

# Chp1 Hello World!

本章是全书的第一章，本章会向你介绍开始学习 Java 之前的准备知识。

## 1 Java 语言介绍

### 1.1 Java 语言的历史

我们从 Java 语言的诞生说起。

1991 年，Sun 公司在一个叫做 James Gosling 的人的带领下，成立了一个项目组，名字叫做“Green”。当时这个项目组成立的时候，是计划开发一种能够运行在消费性电子设备上的编程语言。这种设备的特点是：1、运算能力和运算空间非常有限；2、不同的厂商设计产品时会使用完全不同的 CPU，因此会有完全不同的架构。为了解决这个问题，当时 Green 希望设计出一种具有非常优秀的“跨平台”特性的语言。

Green 项目组的程序员都有着很深厚的 Unix 下 C++编程的背景。事实上，从某种意义上说，Java 语言是脱胎于 C++，在很多基础语法方面，有很多跟 C++类似的地方。当然，C++语言中有很多让人觉得复杂并且难于掌握的特性，而 Sun 公司在设计 Java 语言的时候，把这些特性都摒弃了，而增加了很多优秀的新特性。这些内容随着我们学习的深入会慢慢为大家展开。

1992 年，Green 项目组最初的产品诞生。这个产品一开始被 James Gosling 称之为“Oak”，可能是因为他很喜欢他办公室外面的那颗橡树吧……但是，后来 Sun 公司的同事发现，Oak 已经是另外一种计算机语言的名字。于是，经过讨论，把 Oak 语言改名为 Java 语言。

然而，起初 Java 语言的发展并不是一帆风顺的。在 1994 年之前，Java 语言的优秀特性并没有一个充分发挥的舞台。然而随着时代的进步，一切都开始变得不同了：互联网开始兴起了。由于网络互联互通的需要，因此在多个平台上面运行同样的程序成了一个非常有挑战性但是又非常有意义和价值的东西。Gosling 察觉到了 Java 语言发展的机会，并把 Java 语言由一种在消费设备上运行的语言，修改成为了一种能够在互联网上运行的语言。由于 Java 语言天生具有跨平台特性，Gosling 认为这种特性能够非常好的与互联网结合。这个转折也拉开了 Java 语言发展的序幕。

1995 年 5 月 23 日，在 SunWorld 会议上，Sun 公司对外正式展示了 Java 语言。我们通常把这一天成为 Java 语言的诞生之日。

1996 年，Java 语言发布了第一个正式版本：1.0 版本。这时，Java 语言能够编写的程序称之为 Applet。这种 Applet 只能在集成了 Java 环境的浏览器中运行，当时主要为浏览器来增加各种动态效果，用来美化页面和增强用户与浏览器的交互。应当说，1.0 版本并不能算成功，这个版本非常不成熟。很快的，1.1 版本发布。这个版本修正了 1.0 版本中大量的 bug，并完善了 1.0 版本中的很多缺失的部分。然而，与 1.0 版本一样，1.1 版本同样具有非常大的局限性。

事情到 1998 年有了比较大的改变。1998 年 12 月，Java1.2 版本发布。Sun 公司对这个版本的 Java 做了非常大的结构方面的调整，原有的体系几乎推倒重来。为了表明这是一个非常重大的更新，Sun 公司把 Java1.2 版本，也称之为 Java 2 Platform，用来表示这是一个全新的 Java 平台。

与此同时，Sun 公司还把 Java 2 Platform 进行了细分。对 1.0 和 1.1 扩展之后得到的部分称之为“标准版”，也就是所谓的“Java 2 Standard Edition”，简称“J2SE”。标准版可以

用来写 Applet，也可以用来编写脱离浏览器，独立运行的程序（Application）。这些都是对于一个语言来说，比较基础和比较标准的模块。除了标准版之外，Java 2 还包括“企业版”（J2EE）和“微型版”（J2ME）。企业版主要提供了服务器端编程的功能，而微型版主要提供了在一些资源受限制的平台上（例如手机）运行 Java 的功能。

