第1章 概 述

1.1 什么是 Java 语言

1991 年，美国 Sun Microsystems 公司的 Jame Gosling、Bill Joe 等人为在电视、烤箱等家用消费类电子产品上进行交互式操作而开发了一个名为 Oak 的语言。之后 Sun 的开发人员将 Oak 改为 Java。互联网的出现给 Java 带来了生机。因此，在 1995 年，Sun 向公众推出 Java 时曾引起业界的巨大轰动。Java 语言面向网络应用，其类库不断丰富，性能不断提高，应用领域也不断拓展，到目前已成为当今最通用最流行的软件开发语言之一，是许多专业人员首选的开发语言。Sun 公司为开发人员提供了软件开发工具包(Software Development Kit，SDK)，并不断进行更新。目前提供的是 Java 2 平台，相应的工具包版本是 J2SDK，读者可以从 Sun 公司的网站上 http://java.sun.com/j2se 查询当前最新的版本。

那么，Java 到底是什么？为什么它一问世就引起计算机界如此强烈的反响呢？实际上，Java 是一种功能强大的程序设计语言，它既是开发环境，又是应用环境，它代表一种新的计算模式。特别是从 1993 年开始互联网的流行，为 Java 提供了发挥潜能的机会。图 1-1 说明了 Java 语言的基本概念。

Java 语言	面向对象的程序设计语言
	与机器无关的二进制格式的类文件
	Java 虚拟机(用来执行类文件)
	完整的软件程序包(跨平台的 API 和库)

图 1-1 Java 的基本概念

Java 是简单的、面向对象的语言，它具有分布性、安全性和健壮性等特点。它的最初版本是解释执行，现在的版本中增加了编译执行，所以它具有高性能；它是多线程的、动态的语言；最主要的是它与平台无关，解决了困扰软件界多年的软件移植问题。Java 语言既具有 C++ 的功能，又具备对类型进行严格检查的安全特性。

1.1.1 Java 语言的特点

Java 语言自诞生之日起，就受到了全世界的关注。Java 的出现标志着一个新的计算时代的到来，即 Java 计算时代。Java 的众多突出特点使得它受到了大众的欢迎。实际上，Java 符合目前面向对象程序设计的主流，它与 Web 及 Internet 的结合紧密，具有动画、声音等功能，能实时处理信息。它有如下显著的特点：

1. 语法简单，功能强大

Java 是一种类似于 C++ 的语言，熟悉 C++ 的程序设计人员，不需花费太多的精力就可以掌握 Java。Java 去掉了 C++ 中不常用且容易出错的地方。例如，Java 中没有指针、结构和类型定义等概念，不再有全局变量，没有 #include 和 #define 等预处理器，也没有多重继承的机制。程序员不用自己释放占用的内存空间，因此不会引起因内存混乱而导致的系统崩溃。

Java 强调了面向对象的特性，是一个完全的面向对象的语言，它对软件工程技术有很强的支持。Java 语言的设计集中于对象及其接口，它提供了简单的类机制以及动态的接口模型。Java 的对象中封装了它的状态变量以及相应的方法，实现了模块化和信息隐藏。另一方面，Java 的类提供了一类对象的原型，通过继承机制，子类可以使用父类所提供的方法，实现了代码的复用。因此使用 Java 可以编制出非常复杂的系统。即便如此，Java 的解释器只占用很少的内存，适合在绝大多数类型的机器上运行。

2. 分布式与安全性

Java 从诞生之日起就与网络联系在一起，它强调网络特性，内置 TCP/IP、HTTP、FTP 协议类库，便于开发网上应用系统。Java 程序在语言定义阶段、字节码检查阶段及程序执行阶段进行的三级代码安全检查机制，对参数类型匹配、对象访问权限、内存回收、Java 小应用程序的正确使用等都进行了严格的检查和控制，可以有效地防止非法代码的侵入，阻止对内存的越权访问，能够避免病毒的侵害。

