新人的未来成长之路
成手的进阶修炼秘籍
高手的查缺补漏宝典
本书，在等你……

Broadview
www.broadview.com.cn

Java

深入解析

透析Java本质的36个话题

梁 勇 阮丽珍 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Java 深入解析

透析Java本质的36个话题

梁 勇 阮丽珍 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书分为 5 章，分别为“基本概念”，“运算符与表达式”，“String 类”，“方法、构造器与变量”，“类与接口”。

通过以上方面介绍那些常见、常用却少为人知的 Java 知识。虽然内容相对基本，但都不是容易解答的。目前更多的开发人员（不乏多年开发经验者），也仅仅停留在 Java 表面的层次，而本书，将更深入一层地去讨论 Java 的话题，令读者耳目一新，知其然，更知其所以然。

图书在版编目（CIP）数据

Java 深入解析：透析 Java 本质的 36 个话题 / 梁勇，阮丽珍编著. —北京：电子工业出版社，2013.11
ISBN 978-7-121-21601-5

Ⅰ. ①J… Ⅱ. ①梁… ②阮… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2013）第 235392 号

责任编辑：付 睿
印 刷：北京中新伟业印刷有限公司
装 订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×980 1/16 印张：19 字数：424 千字
印 次：2013 年 11 月第 1 次印刷
定 价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

众所周知，Java 语言已经是当今主流的语言之一，也几乎是大学计算机系必修的一门计算机语言课程。Java 语言凭借其良好的跨平台性、多线程并发、安全等特征，得到了越来越多的爱好者去学习与使用，可以认为，会 Java 的人很多。不过，这种“会”是相对而言的。在众多从事 Java 领域工作的人群中（其中也不乏一些具有多年开发经验的“老人”），对 Java 的实现或原理并不了解，也就是说，对于一些常见的现象或技术，很多人接触过，但大多数却无法说出具体的原因。而这，也正是本书的写作初衷。

也许有人认为这些知识无足轻重，这是大错特错的……

所谓“合抱之木，生于毫末。”再大的项目，也是一个个模块所组成的。再难的谜题，也是若干简单问题的分解，如果不具备逐个解决简单问题的能力，又如何能处理复杂的项目工程？一屋不扫，何以扫天下？

所谓“千里之堤，溃于蚁穴。” 很多错误就是由于对这些“微不足道”的现象不够了解所致。抽不出时间休息的人，迟早要抽出时间养病。同样的道理，如果今天您没有抽出数小时学习本书的内容，那么，在将来的项目开发中，您可能要花费数天甚至更长时间纠结于更多始料未及的错误。

所谓“养兵千日，用兵一时”，每一点知识的累积，终会有用武之地。也许，它会使您在面试过程中正确地回答一道面试题目；也许，它会使您在开发过程中避开一个陷阱；也许，它会让您更加清楚 Java 底层的实现方式；也许，它能令您在学业上感到更加充实……实际上，也许牛津字典中某一个鲜为人知的生僻单词，在某人的一生中也只接触了某一次，但是，那偏偏出现在 GRE 考试的试卷中。如果本书的内容能够在某个时刻满足您的不时之需，那便是笔者最大的欣慰。

本书的特点：

★内容新颖，独树一帜

本书将深入介绍这些实用但却被大多数人所忽视的话题，对于这些话题，很多人都不甚了

解，而市面上介绍这些内容的书籍也相对较少。针对以上情况，渗透其本质给予解答。有些话题看似简单，但是每个话题背后，都能够透析出非同一般的本质。

★分门别类，内容丰富

本书对这些话题，分为 4 个类别：面试、误区、实现与扩展，并给予突出显示标出。其中，“面试”为该问题可能在求职面试的过程中遇到；“误区”为该问题比较容易犯错；“实现”为从编译器角度或者源代码角度来解析某种原理；“扩展”则是对知识展开讨论，或者更深一层去介绍。不过，4 个类别仅供学习参考，四者之间并没有明显的界限，例如，面试中的问题，一样可能是开发中的误区。

★通俗易懂，由浅入深

对于每个话题，本书都是由浅入深地进行介绍的，开篇即列出其表现形式，并开始对该话题进行分析以及处理，提供程序以供参考说明。对于 Java 中的专有词汇，本书尽量使用最通俗的解释。

