第一章

前言

Java 语言是一种通用的、并发的、基于类的、面向对象的语言。 它被设计得足够简单，以至于许多程序员都能熟练使用该语言。Java 语言与 C 语言和 C++语言相关，但又不同，它有一些 C 和 C++语言省略的特性，也含有一些从其他语言借鉴的特性。它是一个用于生产领域的语言，而不是一个用于研究领域的语言，因此，正如 C.A.R.Hoare 在他介绍 Java 语言设计的经典论文中所说的一样，Java 语言的设计避免了一些新的以及未经测试的特性。

Java 语言是一种强类型及静态类型的语言。本文清楚地解释了编译时错误和运行时错误的区别。编译时通常包括将程序转换为与机器无关的字节码表示形式。运行时活动包括加载和链接执行程序所需的类、生成可选的机器代码和程序的动态优化，以及程序的实际执行。

Java 语言是一种高级语言，因为通过该语言无法获得机器表示的细节。它包括自动内存管理，通常使用垃圾回收期，来避免显式释放的安全问题（如 C 语言的 free 或者 C++语言的 delete）。高性能的垃圾收集机制会导致应用程序的停顿。这门语言不包括不安全的数据结构，例如不检查数组下标的数组访问，因为这样不安全的数据结构会导致程序行为超出预期。

Java 语言通常被编译成字节码指令集，具体方式在《Java 虚拟机规范，Java SE 19 版》里介绍。

1.1 内容简介

第二章 介绍语法和符号

第三章 介绍词法结构。该语言是用 Unicode 字符集编写的。它支持在只支持 ASCII 的系统上写入 Unicode 字符。

第四章 介绍类型、值和变量。类型又分为原生类型和引用类型。原生类型被定义为在所有机器和实现上都是相同的，具有不同大小的两种整数类型实现，IEEE754 浮点数，一个布尔类型，一个 Unicode 字符类型。原生类型变量值不共享。

引用类型包括类类型、接口类型和数组类型。引用类型是通过动态创建的对象实现的，它们是类或数组的实例，每个对象可以存在多个引用。所有对象（包括数组）支持 Object 类的方法，Object 类是类继承结构唯一的根节点。预定义的 String 类支持 Unicode 字符串。

类的存在是为了在对象中包装原生类型的值。大多数情况下，装箱/拆箱都是编译器自动完成的。类和接口可以是泛型，因此，他们可以被引用类型实例化。这样的类和接口的参数化类型可以用特定的类型参数来调用。

变量是类型化的存储位置。原生类型的变量包含该原生类型的值。类类型的变量可以持有空引用或对象的引用，该对象是指定类的实例或该类的任何子类。接口类型的变量包含空引用或者实现该接口的实例的引用。数组类型的变量包含空引用或者数组对象的引用。Object 类变量包含空引用或者任何对象的引用，无论该对象是类实例还是数组。

第五章 介绍类型转换。类型转换改变编译时类型，有时改变表达式的值。这些转换包括在原生类型和引用类型之间的装箱和拆箱。数值提升用于将数值运算符的操作数转换为可执行操作的普通类型。语言本身没有漏洞，运行时检查引用类型之间的转换以保证类型安全。

第六章 介绍声明和名称，以及如何确定名称的含义（即名称表示哪个声明）。Java 语言并不需要类和接口或者它们的成员，在使用前声明。声明顺序只对局部变量、局部类、局部接口以及类或接口中的字段初始化器是重要的。这里描述了使程序更具可读性的推荐命名约定。

第七章 介绍程序结构，它被组织成包。包的成员是类、接口和子包。包和它们的成员，在层次命名空间中有不同的名字；DNS 通常可以用来形成唯一的包名。编译单元包含指定包成员类和接口声明，也可以从其他包导入类和接口，给它们重新命名（为简称）。

许多包组织成不同的模块，作为构建大项目的基本组成部分。模块的声明指定编译和运行它自己的包需要哪些其他模块、包、类或者接口。

Java 语言支持从外部访问包、类和接口内部成员的限制。包的成员仅仅能够被同一个包内的其他成员，被同一个模块其他包的成员，或者不同模块的包的成员访问。类和接口的成员也有同样的限制。