在本书中，我们不会涉及企业版和微型版。需要注意的是，J2SE 是学习其他两个版本的基础，是学习 Java 语言的第一门课程。

之后，Java 语言发布了 1.3 和 1.4 版本。这两个版本主要修正了 bug，完善了 Java 的类库，但是并没有对 Java 语言进行什么革命性的变化。因此，这两个版本依然是属于 Java 2 平台的范畴。此时，Java 语言进入了真正成熟期，逐渐成为了世界排名第一的语言。大量的企业级应用采用了 Java 语言来开发服务器端软件。

2004 年，Java 推出了一个新版本。这个版本是 1.1 之后，Java 第一次对语言的基础类库做出重大改进的版本。一开始，这个版本被称之为 1.5，后来，Sun 公司为了表明这个版本具有非常强大的功能，把这个版本称之为 5.0。这也就意味着，这个版本与原来的 Java 2 平台相比有着很多不同的地方，因此，现在 Java 2 Platform 已经逐渐成为了历史，而原来的三个平台：J2SE、J2EE、J2ME 也被改名成为了 Java SE、Java EE、Java ME。

2006 年底，Java 发布了版本 6。这个版本修正了很多 5.0 版本中的 bug，改善了性能并增强了类库。

那么，Java 语言究竟有什么魔力，能够在众多语言中脱颖而出，成为世界第一大语言呢？主要源于下面的一些语言特性：

- 纯面向对象

相对于另一种面向对象的语言 C++，Java 语言是一种非常纯粹的面向对象的语言。对于 C++ 而言，写程序除了可以使用面向对象的方式之外，还可以采用面向过程、面向模板等多种方式；而相对的，Java 语言只能采用面向对象的方式进行编程。

- 简单 VS 复杂

由于 Java 语言相对 C++ 来说是一种纯粹的面向对象的语言，因此 Java 语言在理解和学习方面，都要比 C++ 语言更简单。Java 的简单性，指的是 Java 语言的这种特点：1、Java 语言本身的特点非常简单，没有复杂和晦涩的语法细节；2、Java 语言倾向于让程序员能够简洁清晰的完成任务。

而 Java 语言同时也是复杂的，体现在：1、虽然 Java 语言本身非常简单，但是它有大量强大而扎实的类库，这些类库极大的丰富了 Java 语言的特性；2、Java 语言最主要的阵地是企业级应用，这种应用本身，由于涉及到多线程、分布式、数据库、网络等各种各样的领域，因此，需求非常复杂。为了应对这种复杂的需求，Java 语言也提供了各种对应的特性，因此从这个角度来看，Java 语言是复杂的。

或许，我们可以拿一句英语来总结 Java 语言的简单和复杂：“Simple thing should be simple, Complex thing should be possible”。

- 开放性

Java 语言是一种开放的语言。这种开放集中体现在 Sun 公司：Sun 公司已经对 Java 语言开源，任何人都可以读到 Java 语言的代码；Sun 公司接受任何人提交的 JSR，也就是说，任何程序员都可以对 Java 语言未来的发展提出自己的开发和建议；Sun 公司提供了 Java 语言的免费下载。

现在，Java 社区有大量开源、免费的东西可供下载和使用，这在一定程度上也帮助了 Java 语言的发展和推广。

- 跨平台性

这是溶入 Java 血液中的一个机制。这个机制决定了，运行 Java 可以在各种平台上面，包括常见的桌面 Windows 系统，也包括企业级应用需要的 Unix 系统。这意味着 Java 语言既能够轻快的在桌面上运行，也能够扎实稳定的在企业级操作系统中运行。这种在不同平台下运行的能力使 Java 语言在企业级应用中有着深远的影响。

