Java 语言程序设计(连载二)

(第一篇)　三、Java 的开发环境

Java 自 95 年 5 月向外推出后，已取得了上百个公司的支持，由于诸多公司的参与，Java 已进入其迅速发展、完善的阶段，新的开发环境，新的类库，新的应用领域的 API，不断出现，下面，我们只对 SUN 公开发行的 Windows 95 Java 1.01 的主要开发工具和类库作简要介绍。在 Windows 95 中，Java 的开发工具是运行 DOS Shell 窗口下的程序：

- Javac 与 javac－g 是 java 编译器，在没有错误的情况下，它从 java 程序生成 Java 的字节代码，即从 .java 文件生成 .class 文件。javac－g 是非优化的 java 编译程序，提供用于调试的字节代码。
- java 与 java－g 是 java 的字节代码的解释执行程序，java－g 是非优化的 java 解释程序，用于 java 程序的调试。Java 命令可有不同的选项，在使用 －prof 选项时，java 便于从字节代码中抽取 profiling 数据，供 javapof 使用。
- javaprof 是 java profing 数据的整理程序，它利用命令 java －prof 输出的 profiling 数据，整理输出程序中每个方法调用所花费的时间、程序中每种数据类型所占用的内存容量。
- javap 是 java 类信息的抽取程序，根据不同的选项，分别抽取输出类的父类，公共域数据和方法或者私有域的数据和方法。在使用 －c 选项时，javap 还输出方法的字节代码。
- javah 是编写出 java 方法 C 语言程序的工具，它为 C 语言程序调用 java 类数据、方法提供接口，即生成 C 语言所需要的 .h 文件。

对于一种面向对象的软件开发系统，能否快速开发应用程序，除了语言的功能除外，是否带有丰富的类库和例程也是一个重要的因素。Java 系统为了方便用户在应用开发的编程，提供了相关的类库和例程。Java 的类库是以包(packege)的形式，提供用户使用，这些包是(Java 1.01 for Windows 95)：

java.awt、java.awt.image、java.awt.peer、java.net、java.util、java.applet、java.lang。新的版本还包含如下 API：数据库和 CORBA API、安全 API、商业 API 及 Applet API。

目前集成化的 JAVA 开发环境还有 Symantec 的 Cafe JAVA 开发环境和 Just in Time Compiler，Borland C ++ 5.0 中的 java 开发与调试环境。

第二篇 Java 语法及其特点

杨乔林　孟　焱　(中国科学院计算所)

在讨论语法之前，我们首先简要介绍一下 Java 的词法和基本数据类型。

一、Java 词法

Java 是一种自由格式语言。在 Java 编译过程中，其源代码被分解成一系列标记(Token)，标记分为六类：关键字、标识符、运算符、字面量、分隔符、和注释。

1. 关键字

Java 关键字如下表所示，其中带"＊"的保留字，目前并未使用。

abstract	default	goto＊	null	synchronized
boolean	do	if	package	this
break	double	implements	private	threadsafe
byte	else	inport	protected	throw
byvalue＊	extends	instanceof	public	throws
case	false	int	return	transient
catch	final	interface	short	true
char	finally	long	static	try
class	float	native	super	void
const＊	for	new	switch	while
continue				

2. 字面量

Java 字面量共有五种：整型字面量、浮点型字面量、布尔型字面量、字符字面量和串字面量。有关字面量的规定与 C 基本类似，这里只强调以下几点：

. 在 32 位二进制表示范围之内，整型字面量为 int 型，否则为 long 型。

. 字符字面量为 char 类型，宽度 16bit，属于 Uni-

code 字符集。和 C 一样，Java 也支持反斜线字符常量，如："\ r"代表回车，"\ xdd"为字符的十六进制表示等。同时增加了转义序列格式"\ udddd"以表示双字节 Unicode 字符。

. 串字面量被视为 String 类型的对象，而不是字符数组，这一点与 C 明显不同。

3. **标志符**

Java 标识符由以"字母"(包括'a'～'z'、'A'～'Z'、'$'、'-'以及编码小于 0x00C0 的 Unicode 字符)打头的字符数字串组成。由于 Java 支持双字节字符集 Unicode，故除特殊字符(例如'π')外，都可作为字符。

