第1章　进入Java

1.1　计算机与程序

鉴于这是对计算机程序与计算机科学的介绍，你可能希望我对计算机给出一个直截了当的定义。不过，对计算机的一个好定义并不是学习计算机科学的起始点，它也许是一个中间点。目前我们还是限于日常的理解吧：计算机就是机器，能够计算税款，帮我们写学期论文，监视飞机，控制汽车引擎，帮助寻找生命的基因序列，让我们能以各种方式娱乐。

因此，在这本书中，我们的关注点是程序——能够使计算机执行任务的文本。程序是由特殊语言写成的，那种语言称为编程语言。程序的内容叫做代码。当计算机实现或运行程序时，我们说它在执行代码。我们的目的教你如何运用Java编程语言来读和写程序。

在数百种编程语言中，Java是后来者。由于它是最近开发出来的，其设计反映了近几十年来在程序开发和编程语言设计领域所取得的智慧。这些智慧反映在Java的设计中，使Java成为面向对象的语言。

为了对Java语言代码有个印象，我们快速浏览一个小的Java程序：

```
public class Program0 {
    public static void main(String[] arg) {
        System.out.println("Welcome To Java!");
    }
}
```

这个程序在计算机屏幕上显示欢迎词“Welcome To Java!”。如果进一步观察，你可以看出，这句欢迎词包含在程序的第三行中。很清楚，在屏幕显示文字这一现实事件与这段代码有着联系。

这一章，我们将探究这种联系并且向你展示如何运用Java对象编写你的第一个Java程序。

1.2　程序与模型

几乎所有的程序都要针对某件事建模。模型是什么意思呢？模型是一种简化的描述，它包含对用户来说比较重要的特性，而忽略其他特性。比如，当第一个例子显示

“Welcome To Java!”时,我们关心的只是显示的文字,不关心文字的颜色、屏幕的亮度以及任何其他显示属性。一个儿童塑料汽车模型可能表现了车轮的外部细节,却完全不考虑引擎和传动部分。一个更精致的模型也许包括可以工作的引擎和逼真的内部细节。当然,模型越逼真和详细,制造时付出的努力和花费也就越大。

早在计算机出现以前,人们就开始建模。请考虑下面这个例子:

Keyspan 煤气公司的服务调度员跟踪 Brooklyn 地区的 43 个维修工程车。她用一张大地图和一些高顶图钉来做这件事:1 号图钉代表 1 号维修车,2 号图钉代表 2 号维修车,依此类推。将图钉按在地图上表示一辆维修车最近所在的位置。每当一个维修车从一个新位置呼叫时,就移动代表这辆车的图钉。

这个调度员还必须跟踪客户的服务请求。当来了一个要求服务的电话时,调度员就将一个按钉按在客户的位置。当服务完成后,按钉就被拿掉。

很显然,图钉不是维修车,但调度员不管是在心中还是与同事交谈中都将它看作维修车。然而,当某天她来上班时,如果发现图钉换成了地毯钉,她也会毫无疑问地将地毯钉看作“维修车”。

模型不是一个完整的表示。例如,地图上的按钉并不能表示客户问题的种类。尽管有这些小的不足,但只要模型抽象了重要的细节进而能够完成工作,它就是可以接受的。

虽然在模型的元素和它们表示的事物间有着逻辑对应关系,但是在物理上模型与它表示的事物很不相同。离开特定环境去看调度员的地图,任何人也不可能知道图钉意味着什么。有人也许猜测那是上个月抢劫案的发生地,或者也许是严重的坑洼,或者特别好的餐馆。图钉的含义不是地图上固有的,而是调度员赋予的。

模型还可以表示像“未来城市”这样虚构的或假想的世界。此外,并不是所有模型都是可触摸的实物。一个模型可能只存在于纸上,甚至只在某人的脑子里——精神模型。

例如,某人计划筹措大学教育的经费时,也许假想她将找到一定数量的工作:在学期中的兼职工作,在夏季的全职工作,一定数量的政府贷款、学校资助,等等。将这些写下来,加起来(希望这些等于学费或超过学费),这就是一种建模行为。写下来的数字代表着假想的钱数。