## 1.2 Java 语言的运行机制

上一节为读者介绍了 Java 语言的发展历史和一些基本特点，下面要介绍的就是 Java 语言的运行机制。

首先，我们简单介绍一下什么是计算机语言。对于计算机来说，真正能够直接执行的是所谓的“计算机指令”。这种计算机指令，一方面跟操作系统有关系，也就是说，Windows 系统和 Linux 系统下的指令不同；另一方面，也跟计算机的硬件有关系，不同的 CPU 具有不同的指令集。

直接操作计算机指令，使用的是机器语言以及汇编语言。然而，对于程序员来说，直接使用汇编语言来编写程序，开发起来非常的慢，也非常的辛苦。为了能让程序开发的速度提升，我们设计出了计算机高级语言。

所谓的计算机高级语言，实际上指的：人为的规定一些语法。然后，在遵循这些语法的前提下，写出一个文本文件，最后利用某种方式，把文本文件转化为机器指令进行执行。我们现在所谓的编程，往往指的就是编写文本文件的这个部分。这个文本文件一般我们称之为源文件。

那么，应当如何把一个源文件转化为机器指令进行执行呢？在现代计算机语言中，主要有两种方式：一种是编译型，一种是解释型。

什么叫编译型语言呢？这指的是，通过一个编译器软件，把源文件转化为可执行文件。可执行文件的内容，就是一些机器指令，以及相关的一些数据。在 Windows 中，可执行文件往往以 .exe 作为后缀名。在执行程序的时候，不需要源代码文件，只需要可执行文件即可。示意如下：

源文件 -- 编译器--> 可执行文件    运行可执行文件 --> 机器指令

与编译型语言相对的是解释型语言。解释型语言需要一个解释器软件，这个软件会读源文件，在读文件的过程中，同时完成将源文件内容翻译成机器指令以及执行的过程。换句话说，解释器将读取源文件、翻译成机器指令、执行指令这三步同时完成。示意如下：

文本文件 ---解释器-> 直接翻译成机器指令

由上可知，编译型语言在将源文件编译成可执行文件之后，运行程序只需要可执行文件，不再需要重复编译的过程。而解释型语言每次运行时必须重复翻译源文件，因此从运行效率上来说，解释型语言远远不如编译型语言。

当然，解释型语言也有自己的优势：跨平台性较好。由于编译型语言运行时只需要可执行文件，而可执行文件又与平台紧密相连，这也就意味着，对于不同的平台，必须要有不同的可执行文件才行。而相对而言，解释型语言就没有这么麻烦，对于不同的平台，只需要有不同的解释器就可以了，源代码几乎不用进行修改。

而 Java 语言，则兼具有编译型和解释型两种语言的特点：Java 语言运行时，采用的是先编译、后解释的方式运行。

首先，Java 源代码要写在后缀名为 .java 的源文件中。然后，通过一个编译器，编译生成 .class 文件，这个文件被称为“二进制字节码文件”。

而.class 文件并不能够直接在机器上执行。执行.class 文件，需要一个解释器，这个解释器会把.class 中的指令翻译成真正机器上的指令。也就是说，需要解释执行.class 文件。

示意如下：

.java 源文件 -- 编译 --> .class 字节码文件 -- 解释执行 --> 真正的机器指令

字节码文件是平台中立的，也就是说，运行在不同平台上的.class 文件内容相同，与所在平台无关。

那么.class 文件中保存的是什么呢？这个文件中保存的也是计算机指令，所不同的是，这些计算机指令不是真实计算机所拥有的指令，而是一些虚拟的指令。在解释执行.class 文件的指令时，为了能让这些虚拟的计算机指令能够转换成真正的计算机指令，我们需要一个 Java 虚拟机（Java Virtual Machine，简称 JVM）。

JVM 事实上是一个软件，这个软件为 Java 程序模拟出一个统一的运行环境。Java 程序只需要适应这个虚拟的环境，而与底层真正的硬件环境及操作系统环境无关。换句话说，JVM 的作用在于，它屏蔽了底层不同平台的差异。