3. 与平台无关

提到 Java，有一句著名的口号："一次编写，到处运行"。这反映了 Java 的平台无关性，实现了编程人员多年的梦想。Java 可以跨平台使用，因此更适合于网络应用。Java 语言规定了统一的数据类型，有严格的语言定义，为 Java 程序跨平台的无缝移植提供了很大的便利。Java 编译器将 Java 程序编译成二进制代码，即字节码(bytecode)。不同的操作系统有不同的虚拟机，它类似于一个小巧而高效的 CPU。字节码就是虚拟机的机器指令，与平台无关。字节码有统一的格式，不依赖于具体的硬件环境。在任何安装 Java 运行环境的系统上，都可以执行这些代码。运行时环境针对不同的处理器指令系统，把字节码转换为不同的具体指令，保证了程序的"到处运行"。

4. 解释编译两种运行方式

Java 程序可以经解释器得到字节码，所生成的字节码经过了精心设计，并进行了优化，因此运行速度较快，突破了以往解释性语言运行效率低的瓶颈。在现在的 Java 版本中又加入了编译功能(即 just-in-time 编译器，简称 JIT 编译器)，生成器将字节代码转换成本机的机器代码，然后可以以较高速度执行，使得执行效率大幅度提高，基本达到了编译语言的水平。

5. 多线程

单线程程序一个时刻只能做一件事情，多线程程序允许在同一时刻同时做多件事情。大多数高级语言，包括 C、C++ 等，都不支持多线程，只能编写顺序执行的程序（除非有操作系统 API 的支持）。Java 内置了语言级多线程功能，提供现成的类 Thread，只要继承这个类就可以编写多线程的程序，可使用户程序并行执行。Java 提供的同步机制可保证各线程对共享数据的正确操作，完成各自的特定任务。在硬件条件允许的情况下，这些线程可以直接分布到各个 CPU 上，充分发挥硬件性能，减少用户等待时间。

通过使用多线程，程序设计者可以分别用不同的线程完成特定的行为，而不需要采用全局的事件循环机制，这样就很轻松地实现网络上的实时交互行为。

6. 动态执行

Java 执行代码是在运行时动态载入的，程序可以自动进行版本升级。在网络环境下，可用于瘦客户机架构，减少维护工作。另外，类库中增加的新方法和其他实例，不会影响到原有程序的执行。

7. 丰富的 API 文档和类库

Java 为用户提供了详尽的 API 文档说明。Java 开发工具包中的类库包罗万象，应有尽有，程序员的开发工作可以在一个较高的层次上展开。这也正是 Java 受欢迎的重要原因之一。

1.1.2 Java 的三层架构

计算机诞生早期以主机架构为主。它的特点是集中处理，集中管理，各用户分享使用计算机资源。这种模式下，大部分应用的可移植性差，扩充系统计算能力所花的费用较大，当系统进行维护或因客观环境等因素导致的系统关机会给用户带来不便。20 世纪 70 年代末 80 年代初出现的个人计算机（PC），全面改变了以往的计算模式，实行本地处理，本地管理，各用户独占系统资源，不共享，机器使用时间完全按自己的日程而定。这种机制虽然满足了个性化的要求，但随着 PC 机数量的日益增多及其应用操作日益复杂，PC 机的管理维护费用直线上升。据调查，PC 机的平均管理维护费用远远超出了其最初购置费。

Java 计算模式结合了上述两种模式的优势，它可用于客户/服务器架构，将公共使用的程序放到应用程序服务器上，用户使用时从服务器上下载到客户端，各用户独立使用设备和程序。当程序更新时，只需在服务器上进行。之后，客户再使用时，下载的就是更新后的版本，系统管理员不必在客户端做任何维护工作，达到“零管理”的理想目标。

