

# 第 1 章 Java 程序设计概述

- ▲ Java 程序设计平台
- ▲ Java 发展简史
- ▲ Java “白皮书”的关键术语
- ▲ 关于 Java 的常见误解
- ▲ Java applet 与 Internet

1996 年 Java 第一次发布就引起了人们的极大兴趣。关注 Java 的人士不仅限于计算机出版界，还有诸如《纽约时报》《华盛顿邮报》《商业周刊》这样的主流媒体。Java 是第一种也是唯一一种在 National Public Radio 上占用了 10 分钟时间来进行介绍的程序设计语言，并且还得到了 \$100 000 000 的风险投资基金。这些基金全部用来支持用这种特别的计算机语言开发的产品。重温那些令人兴奋的日子是很有意思的。本章将简要地介绍一下 Java 语言的发展历史。

## 1.1 Java 程序设计平台

本书的第 1 版是这样描写 Java 的：“作为一种计算机语言，Java 的广告词确实有点夸大其辞。然而，Java 的确是一种优秀的程序设计语言。作为一个名副其实的程序设计人员，使用 Java 无疑是一个好的选择。有人认为：Java 将有望成为一种最优秀的程序设计语言，但仍需要一个相当长的发展时期。一旦一种语言应用于某个领域，与现存代码的相容性问题就摆在了人们的面前。”

我们的编辑手中有许多这样的广告词。这是 Sun 公司高层的某位不愿透露姓名的人士提供的（Sun 是原先开发 Java 的公司）。Java 有许多非常优秀的语言特性，本章稍后将会详细地讨论这些特性。由于相容性这个严峻的问题确实存在于现实中，所以，或多或少地还是有一些“累赘”被加到语言中，这就导致 Java 并不如想象中的那么完美无瑕。

但是，正像我们在第 1 版中已经指出的那样，Java 并不只是一种语言。在此之前出现的那么多钟语言也没有能够引起那么大的轰动。Java 是一个完整的平台，有一个庞大的库，其中包含了很多可重用的代码和一个提供诸如安全性、跨操作系统的可移植性以及自动垃圾收集等服务的执行环境。


作为一名程序设计人员，常常希望能够有一种语言，它具有令人赏心悦目的语法和易于理解的语义（C++ 不是这样的）。与许多其他的优秀语言一样，Java 完全满足了这些要求。有些语言提供了可移植性、垃圾收集等，但是，没有提供一个大型的库。如果想要有奇特的绘图功能、网络连接功能和数据库存取功能就必须自己动手编写代码。Java 具备所有这些特性，它是一种功能齐全的出色语言，是一个高质量的执行环境，还提供了一个庞大的库。正是因为它集多种优势于一身，所以对广大的程序设计人员有着不可抗拒的吸引力。

## 1.2 Java “白皮书”的关键术语

Java 的设计者已经编写了颇有影响力的“白皮书”，用来解释设计的初衷以及完成的情况，并且发布了一个简短的摘要。这个摘要用下面 11 个关键术语进行组织：

- |           |         |
|-----------|---------|
| 1) 简单性    | 7) 可移植性 |
| 2) 面向对象   | 8) 解释型  |
| 3) 分布式    | 9) 高性能  |
| 4) 健壮性    | 10) 多线程 |
| 5) 安全性    | 11) 动态性 |
| 6) 体系结构中立 |         |

本节将提供一个小结，给出白皮书中相关的说明，这是 Java 设计者对各个关键术语的论述，另外还会根据我们对 Java 当前版本的使用经验，给出对这些术语的理解。

 **注释：**写这本书时，白皮书可以在 [www.oracle.com/technetwork/java/langenv-140151.html](http://www.oracle.com/technetwork/java/langenv-140151.html) 上找到。对于 11 个关键术语的论述请参看 <http://horstmann.com/corejava/java-an-overview/7Gosling.pdf>。

