第 1 章　Java 程序设计概述

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

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，但还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。然而，现在看起来，当初的这些预测还是有一定准确性的。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种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 恰恰满足了这些要求。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 这种功能齐全的出

色语言，具有高质量的执行环境以及庞大的库。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
2）面向对象
3）网络技能（Network-Savvy）
4）健壮性
5）安全性
6）体系结构中立
7）可移植性
8）解释型
9）高性能
10）多线程
11）动态性

本节将论述下列主要内容：

- 给出白皮书中对每个关键术语的概述，这是Java设计者对相关术语的论述。
- 凭借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.上找到。对于11个关键术语的论述请参看http://labs.oracle.com/features/tenyeares/volcd/papers/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，那里比较详细地解释了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。如果知道C++就会发现可以轻而易举地将其转换成Java。

如果已经习惯于使用可视化的编程环境（例如Visual Basic），你就不会觉得Java简单了。Java有许多奇怪的语法（尽管掌握其要领并不需要很长时间），更重要的是，使用Java需要自己编写大量的程序。Visual Basic的魅力在于它的可视化设计环境几乎自动地为应用程序提供了大量的基础结构。而使用Java实现同样的功能却需要手工编制代码，通常代码量还相当大。然而，已经有一些支持“拖放”风格程序开发的第三方开发环境。

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）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。

在过去的 40 年里，面向对象已经证明了自身的价值，一种现代的程序设计语言不使用面向对象技术简直让人难以置信。的确，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继承，在 Java 中，取而代之的是简单的接口概念，以及 Java 的元类（metaclass）模型（有关这部分内容将在第 5 章中讨论）。

注释：如果没有使用面向对象程序设计语言的经验，你一定要仔细阅读第 4 章～第 6 章。这些章节解释了什么是面向对象程序设计以及在编程实现复杂的项目时为什么比传统的像 C 或 Basic 这样的面向过程的语言更加有效。

1.2.3 网络技能

Java 有一个扩展的例程库，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。

人们已经看到 Java 的网络能力强大且易于使用。任何曾经试图使用其他语言进行网络编程的人都会惊呼 Java 竟然把类似打开 socket 连接这类繁重的任务都变得如此简单（在本书的卷 II 中介绍网络连接）。另外，远程方法调用机制使得分布式对象之间可以进行通信（也将在卷 II 中介绍）。

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，并消除了有出错倾向的状态……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。

这个特性非常有用。Java 编译器能够检测许多在其他语言中仅在运行时刻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。

如果曾经只使用过 Visual Basic 这类没有显式指针的语言，你就会感觉这么说似乎有些小题大做了。然而，C 程序员就没有这样幸运了。他们需要利用指针存取字符串、数组、对象，甚至文件。在 Visual Basic 中，根本不必使用指针访问这些实体，也不必关心有关内存分配的问题。另一方面，在没有指针的语言中，许多数据结构很难实现。Java 具有双方的优势。它不需要使用指针构造诸如字符串、数组这样的结构。如果必要的话，它也能够具有指针的能力，如链表。Java 绝对是安全的，其原因是永远不会存取一个“坏的”指针，造成内存分配的错误，也不必防范内存泄漏。

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。

本书的第 1 版曾经说过：“永远不要把话说绝！”事实证明这是正确的。在 Java 开发工具箱第 1 版启用后不久，普林斯顿大学的一些安全专家就发现了在 JDK1.0 中的某些安全特性方面存在着一些非常隐蔽的 bug。Sun Microsystems 大力支持对 Java 的安全性的研究，制定了供人们使用的规范，实现了虚拟机和安全库，并迅速地处理了所有已知的安全 bug。在任何情况下，蒙骗 Java 的安全机制都是十分困难的。现在，发现 bug 的技术越来越强，数目越来越少。

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，蠕虫等病毒常用的攻击手段。
- 在自己的处理空间之外破坏内存。
- 未经授权读写文件。

许多安全特性相继不断地加入到 Java 中。自从 Java 1.1 问世以来，Java 就有了数字签名类（digitally signed class）的概念（请参看卷 II）。通过数字签名类，可以确定类的作者。如果信任这个类的作者，这个类就可以在你的机器上拥有更多的权限。

