只使用了 try...catch 哪个地方你觉得它有毛病，你就 try 哪。但是 try 然后，要 catch（捕捉）的。如果事先你想不出它会发生什么异常的话，就用 finally.

实践：

```
class FinallyDemo {  
    static void procA() {  
        try {  
            System.out.println("inside procA");  
            throw new RuntimeException("demo");  
        } finally {  
            System.out.println("procA's finally");  
        }  
    }  
    // 从 try 程序块内返回  
    static void procB() {  
        try {  
            System.out.println("inside procB");  
            return;  
        } finally { //结束  
            System.out.println("procB's finally");  
        }  
    }  
    // 执行一个 try 程序块  
    static void procC() {  
        try {  
            System.out.println("inside procC");  
        }  
    }  
}
```

```

    } finally {
        System.out.println("procC's finally");
    }
}

public static void main(String args[]) {
    try {
        procA();
    } catch (Exception e) {
        System.out.println("异常捕捉");
    }
    procB();
    procC();
}
}

```

如果将方法里抛出异常抛出，使用 throws 关键字 `public void abc () throws exception` 也是要用 catch 来捕捉的。

实践：

```

class ThrowDemo {
    static void demoproc() {
        try {
            throw new NullPointerException("demo");
        } catch (NullPointerException e) {
            System.out.println("Caught inside demoproc.");
            throw e; //重新抛出异常
        }
    }
    public static void main(String args[]) {
        try {
            demoproc();
        } catch (NullPointerException e) {
            System.out.println("Recaught: " + e);
        }
    }
}

```

异常类除了 jdk 提供我们的那些之外，我们自己还可以自定义的。Jdk 提供的刚才我们已经见过几个了 `ArrayIndexOutOfBoundsException`（数组边界溢出），`NullPointerException`（空指针异常）。要是 jdk 没有的，我们只有自己定义了。比如说我们现在要用 XML 开发，那么 jdk 没有写这方面的异常类，我们就得自己写一个关于 XML 的异常了。我们下节课讲自定义异常类。

java 初学者实践教程 21—自定义异常类

上节课留下了一个概念，自定义异常类。为什么要自己编写异常类，上节课做了简要的说明。如果 jdk 里面没有提供的异常，我们就要自己写。我们常用的类 `ArithmeticException`, `NullPointerException`, `NegativeArraySizeException`, `ArrayIndexOutOfBoundsException`, `SecurityException` 这些类，都是继承着 `RuntimeException` 这个父类,而这个父类还有一个父类是 `Exception`。那么我们自己写异常类的时候，也是继承 `Exception` 这个类的。

实践：

```
class MyException extends Exception { //继承了 Exception 这个父类
    private int detail;
    MyException(int a) {
        detail = a;}
    public String toString() {
        return "MyException[" + detail + "];"
    }
}

class ExceptionDemo {
    static void compute(int a) throws MyException {
        System.out.println("调用 compute(" + a + ")");
        if(a > 10)
            throw new MyException(a);
        System.out.println("常规退出 ");
    }
    public static void main(String args[]) {
        try {
            compute(1);
            compute(20);
        } catch (MyException e) {
            System.out.println("捕捉 " + e); //这样就可以用自己定义类来捕捉异常了
        }
    }
}
```

像是上节课我们说了，如果你开发程序用到好多组件，或其它厂商的东西。那么出现的异常会是莫名其妙的，这样的话会给调试带来很大的不便。往往在开发的过程中会写很多自定义的异常类。

