HZ BOOKS
华章教育

计 算 机 科 学 丛 书

PEARSON

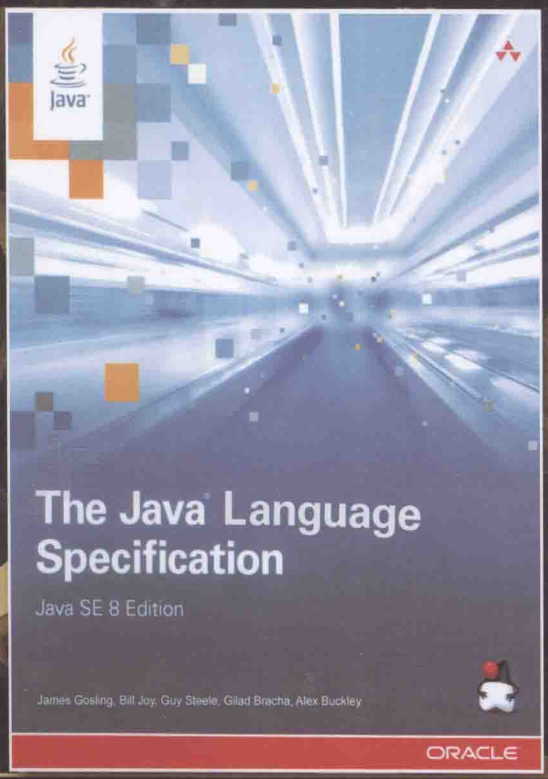
Java语言规范

基于Java SE 8

詹姆斯·高斯林（James Gosling）
比尔·乔伊（Bill Joy）
[美] 盖·斯蒂尔（Guy Steele） 著 陈昊鹏 译
吉拉德·布拉查（Gilad Bracha）
亚历克斯·巴克利（Alex Buckley）

The Java Language Specification

Java SE 8 Edition



机械工业出版社
China Machine Press

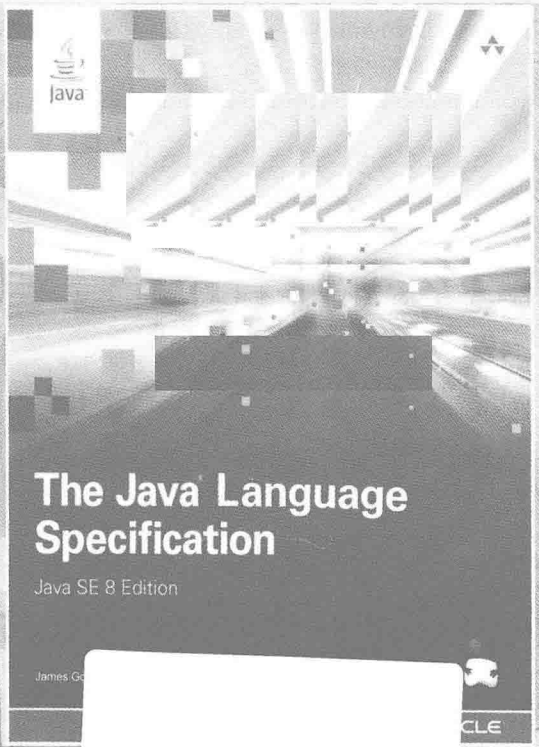
计 算 机 科 学 丛 书

Java语言规范

基于Java SE 8

詹姆斯·高斯林（James Gosling）
比尔·乔伊（Bill Joy）
[美] 盖·斯蒂尔（Guy Steele） 著 陈昊鹏 译
吉拉德·布拉查（Gilad Bracha）
亚历克斯·巴克利（Alex Buckley）

The Java Language Specification
Java SE 8 Edition



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Java 语言规范：基于 Java SE 8/（美）高斯林（James Gosling，）等著；陈昊鹏译．—北京：机械工业出版社，2016.1
（计算机科学丛书）
书名原文：The Java Language Specification, Java SE 8 Edition

ISBN 978-7-111-52399-4

I. J… II. ① 高… ② 陈… III. JAVA 语言－程序设计－英文 IV. TP312

中国版本图书馆 CIP 数据核字（2015）第 294412 号

本书版权登记号：图字：01-2014-5470

本书是 Java 语言最新版本的规范，全面涵盖了从 Java 最基础的文法、类型、变量到高级特性 lambda 表达式、线程与锁等。该规范针对每一项语言特性给出了代表性的示例程序，以帮助读者更容易地理解和掌握这些特性。通过阅读本规范，可以全面系统地了解 Java 语言的各项特性，并充分利用这些特性来编写出更加高效简洁的 Java 程序。读者还可以从 Oracle 公司的 Java 官方网站上看到本书的最新进展和修订，以了解 Java 语言的最新发展动向。