1.1.3 Java 语言的目标

最初开发 Java 语言时有明确的目标，这些也正是它区别于其他语言的独特之处：

- 创建一种面向对象的语言。面向对象的程序设计方法已成为主流，Java 语言的语

法及程序结构正是采用了这样的设计方法。

- 提供一种解释环境，这可以缩短系统开发的编译—连接—装载—测试的周期，提高开发速度。
- 去掉了影响代码健壮性的功能，如指针结构及程序员负责的内存分配与释放。
- 为程序运行多线程提供了方法。
- 允许程序下载代码模块，这样在程序运行生命期内可以动态修改它们。
- 检查下载的代码模块，提供了保证安全的一种手段。

Java 虚拟机、Java 独特的垃圾收集机制及三级代码安全检查机制，使得 Java 得以实现这个目标。后面将分别讲述这些问题。

1.1.4 Java 虚拟机

1. Java 虚拟机

Java 虚拟机(Java virtual machine，JVM)是运行 Java 程序必不可少的机制。编译后的 Java 程序指令并不直接在硬件系统的 CPU 上执行，而是由 JVM 执行。JVM 是编译后的 Java 程序和硬件系统之间的接口，程序员可以把 JVM 看作一个虚拟的处理器。它不仅解释执行编译后的 Java 指令，而且还进行安全检查。它是 Java 程序能在多平台间进行无缝移植的可靠保证，同时也是 Java 程序的安全检验引擎。

Java 虚拟机规范中给出了 JVM 的定义：JVM 是在一台真正的机器上用软件方式实现的一台假想机。JVM 使用的代码存储在 .class 文件中。JVM 的某些指令很像真正的 CPU 指令，包括算术运算、流控制和数组元素访问等。

Java 虚拟机规范提供了编译所有 Java 代码的硬件平台。因为编译是针对假想机的，所以该规范能让 Java 程序独立于平台。它适用于每个具体的硬件平台，以保证能运行为 JVM 编译的代码。JVM 不但可以用软件实现，而且可以用硬件实现。

JVM 的具体实现包括：指令集(等价于 CPU 的指令集)、寄存器组、类文件格式、栈、垃圾收集堆、内存区。

JVM 的代码格式为压缩的字节码，因而效率较高。由 JVM 字节码表示的程序必须保持原来的类型规定。Java 主要的类型检查是在编译时由字节码校验器完成的。Java 的任何解释器必须能执行符合 JVM 定义的类文件格式的任何类文件。

Java 虚拟机规范对运行时数据区域的划分及字节码的优化并不做严格的限制，它们的实现依平台的不同而有所不同。JVM 的实现叫做 Java 运行时系统或运行时环境(runtime environment)，简称为运行时。Java 运行时必须遵从 Java 虚拟机规范，这样，Java 编译器生成的类文件才可被所有 Java 运行时系统下载。嵌入了 Java 运行时系统的应用程序，就可以执行 Java 程序了。目前有许多操作系统和浏览器都嵌入了 Java 运行时环境。

2. Java 虚拟机的性能

Java 在问世之初，因其没有完全优化，并且是解释执行，所以 Java 程序的运行效率较

低。同时，有着较长发展史、已非常成熟的 C++ 语言仍在开发界扮演着主要角色，人们往往拿 C++ 的性能效率与刚诞生的 Java 相比较，这当然失之偏颇。

Java 解释器经过不断的优化，字节码的执行速度已有很大提高。另外，在字节码执行之前可以先经过 JIT 编译器进行编译，生成针对具体平台的本机执行代码。它的执行效率比解释执行的效率有了大幅度提高。现在许多厂商都提供 JIT 编译器，这项技术已非常成熟。由于字节码与平台无关，所以经过编译的 Java 仍不失跨平台的特点。