总结：

异常处理机制是保证java程序正常运行、具有较高[安全](#)性的重要手段。对于开发良好的编程习惯是非常重要的。

java 初学者实践教程 22—输入/输出

输入/输出 (I/O) 是每一项计算机语言，必须有的东西。不让人输入数据的话，计算机怎么处理数据呢？在 java 语言中，I/O 的方式是流的方式。流 (stream) 这是个学习 java 输入输出的最基本的概念。流是字节从源到目的的有序序列。一方面是字节，一方面是有序的。流描述的是一个过程，顺序严格。一个需要键盘输入的程序可以用流来做到这一点。两种基本的流是：输入流和输出流。你可以从输入流读，但你不能对它写。要从输入流读取字节，必须有一个与这个流相关联的字符源。这些东西都放在 java.io.* 这个包里了。io 是 java 的第一大包。在 java.io 包中，有一些流是结点流，即它们可以从一个特定的地方读写，例如磁盘或者一块内存。其他流称作过滤流。一个过滤器输入流是用一个到已存在的输入流的连接创建的。此后，当你试图从过滤输入流对象读时，它向你提供来自另一个输入流对象的字符。

常见的几种流：

- 字节流：传字节的。以 8 位字节为单位进行读写，以 InputStream 与 OutputStream 为基础类
- 字符流：传字符的。以 16 位字符为单位进行读写，以 Reader 与 Writer 为基础类
- 文件流：传文件的。属于节点流，对文件读写，传输。里面的类很多。
- 序列化：传对象的。一个对象怎么读啊，只有变成二进制才可以读，这就是序列化。

实践：

```
//这是一个字节流的例子，以 InputStream 与 OutputStream 为基础类
import java.io.*;
class ByteArrayOutputStreamDemo {
    public static void main(String args[]) throws IOException {
        ByteArrayOutputStream f = new ByteArrayOutputStream();
        String s = "This should end up in the array";
        byte buf[] = s.getBytes();
        f.write(buf);
        System.out.println("Buffer as a string");
        System.out.println(f.toString());
        System.out.println("Into array");
        byte b[] = f.toByteArray();

        for (int i=0; i<b.length; i++) {
            System.out.print((char) b[i]);
            System.out.println("\nTo an OutputStream()");
        }
    }
}
```

```

//输出到文件 test.txt 中
OutputStream f2 = new FileOutputStream("test.txt");
f.writeTo(f2);
f2.close();
System.out.println("Doing a reset");
f.reset();
for (int i=0; i<3; i++)
f.write('X');
System.out.println(f.toString());}}
//字符流的例子，以 Reader 与 Writer 为基础类
import java.io.*;
public class CharArrayReaderDemo {
    public static void main(String args[]) throws IOException {

        String tmp = "abcdefghijklmnopqrstuvwxyz";
        int length = tmp.length();
        char c[] = new char[length];
        tmp.getChars(0, length, c, 0);
        CharArrayReader input1 = new CharArrayReader(c);
        CharArrayReader input2 = new CharArrayReader(c, 0, 5);

        int i;
        System.out.println("input1 is:");
        while((i = input1.read()) != -1) {
            System.out.print((char)i);
            System.out.println();
            System.out.println("input2 is:");
            while((i = input2.read()) != -1) {
                System.out.print((char)i);
                System.out.println();
            }
        }
    }
}
//文件流的例子
import java.io.*;
class FileInputStreamDemo {
    public static void main(String args[]) throws Exception {
        int size;
        InputStream f =
        new FileInputStream("FileInputStreamDemo.java");
        System.out.println("Total Available Bytes: " +
            (size = f.available()));
        int n = size/40;
        System.out.println("First " + n +
            " bytes of the file one read() at a time");
        for (int i=0; i < n; i++) {

```

```

        System.out.print((char) f.read());
    }
    System.out.println("\nStill Available: " + f.available());
    System.out.println("Reading the next " + n +
        " with one read(b[])");
    byte b[] = new byte[n];
    if (f.read(b) != n) {
        System.err.println("couldn't read " + n + " bytes.");
    }
    System.out.println(new String(b, 0, n));
    System.out.println("\nStill Available: " + (size = f.available()));
    System.out.println("Skipping half of remaining bytes with
skip()");
    f.skip(size/2);
    System.out.println("Still Available: " + f.available());
    System.out.println("Reading " + n/2 + " into the end of array");
    if (f.read(b, n/2, n/2) != n/2) {
        System.err.println("couldn't read " + n/2 + " bytes.");
    }
    System.out.println(new String(b, 0, b.length));
    System.out.println("\nStill Available: " + f.available());
    f.close();
}
}