4. **运算符和分隔符**

其规定与 C 中大体相同。

5. **注释**

Java 除支持 C 和 C++ 注释方式外，还支持"/**"...."*/"形式，该形式用于建立 Java 文档，其中的内容可由 JDK 中的 Javadoc.exe 提取出来形成文档。

二、数据类型

Java 中的变量和表达式的类型大致可分为三类：简单类型，数组，类/接口。这里只讨论简单类型，类/接口和数组将在后面结合语法详细讨论。

在 Java 中，简单类型对应于数据结构中的术语"原子类型"，它们由 Java 语言本身提供并且一般不做进一步分割。这些类型是：

数值类型：

整数型(均为有符号整数)：

byte (8 bit)　　short (16 bit)

int (32 bit)　　long (64 bit)

浮点型：

float (32 bit，单精度)

double (64 bit，双精度)

字符型(Unicode 字符)：

char (16 bit，无符号)

布尔类型：

boolean

与 C 语言不同，Java 中各种简单类型的宽度都是固定的，其二进制位数不随具体的编译程序和运行机器而改变，这也是 Java 实现其平台无关性的一个必要条件。

三、Java 程序结构

任何 Java 源程序均由一个或多个编译单元(Compilation Unit)组成，每个编译单元是一个单独编译的 Java 源程序文件，其扩展名一般为 ".Java"。该文件只能包含下述几部分：

. 程序包(package)语句

. 引入(import)语句

. 类声明(Class Declaration)

. 接口声明(Interface Declaration)

Java 编译单元可形式地描述为：

Java 编译单元 = package 语句? import 语句 * 类型声明 *

类型声明 = 类声明|接口声明|';'

这里使用的是正规表达式，其中符号"|"表示选择；后缀"*"表示其前面的成分可出现 0 次或多次；后缀"?"表示可出现 0 次或 1 次；终结符号括在单引号中，以区别于语法成份。

package 语句指明本编译单元属于哪一个程序包(package)，其语法描述为：

package 语句 = 'package'程序包名';'

程序包是 Java 用于管理大的名字空间及避免冲突的工具，一个程序包由若干类声明和接口声明组成，例如：Sun 公司提供的"java.lang" 程序包中声明了许多与 Java 语言及其运行环境密切相关的系统类(区别于程序员自定义的类)如：Object，Class，Sytem，Thread 等，这些类所对应的 Java 源文件(编译单元)中均有"package java.lang;"这条语句。属于同一程序包的类/接口之间可以无条件地访问对方的非私有成员(包括数据成员和方法)，就好象它们位于同一个编译单元中一样。package 语句是可选的，缺省时 Java 编译器认为当前编译单元，属于一个无名程序包。若 package 语句出现在文件中，它必需位于除注释之外的第一个非空程序行。

import 语句使得当前编译单元可以共享其它程序包中的类型声明。在程序中它可以多次出现。语法描述为：

import 语句 = 'import'程序包名'.''*'';'

|'import'(类名|接口名)';'

例如："import Java.io.*;" 这一语句使得当前编译单元可以使用程序包"Java.io"中的所有声明为 public 的类型。读者可能会联想到 C 中的"extern"关键字，但二者之间显然存在很大的区别。

四、Java 程序的主体——类型声明

Java 编译单元的核心部分是类型声明(Type Declaration)，习惯上分为类声明(Class Declaration)和接口声明(Interface Declartion)两部分。与 C++不同，Java 不支持全局函数和孤立变量，所有的方法(函数)和变量只能存在于类和类实例中。因此任何 Java 程序的骨架应是类/接口声明。在传统的结构化程序设语言如 C 或 Pascal 中，所有工作都由过程/函数来完成，而在 Java 这一"彻底"的面向对象的语言中，方法为你作所有一切事情。写 Java 程序的过程就是声明类和接口的过程，在一个 Java 编译单元中可以出现多个类/接口声明，但其中只能有一个声明具有 public 修饰符。

1. 类声明

Java 的类声明与 C++ 的略有不同，其语法如下：

类声明 = 修饰符 * 'class' 类标识符('extends' 基类名)

('implements' 接口名(',' 接口名) *)?