Java 虚拟机接收.class 文件中的虚拟指令，这些指令很类似于真正的汇编语言指令，但这些指令与底层的操作系统平台和硬件平台无关，完全是另外设计出的一套独立体系。而不同平台下的 Java 虚拟机，在执行时，会把.class 文件中的虚拟机指令翻译成对应平台上真正的计算机指令。因此，我们可以修改上面的示意如下：

.java 源文件 -- 编译 --> .class 字节码文件 --在 JVM 中解释执行--> 真正的机器指令

Java 语言这种“先编译，后解释”的运行机制，使得其同时拥有了编译型语言的高效性和解释型语言的跨平台性，Sun 公司给出了最好的注解：“Write once , run anywhere”

### 1.3 JRE 与 JDK

如果一个程序员要发布 Java 程序，一般来说，会发布.class 文件。而如果要运行 Java 程序，同样指的是运行.class 文件。因此，运行 Java 程序，只需要 Java 虚拟机和解释器就可以运行。即 JRE，也就是 Java Runtime Environment 的缩写，指的是 Java 的运行环境。包括 JVM 和 Java 解释器。

但是仅仅有 JRE，只能是完成从.class 文件到真正的机器指令这一步，而无法把一个源文件编译成一个.class 文件。在 Sun 公司的网站上，有一个术语叫做 JDK。所谓的 JDK，指的是 Java Development Kit，Java 开发工具包。从内容上说，

JDK = JRE + 工具（编译器、调试器、其他工具……） + 类库

我们进行 Java 开发，至少应当有 JDK。可以到 Sun 公司的网站上进行 JDK 的下载（在撰写本书时，下载链接为：<http://java.sun.com/javase/downloads/index.jsp>）。JDK 的安装也可以参考 Sun 公司的网站（链接为：<http://java.sun.com/javase/6/webnotes/install/index.html>）。对于 Windows 系统而言，下载 JDK 之后，安装起来与其他的软件安装时并没有太大区别，在此不多赘述。

## 2 Java 开发环境配置

安装完了 JDK 之后，还不能马上进行 Java 开发，还需要进行一些环境变量的配置。配置完之后，才能真正进行 Java 程序的开发。

## 2.1 三个环境变量

Java 的环境配置，其实主要是对三个环境变量进行配置。这三个变量分别为：JAVA\_HOME、PATH 和 CLASSPATH。

JAVA\_HOME 环境变量，表示的是 Java 的安装目录。这个变量是用来告诉操作系统 Java 的安装路径的，当其他的程序需要 Java 进行支持的时候（例如一些 Java 的服务器、Java 的数据库客户端等），会通过 JAVA\_HOME 来寻找 Java 的安装路径。

PATH 环境变量，是在命令行上输入 Java 命令时，用来指示操作系统去哪个路径下找 Java 的相关程序。往往会把 PATH 变量配成 Java 的安装路径/bin 目录。