```

代码很多如有不明白的地方请访问[技术论坛](#), 还有序列化的例子没有举出, 序列化在java中是个很重要的概念哦。我们下节课。具体举例讲解。

java 初学者实践教程 23—序列化

上节课我们讲了 4 种流，只有序列化的这个没有细讲。它是传对象的，如果想把一个对象保存在硬盘上，就只能使用这种方式。它的关键是将它的状态以一种串行格式表示出来，以便以后读该对象时能够把它读出来。对象的串行化对于大多数 java 应用是非常重要的：

[Java](#)的远程方法调用(RMI)，通过socket通信。这个东西我们会在后面的教程讲到。

对象永久化，就是把对象存硬盘上，或外存设备上。以便以后使用。

它的基础类是 `ObjectInputStream` 和 `ObjectOutputStream`，这两个流称为对象流。

实践：

```
//这是一个保存对象的例子
import java.io.*;
import java.util.Date;
public class SerializeDate {
    SerializeDate() {
        Date d = new Date ();
        try {
            FileOutputStream f =
                new FileOutputStream ("date.ser"); //输出到 date.ser 这个文件
            ObjectOutputStream s =
                new ObjectOutputStream (f);
            s.writeObject (d); //写对象,将对象 d 写成是 date.ser 文件
            s.close (); //关闭流
        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
    public static void main (String args[]) {
        new SerializeDate();
    }
}
```

如图所示 23-1，执行之后

```
E:\java资料\原创教程\javacode\23>javac SerializeDate.java
E:\java资料\原创教程\javacode\23>java SerializeDate
E:\java资料\原创教程\javacode\23>
```




图 23-1

在 DOS 窗口中没有看到结果，但是在这个文件夹内发现了一个 date.ser 的文件。它就是对象 d 写入磁盘的状态。如图 23-2

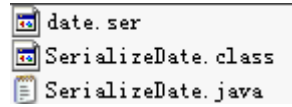


图 23-2

那么保存了之后怎么在把这个 date.ser 文件读出来呢？

实践：

```
import java.io.*;
import java.util.Date;
public class UnSerializeDate {
    UnSerializeDate () {
        Date d = null;
        try { //使用 FileInputStream 类
            FileInputStream f =
                new FileInputStream ("date.ser");
            ObjectInputStream s =
                new ObjectInputStream (f);
            d = (Date) s.readObject ();//读对象
            s.close ();
        } catch (Exception e) {
            e.printStackTrace (); }
        System.out.println(
            "从 date.ser 文件，读取 Date 对象 ");
        System.out.println("日期是: "+d);
    }
    public static void main (String args[]) {
        new UnSerializeDate();
    }
}
```

如图 23-3 所示读出时间

```
E:\java资料\原创教程\javacode\23>javac UnSerializeDate.java
E:\java资料\原创教程\javacode\23>java UnSerializeDate
从date.ser文件，读取Date对象
日期是: Wed Jul 18 10:08:58 CST 2007
E:\java资料\原创教程\javacode\23>
```