注释：来自微软的基于 ActiveX 技术的竞争代码传输机制，其安全性完全依赖于数字签名。这显然是不够的，因为微软自身产品的任何用户都可以证实，来自知名提供商的程序会崩溃并对系统产生危害。Java 的安全机制比 ActiveX 要强得多，因为它是在应用程序运行时加以控制并制止恶意性破坏的。

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，就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，而且还可以迅速地翻译成本地机器的代码。

这并不是什么新的思路。40 多年以前，Niklaus Wirth 实现的原始 Pascal 以及 UCSD Pascal 系统都使用了这种技术。

当然，解释字节码肯定会比全速运行机器指令慢很多。所以说，这是不是一个好的思路还很难讲！然而，虚拟机有一个选项，可以将使用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。这一策略已经证明十分有效，致使微软的 .NET 平台也依赖于虚拟机。

虚拟机还有一些其他的优点。虚拟机可以检测指令序列的行为，以增强其安全性。有些程序还可以快速地生成字节码，并动态地增强所运行程序的处理能力。

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算法都做了明确的说明。

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 类给出了在 UNIX、Windows 和 Macintosh 环境下的不同实现。

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箱。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（而且，在不同平台的图形实现中有不同的 bug）。不过，这毕竟是个开端。对于许多应用问题来说，可移植性比华而不实的用户界面更加重要；而且这些应用程序从 Java 的早期版本中获益匪浅。现在，用户界面工具箱已经完全重写了，不再依赖于主机的用户界面。现在的 Java 版本比早期版本更加稳定，更加吸引人。

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。

增量式链接有其优势，但给开发过程带来的好处显然是言过其实了。事实上，早期的 Java 开发工具的速度相当慢。现在，使用即时编译器将字节码翻译成机器码。

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（在运行时刻）快速地翻译成运行这个应用程序的特定 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，其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“内嵌”）。即时编译器知道哪些类已经加载。如果基于当前加载的类集，且特定的函数不被覆盖的话就可以内嵌。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。

如果曾经使用过其他语言编写多线程的应用程序，就会对Java多线程处理的便捷性惊叹不已。只要操作系统支持，Java中的线程就可以利用多个处理器。在底层，主流平台的线程实现机制各不相同，Java并没有花费太大的力气对此实现平台无关性。在不同的机器上，只是调用多线程的代码完全相同；Java把多线程的实现交给了底层的操作系统或线程库来完成。尽管如此，多线程编译的简单性是Java成为颇具魅力的服务器端开发语言的主要原因之一。

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上下载代码，然后在浏览器上运行。在Java 1.0中，不能直接获得运行时的类型信息，而Java的当前版本允许程序员知道对象的结构和行为。这对于必须在运行时分析对象的系统来说非常有用。这些系统有：Java GUI构建器、智能调试器、可插拔组件以及对象数据库。

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被称为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。甚至还有一种J#语言可将J++的应用迁移到使用C#的虚拟机上。本书不准备介绍J++、C#或J#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Java程序称为applet。为了使用applet，需要启用Java的Web浏览器执行字节码。由于Sun公司负责发放Java源代码的许可证，并坚持不允许对语言和基本类库的结构做出任何修改，因此，Java的applet应该可以运行在任何启用Java浏览器上，并且无论何时访问包含

applet 的网页，都会得到程序的最终版本。最重要的是，要感谢虚拟机的安全机制，它让我们不必再担心来自恶意代码的攻击。

用户下载一个 applet 就如同在网页中嵌入一幅图片。applet 成了页面的一部分。文本环绕着 applet 所占据的空间周围。关键一点是图片是活动的。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示例。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，以便更加细致地理解分子结构。用静态网页将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）。