'{' 域声明 * '}'

总的来说一个类的声明要完成以下几方面工作：

·用修饰符限定该类是"public"，"abstract"还是"final"的

·用类标识符给出类名

·在"extends"后给出基类(superclass，或"超类")的名字，指出该类由哪一个类派生而来

·在"implement"后面给出一个或多个接口(interface)名，表明该类实现了哪些接口

·在域声明中声明成员变量和类的方法

读者可能已经注意到这一语法中的几个新关键字，下面我们逐一讨论。

修饰符"abstract"表明被修饰的类是抽象类，也就是说该类中存在抽象方法——未实现的方法。当在一个类中声明抽象方法时，并不给出其方法体，而将其留给它的子类(但不一定是直接子类)去完成。抽象方法会被子类继承下去，只到最终被某个子类重写(override，或称"覆盖")并给出完整方法体为止，在此之前它仍是抽象的。任何含有抽象方法的类都是抽象类，在程序中如果试图创建一个抽象类的实例或直接调用某个抽象方法，都将引起编译错误。

修饰符"final"表明被修饰的类不可以再有子类。因而这个类将成为 Java 继承树中的一个叶节点。"final"也可以修饰方法和数据成员，这时分别表示一个方法不能被重写或一个数据成员是常量，使用"final"修饰符可使编译器完成多种优化，提高程序性能。

"extends"关键字后面紧跟的是当前类的唯一基类的名字，从字面上说是当前类"扩展"它的基类。与 C++ 不同，Java 不支持多继承(Multiple Inheritance)，因而每个类最多只有一个基类。事实上，所有 Java 的类均是由同一个根派生出的，这个共同祖先的名字是"Object"(这一点很象 Small talk)，Object 的声明参见 java.lang 程序包中的 Object.java 源文件。当一个类声明缺省 extends 分句时 Java 默认该类为 Object 的直接子类。Java 运行系统本身提供了丰富的预定义的系统类，其中也包括 Obiect 在内。事实上，Java 语言的许多核心概念，如字符串、线程、异常等都是以系统类的形式实现的，这些类通过其相互派生关系形成了 Java 的继承树，这是一棵以 Object 为根的严格意义上的"树"，而不象 C++ 的类等级那样可以是森林或网状结构。程序员自定义的类谱系必然是 Java 继承树的子树，从这一意义上说：任何 Java 编程都是对这棵树进行某种的扩充。

一个很自然的问题是：在 Java 摒弃了多继承后，如果我们希望类继承树中，不同子树的多个类具有类似行为的话，应该有其补救的方法。例如，在一个学籍管理系统中，我们声明了代表学生主修专业的类 Speciality 和代表本人原籍省份的类 Province，二者都是 Object 的直接子类。在设计时我们希望这两个类都能提供某个方法来打印出自己的中文简称，假设这个方法名为 abbreviation()。惯用 C++ 的程序员会立刻想到在 Speciality 和 Prvince 的某个共同祖先中声明 abbreviation()方法，并通过继承将其分别传给它们。遗憾的是，这两个类只有一个共同祖先，即 Object。象其他系统核心类一样，Object 是不能被重写的。同样我们也无法再为二者另行创建一个共同基类，否则就将出现多继承。幸好 Java 从 Object C 那里引进了接口(interface)这一概念。使用它很容易解决这类问题。

2. 接口和接口声明

接口描述了一个方法集合，但并不为它们编写方法体(函数体)，而仅仅给出原型。这些方法的方法体将由实现该接口的类去完成。接口声明的语法如下：

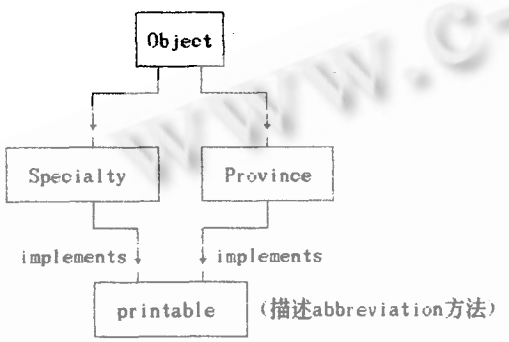
接口声明 = 修饰符 * 'interface' 接口标识符

('extends' 接口名(',' 接口名) *)?