每一个模型,无论代表现实世界还是假想世界,都具有下列特征:

- 模型中的元素代表着另一个更复杂的事物,例如,图钉可以用于表示维修车。
- 这些模型元素呈现统一的行为,例如,图钉指明位置并且可移动。
- 模型元素可以按它们的共同行为分组,成为不同的类,例如,按钉可以出现和消失。按钉一旦按到地图上,直到被取走,一直是不会移动的。相反,图钉始终在地图上,但是可以移动(这反映出这样的现实:维修车会行走,而客户所在的位置只会是需要服务,或者不需要服务)。
- 由模型元素之外的动作导致模型元素的行为,例如,用手移动图钉。

在前面的例子中,模型中的元素都是物理对象(高顶图钉、按钉),外部动作来自于人,甚至行为部分也由人来执行。在编程时我们省略这些物理手段,而是在程序中表示模型元素和行为。首先,我们使用的模型比这个例子中的要简单得多。不过,当我们在后续章节中获得更多的工具时,我们就能在程序中建立越来越复杂的模型。

1.3　对象、类和消息

Java 程序中的模型元素称为对象。具有共同行为的对象可以分组为不同的类别，称为类。为了完成一项任务而共同工作的对象，通过互相发送消息进行通信。在这一节，我们通过 Java 程序与一个企业之间的类比，开始探索这些基本概念。

1.3.1　对象

让我们专门来考虑一个大型软件开发企业中的一组雇员。我们只考虑几种类型的雇员，以及这些雇员可能要完成的几种任务，以简化这种业务模型。这种企业可能有一个总裁和两个副总裁，他们每个人分别配有一组执行助理。在这个人员层次结构中向下走若干级，可能会找到很多程序员，一组组的程序员由项目经理管理。

在 Java 中，我们将模型中的每一个人称为一个对象。实际上，假如我们编写 Java 程序模拟这家公司的一些活动，我们应该将每个人模拟为一个 Java 对象：一个 President 对象、两个 VicePresident 对象、一些 ExecutiveAssistant 对象、很多 Programmer 对象，等等。

1.3.2　行为

虽然程序员们在公司里从事不同的项目，但他们的行为在模型中是类似的：他们生产代码，经常加班，在桌子底下睡觉。在 Java 中，我们说 Programmer 对象共享共同的行为，这意味着它们做着同类事情。副总裁做的事情也是类似的，即使他们负责的领域不同：他们向员工要数据，产生报告，在股东的压力下辞职。因此，VicePresident 对象也共享共同的行为。但是并不奇怪，VicePresident 对象共享的行为与 Programmer 对象共享的行为是不同的。这两种不同的对象模拟了不同类别的事物（执行官和程序员）。在模拟这家公司的 Java 程序中，我们应该用具有不同行为的对象来模拟不同类型的雇员。

1.3.3　信息

正像如果没有清晰的通信渠道，哪个组织都无法有效工作，Java 的对象也必须能够协同工作。现代化企业中多数日常通信都是以电子邮件或语音邮件的形式实现的：如果一个人要将消息传给另一个人，或者需要另一个人完成什么任务，它可能会简单地发一个恰当的电子邮件，或者留个语音邮件。这正是 Java 对象彼此通信的方式：发送消息（注意对象间的消息并不是真的电子邮件消息！）。

当然，公司的雇员受到的培训和个人能力都不尽相同，所以不是每个人都能正确回复发来的每个消息。虽然请一位程序员准备在下周二做一个项目演示是完全有意义的，但将同样的请求给副总裁，结果最多是不可预料。同样我们不会要求一个程序员收集公司的财务数据，但是副总裁可能非常胜任这项任务。很清楚，对象的行为与它能够回复的消息类型密切相关。