图 23-3

对于一个可以被序列化的类，它会实现一个 `Serializable` 的接口。那是个空接口，什么方法也没有只是一个标志而已。这在 J2EE，（现在叫 `java EE`）中，使用 EJB 时是非常重要的。如果大家以后能继续学习学到 EJB 的时候，再具体了解。

java 初学者实践教程 24—反射

还是那样的，java 的概念就是多，有时候多的还没等你反应过来又给你出来一个新的概念。反射是个很重要的概念，这是一种机制，不只是 java 里面有，很多语言里面都有。这个概念是一个叫 Smith 的大师，由 1982 年提出来的。指的是一类应用，它们能够自描述和自控制。这样说太抽象了。我们看个例子，实践：

```
import java.lang.reflect.*;
public class Refl {
    public static void main(String args[]) {
        try {
            //Class.forName() 这是反射的一种方式。将类在运行时自动
            加载进来
            Class c = Class.forName("java.lang.String");
            // getDeclaredMethods()获取这个类中定义了的方法列表
            Method m[] = c.getDeclaredMethods();
            for (int i = 0; i < m.length; i++)
                System.out.println(m[i].toString());
        } catch (Throwable e) {
            System.err.println(e);
        }
    }
}
```

执行的时候发现输出了，String 类的所有方法打印了出来。重要的是，Class.forName 这句话它是反射的一种方式。就是在运行时改变 Refl 类的状态，通过“java.lang.String”改变。

[Java](#)语言提供了一套反射类，java.lang.reflect.*;这些类可以用做：

构造新类实例和新数组

访问并修改对象（Object）和类的字段(Field)

调用对象和类中的方法（Method）

访问并修改数组的元素

反射是一种强大的工具，但也存在一些不足。一个主要的缺点是对性能有影响。使用反射基本上是一种解释操作，我们可以告诉JVM，我们希望做什么并且它满足我们的要求。这类操作总是慢于只直接执行相同的操作。一边执行的时候，一边加载其它类，肯定会慢的。但是它有很强的扩展性，具有开放性的系统很多

都采用这种机制，因为在[安全](#)允许的情况下它可以随意加载类，和调用方法。在 windows 编程里面的 dll 与它几乎是一个意思。

java 初学者实践教程 25—多线程

[Java](#)语言中有一个重要的特性是支持多线程。多线程是java的一项高级技术，它涉及到操作系统里面的知识，层次贴近系统层面。对于普通程序员一般很少碰它。而且目前就是在java EE（原来的J2EE）的相关框架里，对线程这个东西都是尽量回避。程序员最理想的状态是专注业务逻辑，而不是天天想着线程这个东西怎么写。

思考一个问题程序的本质是什么？是 CPU 的指令序列的集合。到底什么顺序是程序员编写的让计算机赋值，它就赋值、写个循环它就循环、写个分支语句它就分支、写个跳转它就跳转。每个指令流就是一个线程，并发执行多个指令流就是多线程。大家想，只有一个 CPU 怎么可能同时发出多个指令流呢？是的，并发只是“逻辑”上的，物理上是不可能的除非是两个以上的 CPU。

多线程和传统的单线程的区别是由于各个线程的控制流彼此独立，使得各个线程之间的代码是乱序执行的，出现了并发访问带来的一切问题。正像是三个和尚的故事，和尚多了未必是好事。也就是刚才说的，程序员一般都不让他们碰这个东西。

在 java 中如何写线程呢，在 java 中就是很简单了。有两种方法：第一、继承 `java.lang.Thread` 第二、实现 `Runnable` 接口。

实践：

```
//继承 Thread 而重写了 run()方法
public class Hello extends Thread{
    int i;
    public void run(){
        while(true){
            System.out.println("Hello "+i++);
            if(i==10) break;
        }
    }
}

public class HelloThread {
    public static void main(String[] args){
        Hello h1 = new Hello();
        Hello h2 = new Hello();
        h1.start(); //用两个线程执行那 10 次循环
        h2.start();
    }
}

//上面的例子是第一种方法，下面是第二种方法
public class TestThread {
    public static void main(String args[]) {
        XYZ r = new XYZ();
    }
}
```

```

    XYZ r1 = new XYZ();
    Thread t1 = new Thread(r);
    Thread t2 = new Thread(r1);
    t1.start();//用两个线程执行那 50 次循环
    t2.start();
} } //实现 Runnable 接口
class XYZ implements Runnable {
    int i;
    public void run() {
        i = 0;
        while (true) {
            System.out.println("Hello " + i++);
            if ( i == 50 ) {
                break;
            }
        }
    }
}
}
}
}
}