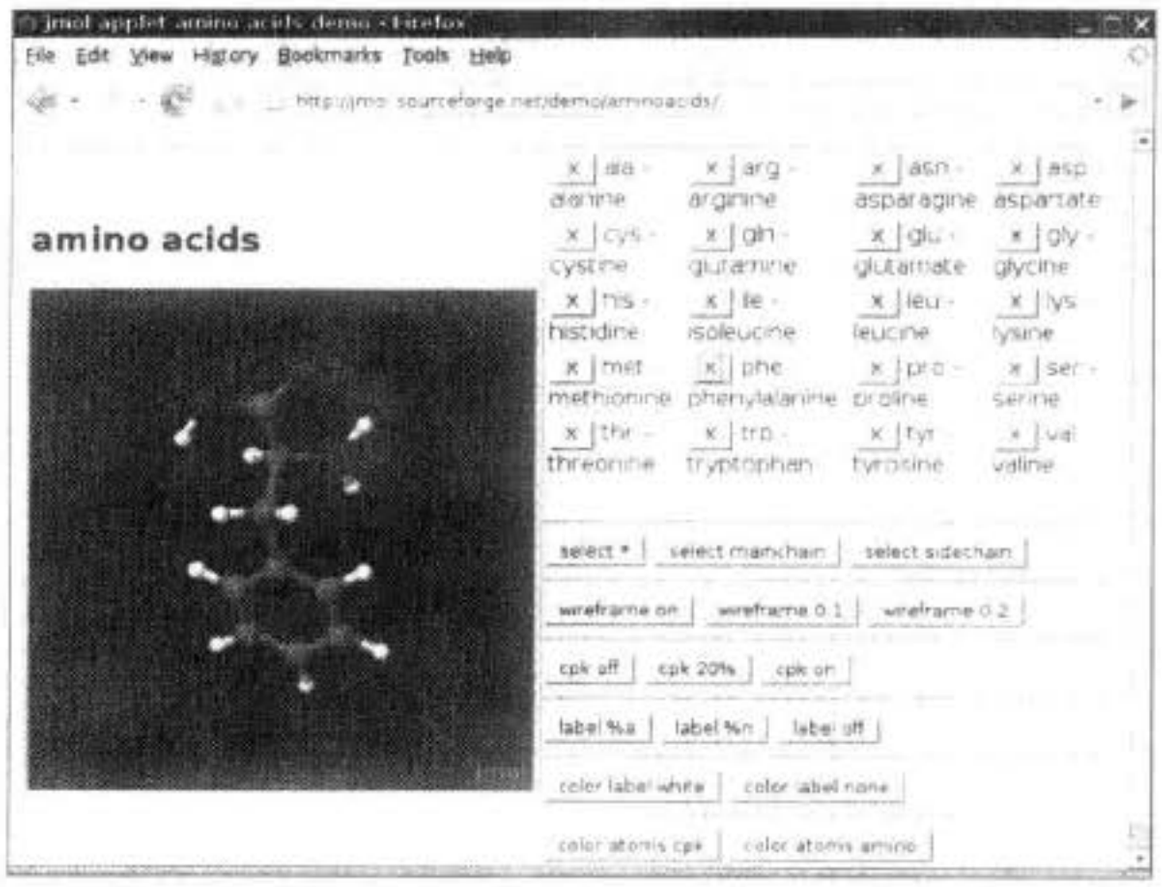

图 1-1 Jmol applet

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。今天，当需要在浏览器中显示动态效果时，大多数网页都直接使用 JavaScript 或 Flash。另外，Java 已经成为用来开发服务器端应用程序的最流行的语言，使用这些服务器端应用程序可以产生网页、运行后端逻辑。

1.4 Java 发展简史

本节将介绍 Java 的发展简史。这些参考资料来源于多方面的出版物（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 及其伙伴 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能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联想起很早以前的一种模型，某些 Pascal 的实现曾经在早期的 PC 上尝试过这种模型。以 Pascal 的发明者 Niklaus Wirth 为先驱，率先设计出一种为假想的机器生成中间代码的可移植语言（假想的机器称为虚拟机——Java 虚拟机即 JVM 的命名由此而来）。这种中间代码可以应用于所有已经正确安装解释器的机器上。Green 项目工程师也使用了虚拟机，从而解决了课题中的主要问题。

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 Pascal。特别是这种语言是面向对象的，而不是面向过程的。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。事实证明这是一个极好的选择。