我们来稍微多想一想是什么构成了消息：如果总裁需要一个员工准备一份报告，他怎

么构思这个消息呢? 显然消息要有一个发送者(总裁)和一个接收者(指定的雇员)。还得说明要完成的任务:准备一份报告。如果这就是消息的全部,那么雇员很难搞清楚要做什么,它需要更多的细节信息,例如,说明这份报告应该汇总六月份的收入和支出。

在 Java 中,消息的构成也是类似的:发送的每个消息都要指明接收它的对象(在 Java 中,对象以引用来指定),响应消息时应该执行什么任务,以及为了充分描述任务所必须提供的任何进一步细节。

1.3.4 Java 程序

现在离开我们曾使用的类比,关于 Java 和 Java 程序我们学到了什么? 我们知道在计算机上运行的 Java 程序,就是与要解决的问题或要执行的计算中的重要元素相对应的对象集合。对象是主要演员:每个对象有其能够完成的一组特定任务(或行为)。例如,下一节我们会看到 PrintStream 类型的对象,它们能够在计算机屏幕上显示文字。对象也会像其他对象发送消息,请求它们完成指定给它们的任务。对象的行为取决于它代表着什么事物;没有哪个对象可以完成每个任务,但是我们希望代表类似事物的对象具有类似的行为。

1.3.5 类

在 Java 中一个模型元素的类型称为类。Java 程序员的基本工作是提供类定义,或者说是描述类中的对象必须如何表现。这样的类定义描述类中的对象可以接收哪些种类的消息,以及如何响应这些消息。当一个类在程序中被定义以后,程序运行时就可以创建那个类的对象。属于一个类的每一个对象都被称为该类的一个实例。用 Java 编程就等于编写类的定义并用类创建对象。

有时候我们会提到类的职责。那意思就是指类提供的行为(或负责的事情)。这只是用另一种方式来说明类模拟的是什么。当我们说模拟时,我们关心的是这个类代表哪一类对象;当我们说到职责时,我们集中关注对象的行为。

如果我们用 Java 模拟软件公司,我们应该定义一个 Programmer 类。它提供 Programmer 对象的行为并使我们可以创建 Programmer 对象,每一个对象都是 Programmer 类的一个实例。我们还要定义一个 President 类,它定义 President 对象的行为并使我们可以创建 President 对象。

1.3.6 预定义的对象和类

很多程序都有共同的行为。例如,很多程序都在屏幕上显示文字。难道对每一个这样的程序我们都必须从定义类以提供共同行为开始吗? 幸运的是,我们不必经常重复此事:Java 自带了预定义的类和对象。并且,我们还可以使用我们自己或其他程序员已经创建的可用的类和对象。这样,程序代码就可以被重用,以生产功能更强大的程序。

为了开始探索如何在 Java 中使用对象,我们将从这些预定义对象开始。

1.4　第一个对象：PrintStream 对象

程序的目的是提供信息。我们要编写的大多数程序都要将信息显示在监视器上，即在计算机屏幕上显示信息。在这一节中我们学习如何利用 Java 的预定义对象来控制监视器。

监视器扮演两种角色。对人来说，监视器是读取信息的设备。从程序的角度来看，监视器最重要的行为是(程序)可以告诉它显示一些字符(字母、数字、标点符号、空格等)，然后再显示多一些字符、再多一些，等等。因此，对程序来说监视器是一个可以输出连续字符流的设备。

Java 提供一个预定义的类，模拟像显示器这样可以印出字符流的设备。这个类(恰当地)被称为 PrintStream 类。PrintStream 类负责显示字符流。PrintStream 类的实例对象可以接受 println()消息，这个消息要求将一组给定的字符打印出来。所有 PrintStream 对象共享这个 ptintln()行为。