```

上面两种方法继承 Thread 类，是比较简单的，代码也比较少。但是我们不提倡使用这种方法。而第二种实现 Runnable 接口，更符合面向对象思想，Thread 是把虚拟的 CPU 看成一个对象，封装了 CPU 的细节。但是 Thread 的构造线程的子类的方法中与 CPU 不相关，没有必要把 CPU 的细节都继承来。而实现 Runnable 则不影响 java.lang.Thread 的体系。而且便于其它类的继承。

线程并发的代码和数据的执行顺序混乱，我们也需要自己调度和控制它们。请看附加教程，线程调度和并发。

java 初学者实践教程 26—网络程序

[Java](#)在网络编程这个地方做的很好，java的主要目的也是为了网络而生的，它能方便的访问网络上的资源。我们这节课来介绍网络通讯的两种机制：URL通信机制，Socket通信机制。

URL表示了Internet上一个资源的引用或地址。[Java](#)网络应用程序也是使用URL来定位要访问的Internet的资源。在jdk里面java.net.URL也是一个类，它来封装URL的一些细节。目前大家可以把URL理解为网址，default.aspx 这就是个URL.http是[协议](#)名（超文本传输[协议](#)）用“://”隔开[www.chinaitlab.com](#) 是主机名。Default.aspx是文件名。它的端口号没有写，默认是 80。

实践：

```
import java.net.*;
public class ParseURL {
    public static void main(String[] args) throws
    MalformedURLException{
        URL url = new URL("http://www.100jq.com:45175/default.aspx");
        System.out.println("协议是 "+url.getProtocol());
        System.out.println("主机是 "+url.getHost());
        System.out.println("文件名是 "+url.getFile());
        System.out.println("端口号是 "+url.getPort());
    }
}
/*
URL 这个对象中提供了很多方法像是
getProtocol()
getHost()
getFile()
getPort()
*/
```

我们可以通过 URL 对文件或资源读取，也可以通过 URLConnection 读取，也可以通过这个写入数据限于 cgi 脚本。

实践：

```
import java.net.*;
import java.io.*;
public class URLConnectionReader {
    public static void main(String[] args) throws IOException {
        URL google = new URL("");
    }
}
```

```

        URLConnection g = google.openConnection();
        BufferedReader in = new BufferedReader(new
InputStreamReader(g.getInputStream()));
        String inputLine;
        while ((inputLine=in.readLine())!=null)
            System.out.println(inputLine);
            in.close();
    }
}

```

URL 和 URLConnection 类提供了较高层次的网络访问。有时候需要进行较低层次的访问。编写 C/S 模型的程序时，就要使用 Socket 通信机制了。因为在网络上不一定非得访问文件。

实践:

```

//先写个客户端的应用
import java.net.*;
import java.io.*;
public class SimpleClient {
    public static void main(String args[]) {
        try {
            // 在 5432 端口打开服务器连接
            // 在这里用 localhost 与 127.0.0.1 是一个意思
            Socket s1 = new Socket("127.0.0.1", 5432);
            // 对这个端口连接一个 reader,注意端口不能够占用别的
            BufferedReader br = new BufferedReader(
                new InputStreamReader(s1.getInputStream()));
            // 读取输入的数据并且打印在屏幕上
            System.out.println(br.readLine());
            //当完成时关闭流和连接
            br.close();
            s1.close();
        } catch (ConnectException connExc) {
            System.err.println("Could not connect to the server.");
        } catch (IOException e) {
            // ignore
        }
    }
}
//这是服务端的应用
import java.net.*;
import java.io.*;
public class SimpleServer {
    public static void main(String args[]) {
        ServerSocket s = null;
        // 注册服务端口为 5432
    }
}