### 1.2.1 简单性

人们希望构建一个无须深奥的专业训练就可以进行编程的系统，并且要符合当今的标准惯例。因此，尽管人们发现 C++ 不太适用，但在设计 Java 的时候还是尽可能地接近 C++，以便系统更易于理解。Java 剔除了 C++ 中许多很少使用、难以理解、易混淆的特性。在目前看来，这些特性带来的麻烦远远多于其带来的好处。

的确，Java 语法是 C++ 语法的一个“纯净”版本。这里没有头文件、指针运算（甚至指针语法）、结构、联合、操作符重载、虚基类等（请参阅本书各个章节给出的 C++ 注释，其中比较详细地解释了 Java 与 C++ 之间的区别）。然而，设计者并没有试图清除 C++ 中所有不适当的特性。例如，switch 语句的语法在 Java 中就没有改变。如果你了解 C++ 就会发现可以轻而易举地转换到 Java 语法。

Java 发布时，实际上 C++ 并不是最常用的程序设计语言。很多开发人员都在使用 Visual Basic 和它的拖放式编程环境。这些开发人员并不觉得 Java 简单。很多年之后 Java 开发环境才迎头赶上。如今，Java 开发环境已经远远超出大多数其他编程语言的开发环境。

简单的另一个方面是小。Java 的目标之一是支持开发能够在小型机器上独立运行的软件。基本的解释器以及类支持大约仅为 40KB；再加上基础的标准类库和对线程的支持（基本上是一个自包含的微内核）大约需要增加 175KB。

在当时，这是一个了不起的成就。当然，由于不断的扩展，类库已经相当庞大了。现在有一个独立的具有较小类库的 Java 微型版（Java Micro Edition），这个版本适用于嵌入式设备。

### 1.2.2 面向对象

简单地讲，面向对象设计是一种程序设计技术。它将重点放在数据（即对象）

和对象的接口上。用木匠打一个比方，一个“面向对象的”木匠始终关注的是所制作的椅子，第二位才是所使用的工具；一个“非面向对象的”木匠首先考虑的是所用的工具。在本质上，Java 的面向对象能力与 C++ 是一样的。

开发 Java 时面向对象技术已经相当成熟。Java 的面向对象特性与 C++ 旗鼓相当。Java 与 C++ 的主要不同点在于多重继承，在 Java 中，取而代之的是更简单的接口概念。与 C++ 相比，Java 提供了更丰富的运行时自省功能（有关这部分内容将在第 5 章中讨论）。

### 1.2.3 分布式

Java 有一个丰富的例程库，用于处理像 HTTP 和 FTP 之类的 TCP/IP 协议。Java 应用程序能够通过 URL 打开和访问网络上的对象，其便捷程度就好像访问本地文件一样。

如今，这一点已经得到认可，不过在 1995 年，主要还是从 C++ 或 Visual Basic 程序连接 Web 服务器。

### 1.2.4 健壮性

Java 的设计目标之一在于使得 Java 编写的程序具有多方面的可靠性。Java 投入了大量的精力进行早期的问题检测、后期动态的（运行时）检测，并消除了容易出错的情况……Java 和 C++ 最大的不同在于 Java 采用的指针模型可以消除重写内存和损坏数据的可能性。

Java 编译器能够检测许多在其他语言中仅在运行时才能够检测出来的问题。至于第二点，对于曾经花费几个小时来检查由于指针 bug 而引起内存冲突的人来说，一定很喜欢 Java 的这一特性。

### 1.2.5 安全性

Java 适用于网络 / 分布式环境。为了达到这个目标，在安全方面投入了很大精力。使用 Java 可以构建防病毒、防篡改的系统。

从一开始，Java 就设计成能够防范各种攻击，其中包括：


- 运行时堆栈溢出。如蠕虫和病毒常用的攻击手段。
- 破坏自己的进程空间之外的内存。
- 未经授权读写文件。

原先，Java 对下载代码的态度是“尽管来吧！”。不可信代码在一个沙箱环境中执行，在这里它不会影响主系统。用户可以确信不会发生不好的事情，因为 Java 代码不论来自哪里，都不能脱离沙箱。