PrintStream 类并不模拟实际监视器具有的全部特性。例如，它不提供任何改变监视器背景颜色的行为。它只模拟监视器显示字符流的能力。

与预定义的 PrintStream 类一起，Java 还提供了一个预定义的 PrintStream 类实例，代表计算机自己的监视器。在 Java 程序中，通过短语 System. out 引用这个预定义的 PrintStream 类对象。下一节，我们将看到 Java 程序员如何使用 System. out 发送 println()消息给这个 PrintStream 对象，进而引发 println()行为。

我们在上面提到过，System. out 引用了一个预定义的 PrintStream 对象。我们说 System. out 是那个对象的引用。图 1.1 显示了使用中的 System. out。Java 中的引用是一个引用到对象的短语。

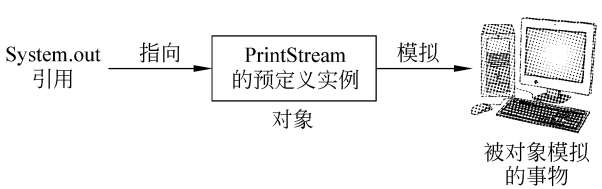

图 1.1　一个引用的例子。这里 System. out 引用一个预定义的 PrintStream 对象，以模拟一个监视器

1.5　发送一个消息到 System. out 对象

Java 中的消息是一个请求，向对象请求期望的行为。一个消息的组成如下：

- 期望的行为名(在 PrintStream 类中是 println())。
- 由圆括号包围的细节。

对“细节”的需求取决于特定的行为。例如，PrintStream 对象的 println()行为输出

一个字符流,需要知道要输出的字符是什么。所以,给 PrintStream 对象的 println()消息看起来是像这样的:

```
println( some - characters )
```

在 Java 中,我们可以用双引号括起来指定一组字符。这样的表达式称为字符串。所以,为了使监视器向一对用户 Millicent 和 Micah 显示欢迎词,我们需要向 PrintStream 对象发送如下消息:

```
println("Welcome Millicent and Micah" )
```

这一 Java 消息如图 1.2 所示。

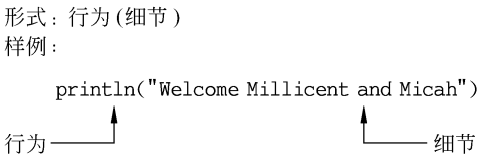

图 1.2 Java 中的消息。一个消息包括期望的行为和括号中进一步的细节

为了发送一个消息,我们必须指定接收者,即接收消息的对象。在 Java 中我们写一个由下列部分组成的短语:

- 接收对象的引用(例如,System. out)。
- 一个句点。
- 要发送的消息。

所以要将我们的 println()消息发送给 PrintStream 对象,必须像下面这样写:

```
System. out. println("Welcome Millicent and Micah" )
```

图 1.3 说明了 Java 消息的各部分。

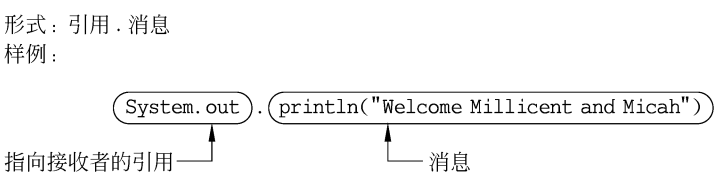

图 1.3 Java 中的消息。“引用.消息”格式用于将消息发往接收者

执行的时候,System. out 对象接收 println("Welcome Millicent and Micah")消息并响应消息,将欢迎词显示在屏幕上(见图 1.4)。