本书可作为高等院校计算机及相关专业程序设计课程的教材，也可作为 Java 语言爱好者的参考资料。

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码 100037）
责任编辑：李　艺　　责任校对：董纪丽
印　　刷：三河市宏图印务有限公司　　版　　次：2016 年 1 月第 1 版第 1 次印刷
开　　本：185mm×260mm　1/16　　印　　张：33
书　　号：ISBN 978-7-111-52399-4　　定　　价：129.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037

华章科技图书出版中心

译者序

The Java Language Specification, Java SE 8 Edition

本书是Java最新版本的规范，也是Java语言的创始人在程序员们面对各种所谓“下一代语言”的纷乱局面时进行的高调回应。在第8版中，无论是lambda表达式还是增强的注解，都给予了Java新鲜的活力，能够帮助我们从容应对多核时代并行处理大行其道的局势。第8版与第7版之间的更新时间间隔较以往明显缩短了，这说明Java确实感受到了来自其他语言和新型计算模型的压力。关于第8版是否达到了预期的目的，这是一个仁者见仁的问题，但至少Java还在不断地通过更新来完成自我救赎。时至今日，我们仍旧可以看到每年都有大量新程序员对Java趋之若鹜。因此可以预见，在未来不算短的时间里，Java仍将是一种十分重要的编程语言，并占据大量的市场份额。

本书与其他有关Java的著作不同，它没有华丽的辞藻和生动的修辞，它强调的是规范的科学性和严谨性。因此，阅读本书在某种程度上会显得有些乏味。如果你将它作为Java的入门书籍，那必定会感到阅读本书艰涩而无趣，但是如果你将本书当作Java语言的工具书，用它来寻找语言中的各种设计细节，那么你就会觉得它十分权威而且全面。

正是因为其科学性和严谨性，使本书的翻译难度明显大于其他著作。因此，尽管我已经翻译过多本有关Java的经典书籍，但是在翻译本书时仍然觉得非常棘手，很多地方要反复推敲，力求准确，保持其严谨性。说到这里，我仍旧需要不可免俗地讲：“由于水平有限，书中错误在所难免，敬请批评指正。”但是在这部译本中，这句话确实并非只是走过场，而是肺腑之言，因为面对这样一部规范，我的信心确实比以往更显不足。

最后，希望各位读者通过阅读本书能够获益，也能够对Java收获信心和希望，成为越来越优秀的Java程序员。

陈昊鹏

1996 年，James Gosling、Bill Joy 和 Guy Steele 为《Java 语言规范》第 1 版写下了下面的话：

“我们相信 Java 编程语言是一种成熟的语言，已经为广泛使用做好了准备。但是，我们还是期望该语言的某些演化会在未来几年内出现。我们希望能够以与已有应用完全兼容的方式来管理这种演化。”

Java SE 8 是 Java 语言在其历史上最大的演化版本。一组数量相对较小的特征，包括 lambda 表达式、方法引用和函数型接口，组合起来提供了融合面向对象风格和函数型风格的编程模型。在 Brian Goetz 的领导下，这种融合是以鼓励最佳实践的方式完成的，包括不可变性、无状态性、组合性；同时，保留了“Java 的感觉”，即可读性、简单性和普适性。

重要的是，Java SE 平台的库在随 Java 语言一起演化。这意味着使用 lambda 表达式和方法引用来表示行为，例如应用到列表中每个元素上的操作，具有“打破常规”的高生产率和高性能。按照类似的模式，Java 虚拟机也在随 Java 语言一起演化，以确保在分离编译的约束下，支持库演化的缺省方法在编译时和运行时都尽可能地保持一致。

向 Java 语言中添加第一流功能的倡议始于 20 世纪 90 年代。BGGA 和 CICE 提出的 circa 2007 为该话题带来了新的活力，而 OpenJDK circa 2009 中 Project lambda 的创建更是吸引了人们极大的兴趣。Java SE 7 中向 JVM 添加了方法句柄，这为添加新的实现技术同时保持“编写一次，处处运行”的特性打开了一扇门。最后，由 JSR 335《用于 Java 编程语言的 lambda 表达式》将变化引入到了 Java 语言中，JSR 335 专家组包括 Joshua Bloch、Kevin Bourrillion、Andrey Breslav、Rémi Forax、Dan Heidinga、Doug Lea、Bob Lee、David Lloyd、Sam Pullara、Srikanth Sankaran 和 Vladimir Zakharov。