不过，Java 的安全模型很复杂。Java 开发包（Java Development Kit, JDK）的第一版发布之后不久，普林斯顿大学的一些安全专家就发现一些小 bug 会允许不可信的代码攻击主系统。

最初，安全 bug 可以快速修复。遗憾的是，经过一段时间之后，黑客已经很擅长找出安全体系结构实现中的小漏洞。Sun 以及之后的 Oracle 为修复 bug 度过了一段很是艰难的日子。

遭遇多次高调攻击之后，浏览器开发商和 Oracle 都越来越谨慎。Java 浏览器插件不再信任远程代码，除非代码有数字签名而且用户同意执行这个代码。

 **注释：**现在看来，尽管 Java 安全模型没有原先预想的那么成功，但 Java 在那个时代确实相当超前。微软提供了一种与之竞争的代码传输机制，其安全性完全依赖于数字签名。显然这是不够的，因为微软自身产品的任何用户都可以证实，知名开发者的程序确实会崩溃并对系统产生危害。

### 1.2.6 体系结构中立

编译器生成一个体系结构中立的目标文件格式，这是一种编译过的代码，只要有 Java 运行时系统，这些编译后的代码可以在许多处理器上运行。Java 编译器通过生成与特定的计算机体系结构无关的字节码指令来实现这一特性。精心设计的字节码不仅可以很容易地在任何机器上解释执行，而且还可以动态地翻译成本地机器代码。

当时，为“虚拟机”生成代码并不是一个新思路。诸如 Lisp、Smalltalk 和 Pascal 等编程语言多年前就已经采用了这种技术。

当然，解释虚拟机指令肯定会比全速运行机器指令慢很多。然而，虚拟机有一个选项，可以将执行最频繁的字节码序列翻译成机器码，这一过程被称为即时编译。

Java 虚拟机还有一些其他的优点。它可以检测指令序列的行为，从而增强其安全性。

### 1.2.7 可移植性

与 C 和 C++ 不同，Java 规范中没有“依赖具体实现”的地方。基本数据类型的大小以及有关运算都做了明确的说明。

例如，Java 中的 int 永远为 32 位的整数，而在 C/C++ 中，int 可能是 16 位整数、32 位整数，也可能是编译器提供商指定的其他大小。唯一的限制只是 int 类型的大小不能低于 short int，并且不能高于 long int。在 Java 中，数据类型具有固定的大小，这消除了代码移植时令人头痛的主要问题。二进制数据以固定的格式进行存储和传输，消除了字节顺序的困扰。字符串是用标准的 Unicode 格式存储的。

作为系统组成部分的类库，定义了可移植的接口。例如，有一个抽象的 Window 类，并给出了在 UNIX、Windows 和 Macintosh 环境下的不同实现。

选择 Window 类作为例子可能并不太合适。凡是尝试过的人都知道，要编写一个在 Windows、Macintosh 和 10 种不同风格的 UNIX 上看起来都不错的程序有多么困难。Java 1.0 就尝试着做了这么一个壮举，发布了一个将常用的用户界面元素映射到不同平台上的简单工具包。遗憾的是，花费了大量的心血，却构建了一个在各个平台上都难以让人接受的库。原先的用户界面工具包已经重写，而且后来又再次重写，不过跨平台的可移植性仍然是个问题。

不过，除了与用户界面有关的部分外，所有其他 Java 库都能很好地支持平台独立性。你可以处理文件、正则表达式、XML、日期和时间、数据库、网络连接、线程等，而不用操心底层操作系统。不仅程序是可移植的，Java API 往往也比原生 API 质量更高。

### 1.2.8 解释型

Java 解释器可以在任何移植了解释器的机器上执行 Java 字节码。由于链接是一个增量式且轻量级的过程，所以，开发过程也变得更加快捷，更加具有探索性。

这看上去很不错。用过 Lisp、Smalltalk、Visual Basic、Python、R 或 Scala 的人都知道“快捷而且具有探索性”的开发过程是怎样的。你可以做些尝试，然后就能立即看到结果。Java 开发环境并没有将重点放在这种体验上。