Hotspot 技术是 Sun 公司推出的另一个有特色的技术。它提供对代码的运行时选择，为的是从根本上解决 Java 程序的效率问题。在程序执行时，Hotspot 对每个字节码指令进行分析，根据它的执行次数，动态决定它的执行方式。比如，一段指令需要多次重复执行，则立即编译为可执行代码。如果是只执行一次的简单指令，且解释执行的效率更高，则使用解释执行的方式。有了这项技术，Java 的效率问题基本上可以得到解决。

1.1.5 垃圾收集

许多程序设计语言允许在运行时动态分配内存。分配内存的过程因各种语言的语法不同而有所不同，但总要返回指向内存块开始地址的指针。

一旦不再需要所分配的内存(指向内存的指针超出使用范围)，程序或运行时环境最好将内存释放，避免内存越界时得到意外结果。

在 C 和 C++(及其他语言)中，由程序开发人员负责内存的释放。这是个很恼人的事情，因为程序开发人员并不总是知道内存应该在何时释放。如果不释放内存，那么当系统中没有内存可用时程序会崩溃。这些程序被称为有“内存漏洞”。

在 Java 中，程序员不必亲自释放内存，它提供了后台系统级线程，记录每次内存分配的情况，并统计每个内存指针的引用次数。在 Java 虚拟机运行时环境闲置时，垃圾收集线程将检查是否存在引用次数为 0 的内存指针，引用次数为 0 即意味着没有程序再使用这块内存；如果有这样的内存的话，则垃圾收集线程把该内存“标记”为“清除”(释放)，就是要归还给系统，留待下次再分配给其他的内存申请。

在 Java 程序生存期内，垃圾收集将自动进行，无需用户释放内存，从而消除了内存漏洞。编制程序时，程序员可以将注意力放在更需注意的地方，不仅如此，程序的调试也更加方便。

1.1.6 代码安全

图 1-2 说明了 Java 程序环境，我们来看看它是如何在这个环境中保证代码安全的。

Java 语言是解释执行的，但从某种意义来讲，Java 文件是“编译”的，因为它也生成了中间语言形式的文件。经过“编译”的 Java 目标代码，称为字节码，存储在.class 文件中。字节码是不依赖于机器硬件平台的二进制代码。

运行时，类下载器下载组成 Java 程序的字节码，在解释器中检查并运行它们。在某些 Java 运行时环境中，验证过的字节码也可以被编译为本地的机器码，并直接在硬件平台上执行。