第八章 介绍类。类的成员可以是类、接口、变量或者方法。在每个类，类变量只存在一次。类方法不用特定对象的引用就能调用（这里说的应该是静态类）。实例变量是对象动态创建的，是类的实例。实例方法通过类的实例调用；在执行过程中这种实例成为当前对象 this，它是面向对象编程的支撑。

类支持单继承，每个类都有一个父类。每个类从它的父类继承，最终从 Object 类继承。类类型的变量可以引用该类或该类子类的实例，允许新类和现有方法一起使用，这就是多态的使用方式。

类使用 synchronized 方法可以支持并发编程。方法声明在执行过程中可能产生的受检异常，从而允许进行编译时检查，以确保异常条件得到处理。对象可以声明一个 finalize 方法，该方法将在垃圾回收器回收对象之前被调用，通常对象在该方法内清除它们的状态。

为简单起见，该语言没有声明”头”与类的实现分开，也没有单独的类型和类层次结构。

枚举是类的一种受限形式，它支持以类型安全的方式定义小值集合及其操作。不像其他语言的枚举，Java 的枚举常量是拥有自己方法的对象。

第九章 介绍接口。接口的成员包括类、接口、常量和方法。在其他方面不相关的类可以实现相同的接口。接口类型的变量包含任何实现该接口的对象的引用。

类和接口支持接口多继承。一个类实现了一个或多个接口，可以从它的父类和父接口继承实例方法。

注解是可以应用于程序中的声明的元数据，也可以应用于声明和表达式中类型的使用。注解的形式由注解接口(一种专门的接口)定义。这种声明不能用任何方式影响程序的语义。然而， 它们提供不同工具的有用的输入。

第十章 介绍数组。数组访问包括边界检查。数组是动态创建的对象，可以赋值给 Object 类型的变量。Java 语言支持数组的数组，而不是多维数组。

第十一章 介绍异常。它是不可恢复的，并且和语言的语义和并发机制完全集成。有三种类型的异常：受检异常，运行时异常和错误。编译器要求只有在方法或构造函数声明了受检异常的情况下，才能正确处理受检异常。这提供了在编译时检查异常处理程序是否存在的功能，并将大大帮助编程。大多数用户定义的异常都是受检异常。Java 虚拟机发现的非法操作将导致运行时异常，例如 NullPointerException。错误来自于 Java 虚拟机发现的失败，例如 OutOfMemoryError。大多数简单的程序并不处理错误。

第十二章 介绍执行程序过程中发生的活动。程序通常存储为表示已编译类和接口的二进制文件。这些二进制文件可以被加载到虚拟机中，链接到其他类和接口，并且初始化。

初始化后，类方法和类变量就可以使用了。为了创建一个类类型的新对象，可能需要初始化很多类。类实例对象包含一个父类的对象，对象创建包括递归创建这些父类实例。

当对象不再被引用，它将被垃圾回收器回收。如果对象声明了 finalize 方法，该方法在对象被回收之前执行，这是对象清理未释放资源的最后的机会。当不再需要对象时，对象将被销毁。

第十三章 介绍二进制兼容性，指定类和接口的更改对使用已更改的类和接口但未重新编译的其他类和接口的影响。这些考虑项对类和接口的开发人员来说是很重要的，因为程序将通过一系列版本被发布到网络的各个角落。当类或接口改变时，好的开发环境会自动重编代码，因此大多数的开发人员不需要关心这些细节。

第十四章 介绍程序块和语句，它们是基于 C 和 C++语言。Java 语言没有 goto 语句，但是包含 break 和 continue 语句。和 C 语言不同的是，在控制流语句中，Java 需要 boolean 或 Boolean 类型的表达式，并不隐式转换类型为 boolean（除非通过拆箱），这样做的目的是为了在编译时期捕获更多错误。一个 synchronized 语句提供了基本对象级别的监视锁。try 语句可以包含 catch 和 finally 子句，以防止非局部控制传输。第 14 章还描述了在语句(和表达式)中使用的模式，以有条件地声明和初始化局部变量。

第十五章 介绍表达式。为了增加确定性和可移植性，本文档完全指定了表达式求值的(明显的)顺序。重载方法和构造函数在编译时解析，方法是从适用的方法或构造函数中选择最特定的方法或构造函数。