'{' 域声明 * '}'

接口的域声明中允许出现方法和常量两类成员，但不鼓励对成员使用修饰符。Java 隐含地规定接口中所有方法为 public 和 abtract 的，常量为 public, Static 和 final 的。由于接口内的方法是抽象的，因此它们没有方法体，而直接以分号结束。除此之外，接口与类是相似的，在 Java 中都被作为"类型"对待。

接口声明的 "extends" 关键字后可以有多个接口名，这看上去好象表明接口是"多继承"的，事实上有时也确实这么说，但其含义与类的多继承是有区别的。当一个接口"扩展"一个或多个接口时，只表明当前接口将包含所列接口中的所有成员(包括方法和常量)，类似于对几个成员集合进行"并"操作。



当一个类声明中出现 "implements" 分句时，表明该类实现了此关键字后所列的一个或多个接口。也就是说，当前类的成员中必然包括上述接口中所描述的所有方法。如果几个类实现了同一个接口，这些类也就具有了一组相同的方法。于是我们找到了前面所举的学籍管理例子的解决办法：声明一个接口 "printable"，在其中描述 abbreviation() 方法的原型，之后我们令 Speciality 和 Province 分别实现这一接口，即在它们的类声明中加上" implements pintable"分句，然后在各自的成员声明中编写它们自己的 abbreviation() 方法。事实上，任何声明实现了 printable 接口的类都必须提供这一方法。因而这些类的对象都可以打印出自己的中文简称，在程序中他们也都可以被赋值给类型为 printable 的变量(或者说，都可以被当做 printable 类型的对象来引用)，就好象 printable 是它们的基类一样，例如，下面的程序是合法的：

```
Plintable P; /* 声明 p 为 printable 类型的变量 */
Speciality S= new Speciality();/* 创建一个 Speciality 类型 */   的对象 S */
... ...
P = S; /* 将 S 赋值给变量 P */
P.abbreviation();/ *打印 S 的中文简称 */
```

接口中的函数是在运行时联编的，类似于 C++ 中的虚函数。当一个程序员利用接口来引用某个类中的方法时，可以对该类的其它细节一无所知。同样，当这个类因升级等原因被重写时，只要它仍实现原有接口，则通过接口引用其方法的 Java 程序也无需重新编译。

3. **域声明**(Field Declaration)

类和接口声明的最后一部分是包在一对花括号内的若干条域声明，域声明一般分为变量声明和方法声明。除此之外，域声明还可以是一个静态初始化程序段(static Initializer)。形式描述为：

域声明 = 方法声明|构造方法声明|变量声明|静态初始化程序段|';' 这里将构造方法(constructor)单独列出以区别于普通方法，构造方法没有返回类型。

一个静态初始化程序段是由关键字'static'标出的语句块，语法为：

'static' '{' 语句 * '}'

该程序段将在它所在的类初次载入运行系统时执行，其主要作用是初始化类中的静态变量。由于此时还没有该类的实例存在，因此在静态初始化段不允许访问类的实例变量和非静态方法。

(1)变量声明

Java 变量声明与 C/C++ 大致相同，语法为

变量声明 = 修饰符 * 类型 变量标识符('[]') * ('='变量初始化段)?

其中变量初始化段(Variable Initializer)一般为表达式，当变量为数组类型时，则可能是由花括号括起的表达式表。其作用是为变量赋初值。

Java 语言支持八种简单类型：boolean, byte, char, short, int, float, long, double。除此之外，变量也可以是类/接口或者数组。Java 对类型转换有严格的要求，子类对象可以转换成基类，但将基类的对象赋值给子类的变量时，必须显式地进行强制类型转换并将受到严格的运行时检查以确保类型无误。兄弟类之间的强制转换将引起编译错误。此外，Java 中对数组的处理以及对象的创建和回收都有其独特之处。

·数组

与 C 不同，Java 支持"真正的"数组，它取代了 C 中的指针运算。尽管实际上 Java 中的所有对象(包括数组)仍是通过指针来引用的，但这些"指针"不能象在 C