### 1.2.9 高性能

尽管对解释后的字节码性能已经比较满意，但在有些场合下还需要更加高效的性能。字节码可以（在运行时时刻）动态地翻译成对应运行这个应用的特定 CPU 的机器码。

使用 Java 的头几年，许多用户不同意这样的看法：性能就是“适用性更强”。然而，现在的即时编译器已经非常出色，以至于成了传统编译器的竞争对手。在某些情况下，甚至超越了传统编译器，原因是它们含有更多的可用信息。例如，即时编译器可以监控经常执行哪些代码并优化这些代码以提高速度。更为复杂的优化是消除函数调用（即“内联”）。即时编译器知道哪些类已经加载。基于当前加载的类集，如果特定的函数不会被覆盖，就可以使用内联。必要时，还可以撤销优化。

### 1.2.10 多线程

多线程可以带来更好的交互响应和实时行为。

如今，我们非常关注并发性，因为摩尔定律行将完结。我们不再追求更快的处理器，而是着眼于获得更多的处理器，而且要让它们一直保持工作。不过，可以看到，大多数编程语言对于这个问题并没有显示出足够的重视。

Java 在当时很超前。它是第一个支持并发程序设计的主流语言。从白皮书中可以看到，它的出发点稍有些不同。当时，多核处理器还很神秘，而 Web 编程才刚刚起步，处理器要花很长时间等待服务器响应，需要并发程序设计来确保用户界面不会“冻住”。

并发程序设计绝非易事，不过 Java 在这方面表现很出色，可以很好地管理这个工作。

### 1.2.11 动态性

从各种角度看，Java 与 C 或 C++ 相比更加具有动态性。它能够适应不断发展的环境。库中可以自由地添加新方法和实例变量，而对客户端却没有任何影响。在 Java 中找出运行时类型信息十分简单。

当需要将某些代码添加到正在运行的程序中时，动态性将是一个非常重要的特性。一个很好的例子是：从 Internet 下载代码，然后在浏览器上运行。如果使用 C 或 C++，这确实难度很大，不过 Java 设计者很清楚动态语言可以很容易地实现运行程序的演进。最终，他们将这一特性引入这个主流程序设计语言中。

**■ 注释：**Java 成功地推出后不久，微软就发布了一个叫做 J++ 的产品，它与 Java 有几乎相同的编程语言以及虚拟机。现在，微软不再支持 J++，取而代之的是另一种名为 C# 的语言。C# 与 Java 有很多相似之处，然而使用的却是完全不同的虚拟机。本书不准备介绍 J++ 或 C# 语言。

### 1.3 Java applet 与 Internet

这里的想法很简单：用户从 Internet 下载 Java 字节码，并在自己的机器上运行。在网页中运行的 Java 程序称为 applet。要使用 applet，需要启用 Java 的 Web 浏览器执行字节码。不需要安装任何软件。任何时候只要访问包含 applet 的网页都会得到程序的最新版本。最重要的是，要感谢虚拟机的安全性，它让我们不必再担心来自恶意代码的攻击。

在网页中插入一个 applet 就如同在网页中嵌入一幅图片。applet 会成为页面的一部分。文本环绕着 applet 所占据的空间周围。关键的一点是这个图片是活动的。它可以对用户命令做出响应，改变外观，在运行它的计算机与提供它的计算机之间传递数据。

图 1-1 展示了一个很好的动态网页的例子。Jmol applet 显示了分子结构，这将需要相当复杂的计算。在这个网页中，可以利用鼠标进行旋转，调整焦距等操作，以便更好地理解分子结构。用静态网页就无法实现这种直接的操作，而 applet 却可以达到此目的（可以在 <http://jmol.sourceforge.net> 上找到这个 applet）。

