**3.1、包（重点）**

**3.1.1、包的定义**

在Java程序之中的包，主要的目的是可以将不同功能的文件进行分割。在之前的代码开发之中，所有的程序都保存在了同一个目录之中，这样一来所带来的问题：如果出现了同名文件，那么会发生覆盖问题，因为在同一个目录之中不允许有重名文件，而在不同的目录下可以有重名文件，所谓的包实际上指的就是文件夹。

在Java之中使用package关键字来定义包，其定义如下：

**范例：**定义Hello.java

|  |
| --- |
| **package cn.mldn.demo ; // 定义包**  public class Hello {  public static void main(String args[]) {  System.out.println("Hello World .") ;  }  } |

一旦包定义完成之后，那么这个类的名字就成了“cn.mldn.demo.Hello”，即这是完整的类名称，而在进行程序编译的时候也需要将\*.class文件保存在包之中，于是为了方便开发，那么也就提供了一个打包编译的操作：

|  |
| --- |
| javac -d . Hello.java |

这个命令之间有空格，每一个操作的含义：

· -d：表示的是生成文件目录，根据package的定义生成；

· .：表示在当前所在的目录下生成。

这个时候类的名字必须带上包的名称，所以执行类的时候：java cn.mldn.demo.Hello，也就是说完整类的名称就是“包.类”，而在所有的开发之中，没有包的类是绝对不存在的，只要是程序一定要有包。

**3.1.2、包的导入**

既然使用包可以将一个大型的程序拆分成不同的功能目录保存，那么这些不同的包之间也一定会存在包的导入问题，而导入包在程序之中使用import完成，下面通过一个程序来进行演示。

**范例：**定义一个Message.java的类

|  |
| --- |
| package cn.mldn.util ;  class Message {  public String getInfo() {  return "Hello World ." ;  }  } |

**范例：**随后定义另外一个类，使用Message类

|  |
| --- |
| package cn.mldn.test ;  import cn.mldn.util.Message ; // 导入包  public class Test {  public static void main(String args[]) {  **Message msg = new Message() ;**  **System.out.println(msg.getInfo()) ;**  }  } |

这个时候上面的两个类应该是按照顺序编译：

· 应该首先编译Message.java程序：javac -d . Message.java；

· 再次编译Test.java程序：javac -d . Test.java，但是这个时候出现了以下的错误提示：

|  |
| --- |
| Test.java:2: 错误: Message在**cn.mldn.util**中不是公共的; 无法从外部程序包中对其进行访问  import cn.mldn.util.Message ; // 导入包 |

**提示：**关于public class和class定义类的区别

· public class：文件名称和类名称保持一致，在一个\*.java文件之中只能存在一个public class定义，如果一个类要想被外部的包所访问必须定义为public；

· class：文件名称可以和类名称不一致，在一个\*.java之中可以同时存在多个class定义，并且编译完成之后会形成多个\*.class文件，使用class定义的类只能够在一个包中访问，不同包无法访问。

**范例：**修改Message.java程序

|  |
| --- |
| package cn.mldn.util ;  **public** class Message {  public String getInfo() {  return "Hello World ." ;  }  } |

但同时也可以发现一个问题，现在这些类编译的时候要有顺序，实在是麻烦了，为此在Java之中专门提供了一个可以进行自动联编的操作，编译的时候使用\*.java：javac -d . \*.java，将一个目录之中的所有的\*.java文件进行编译。

但是以上的代码还有一个小问题：程序在进行导入的时候使用了“包.类”的完整名称完成的，但是如果在一个程序之中要同时导入一个包的多个类的时候，那么分开去编写实在是很麻烦，为此可以使用通配符“\*”完成导入。

|  |
| --- |
| package cn.mldn.test ;  **import cn.mldn.util.\* ; // 导入包**  public class Test {  public static void main(String args[]) {  Message msg = new Message() ;  System.out.println(msg.getInfo()) ;  }  } |

但是需要注意的是，在Java之中是使用“\*”或者是单独导入，其从实际的操作性能上来讲是没有任何区别的，因为即使使用了\*也表示只导入所需要的类，不需要的不导入。

可是在导入包的时候也会遇到一种比较麻烦的问题：会导入不同包的同名类，例如：对于Message类，现在在两个包中都有：cn.mldn.util、cn.mldnjava.info。

|  |
| --- |
| package **cn.mldnjava.info** ;  public class **Message** {  public String getMsg() {  return "世界，你好！" ;  }  } |

**范例：**由于某种需要，同时导入两个包，这个时候要使用Message类的时候必须加上类的全名。

|  |
| --- |
| package cn.mldn.test ;  import cn.mldn.util.\* ; // 导入包  import cn.mldnjava.info.\* ; // 导入包  public class Test {  public static void main(String args[]) {  **cn.mldn.util.Message msg = new cn.mldn.util.Message() ;**  System.out.println(msg.getInfo()) ;  }  } |

这种情况在开发之中经常可见到，习惯了就好。

**3.1.3、jar命令**

Jar是一种Java给出的压缩格式文件，即：可以将\*.class文件以\*.jar压缩包的方式给用户，这样方便程序的维护。如果要使用jar的话，可以直接利用JDK给出的jar命令完成。