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（其能力相当于长 6 英寸、宽 4 英寸、高 4 英寸的 SPARC 工作站）。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 公司中继续进行的时候，Internet 的万维网也日渐发展壮大。Web 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：“我们能够建立一个最酷的浏览器。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。然而，设计者也非常清

楚现在所谓的 applet 的威力，因此他们让 HotJava 浏览器具有执行网页中内嵌代码的能力。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 Java1.1 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、自动装箱和元数据。

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，而决定将重要的改进推迟到 Java 8，该版本将在 2013 年发布。

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	无	1524
1.3	2000	无	1840
1.4	2002	断言	2723

（续）

版 本	年 份	语言新特性	类与接口的数量
5.0	2004	泛型类型、“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、变形操作符、二进制字面量、异常处理改进	4024

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，就不需要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4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与几万个方法。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。的确，除了微软之外的每一个厂商都希望如此。然而，有很多在桌面计算机上已经工作良好的应用程序，在其他设备上或浏览器中或许不能正常地工作。同时，在编写这些应用程序时，利用了相应处理器的速度和本地的用户接口库，而且它们已经移植到所有重要的平台上。这类应用程序包括：字处理程序、图片编辑器以及Web浏览器。它们通常是用C或C++编写的，采用Java语言重新编写似乎对最终的用户不会带来什么特别的好处。

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？是否拥有强大的程序设计能力与调试工具？语言和工具是否能够与计算机的其他基础结构整合在一起？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：联网和多线程。Java减少了指针错误，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）现在有了C#，Java过时了。

C#借鉴了Java许多好的思想，例如：清晰的语言结构、虚拟机和垃圾收集。无论怎样，C#还是省去了一些好的特性，其中最重要的是安全性和平台无关性。如果确定使用Windows，C#就更有意义。但是，从求职广告判定，Java仍然是大多数开发者选择的语言。

7）Java是专用的，应该避免使用。

Sun Microsystems负责将Java的许可发放给销售者以及最终用户。尽管Sun公司通过Java Community Process最终控制着Java，但他们同时与许多其他的公司联手一起进行着语言修订版的开发及新类库的设计。虚拟机和类库的源代码都可以免费获取，但是，只能查阅，不能修改，也不能再发布。因此，Java是“闭源，但运转良好”。

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下可用。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。

8）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。现在除了像手机这样的“微型”平台之外，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。

Java有一些C++没有的额外开销。虚拟机的启动时间要慢一些，并且Java GUI要比本地的GUI慢一些，这是因为它们采用了与平台无关的绘图方式。

对于Java比C++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。就这一点来说，那些抱怨听起来有点像狐狸抱怨葡萄酸，有些人已经转过来攻击Java用户界面不够漂亮而不再攻击速度慢了。

9）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，即一种在网页中运行的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。

10）Java程序是主要的安全风险。

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。大多数安全问题都存在于Java的特定浏览器中。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。据我们所知，任何实用系统都有安全危机。想想看：毫不夸张地说，有数百万种病毒攻击着 Windows 的可执行文件和 Word 宏，这给系统造成了巨大的损害，但却很少有人批评被攻击平台的脆弱。同样，Internet Explorer 中的 ActiveX 机制始终作为被攻击的目标，但由于阻止这种攻击非常简单，所以人们也就懒得将它们公布于众了。

有些系统程序员在公司的浏览器中禁用 Java，却允许其用户下载可执行文件、ActiveX 控件和 Word 文档。这是多么荒唐可笑啊！事实上这会带来更大的风险。尽管距离 Java 的诞生已经 15 年之久，但与其他常用的执行平台相比，还是 Java 安全得多。

11）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。

12）使用 Java 可以用价值 500 美元的 Internet 设备取代电脑。

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。从本书的第 1 版开始，我们就已经认定“家庭用户将会放弃功能强大且便利的桌面系统，而使用没有本地存储的网络设备”是一种荒谬的想法。我们发现基于 Java 的网络计算机，对利用“零管理”降低计算机所有者的商业成本是一种很好的选择。即便如此，这种好事也没有发生。新一代的平板电脑并没有使用 Java。