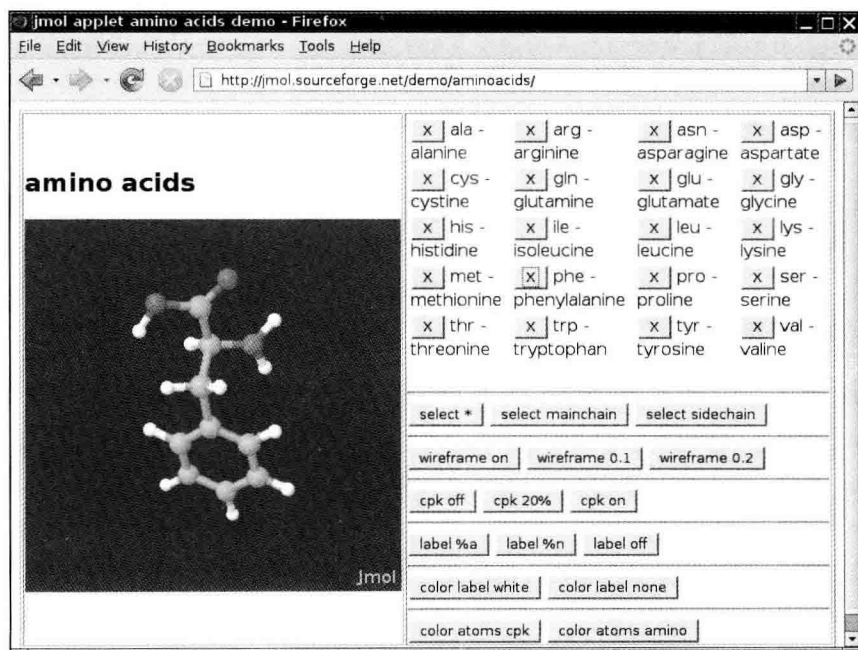


图 1-1 Jmol applet

当 applet 首次出现时，人们欣喜若狂。许多人相信 applet 的魅力将会导致 Java 迅速地流行起来。然而，初期的兴奋很快就淡化了。不同版本的 Netscape 与 Internet Explorer 运行不同版本的 Java，其中有些早已过时。这种糟糕的情况导致更加难于利用 Java 的最新版本开发 applet。实际上，为了在浏览器中得到动态效果，Adobe 的 Flash 技术变得相当流行。后来，Java 遭遇了严重的安全问题，浏览器和 Java 浏览器插件变得限制越来越多。如今，要在浏览器中使用 applet，这不仅需要一定的水平，而且要付出努力。例如，如果访问 Jmol 网站，可能会看到一个消息，警告你要适当地配置浏览器允许运行 applet。

## 1.4 Java 发展简史

本节将介绍 Java 的发展简史。这些内容来自很多出版资料（最重要的是 SunWorld 的在线杂志 1995 年 7 月刊上对 Java 创建者的专访）。

Java 的历史要追溯到 1991 年，由 Patrick Naughton 和 James Gosling（一个全能的计算机奇才）带领的 Sun 公司的工程师小组想要设计一种小型的计算机语言，主要用于像有线电视转换盒这类的消费设备。由于这些消费设备的处理能力和内存都很有限，所以语言必须非常小且能够生成非常紧凑的代码。另外，由于不同的厂商会选择不同的中央处理器（CPU），因此这种语言的关键是不与任何特定的体系结构捆绑在一起。这个项目被命名为“Green”。

代码短小、紧凑且与平台无关，这些要求促使开发团队设计一个可移植的语言，可以为虚拟机生成中间代码。

不过，Sun 公司的人都有 UNIX 的应用背景。因此，所开发的语言以 C++ 为基础，而不是 Lisp、Smalltalk 或 Pascal。不过，就像 Gosling 在专访中谈到的：“毕竟，语言只是实现目标的工具，而不是目标本身”。Gosling 把这种语言称为“Oak”（这么起名的原因大概是因为他非常喜欢自己办公室外的橡树）。Sun 公司的人后来发现 Oak 是一种已有的计算机语言的名字，于是，将其改名为 Java。事实证明这是一个很有灵感的选择。