Java 语句

发送一个消息给对象是程序员说明的一个动作,当程序运行时由计算机执行。在 Java 中,所有的动作都用语句来说明。

在末尾添加一个分号,便将发送消息给 Java 对象的短语转变为一个完整的 Java

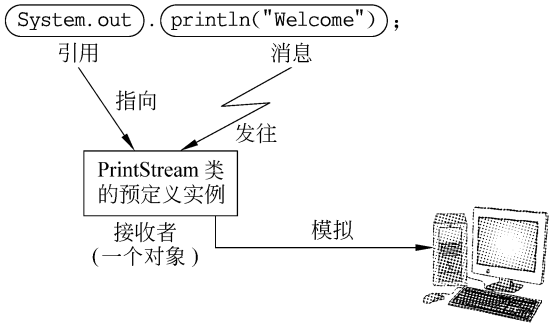

图 1.4　发送消息的过程中接收者的角色

语句：

```
System.out.println("Welcome Millicent and Micah" );
```

这就像我们向完整的英语语句添加句号一样(见图 1.5)。

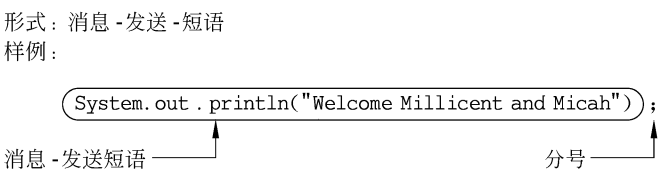

图 1.5　发送消息的语句

1.6　一个 Java 程序

编写程序的很大乐趣在于看到程序执行。即使一个程序只显示一个通告也是重要的第一步，并且它确实模拟了什么——一个发出通告的人。假设我们想写一个程序，在屏幕上显示文字“This is my first Java program”，在其下方显示“but it won’t be my last.”下面的思想给出了编写程序的指南。

- 有一个预定义的对象(通过 System.out 引用)，其行为包括了在屏幕上显示字符。
- 得到这个行为的办法是发送一个消息。
- 消息的第一个部分必须是提供这种行为的方法名(println())。
- 消息的其他部分必须由参数组成，参数中包含了对象完成工作所需要的信息(要显示的行)。

因为有两行文字要显示，我们必须发送两个消息给 System.out 对象。这就产生了下列 Java 程序的核心部分，由一对语句构成：

```
System.out.println("This is my first Java program" );
System.out.println("but it won't be my last." );
```

要将这些变成合法的 Java 程序，我们必须为程序选择一个名字。Java 中的名字称为“标识符”。

标识符是一个由字母、数字或下划线构成的序列。Program1 便是这个程序的一个好名字。使用这个名字,添加一些需要的标记,我们可以写出如下 Java 程序

```
public class Program1{
  public static void main(String[] arg){
        System.out.println("This is my first Java program" );
        System.out.println("but it won't be my last." );
    }
}
```

在这一章的其余部分和第 2 章中,前两行

```
public class Program1{
  public static void main(String[] arg){
```

和最后两行

```
    }
}
```

将是我们写的每个程序的开始和结束。

附录 B 介绍了输入和运行这个程序所必需的步骤。如果你手边有计算机并且想实践,你可以跳转到那部分去,以学习如何运行这个程序。

1.7 Java 小插曲 标识符、语句顺序、格式和注释

这本书的目的是教授编程,为了避免所有重要的目标受到其他因素(包括 Java 语言细节)干扰,我们在讨论中会经常省略一些细节和变化。为了弥补这些缺口和复习总结,我们将周期性地给出 Java 小插曲,就像本节这样,仅关注于语言本身。

1.7.1 Java 规则

就像英语有一系列规则告诉我们什么是一个可接受的句子,Java 也有一系列规则告诉我们什么是一个可接受的程序。如果代码不违反这些规定,我们就说它是合法的。

1.7.2 标识符

一个类或行为必须有名字——标识符。小心:Java 区分大写和小写字母。这意味着 system 和 System 是不同的标识符!当你要表示 System 时如果写成 system,你的程序就不能工作了。在 Java 中,通常每个类和程序名以大写字母开头;其他标识符(如方法名)以小写字母开头。

1.7.3 关键字

关键字是 Java 语言中具有预定义含义的特殊单词。像 class、public、static 和 void 这些单词都是关键字。PrintStream 不是关键字——它是预定义的类的名字。