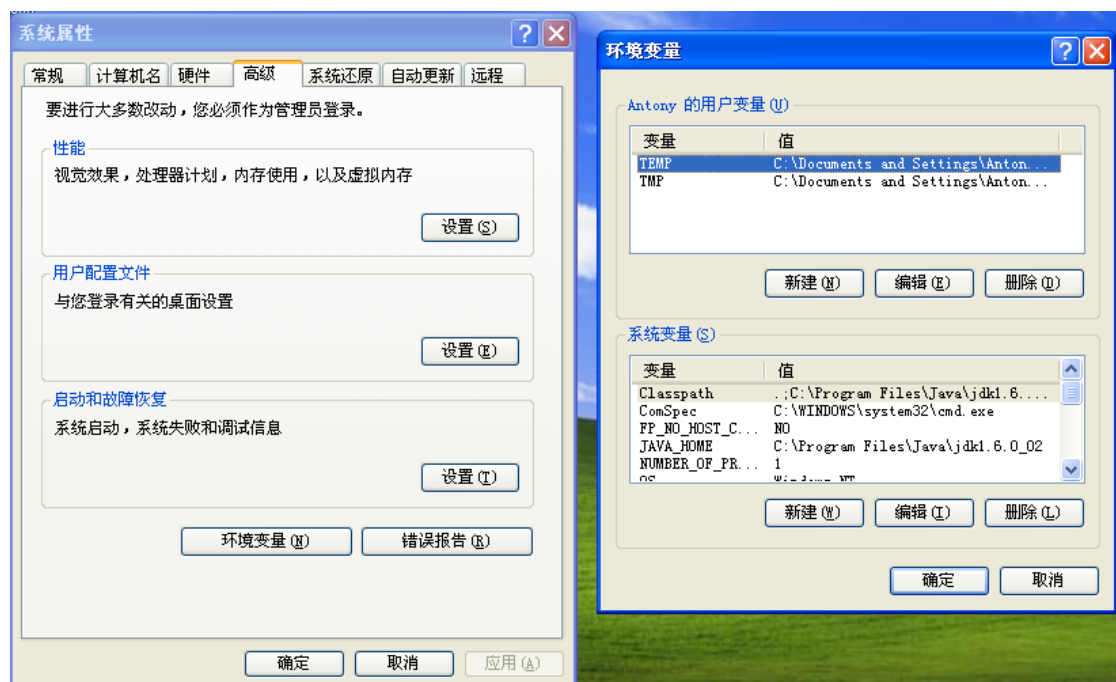
CLASSPATH 是用来指示编译器和 JVM 去哪个目录寻找.class 文件。当我们运行 Java 程序时，必然会需要获取.class 文件的信息，而且往往还需要不止一个.class 文件的信息。此时，我们就需要在硬盘中寻找相应的.class 文件。而硬盘中的文件成千上万，JVM 如何寻找呢？为了能够让 JVM 有的放矢，我们需要设置 CLASSPATH 环境变量，指定一些目录，让 JVM 寻找.class 文件时，只需要寻找这些我们指定的目录即可。

我们分 Windows 和 Linux 两个平台，来具体的介绍如何配置环境变量。

## 2.2 环境变量的配置

### 2.2.1 Windows

右键点击“我的电脑” -- 选择“属性” -- 选择“高级” -- 选择“环境变量”

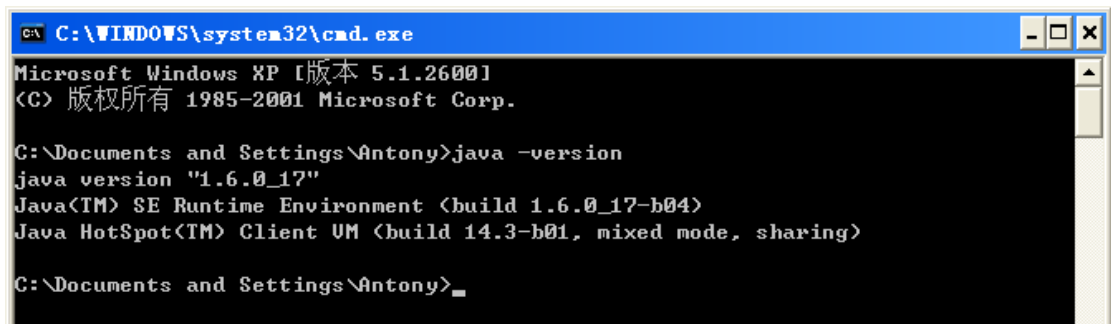


在系统变量中，点击“新建”，增加 JAVA\_HOME 系统变量，这个变量的值设为 Java 的安装目录。假设 Java 安装在 C:\Program Files\Java\jdk1.6.0\_02。

然后，在系统变量中，查找 PATH 变量（不区分大小写）。在 Path 变量的末尾，增加一句：“;C:\Program Files\Java\jdk1.6.0\_02\bin”。注意两个要点：1、在 Path 末尾增加，千万不要把原有的内容去掉；2、增加的值为 Java 安装目录下的 bin 目录。