编程语言设计一般总是会涉及与复杂程度的角力，而这种复杂程度是完全向语言的用户屏蔽的。（因此，编程语言的设计经常被比作冰山：90% 是不可见的，因为它们“在水下”。）在 JSR 335 中，最大的复杂性潜藏于隐式类型的 lambda 表达式与重载解析的交互中。在这个领域以及其他许多领域，Oracle 公司的 Dan Smith 完成了一项杰出的工作，即透彻地描述了期望的行为。他的话在本规范中通篇都可以找到，包括全新的第 18 章。

在 Java SE 8 中另一项创意是提升注解的效用，而注解是 Java 语言中最流行的特性之一。首先，Java 文法已经扩展为允许对许多语言结构的类型使用注解，从而形成了如 Checker Framework 等新型静态分析工具的基础。这个特性是通过由 Michael Ernst 和包括我自己、Doug Lea 以及 Srikanth Sankaran 组成的专家组主导的 JSR 308《Java 类型上的注解》描述的。本规范涉及的变更是广泛的，而且 Michael Ernst 和 Werner Dietl 多年来所做出的不懈努力得到了认可。其次，注解可以在同一个语言结构上“重复”，使得使用注解类型建模特定领域配置的 API 大受裨益。Michael Keith 和 Bill Shannon 在 Java EE 中首倡并指导了这项特性。

Oracle 公司 Java 平台组的许多同事都为本规范提供了非常有价值的支持，他们是：Leonid Arbouzov、Mandy Chung、Joe Darcy、Robert Field、Joel Franck、Sonali Goel、Jon

Gibbons、Jeannette Hung、Stuart Marks、Eric McCorkle、Matherey Nunez、Mark Reinhold、Vicente Romero、John Rose、Georges Saab、Steve Sides、Bernard Traversat 和 Michel Trudeau。

也许，最诚挚的感谢必须献给使本规范变得“真实”的编译器工程师。Oracle 公司的 Maurizio Cimadamore 从早期对 lambda 表达式的设计及其在 `javac` 中的实现开始，就一直进行着超人般的工作。在 Eclipse 中支持 Java SE 8 的特性是由下列人员实现的：Jayaprakash Arthanareeswaran、Shankha Banerjee、Anirban Chakraborty、Andrew Clement、Stephan Herrmann、Markus Keller、Jesper Møller、Manoj Palat、Srikanth Sankaran 和 Olivier Thomann；在 IntelliJ 中支持 Java SE 8 的特性是由下列人员实现的：Anna Kozlova、Alexey Kudravtsev 和 Roman Shevchenko。他们做出的贡献值得整个 Java 社区感谢。

Java SE 8 是 Java 语言的复兴。尽管有些人在追寻“下一种伟大的语言”，但是我们相信用 Java 编程比以往更加令人激动并具有更高的生产率。我们希望它对你而言也是经久耐用的。

Alex Buckley
加利福尼亚州圣克拉拉
2014 年 3 月

第 1 章
The Java Language Specification, Java SE 8 Edition

概　述

Java 编程语言是一种通用的、支持并发的、基于类的面向对象编程语言。它设计得非常简单，使众多程序员都可以熟练地掌握它。Java 编程语言与 C 和 C++ 有着千丝万缕的联系，但是构成却与它们千差万别，Java 摒弃了 C 和 C++ 的许多内容，并且融入了许多其他语言的思想。它的设计者意图使其成为一种用于实际生产的语言而非研究类型的语言，因此，正如 C. A. R. Hoare 在其关于语言设计的经典论文中所建议的那样，其设计绝不包含未经充分测试的新特性。

Java 编程语言是一种强静态类型语言，本规范明确地将可以且必须在编译时探测到的编译时错误与在运行时发生的错误进行了区分。“编译时”通常是指将程序翻译成独立于机器的字节码表示形式的过程，而运行时的活动则包括对执行程序所需的类的加载和链接，非强制性的机器码生成和程序动态优化，以及实际的程序运行。

Java 编程语言是一种相对高级的语言，因为有关机器表示形式的细节在整个语言中是屏蔽的。它特有的使用垃圾回收器的自动内存管理机制，可以避免出现显式释放内存（例如 C 的 `free` 和 C++ 的 `delete`）带来的安全性问题，其高性能的垃圾回收实现方式可以将因垃圾回收导致的暂停时间限定在较小的范围内，从而对系统编程和实时应用提供支持。Java 中不包含任何不安全的结构，例如不带下标越界检查的数组访问，因为这些不安全的结构将会导致程序以某种不详的方式运行。