1.7.4　Java 的语句顺序

语句顺序是重要的，因为它们是按照出现的次序被计算机执行。所以，

```
System.out.println("One two three" );
System.out.println("Four five six" );
```

产生的消息次序和在屏幕上的显示内容与下述语句不同：

```
System.out.println("Four five six" );
System.out.println("One two three" );
```

1.7.5　程序格式和注释

上一节的 Java 程序是使用一种特殊格式编写的。程序中的语句通过多次使用 TAB 键缩格。每条语句只出现在一行，并且一行中绝不超过一条语句。

Java 的规则在格式方面很灵活。主要的格式规则是两个连续的标识符或者关键字必须由至少一个空格分隔。因此下述写法不合法：

```
classProgram1 ...
```

而应该写 class Program1。但是如果将样例程序写成如下形式也是合法的：

```
public class Program1{ public static void main(String[]
arg){ System.out.println(
"This is my first Java program" ); System.out.println(
"but it won't be my last." ); } }
```

虽然合法，但这段代码却很难读。当我们写代码时，目标之一就是让它可读。可读性是重要的，因为程序需要周期性地升级，并且需要调试(检查并更正错误)，调试人员常常不是原来编写程序的人。在此，详细列出明确的格式规则为时尚早，但下面的规则是个开端：

- 每行写一条语句。如果语句太长在一行写不下，请在合理的地方断行，并且使续行相对起始行缩进一个 TAB 位置。
- 缩格时使用 TAB 键而不是空格键。这会很容易地形成整齐的左对齐。
- 总的来说，可以模仿本书的风格。

在我们展示 Java 语言的更多特性时，我们也会说明如何编排其格式。

1.7.6　注释

Java 允许程序员在代码中加入注释。注释是指被计算机忽略，但是出现在程序中向读者解释的说明文字。在 Java 中有两种方法写注释：包围的注释和行注释。

1.7.7　包围的注释

以/ * 开头并以 * /结尾的任何文字都是注释。/ * 和 * /不必在同一行。因此，

```
/*
 * This program prints out several greetings.
 */
```

是一个合法的注释(注意:/和 * 之间不能有空格)。

1.7.8 行注释

一旦//出现在行尾,其后的所有文字就都是注释了。因此,

```
//This program prints out several greetings.
```

也是合法的注释(注意:两个/之间不能有空格)。

因为注释会被计算机忽略,你可以将注释放在程序中任何地方。

就像程序格式一样,虽然关于注释位置的规则十分自由,但是却有一些惯例。这里有几条指导原则:

- 以关键字 class 起始的行之前应该有一个注释,说明代码的目的和行为。
- 注释不解释 Java 如何工作,假定读者已懂得 Java 语言了。更重要的是,注释要给出对代码的理解而不是代码的表象。
- 行注释不能出现在一条语句中间。

例如,我们可以对上一节的程序进行如下注释:

```
/*
 * Program1: 声明我编写 Java 程序的第一次经验以及继续下去的愿望。
 */
public class Program1{
  public static void main(String[] arg){
    System.out.println("This is my first Java program" );
    System.out.println("but it won't be my last." );
    }
}
```

1.8 例 行 步 骤

在纸上编写一个 Java 程序本身并不能使计算机做任何事情。首先,要使计算机可以得到程序。其次,程序要被翻译成计算机可以执行的形式。最后,要指示计算机去执行,也就是说运行翻译以后的程序指令。

幸运的是,这三步非常容易进行,部分原因是已经有程序以近乎自动化的方式来完成每一个步骤。这些程序的精确细节因不同的计算机系统而不同。本节我们将讲述一般步骤。关于一些特殊系统(包括 Windows、UNIX/Linux 和 Macintosh)的细节将在附录中给出。

1.8.1 可访问性

用笔写在纸上的程序不能被计算机访问。要使一个程序可被计算机访问,需要创建