最后，增加一个 CLASSPATH 变量。值为“.”，注意不带双引号。

当所有环境变量都配置好了之后，打开 Windows 中的命令行（对于 WindowsXP 来说：开始菜单 -- 运行 -- cmd），执行 `java -version`。如果得到正确的版本信息，则说明配置正常。如下图：



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600.1]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Antony>java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
Java HotSpot(TM) Client VM (build 14.3-b01, mixed mode, sharing)

C:\Documents and Settings\Antony>
```

## 2.2.2 Linux

Linux 下的配置我们简单的描述一下。假设在 Linux 中 Java 安装在 `/opt/java/jdk6` 目录下。对于使用 `bash` 的 Linux 来说，在用户主目录下，修改 `.bash_profile` 文件，在这个文件的最末尾增加三行：

```
export JAVA_HOME=/opt/java/jdk6

export PATH=$JAVA_HOME/bin:$PATH

export CLASSPATH=.
```

要注意的是，Linux 中的环境变量区分大小写，因此要注意大小写不要写错。

## 3 第一个程序

完成了 Java 环境变量的配置，下面我们来看我们的第一个程序：HelloWorld 程序。

### 3.1 Hello World 程序

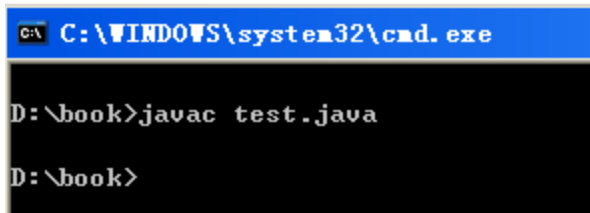
首先我们写出 HelloWorld 程序的代码。在硬盘上创建一个名为 `test.java` 的文件，并用记事本打开这个文件，书写如下内容。注意：区分大小写，并注意空格：

```
class HelloWorld{
    public static void main(String args[]){
        System.out.println("Hello World");
    }
}
```

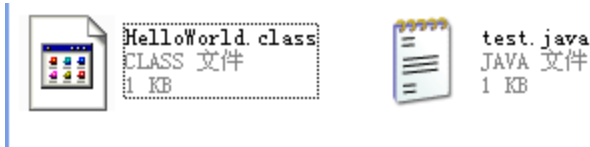
代码写完之后就可以进行保存。然后，在命令行模式下，进入这个 `.java` 文件所在的文件夹，使用下面的命令来进行编译：

```
javac test.java

编译成功时如下图：
```



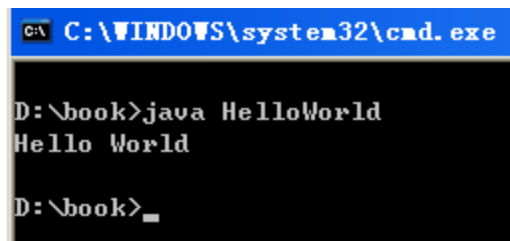
在编译完成之后，会在目录下生成一个相应的 HelloWorld.class 文件，如下图：



之后，在命令行上，用下面的命令来执行：

```
java HelloWorld
```

执行结果如下：



在屏幕上打印出一个“Hello World”的字符串。

请保证在你的机器上看到 Hello World 字符串再继续下一小节的学习。

## 3.2 深入 HelloWorld

### 3.2.1 Hello World !

我们首先来研究和分析一下 test.java 这个程序的代码：