```

```

try {
    s = new ServerSocket(5432);
} catch (IOException e) {
    e.printStackTrace();
}
// 运行监听器并接收，永远循环下去。因为服务器总要开启的
while (true) {
    try {
        // 等待一个连接的请求
        Socket s1 = s.accept();
        // 得到端口的输出流
        OutputStream s1out = s1.getOutputStream();
        BufferedWriter bw = new BufferedWriter(
            new OutputStreamWriter(s1out));
        // 发送一个字符串
        bw.write("百家拳软件项目研究室欢迎您!\n");
        // 关闭这个连接，但不是服务端的 socket
        bw.close();
        s1.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}
}

```

执行这个程序和其它的不太一样，先用 javac 将两个文件编译之后。然后敲 start 开启另一个窗口。用 start 命令开启的窗口继承了原来窗口的特性。如图 26-1 所示



图 26-1

接着在原来的窗口上执行服务端程序 java SimpleServer. 在新窗口中执行 java SimpleClient 就会看到结果了。注意如果如果在启动服务端的时候抛出 bindException 则说明 5432 这个端口已经被别的程序占用着，改成别的端口号就可以了。通常选用端口的时候，其数字最好不要小于 1024，1024 一下的端口很多都是专用的端口。

java 初学者实践教程 27—applet

现在的java界，很多东西叫××let，××let的意思都是些小程序的意思。例如：applet应用程序的小程序，servlet[服务器](#)端的小程序，midlet手机中的小程序，portlet门户容器端的小程序。这节我们介绍applet.这个东西用的不是很多，但是在java的体系结构中是很有意义的。这个东西是能够在浏览器里运行的，可以潜入到HTML页面里。我们知道普通的Application要有main（）作为入口点运行，而Applet要在浏览器里运行，或者开发时查看的时候用appletviewer运行。举个例子，实践：

```
import java.awt.*;
import java.applet.*;
@SuppressWarnings("serial") //抑制警告
//所有的 Applet，都继承了 java.applet.Applet
public class HelloApplet extends Applet {
    public void paint(Graphics g){
        g.drawString("百家拳软件项目研究室!",30,30);
    }
}
```

还需要建立一个html文件，因为刚才说了它可以嵌入在浏览器里面。用记事本建立一个hello.html代码如下：

```
<applet code="HelloApplet.class" width=150 height=150></applet>
```

之后照样用javac编译刚才那个类。最后在命令行中输入appletviewer hello.html 可以看到结果。

这种小程序弥补了B/S模型的不足，用浏览器可以执行客户端的东西。因为它功能强大，所以是个不错的东西。可是功能太强大了，又引发了一些[安全](#)性的问题。所以浏览器也会对applet做了一些[安全](#)性的限制。Applet还有一种叫做沙箱模型的机制，它使得没有访问权限的资源，不能访问。保证了安全性。同时开发时也不是那么方便。Applet又跨平台的特性。