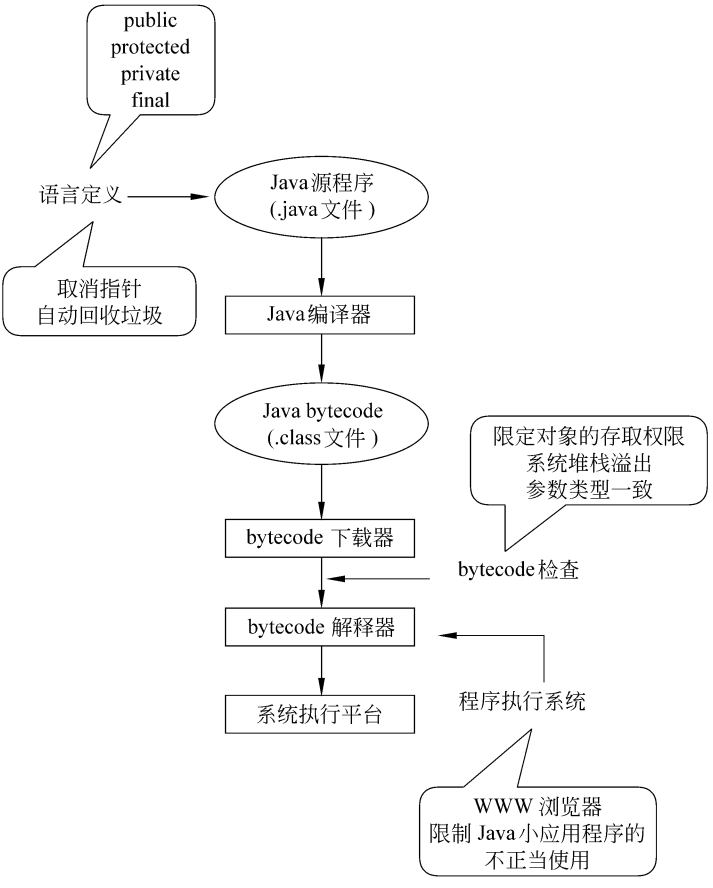

图 1-2　Java 代码安全

对于 Applet，因其是从其他机器上通过网络下载到本机执行的，程序中可能隐藏某些非法操作，所以在 Applet 运行之前，系统要对之进行严格的三级代码安全检查，即验证、分析和跟踪监测。第一级验证是在类下载时完成的，检查从哪里下载文件，是否有权限进到本机系统。然后进行第二级检查，即字节码校验，此时要分析下载的字节码是否合乎规则。如果字节码的格式不合要求，则拒绝执行。完全合乎规则的字节码才允许执行。执行的时候，安全管理器始终监测所执行的每步操作，检查其合法性。Java 的安全检查可以全面提高操作系统的安全等级，经过这三级安全检查的 Java 程序不会受到病毒的侵害。目前很多系统在做安全检查时只做第一步，即看代码下载的权限是否合乎要求。与之相比，Java 的三级安全检查机制要完备得多，它不仅进行静态检查，还要进行动态跟踪。

具体地说，在 Java 的不同版本中，都有不同的安全机制。在最初的 JDK 1.0 版本中，安全模型是所谓的“沙箱”模型，从网络上下载的代码只能在一个受限的环境中运行，这个环境像个箱子一样限制了代码能访问的资源。

在后来的 JDK 1.1 版本中，提出了“签名 Applet”的概念。有正确签名的 Applet 视同本地代码一样，可以使用本地的资源。没有签名的 Applet 还与前一版本一样，只在沙

箱中运行。

在 Java 2 平台下，安全机制又有较大改善。它允许用户自己设定相关的安全级别。另外，对于应用程序，也采取了和 Applet 一样的安全策略，程序员可以根据需要对本地代码或是远程代码进行设定，以保证程序更安全高效地运行。

在 Java 程序环境中，重要的几个组成部分包括 Java 解释器、类下载器及字节码校验器。

1. Java 解释器

Java 解释器只能执行为 JVM 编译的代码。Java 解释器有三项主要工作：

- 下载代码——由类下载器完成。
- 校验代码——由字节码校验器完成。
- 运行代码——由运行时解释器完成。

2. 类下载器

Java 运行时系统区别对待来自不同源的类文件。它可能从本地文件系统中下载类文件，也可能从 Internet 网上使用类下载器下载类文件。运行时系统动态决定程序运行时所需的类文件，并把它们下载到内存中，将类、接口与运行时系统相连接。类下载器把本地文件系统的类名空间和网络源输入的类名空间区分开来，以增加安全性。因为内置的类总是先被检查，所以可以防止起破坏作用的应用程序的侵袭。

所有的类下载完毕后，开始确定可执行文件的内存分配。此时，指定具体的内存地址，并创建查询表。因为内存分配是在运行时进行的，且 Java 解释器阻止访问可能给操作系统带来破坏的非法代码地址，从而增加了保护性。

3. 字节码校验器

Java 代码在机器上真正执行前要经过几次测试。程序通过字节码校验器检查代码的安全性，字节码校验器检测代码段的格式，并使用规则来检查非法代码段——伪造的指针、对目标的访问权限违例或是试图改变目标类型或类的代码。通过网络传送的所有类文件都要经过字节码校验器的检验。