中那样被作为数字量来修改和运算，而只能用于赋值或作为参数传递。Java 对数组进行严格的下标检查，下标越界是一种严重的运行时错误。Java 中数组的定义方法与 C 类似，如："int i[];"声明一个整型数组变量 i。此外还支持"类型[] 标识符"的形式，如"int[] i;"与前面的声明等价。Java 隐含地为每种可能的类型定义一个与之对应的数组类型。例如：int[] 与 int 对应，String[] 与 String 对应。如果我们自定义一个类 province，则就有一个 province[] 类型与之对应。事实上，在 Java 虚拟机模型中，对数组的操作和对类的操作是非常相似的。我们可以这样认为：在 Java 类层次中存在一个代表数组的类"Array"，各种不同的数组类型都是 Array 的子类。而且，若 B 是 A 的子类，则 B[] 是 A[]的子类。Array 有一个实例变量 length，其值为本数组的长度。

Array 是 Object 的直接子类，因此将一个数组对象赋值给 Object 型的变量是合法的，但由于 Array 这个类并没有被显式地声明，因此也不能显式地为其派生出子类。数组的创建和对象一样，也使用 new 操作符。

·对象的创建和回收

在 Java 中声明一个属于某类型的变量与真正创建一个该类的对象是两码事（八种简单类型除外）。例如：声明 "StringBuffer S;" 只意味着变量 S 可用来存放一个对 StringBuffer 类型的对象的引用，在给它赋值之前 S 是空值 null。操作符 new 显式地为一个确定类型的对象分配内存并返回对所创建对象的一个引用。如"S= new stingBuffer(10);"创建一个初始长度为 10 的 stingBuffer 型对象，并将其引用赋值给 S。当创建的是单个对象时，new 后面所跟的一般是对该对象所属类型的构造方法（构造函数）的显示调用，其中可以有参数。new 也用于创建数组对象，此时后面所跟的是该数组类型的完整描述，并指定元素个数。如："i = new int[10];" 创建长度为 10 的整型数组 i。显然，i.length 的值为 10。当一个 Java 对象不再有用时，无需显示地释放其所占内存。Java 的自动垃圾收集程序保存对每一个生存对象的引用计数，当一个对象不可能再被访问到时，就可以回收被它占用的资源。

(2) 方法声明

Java 方法声明用来描述当前类/接口的一个方法，语法如下：

方法声明 = 修饰符 * 返回类型 方法名 '(' 参量表? ')'('[]') *

('throws' 异常列表)?

('{' 语句 * '}' | ';')

从上面描述可以看出在方法声明中允许不出现方法体，而代之以分号，此时其修饰符必须是 abstract 或 native。

方法可有多种修饰符，除 C 中已有的 public, private, protected, static 之外，还有前面提到过的 final 和 abstract 以及下面要讨论的 native 和 synchronized。

·native

当一个方法被声明为 native 时，表示该方法是由平台相关的语言（一般是 C）来实现的。此时其方法体在另外的 C 源文件中编写，并编译成与平台相关的动态可装载库，在 win95 下，就是".DLL" 文件。在 native 方法被调用时，它所对应的动态链接库必须已经载入内存。Java 提供在运行时装载库的方法（见 Java.lang.system.loadLibrary()）。

·synchronized

关键字 Synchronized 表明一个方法或语句块是"临界区"(Critical Section)。换言之，它不能与另一段可能访问相同资源的代码同时执行。Java 是多线程的，在同一段时间内可能有若干个线程在执行，这就存在一个资源的互斥访问的问题。为此，Java 为每个类和类的每个实例都维护一个"监视器"(monitor)或称"锁"。每当一个 Synchronized 方法被调用时，它首先试图获得当前对象的锁（如果这是一个静态方法，那么它试图获得的是当前类的锁），若无法获得，则说明当前系统中有另一段代码正在访问该对象/类的资源，于是这个方法所在的线程将等待这个锁被释放，然后获得它，执行本身代码，再释放之。这样就实现了对资源的"同步"访问。Synchronized 也可修饰语句块，此时需在其后的括号内给出一个对象或类的名字，语法如下：

"Synchronized""("类或对象名")"

"{"语句 * "}"

这时，该语句块试图获得的将是在括号内指定的那个对象或类的锁。由于这种用法破坏了程序的清晰结构，违背了面向对象的程序设计思想，因此在编程中不鼓励使用。（未完待续）