第十六章 介绍语言如何确保局部变量在使用前被定义。由于所有其他变量都被自动初始化为默认值，Java 语言并不自动初始化局部变量，以免掩盖编码错误。

第十七章 介绍线程和锁的语义，它们基于最初由 Mesa 编程语言引入的基于监视器的并发性。Java 编程语言为支持高性能实现的共享内存多处理器指定了一个内存模型。

第十八章 介绍了各种类型推断算法，用于测试泛型方法的适用性，并在泛型方法调用中推断类型。

第十九章 介绍该语言的语法。

1.2 第一个例子程序

文中给出的大多数示例程序都已准备好执行，并且形式类似于:

```
class Test {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.print(i == 0 ? args[i] : " " + args[i]);
        System.out.println();
    }
}
```

在一台安装了 JDK 的机器上,把这个类保存成文件 Test.java,可以通过以下命令编译执行:

```
javac Test.java
java Test Hello, world.
```

产生以下输出:

```
Hello, world.
```

1.3 符号

在本规范中，我们引用了来自 Java SE 平台 API 的类和接口。每当我们使用一个标识符 N 引用一个类或接口(除了在示例中声明的那些)时，预期的引用是指向 java.lang 包中名为 N 的类或接口。我们对 java.lang 以外的包中的类或接口使用规范名称(§6.7)。

旨在阐明规范的非规范性信息以较小的缩进文本给出。

> 这是非规范性信息。它提供直觉、基本原理、建议、例子等。

为了缩短一些规则的描述，特别是那些系统地分析 Java 编程语言结构的规则，习惯上用“iff”来表示

“当且仅当”。

Java 编程语言的类型系统偶尔会依赖于替换的概念。符号[F_1:=T_1,...,F_n:=T_n]表示将 Fi 替换为 Ti，i 从 1 遍历到 n（不包括 1 和 n）。

1.4 预定义类和接口的关系

如上所述，该规范经常引用 Java SE 平台 API 的类和接口。特别是，一些类和接口与 Java 编程语言有特殊的关系。例如 Object, Class, ClassLoader, String 和 Thread 这些类，以及包 java.lang.reflect 里的类和接口等等。该规范限制了这种类和接口的行为，但没有为它们提供完整的规范。读者可以参考 Java SE 平台 API 文档。因此，该规范没有详细描述反射。许多语言结构在核心反射 API (java.lang.reflect)和语言模型 API (javax.lang.model)中都有类似的东西，但这里一般不讨论这些问题。例如，当我们列出创建对象的方式时，我们通常不包括核心反射 API 实现这一功能的方法。读者应该知道这些额外的机制，即使它们没有在文本中提到。

1.5 预览特性

预览特性是指:

- Java 编程语言的一种新特性 ("预览语言特性"), 或者
- 在 java.*或者 javax.*命名空间（“预览 API”）里完全指定、完全实现的一个新模块、包、类、接口、字段、方法、构造函数或枚举常量，但它不是永久的。它可以在给定的 Java SE 平台的实现中使用，以引起开发人员基于现实世界使用的反馈；这可能会导致它在 Java SE 平台的未来版本中成为永久性的。

除非用户指示将在编译时和运行时通过主机系统启用预览特性，否则应该禁用给定 Java SE 版本的预览特性。

Java SE 平台的特定版本定义的预览特性在该版本的 Java SE 平台规范中枚举。预览功能说明如下:

- 在独立的文档中指定了预览语言特性，说明该版本对 Java 语言规范的更改。 当且仅当在编译时启用预览功能时，预览语言功能规范通过引用被纳入 Java 语言规范，并成为其一部分。

  Java SE 19 定义了两个预览语言特性:switch 的模式匹配和记录模式。指定此预览功能的独立文档可在 Oracle 网站上获得，该网站托管 Java 语言规范：https://docs.oracle.com/javase/specs/index.html。

- 预览 API 在该版本的 Java SE API 规范中指定。

预览语言功能的使用规则如下:

- 如果禁用预览功能，那么任何对预览语言功能的源代码引用，或者对使用预览语言功能声明的类或接口的源代码引用，都会导致编译时错误。
- 如果预览特性被启用，那么任何对使用预览语言特性声明的类或接口的源代码引用都会导致预览警告，除非下列情况之一为真:
  - 该引用出现在一个声明中，该声明被注解为禁止预览警告(§9.6.4.5)。
  - 引用出现在导入声明中(§7.5)。