1992 年，Green 项目发布了它的第一个产品，称之为“\*7”。这个产品具有非常智能的远程控制。遗憾的是，Sun 公司对生产这个产品并不感兴趣，Green 项目组的人员必须找出其他的方法来将他们的技术推向市场。然而，没有一个标准消费品电子公司对此感兴趣。于是，Green 项目组竞标了一个提供视频点播等新型服务的有线电视盒的项目，但没有成功（有趣的是，得到这个项目的公司的领导恰恰是开创 Netscape 公司的 Jim Clark。Netscape 公司后来对 Java 的成功给予了很大的帮助）。

Green 项目（这时换了一个新名字——“First Person 公司”）花费了 1993 年一整年以及 1994 年的上半年，一直在苦苦寻求其技术的买家。然而，一个也没有找到（Patrick Naughton，项目组的创立人之一，也是完成此项目大多数市场工作的人，声称为了销售这项技术，累计飞行了 300 000 英里）。1994 年 First Person 公司解散了。

当这一切在 Sun 公司发生的时候，Internet 的万维网也在日渐发展壮大。万维网的关键是把超文本页面转换到屏幕上的浏览器。1994 年大多数人都在使用 Mosaic，这是一个 1993



年出自伊利诺斯大学超级计算中心的非商业化的 Web 浏览器 (Mosaic 的一部分是由 Marc Andreessen 编写的。当时,他作为一名参加半工半读项目的本科生,编写了这个软件,每小时的薪水只有 6.85 美元。他后来成了 Netscape 公司的创始人之一和技术总监,可谓名利双收)。

在接受 SunWorld 采访的时候, Gosling 说在 1994 年中期, Java 语言的开发者意识到:“我们能够建立一个相当酷的浏览器。我们已经拥有在客户机 / 服务器主流模型中所需要的体系结构中立、实时、可靠、安全——这些在工作站环境并不太重要,所以,我们决定开发浏览器。”

实际的浏览器是由 Patrick Naughton 和 Jonathan Payne 开发的,并演变为 HotJava 浏览器。为了炫耀 Java 语言超强的能力, HotJava 浏览器采用 Java 编写。设计者让 HotJava 浏览器具有在网页中执行内嵌代码的能力。这一“技术印证”在 1995 年 5 月 23 日的 SunWorld 上得到展示,同时引发了人们延续至今的对 Java 的狂热追逐。

1996 年年初, Sun 发布了 Java 的第 1 个版本。人们很快地意识到 Java1.0 不能用来进行真正的应用开发。的确,可以使用 Java 1.0 来实现在画布上随机跳动的神经质的文本 applet,但它却没有提供打印功能。坦率地说, Java 1.0 的确没有为其黄金时期的到来做好准备。后来的 Java 1.1 弥补了其中的大多明显的缺陷,大大改进了反射能力,并为 GUI 编程增加了新的事件处理模型。不过它仍然具有很大的局限性。

1998 年 JavaOne 会议的头号新闻是即将发布 Java 1.2 版。这个版本取代了早期玩具式的 GUI,并且它的图形工具箱更加精细而具有可伸缩性,更加接近“一次编写,随处运行”的承诺。在 1998 年 12 月 Java 1.2 发布三天之后, Sun 公司市场部将其名称改为更加吸引人的“Java 2 标准版软件开发工具箱 1.2 版”。

除了“标准版”之外, Sun 还推出了两个其他的版本:一个是用于手机等嵌入式设备的“微型版”;另一个是用于服务器端处理的“企业版”。本书主要讲述标准版。

标准版的 1.3 和 1.4 版本对最初的 Java 2 版本做出了某些改进,扩展了标准类库,提高系统性能。当然,还修正了一些 bug。在此期间, Java applet 采用低调姿态,并淡化了客户端的应用,但 Java 却成为服务器端应用的首选平台。

5.0 版是自 1.1 版以来第一个对 Java 语言做出重大改进的版本(这一版本原来被命名为 1.5 版,在 2004 年的 JavaOne 会议之后,版本数字升至 5.0)。经历了多年的研究,这个版本添加了泛型类型 (generic type)(类似于 C++ 的模板),其挑战性在于添加这一特性并没有对虚拟机做出任何修改。另外,还有几个受 C# 启发的很有用的语言特性:“for each”循环、自动装箱和注解。