Java 编程语言通常会按照《Java 虚拟机规范（Java SE 8 版）》[⊖]中定义的字节码指令集和二进制格式来编译程序。

1.1　本书结构

第 2 章阐述了 Java 语言的文法，以及用来表示词法和句法的表示法。

第 3 章阐述了 Java 编程语言的词法结构，这些结构是基于 C 和 C++ 的。Java 程序是用 Unicode 字符集编写的，它支持在只支持 ASCII 的系统上使用 Unicode 字符编写程序。

第 4 章阐述了类型、值和变量，其中类型又被细分为简单类型和引用类型。

简单类型在任何机器上和任何实现中的定义都相同，具体包括各种大小的以 2 的补码形式表示的整数、单精度和双精度的符合 IEEE 754 标准的浮点数、`boolean` 类型，以及表示单个 Unicode 字符的 `char` 类型。简单类型的值互相之间不共享状态。

引用类型包括类类型、接口类型以及数组类型。引用类型是通过动态创建的对象来实现的，这些对象是类或数组的实例，对每个对象都可以有多个引用。所有对象（包括数组）都支持 `Object` 类的方法，这个类是整个类层次结构的（单）根。Java 有一个预定义的 `String` 类，用以支持由 Unicode 字符构成的字符串。此外还有若干个用于将简单值包装于其对象内部的包装器类。在许多情况下，包装和解包都是由编译器自动执行的（在这种情况下，包装

⊖ 《Java 虚拟机规范（Java SE 8 版）》已由机械工业出版社引进并出版，ISBN 为 978-7-111-50159-6。　——编辑注

（wrapping）被称作装箱（boxing），而解包（unwrapping）则被称为拆箱（unboxing））。类和接口声明可以是泛化的，即可以用其他引用类型来参数化它们。这种声明之后可以用具体的类型引元去调用。

变量是类型化的存储位置。简单类型的变量所持有的值是类型精确地为该简单类型的值；而类类型的变量可以持有一个空引用，或是一个可以指向类型为该类或其任何子类的对象的引用；接口类型的变量可以持有一个空引用，或是一个可以指向实现了该接口的任何类的实例的引用；数组类型的变量可以持有一个空引用，或是一个指向某个数组的引用；`Object`类类型的变量可以持有一个空引用，或是一个指向任何对象的引用，不管该对象是类实例还是数组。

第5章阐述了类型转换和数字提升。类型转换会改变编译时类型，并且有时还会改变表达式的值。这些类型转换包括在简单类型和引用类型之间的装箱与拆箱转换。数字提升用于
2 将数字型操作符的操作数转换为可以使操作得以完成的公共类型。在Java语言中不存在任何类型不安全的漏网之鱼，因为引用类型的类型转换在运行时都会被检查，以确保类型安全。

第6章阐述了声明和名字，以及如何确定名字的含义。Java语言不要求类型或其成员的声明必须位于使用它们的代码之前。声明的顺序只对局部变量、局部类，以及类或接口内的域初始化器的顺序有影响。

Java编程语言提供了对名字作用域的控制，并且支持对从外部对包、类和接口的成员所做的访问进行限制。这样可以将类型的实现与它的用户以及扩展者分离开，从而有助于编写大型的程序。本章还介绍了推荐的命名习惯，它们可以使程序更易于阅读。

第7章阐述了程序的结构，Java程序被组织成与Modula的模块类似的包。包的成员包括类、接口和子包。包被划分成编译单元，后者包含类型声明，并且可以从其他包导入类型以便使用短名字引用它们。包的名字位于层次化名字空间中，互联网域名系统通常可以用来构成唯一的包名。

第8章阐述了类。类的成员可以是类、接口、域（变量）或方法。类变量是指在整个类中只有一个实例的变量，而类方法是指不需要通过指向具体对象的引用就可以运行的方法。实例变量是在对象中动态创建的，这些对象是类的实例。实例方法是在类的实例上调用的方法，在其执行过程中，为了支持面向对象编程风格，这种实例就变成了当前对象`this`。

类支持单一实现继承，这是指每个类的实现都是从单一的超类中派生的，并且追根溯源都是从`Object`类派生的。某个类类型的变量可以引用该类或者其任何子类的实例，从而允许新类型可以多态地适用于现有的方法。