    当启用预览功能时，强烈建议 Java 编译器对每个预览语言功能的源代码引用给出不可抑制的警告。该警告的详细内容超出了 Java 语言规范的范围，但其目的应该是提醒程序员预览语言特性的代码可能会受到未来更改的影响。

  一些预览 API 被 Java SE 平台规范描述为反射，主要在 java.lang.reflect、java.lang.invoke 和 javax.lang.model 包中。使用反射预览 API 的规则如下:

  • 无论启用或禁用预览功能，任何对反射预览 API 元素的源代码引用都会导致预览警告，除非以下情况之一为真:
    - 引用出现的声明与反射预览 API 元素的声明在同一个模块内。
    - 引用出现在注解了预览警告的声明中。
    - 引用出现在导入声明中。

  Java SE 平台规范中没有描述为反射的所有预览 API 都是正常的。普通预览 API 的使用规则如下:

  - 如果预览特性被禁用，那么任何对普通预览 API 元素的源代码引用都会导致编译时错误，除非:

    -引用出现的声明与普通预览 API 元素的声明在同一个模块内。
  - 如果预览特性被启用，那么任何对普通预览 API 元素的源代码引用都会导致预览警告，除非以下情况之一为真:

    -引用出现的声明与普通预览 API 元素的声明在同一个模块内。

    -引用出现在注解了预览警告的声明中。

    -引用出现在导入声明中。

1.6 反馈

欢迎读者向 jls-jvms-spec-comments@openjdk.java.net 报告 Java 语言规范中的技术错误和歧义。有关 javac(Java 编程语言的参考编译器)行为的问题，特别是它是否符合本规范的定义，读者可以发送邮件到 compiler-dev@openjdk.java.net 进行反馈。

1.7 参考文献

Apple Computer. *Dylan Reference Manual*. Apple Computer Inc., Cupertino, California. September 29, 1995.

Bobrow, Daniel G., Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales, and David A. Moon. *Common Lisp Object System Specification*, X3J13 Document 88-002R, June 1988; appears as Chapter 28 of Steele, Guy. *Common Lisp: The Language*, 2nd ed. Digital Press, 1990, ISBN 1-55558-041-6, 770-864.

Ellis, Margaret A., and Bjarne Stroustrup. *The Annotated C++ Reference Manual*. Addison- Wesley, Reading, Massachusetts, 1990, reprinted with corrections October 1992, ISBN 0-201-51459-1.

Goldberg, Adele and Robson, David. *Smalltalk-80: The Language*. Addison-Wesley, Reading, Massachusetts, 1989, ISBN 0-201-13688-0.

Harbison, Samuel. *Modula-3*. Prentice Hall, Englewood Cliffs, New Jersey, 1992, ISBN 0-13-596396.

Hoare, C. A. R. *Hints on Programming Language Design*. Stanford University Computer Science Department Technical Report No. CS-73-403, December 1973. Reprinted in SIGACT/SIGPLAN Symposium on Principles of Programming Languages. Association for Computing Machinery, New York, October 1973.

IEEE. *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2019 (Revision of IEEE 754-2008). July 2019, ISBN 978-1-5044-5924-2.

Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*, 2nd ed. Prentice Hall, Englewood Cliffs, New Jersey, 1988, ISBN 0-13-110362-8.

Madsen, Ole Lehrmann, Birger Moller-Pedersen, and Kristen Nygaard. *Object-Oriented Programming in the Beta Programming Language*. Addison-Wesley, Reading, Massachusetts, 1993, ISBN 0-201-62430-3.

Mitchell, James G., William Maybury, and Richard Sweet. *The Mesa Programming Language, Version 5.0*. Xerox PARC, Palo Alto, California, CSL 79-3, April 1979.

Stroustrup, Bjarne. *The C++ Progamming Language*, 2nd ed. Addison-Wesley, Reading, Massachusetts, 1991, reprinted with corrections January 1994, ISBN 0-201-53992-6.

Unicode Consortium, The. *The Unicode Standard, Version 13.0*. Mountain View, California, 2020, ISBN 978-1-936213-26-9.