```
class HelloWorld{  
    public static void main(String args[]){  
        System.out.println("Hello World");  
    }  
}
```

首先，第一行：class HelloWorld，这句话定义了一个名字叫做 HelloWorld 的类。其中，class 是 Java 语言的关键字，而 HelloWorld 则是定义的类的名字。

至于什么是类，这个概念对于初学者来说比较复杂，在此不多做介绍。对于初学者来说，可以把类理解成：代码的容器。也就是说，在 Java 中绝大部分代码都要写在类的范围之内，要写代码就必须先定义一个类。

在定义完了 HelloWorld 之后，后面有一对花括号，花括号中的内容就表示是这个类中的内容。

在这个花括号里，下面的这行：

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

这一行定义了一个主方法（也叫主函数）。在 Java 中，主方法的定义比较长，但是非常有用，也许你现在还无法理解这些内容，随着学习的深入，你会理解这里面的每一个单词。但是现在，照着写并且记住主方法的写法就行了。

那么主方法有什么用呢？我们把主方法称之为：程序执行的入口。也就是说，Java 程序

在执行的时候，会执行类中的主方法，当主方法执行完毕之后，程序也就退出了。

在主方法内部，有下面这句：

```
System.out.println("Hello World");
```

这是一个 Java 的语句。注意，每一个 Java 语句都应当以分号结尾。

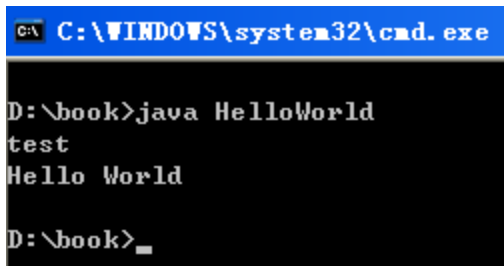
这个语句能够在屏幕上打印出一个字符串，这个字符串的内容，就是括号中的“Hello World”字符串。并且，这个语句在打印完字符串之后，还会打印一个换行符，进行换行。

与 `System.out.println` 对应的，还有 `System.out.print` 语句。这个语句同样完成打印，但是输出不换行。我们比较下面两个程序的区别：

代码：

```
class HelloWorld{
    public static void main(String args[]){
        System.out.println("test");
        System.out.println("Hello World");
    }
}
```

输出结果：



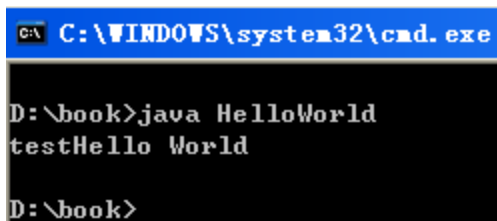
```
C:\WINDOWS\system32\cmd.exe
D:\book>java HelloWorld
test
Hello World
D:\book>
```

输出完 test 之后，换行，再输出 Hello World。

代码 2：

```
class HelloWorld{
    public static void main(String args[]){
        System.out.print("test");
        System.out.println("Hello World");
    }
}
```

输出结果：



```
C:\WINDOWS\system32\cmd.exe
D:\book>java HelloWorld
testHello World
D:\book>
```

输出 test 之后，不换行直接输出 Hello World。

### 3.2.2 类与.class 文件

我们注意到，编译 test.java 之后，会产生一个名为 HelloWorld 的.class 文件。这个.class



文件的文件名与我们在 `test.java` 中定义的类名一模一样。换句话说，一个 `.java` 文件中定义的每一个类，编译后都会对应的生成一个和类名完全一样的 `.class` 文件。

这个 `.class` 文件不是可执行文件，用文本编辑器也无法正常打开。我们也可以在 `test.java` 中定义多个类。例如下面的代码：

```
class HelloWorld{
    public static void main(String args[]){
        System.out.println("test");
        System.out.println("Hello World");
    }
}