版本 6 (没有后缀 .0) 于 2006 年年末发布。同样,这个版本没有对语言方面再进行改进。但是,改进了其他性能,并增强了类库。

随着数据中心越来越依赖于商业硬件而不是专用服务器, Sun Microsystems 终于沦陷,于 2009 年被 Oracle 收购。Java 的开发停滞了很长一段时间。直到 2011 年 Oracle 发布了 Java 的一个新版本, Java 7,其中只做了一些简单的改进。

2014 年, Java 8 终于发布,在近 20 年中这个版本有了最大的改变。Java 8 提供了一种“函



数式”编程方式，可以很容易地表述并发执行的计算。所有编程语言都必须与时俱进，Java 在这方面显示出非凡的能力。

表 1-1 展示了 Java 语言以及类库的发展状况。可以看到，应用程序编程接口（API）的规模发生了惊人的变化。

表 1-1 Java 语言的发展状况

版 本	年 份	语言新特性	类与接口的数量
1.0	1996	语言本身	211
1.1	1997	内部类	477
1.2	1998	strictfp 修饰符	1524
1.3	2000	无	1840
1.4	2002	断言	2723
5.0	2004	泛型类、“for each”循环、可变元参数、自动装箱、元数据、枚举、静态导入	3279
6	2006	无	3793
7	2011	基于字符串的 switch、钻石操作符、二进制字面量、异常处理改进	4024
8	2014	lambda 表达式，包含默认方法的接口，流和日期/时间库	4240

## 1.5 关于 Java 的常见误解

在结束本章之前，我们列出了一些关于 Java 的常见误解，同时给出了解释。

### 1. Java 是 HTML 的扩展

Java 是一种程序设计语言；HTML 是一种描述网页结构的方式。除了用于在网页上放置 Java applet 的 HTML 扩展之外，两者没有任何共同之处。

### 2. 使用 XML，所以不需要 Java

Java 是一种程序设计语言；XML 是一种描述数据的方式。可以使用任何一种程序设计语言处理 XML 数据，而 Java API 对 XML 处理提供了很好的支持。此外，许多重要的第三方 XML 工具采用 Java 编写。有关这方面更加详细的信息请参看卷 II。

### 3. Java 是一种非常容易学习的程序设计语言

像 Java 这种功能强大的语言大都不太容易学习。首先，必须将编写玩具式程序的轻松和开发实际项目的艰难区分开来。需要注意的是：本书只用了 7 章讨论 Java 语言。在两卷中，其他的章节介绍如何使用 Java 类库将 Java 语言应用到实际中去。Java 类库包含了数千种类和接口以及数万个函数。幸运的是，并不需要知道它们中的每一个，然而，要想 Java 解决实际问题，还是需要了解不少内容的。

### 4. Java 将成为适用于所有平台的通用性编程语言

从理论上讲，这是完全有可能的。但在实际中，某些领域其他语言有更出色的表现，比

如，Objective C 和后来的 Swift 在 iOS 设备上就有着无可取代的地位。浏览器中的处理几乎完全由 JavaScript 掌控。Windows 程序通常都用 C++ 或 C# 编写。Java 在服务器端编程和跨平台客户端应用领域则很有优势。

#### 5. Java 只不过是另外一种程序设计语言

Java 是一种很好的程序设计语言，很多程序设计人员喜欢 Java 胜过 C、C++ 或 C#。有上百种好的程序设计语言没有广泛地流行，而带有明显缺陷的语言，如：C++ 和 Visual Basic 却大行其道。

这是为什么呢？程序设计语言的成功更多地取决于其支撑系统的能力，而不是优美的语法。人们主要关注：是否提供了易于实现某些功能的易用、便捷和标准的库？是否有开发工具提供商能建立强大的编程和调试环境？语言和工具集是否能够与其他计算基础架构整合在一起？Java 的成功源于其类库能够让人们轻松地完成原本有一定难度的事情。例如：联网 Web 应用和并发。Java 减少了指针错误，这是一个额外的好处，因此使用 Java 编程的效率更高。但这些并不是 Java 成功的全部原因。