字节码校验器要对程序中的代码进行四趟扫描。这可以保证代码将依从 JVM 规范，并且不破坏系统的完整性。校验器主要检查以下几项内容：

- 类遵从 JVM 的类文件格式。
- 不出现访问违例情况。
- 代码不会引起运算栈溢出。
- 所有运算代码的参数类型总是正确的。
- 不会发生非法数据转换，如把整数转换为指针。
- 对象域访问是合法的。

如果完成所有的扫描之后不返回任何错误信息，就可以保证 Java 程序的安全性了。

1.2 一个基本的 Java 应用程序

1.2.1 开发环境的安装

在介绍 Java 应用程序之前，我们先介绍如何安装开发环境。JDK 是 Sun 公司提供的软件包，其中含有编写和运行 Java 程序的所有工具，包括组成 Java 环境的基本构件：Java 编译器 javac、Java 解释器 java、浏览 Applet 的工具 appletviewer 等。编写 Java 程序的机器上一定要先安装 JDK，安装过程中要正确设置 PATH 和 CLASSPATH 环境变量，这样系统才能找到 javac 和 java 所在的目录。

我们以 Windows 环境为例。首先，登录到下列网址：

http://java.sun.com/j2se/1.5.0/download.jsp

下载 Java2 SDK Standard Edition V5.0，得到文件：jdk-1_5_0_04-nb-4_1-win.exe。双击 jdk-1_5_0_04-nb-4_1-win.exe 直接运行，开始安装 JDK 的过程，如图 1-3 所示，按照安装向导进行即可。

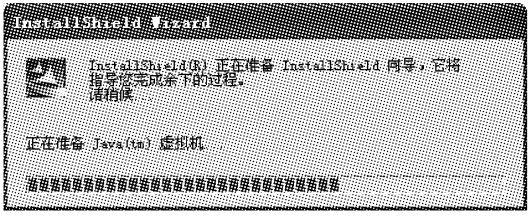

图 1-3 JDK 安装开始

如果 JDK 安装在系统的 C 分区上，安装 JDK 后将产生如下目录结构，如图 1-4 所示。

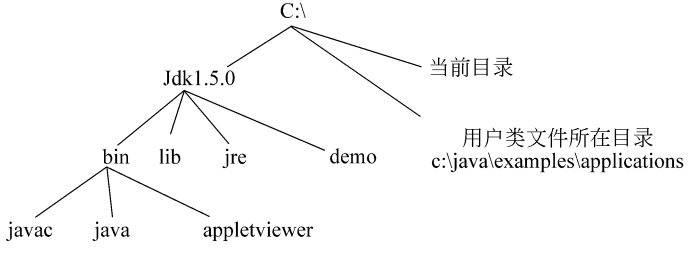

图 1-4 JDK 目录

- \bin 目录：Java 开发工具，包括 Java 编译器、解释器等。
- \demo 目录：一些实例程序。
- \lib 目录：Java 开发类库。
- \jre 目录：Java 运行环境，包括 Java 虚拟机、运行类库等。

bin 目录下的 Java 开发工具包括以下几种：

- Javac：Java 编译器，用来将 Java 程序编译成字节码。
- Java：Java 解释器，执行已经转换成字节码的 Java 应用程序。

- Jdb：Java 调试器，用来调试 Java 程序。
- Javap：反编译，将类文件还原回方法和变量。
- Javadoc：文档生成器，创建 HTML 文件。
- Appletviwer：Applet 解释器，用来解释已经转换成 bytecode 的 Java 小应用程序。

接下来，需要设置环境变量。

PATH = C:\Program Files\Java\jdk1.5.0\bin;

CLASSPATH = . ; C:\ Program Files\Java\jdk1.5.0\LIB;C:\Program Files\Java\jdk1.5.0\demo

如果读者使用的是 Windows 环境，例如 Windows NT/2000/XP，则打开“控制面板”，选择“系统”，进入“高级”，选中“环境变量”，然后分别添加上面两项设置即可。当然，如果读者自己选择独特的安装目录，则上面的设置也要做相应的修改。