在以上的操作之中，往往只使用三个参数：

· -c：创建一个新的文件；

· -v：生成标准的压缩信息；

· -f：由用户自己指定一个\*.jar的文件名称。

**范例：**Message.java

|  |
| --- |
| package cn.mldn.util ;  public class Message {  public String getInfo() {  return "Hello World ." ;  }  } |

· 将Message.java程序进行编译：javac -d . Message.java，生成包.类；

· 将“包.类”压缩成my.jar文件：jar -cvf my.jar cn，出现了一个my.jar包；

**范例：**定义一个测试类，使用以上的程序

|  |
| --- |
| package cn.mldn.test ;  public class Test {  public static void main(String args[]) {  cn.mldn.util.Message msg = new cn.mldn.util.Message() ;  System.out.println(msg.getInfo()) ;  }  } |

现在my.jar和Test.java处于同一目录之中。但是发现找不到my.jar之中定义的内容，这是因为在java之中每一个\*.jar文件都属于一个独立的CLASSPATH路径，如果要想使用，必须配置CLASSPATH。

|  |
| --- |
| SET CLASSPATH=.;d:\javademo\my.jar |

以上的配置只是针对于一个命令行方式下完成的，如果现在希望针对于全局配置，则需要修改环境属性。

**3.2、Java的常用包（了解）**

Java语言最大的特点是提供了大量的开发支持，尤其是经过了这么多年的发展，几乎只要想做的技术，Java都可以完成了，而且有大量的开发包支撑着。而对于JAVA SE部分也提供了一些常见的系统包，下面分别来看。

· java.lang：包含了一些系统的常用类，例如：String、Object、Integer等，JDK 1.1之后此包默认导入；

· java.lang.reflect：Java的反射机制操作包；

· java.util：是一个工具的提供包，里面提供了各种的工具操作类；

· java.util.regex：正则开发包；

· java.io：编写IO操作；

· java.sql：数据库的程序开发包；

· java.net：网络程序开发包；

· java.applet：小应用程序开发包；

在1995年出现java的时候最早就是以小应用程序为主的，小应用程序最大的特点是嵌入在HTML代码之中进行展示，但是这种程序已经随着时间的推移不再开发了，下面演示一个比较简单的程序。

**范例：**定义一个Hello.java类

|  |
| --- |
| package cn.mldn.demo ; // 定义包  import java.applet.\* ;  import java.awt.\* ;  public class Hello **extends** Applet {  public void paint(Graphics g) {  g.drawString("Hello World .",20,20) ;  }  } |

**范例：**定义一个HTML文件，使用以上的程序

|  |
| --- |
| <applet code="cn.mldn.demo.Hello" height="300" width="300"/> |

**提示：**关于Applet和Application的区别？

· Applet是不需要主方法运行的，需要嵌入在HTML代码之中；

· Application是需要通过主方法执行。

· java.awt.、javax.swing：是进行图形界面的开发包，其中java.awt是一组重量级的组件包，而javax.swing是属于轻量级的组件包，并且JDK 1.2的最大特点是加入了javax.swing包；

**3.3、四种访问权限（重点）**

对于封装型实际上在之前只讲解了一个private，而封装性如果要想讲解完整，必须结合四种访问权限来看。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **范围** | **private** | **default** | **protected** | **public** |
| 1 | 在同一包的同一类 | √ | √ | √ | √ |
| 2 | 同一包的不同类 |  | √ | √ | √ |
| 3 | 不同包的子类 |  |  | √ | √ |
| 4 | 不同包的非子类 |  |  |  | √ |

以上可以简单理解为：

· private只能在一个类中访问；

· default：只能在一个包中访问；

· protected：在不同包子类；

· public：所有都可以。

**范例：**观察protected权限

|  |
| --- |
| package cn.aa ;  public class A {  protected String info = "Hello World ." ;  } |

**范例：**在不同的包继承此类

|  |
| --- |
| package cn.bb ;  import cn.aa.A ;  public class B extends A {  public void print() {  System.out.println(super.info) ;  }  } |

**范例：**定义测试类

|  |
| --- |
| package cn.test ;  import cn.bb.\* ;  public class Test {  public static void main(String args[]) {  new B().print() ;  }  } |

实际上对于四种权限本人原则：

· 属性永远都是private；

· 正常类的方法永远都是public；

· protected实际上也属于一种封装，但是这种封装知道特点就行了。

封装：private、default、protected。

**3.4、命名规范（重点）**

命名规范的主要特点就是保证程序之中类名称或方法等名称的标记明显一些，可是对于Java而言，有一些固定的命名规范还是需要遵守的：

· 类名称：每一个单词的开头首字母大写，例如：TestDemo；

· 变量名称：第一个单词的首字母小写，之后每个单词的首字母大写，例如：studetName；

· 方法名称：第一个单词的首字母小写，之后每个单词的首字母大写，例如：printInfo()；

· 常量名称：每个字母大写，例如：FLAG；

· 包名称：所有字母小写，例如：cn.mldnjava.util。

**4、总结**

1、 所有的类都要有包；

2、 虽然访问权限有四种，但是我们只需要使用private、public，99%通用原则。