#### 6. Java 是专用的，应该避免使用

最初创建 Java 时，Sun 为销售者和最终用户提供了免费许可。尽管 Sun 对 Java 拥有最终的控制权，不过在语言版本的不断发展和新库的设计过程中还涉及很多其他公司。虚拟机和类库的源代码可以免费获得，不过仅限于查看，而不能修改和再发布。Java 是“闭源的，不过可以很好地使用”。

这种状况在 2007 年发生了戏剧性的变化，Sun 声称 Java 未来的版本将在 General Public License (GPL) 下提供。Linux 使用的是同一个开放源代码许可。Oracle 一直致力于保持 Java 开源。只有一点美中不足——专利。根据 GPL，任何人都可以得到专利许可，允许其使用和修改 Java，不过仅限于桌面和服务器平台。如果你想在嵌入式系统中使用 Java，就需要另外一个不同的许可，这很可能需要付费。不过，这些专利在未来十年就会到期，那时 Java 就完全免费了。

#### 7. Java 是解释型的，因此对于关键的应用程序速度太慢了

早期的 Java 是解释型的。现在 Java 虚拟机使用了即时编译器，因此采用 Java 编写的“热点”代码其运行速度与 C++ 相差无几，有些情况下甚至更快。

对于 Java 桌面应用速度慢，人们已经抱怨很多年了。但是，今天的计算机速度远比人们发出抱怨的时候快了很多。一个较慢的 Java 程序与几年前相当快的 C++ 程序相比还要快一些。

#### 8. 所有的 Java 程序都是在网页中运行的

所有的 Java applet 都是在网页浏览器中运行的。这也恰恰是 applet 的定义，即一种在浏览器中运行的 Java 程序。然而，大多数 Java 程序是运行在 Web 浏览器之外的独立应用程序。实际上，很多 Java 程序都在 Web 服务器上运行并生成用于网页的代码。

#### 9. Java 程序是主要的安全风险

对于早期的 Java，有过关于安全系统失效的报道，曾经一度引起公众哗然。研究人员将

这视为一种挑战，即努力找出 Java 的漏洞，对 applet 安全模型的强度和复杂度发起挑战。随后，人们很快就解决了引发问题的所有技术因素。后来又发现了更严重的漏洞，而 Sun 以及后来的 Oracle 反应却过于迟缓。浏览器制造商则有些反应过度，他们甚至默认禁用了 Java。客观地来讲，可以想想针对 Windows 可执行文件和 Word 宏有数百万种病毒攻击，并造成了巨大的损害，不过奇怪的是却很少有人批评被攻击平台的脆弱。

有些系统管理员甚至在公司浏览器中禁用了 Java，而同时却允许用户下载可执行文件和 Word 文档，实际上，这些带来的风险远甚于使用 Java。尽管距离 Java 诞生已经 20 年之久，与其他常用的执行平台相比，Java 还是安全得多。

#### 10. JavaScript 是 Java 的简易版

JavaScript 是一种在网页中使用的脚本语言，它是由 Netscape 发明的，原来的名字叫做 LiveScript。JavaScript 的语法类似 Java，除此之外，两者无任何关系。当然，名字有些相像。JavaScript 的一个子集已经标准化为 ECMA-262。与 Java applet 相比，JavaScript 更紧密地与浏览器集成在一起。特别是 JavaScript 程序可以修改正在显示的文档，而 applet 只能在有限的区域内控制外观。

#### 11. 使用 Java 可以用廉价的 Internet 设备取代桌面计算机

当 Java 刚刚发布的时候，一些人打赌：肯定会有这样的好事情发生。一些公司已经生产出 Java 网络计算机的原型，不过用户还不打算放弃功能强大而便利的桌面计算机，而去使用没有本地存储而且功能有限的网络设备。当然，如今世界已经发生改变，对于大多数最终用户，常用的平台往往是手机或平板电脑。这些设备大多使用安卓平台，这是 Java 的衍生产物。学习 Java 编程肯定也对 Android 编程很有帮助。