1.2.2 Java 应用程序

Java 程序分为两种，一种是 Java 应用程序(Java Application)，另一种是 Java 小应用程序(Java Applet)，或叫 Java 小程序。本书的前几章先介绍 Java 应用程序，在第 9 章介绍小应用程序。

和其他任何程序设计语言一样，Java 语言也可以创建通常意义下的应用程序。下面编制一个仅在屏幕上显示字符串“Hello World!”的最小的应用程序，以此说明 Java 程序的基本结构。该程序存储在 HelloWorldApp.java 文件中。

程序 1-1 一个基本的 Java 应用程序。

```
1 //
2 // 简单的应用程序 HelloWorld
3 //
4 public class HelloWorldApp{
5     public static void main (String args[]) {
6         System.out.println ("Hello World!");
7     }
8 }
```

这些程序行包含了在屏幕上打印“Hello World!”所需的最基本的内容。

程序的前三行是注释行：

```
1 //
2 // 简单的应用程序 HelloWorld
3 //
```

第 4 行为：

```
4 public class HelloWorldApp{
```

这一行说明了一个公有类，类的名字为 HelloWorldApp。类的内容从类名后的花括号“{”开始，到与之匹配的“}”(第 8 行)为止。编译正确后，系统在当前工作目录下创建一

个 classname.class 文件，其中 classname 即用户在源文件中指定的类名。此处，编译器创建的文件名为 HelloWorldApp.class。这是二进制格式的字节码文件。

程序从第 5 行开始执行。

```
5     public static void main (String args[]) {
```

程序执行时，程序名之后输入的内容称为命令行参数，它是动态传递给程序的参数。如果程序执行时给定命令行参数，则这些参数将放在称为 args 的字符串数组中传给 main()方法。本例中，没有命令行参数，args 数组为空。

C 和 C++ 语言使用 main()作为程序运行的入口点，Java 亦是如此。Java 解释器在执行前查找该方法，然后从此处开始执行；如果找不到该方法，程序就不会执行。作为 Java 程序中应用程序执行的入口点的主函数 main()，它的前面有三个修饰符，这是 main 方法规定的，不能缺少，也不能替换为其他内容。熟悉 C 或 C++ 语言的读者能够理解 main()之前各关键字的意义。如果读者不完全理解，也不要担心，本书后面会介绍的。Java 中这三个修饰符的次序可以稍有变化，可以如程序中所示，也可以写为：

```
static public void
```

这一行各要素的具体含义是：

- public——该关键字说明方法 main()是公有方法，它可被任何方法访问，包括 Java 解释器。实际上，main()方法只被 Java 解释器调用，其他方法一般不调用它。
- static——该关键字告诉编译器 main()方法是静态的，可用在类 HelloWorldApp 中，不需要通过该类的实例来调用。如果方法不是静态的，则必须先创建类的实例，然后调用实例的方法。有关类和实例的内容请参看本书后面相关章节。
- void——指明 main()方法不返回任何值。这很重要，因为 Java 要进行谨慎的类型检查，包括对调用方法所返回的类型和它们说明的类型之间的检查。如果方法没有返回值，必须说明为 void，不可省略。如果方法有返回值，则以返回值类型替换 void。
- String args[]——表示命令行参数。运行一个 Java 程序的方式是在命令行中键入如下的命令：

  ```
  java 程序名 [参数列表]
  ```

这里，程序名也就是类的名字，参数列表是可选的。如果想要传送给程序不同的参数值，则可以把这些参数依次列在程序名字的后面，系统将这些参数依次放到 args 数组中。该数组各元素是 String 类型的。如键入如下的命令行：

```
$ java HelloWorldApp arg1 arg2
```

数组元素 args[0]中存储参数 arg1，args[1]中存储参数 arg2，args.length 表示命令行参数的个数。本例中，args.length 的值为 2，在命令行键入时由系统自动赋值。