类通过`synchronized`方法支持并发编程。方法可以声明在其执行时可能会产生受检类型的异常，这样可以通过编译时的检查来确保异常情况会得到处理。对象还可以声明`finalize`方法，该方法会在对象被垃圾回收器丢弃之前被调用，对象可以在这个方法中清除状态。

为简便起见，Java语言没有从类的实现中分离出“声明头文件”，也没有分离的类型和
3 类层次结构。

类有一种特殊形式，即枚举，它支持以类型安全的方式定义一个小的取值集以及可以施加于其上的操作。与其他语言的枚举不同，枚举是对象，并且可以拥有自己的方法。

第9章阐述了接口类型，即声明了一组抽象方法、成员类型和常量的类型。在其他方面没有关联的多个类可以实现相同的接口类型。接口类型的变量可以是一个指向实现了该接口

的任何对象的引用。Java 支持多重接口继承。

注解类型是特殊化的接口，用来注解各类声明。这些注解不会对 Java 程序的语义造成任何影响，但是，它们对各种各样的工具提供了有用的输入。

第 10 章阐述了数组。Java 中对数组的访问包含越界检查。数组是动态创建的对象，并且可以赋值给 `Object` 类型的变量。Java 语言支持的是元素为数组的数组，而不是多维数组。

第 11 章阐述了异常。Java 的异常机制会改变程序执行的流程，而不会尝试继续执行出错的程序，并且该机制与 Java 语义和并发机制实现了彻底的集成。Java 中共有 3 种异常：受检异常、运行时异常和错误。编译器会确保受检异常能够得到正确的处理，因为它要求一个方法或构造器只有在声明它有可能会抛出某个受检异常时，才能抛出这个异常。这种机制提供了对异常处理器是否存在的编译时检查，从而对编程起到很大的辅助作用。大多数用户定义的异常都应该是受检异常，而被 Java 虚拟机探测到的程序中的无效操作都会产生运行时异常，例如 `NullPointerException`。由 Java 虚拟机探测到的故障会产生错误，例如 `OutOfMemoryError`，但是大多数简单程序都不必费力去处理错误。

第 12 章阐述了在程序执行过程中发生的活动。Java 程序通常存储为二进制文件，这些文件表示编译过的类和接口，并且可以加载到 Java 虚拟机中，与其他类和接口进行链接，以及初始化。

在初始化之后，类方法和类变量就可以使用了。有些类可以被实例化，以创建类型为这种类的新对象。如果一个对象是某个类的实例，那么对于这个类的每个超类，该对象都包含一个相应的引用，并且该对象的创建将包括对这些超类实例的递归创建。

当某个对象不再被引用时，它可以被垃圾回收器回收。如果这个对象声明了终结器，那么终结器将会在对象被回收之前执行，从而给了对象最后的机会去清理资源，否则这些资源就有可能得不到释放。当不再需要某个类时，这个类也可以被卸载。 4

第 13 章阐述了二进制可兼容性，详细说明了修改类型对其他类型的影响，后者是指那些会使用修改后的类型但是自身并没有被重新编译的类型。这些讨论关乎类型开发者的利益，因为他们开发的类型经常会以一系列连续版本的形式，经由互联网广泛发布。优秀的程序开发环境会在一个类型被修改后自动重新编译依赖它的代码，因此大多数程序员无需关心这些细节。

第 14 章阐述了语句块和语句，它们的概念都源于 C 和 C++。Java 语言没有任何 `goto` 语句，但是包含标记为 `break` 和 `continue` 的语句。与 C 不同，Java 编程语言要求在控制流语句中必须是 `boolean` 或 `Boolean` 表达式，并且不会将其他类型隐式地转换为 `boolean` 类型（但是可以将 `Boolean` 拆包为 `boolean`，这是仅有的例外），这种设计是因为希望在编译时能够捕获更多的错误。`synchronized` 语句提供了基本的对象级别的监视加锁机制。`try` 语句可以包括 `catch` 和 `finally` 子句，以便将控制流限定在程序中当前正在运行的局部内，防止控制流转移到该局部之外。

第 15 章阐述了表达式，其中详细说明了表达式赋值的顺序，以便提高其确定性和可移植性。重载的方法和构造器是在编译时解析的，即在众多可应用的候选者中挑选最具体的方法或构造器。

第 16 章阐述了 Java 语言能够确保局部变量在使用前一定先被设置过的精确工作方式。尽管其他所有变量都会自动初始化为缺省值，但是 Java 编程语言不会自动初始化局部变量，这样做是为了避免掩盖编程错误。