class Welcome{
    public static void main(String args[]){
        System.out.println("Welcome to learn java");
    }
}
```

编译之后，会生成两个 `.class` 文件，一个 `HelloWorld.class`，一个 `Welcome.class`。

在运行时，运行的是 `.class` 文件。但是需要注意的是，使用 `java + 类名` 运行，而不能有 `.class` 这个后缀。

比如，我们要运行 `Welcome` 类，则使用的命令应当是：

`java Welcome`

而不是

`java Welcome.class`

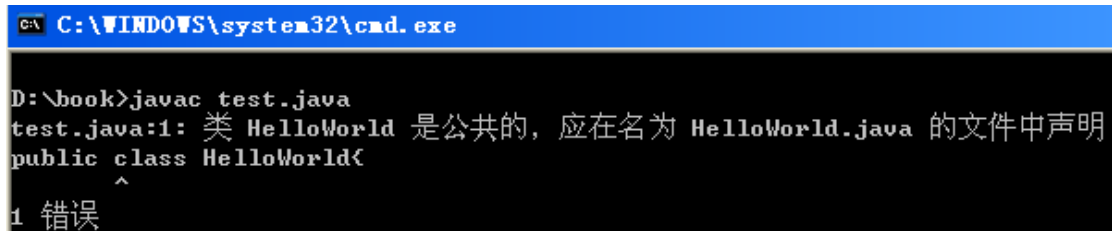
要注意的是，我们使用 `java Welcome` 运行 `Welcome` 类时，JVM 需要在硬盘上找到相应的 `Welcome.class` 文件。此时，JVM 会通过 `CLASSPATH` 变量的指示，来寻找 `.class` 文件。由于我们把 `CLASSPATH` 配置成了一个“.”，这表示当前目录，因此 JVM 就会在当前目录下寻找 `Welcome.class` 文件。

### 3.2.3 类与公开类

如果我们给 `HelloWorld` 类增加一个前缀：`public`，则此时，`HelloWorld` 就不是一个普通的类，而变成了一个公开类。代码如下：

```
public class HelloWorld{
    public static void main(String args[]){
        System.out.println("test");
        System.out.println("Hello World");
    }
}
```

公开类有自己的特殊性。此时再编译 `test.java`，则会产生一个编译时的错误，错误如下：



```
C:\WINDOWS\system32\cmd.exe

D:\book>javac test.java
test.java:1: 类 HelloWorld 是公共的, 应在名为 HelloWorld.java 的文件中声明
public class HelloWorld<
^
1 错误
```

这个错误说明，如果要使用一个公开类，则有一个要求：公开类的类名必须与.java 文件的文件名相同（包括大小写）。

为了修正这个错误，我们必须把原来的 `test.java` 改名为 `HelloWorld.java`，再次编译才能编译通过。

由于一个.java 文件只能有一个文件名，因此一个.java 文件中，最多只能有一个公开类。当然，如果不是公开类的话，一个.java 文件中可以有多个类。

## 4 初学者忠告

至此，我们对 Java 的基本介绍，以及 Java 环境的配置就全部介绍完了。在开始下一章的内容之前，下面是对初学者的一些忠告：

1、动手敲代码。不论是书本上的例子代码，还是练习中的代码，请每一个都自己敲一遍。有些题目很简单，但是在做这些简单练习的时候，或许你会犯一些低级错误，而提前犯这些错误能够避免你在解决难题时被这种低级错误纠缠。有些题目很难，你凭借自己的能力无法完成，那欢迎你看答案。但是看完答案之后，请记住，自己再敲一遍代码，巩固一下。

2、不要使用 IDE。Java 语言的流行，使得有大量的好用的开发工具，比如 `eclipse`，`netbeans` 以及 `jbuilder` 等。这些工具能够极大的提高程序员的开发效率，是程序员的好帮手，但是不适合初学者。初学者需要更多的锻炼和磨砺，才能够打下扎实的基础，才能够更好的掌握这些工具。如果一开始你就使用这些工具的话，很有可能你会一直被这些工具束缚住，对他们产生依赖，而影响你对 Java 的掌握。

建议你使用一个文本编辑器 + 命令行的方式，学习 Java 的开发。起码在学习 Java 的前半个月应该这么做。

3、选择一个有语法高亮的文本编辑器。如 `notepad++` 和 `notepad2`，你也可以选择更加强大的 `UltraEdit`。有一个支持 java 语法的编辑器会让你愉快很多。