而且微软的IE浏览器里面在运行applet的时候速度不是很快，不如activex的方便。界面也不是太漂亮。不过它的这种在浏览器中运行的思想还是比较不错的。

再看个有意思的例子吧：如图 27—1 所示



图 27-1

JAVA 基础知识精华总结

1、对象的初始化

(1) 非静态对象的初始化

在创建对象时，对象所在类的所有数据成员会首先进行初始化。

基本类型：`int` 型，初始化为 0。

如果为对象：这些对象会按顺序初始化。

※在所有类成员初始化完成之后，才调用本类的构造方法创建对象。

构造方法的作用就是初始化。

(2) 静态对象的初始化

程序中主类的静态变量会在 `main` 方法执行前初始化。

不仅第一次创建对象时，类中的所有静态变量都初始化，并且第一次访问某类（注意此时未创建此类对象）的静态对象时，所有的静态变量也要按它们在类中的顺序初始化。

2、继承时，对象的初始化过程

(1) 主类的超类由高到低按顺序初始化静态成员，无论静态成员是否为 `private`。

(2) 主类静态成员的初始化。

(3) 主类的超类由高到低进行默认构造方法的调用。注意，在调用每一个超类的默认构造方法前，先进行对此超类进行非静态对象的初始化。

(4) 主类非静态成员的初始化。

(5) 调用主类的构造方法。

3、关于构造方法

(1) 类可以没有构造方法，但如果有多多个构造方法，就应该要有默认的构造方法，否则在继承此类时，需要在子类中显式调用父类的某一个非默认的构造方法了。

(2) 在一个构造方法中，只能调用一次其他的构造方法，并且调用构造方法的语句必须是第一条语句。

4、有关 public、private 和 protected

(1) 无 public 修饰的类，可以被其他类访问的条件是：a.两个类在同一文件中，b.两个类在同一文件夹中，c.两个类在同一软件包中。

(2) protected：继承类和同一软件包的类可访问。

(3) 如果构造方法为 private，那么在其他类中不能创建该类的对象。

5、抽象类

(1) 抽象类不能创建对象。

(2) 如果一个类中一个方法为抽象方法，则这个类必须为 abstract 抽象类。

(3) 继承抽象类的类在类中必须实现抽象类中的抽象方法。

(4) 抽象类中可以有抽象方法，也可有非抽象方法。抽象方法不能为 private。

(5) 间接继承抽象类的类可以不给出抽象方法的定义。

6、final 关键字

(1) 一个对象是常量，不代表不能转变对象的成员，仍可以其成员进行操作。

(2) 常量在使用前必须赋值，但除了在声明的同时初始化外，就只能在构造方法中初始化。

(3) final 修饰的方法不能被重置（在子类中不能出现同名方法）。

(4) 如果声明一个类为 final，则所有的方法均为 final，无论其是否被 final 修饰，但数据成员可为 final 也可不是。

7、接口 interface（用 implements 来实现接口）

(1) 接口中的所有数据均为 static 和 final 即静态常量。尽管可以不用这两个关键字修饰，但必须给常量赋初值。

(2) 接口中的方法均为 public，在实现接口类中，实现方法必须可 public 关键字。

(3) 如果使用 public 来修饰接口，则接口必须与文件名相同。

8、多重继承

(1) 一个类继承了一个类和接口，那么必须将类写在前面，接口写在后面，接口之间用逗号分隔。

(2) 接口之间可多重继承，注意使用关键字 `extends`。

(3) 一个类虽只实现了一个接口，但不仅要实现这个接口的所有方法，还要实现这个接口继承的接口的方法，接口中的所有方法均须在类中实现。

9、接口的嵌入

(1) 接口嵌入类中，可以使用 `private` 修饰。此时，接口只能在所在的类中实现，其他类不能访问。

(2) 嵌入接口中的接口一定要为 `public`。

10、类的嵌入

(1) 类可以嵌入另一个类中，但不能嵌入接口中。

(2) 在静态方法或其他方法中，不能直接创建内部类对象，需通过手段来取得。

手段有两种：

```
class A { class B {} B getB () { B b = new B (); return b ; }  
} static void m () { A a = new A (); A.B ab = a.getB (); // 或者是 A.B ab  
= a.new B (); }
```

(3) 一个类继承了另一个类的内部类，因为超类是内部类，而内部类的构造方法不能自动被调用，这样就需要在子类的构造方法中明确的调用超类的构造方法。接上例：

```
class C extends A.B { C () { new A () super (); // 这一句就实现了对内  
部类构造方法的调用。 }}
```

构造方法也可这样写：

```
C (A a ) { a.super (); } // 使用这个构造方法创建对象，要写成 Cc = new C  
(a ); a 是 A 的对象。
```

11、异常类

JAVA 中除了 `RunTimeException` 类，其他异常均须捕获或抛出。

备注：

1. 文中资料来源于 <http://java.chinaitlab.com/special/shijian/>