★归纳细致，重点突出

在本书中，在每小节的结尾，笔者归纳了所有重要的知识要点，这样总结细致，便于读者理解，同时能够突出重点，使读者一目了然，也省去了读者自行总结的时间与不必要的麻烦。

本书的内容安排：

本书分为 5 章，分别为“基本概念”，“运算符与表达式”，“String 类”，“方法、构造器与变量”，“类与接口”。通过这些方面，介绍那些常见、常用却少为人知的 Java 知识。多数内容都相对基本，但却不是那么容易解答的。相信读过本书之后，读者会对自己、对本书有一个新的权衡，更加深入地熟知 Java 本质。

适合的读者人群：

- Java 程序开发人员
- Java 面试人员
- 对 Java 本质精益求精的人员
- 编程爱好者

对于书中的程序，如果无特殊说明，都是在 JDK1.7 环境下编译运行的。虽然笔者在写作的过程中，尽可能地校验本书的内容，然而，由于时间仓促，加之笔者的水平有限，书中疏漏之处在所难免，欢迎广大读者批评指证，笔者的邮箱是 floatview@foxmail.com。如果您对本书有什么意见或建议，均可联系笔者，来函必复！

目　录

第 1 章

基本概念

话题 1　开门见山——测试你的 Java 水平

抛砖引玉：

在本书初始，笔者准备了若干个 Java 题目，旨在测试读者的 Java 水平。请读者认真阅读并给予回答。

重点摘要：

- 对自身能力进行权衡。

测试自己的水平

1．float 类型在 Java 中占用 4 字节，long 类型在 Java 中占用 8 字节，为什么 float 类型的取值范围比 long 类型的取值范围还大？

2．使用“+”可以连接两个字符串（String 对象），那么，是怎样进行连接的？

3．构造器是否创建了对象？该怎样来证明这一点？

4．如果没有在类中显示声明构造器，则编译器会自动生成一个无参的构造器，那么编译器为什么要自动生成这个无参的构造器呢？有什么作用？

5．i++与++i 到底有什么不同？仅仅是先加与后加的区别吗？

6．移位运算：5 << 35，会首先进行 35 % 32 的求余运算吗？如果是这样，那么 5 << –2 的结果是多少呢？

7. 如果重写了 equals 方法，为什么还要重写 hashCode 方法？如果没有重写 hashCode 方法，会有什么问题？

8．从 JDK1.7 起，switch 语句可以支持 String 类型，那么在底层是如何实现的？

9．静态方法是否可以重写？方法重写与方法隐藏有什么不同？

10．为什么不能在静态方法中使用 this？this 指代的是当前对象，但是，这个所谓的“当前对象”到底在哪里？

11．在 Java 中，类型会在什么时间、什么条件下由 JVM 加载？加载后一定会初始化吗？

12．比起 C / C++中的枚举，Java 中的枚举有什么不同（优势）？枚举是怎样实现的？

13．为什么要为 String 对象建立常量池？String 常量池有什么好处？

14．每个基本数据类型都对应一个包装类型，这些包装类型有什么用？

15．内部成员类是如何绑定外围类对象的？

对于以上问题，如果你能回答 0～5 个，那说明你是一个 Java 新手，处于学习阶段，对 Java 语言有初步的了解。

如果你能回答 6～10 个，那你属于 Java 中级水平，处于成长阶段，具有一定的 Java 功底。

如果你能回答 11～14 个，你属于 Java 中高级水平，处于成熟阶段，对 Java 有着较为深入的理解。

如果你能回答全部问题，并且理解透彻，那想必你一定已经从事 Java 研究多年，是一个经验丰富的“高手”。

不过，不管你处在什么水平，都可以进行下面的阅读，因为以上仅仅是冰山一角，本书所阐述的话题还不仅于此。对于初、中级学者，本书的内容可以令你快速成长，对 Java 语言理解更加透彻，从而在以后的学习中事半功倍；对于“高手”而言，本书的内容也可以令你温故而知新，使你“百尺竿头，更进一步”！

以上问题都可以在本书中找到答案，下面，就开始我们真正的 Java 话题吧。

话题 2　世外隐者——隐居深山的“关键字”

抛砖引玉：

这是一个非常典型的面试题：在 Java 中有没有 goto？

除了 goto，今天我们还要谈到几个特别的标识，分别是 const、true、false、null。这些标识在 Java 语言中，是不是关键字呢？如果是，该如何使用？如果不是，与普通的标识符是否又存在什么区别？

重点摘要：

- goto 与 const 是否为关键字
- true、false 与 null 是什么
- Java 中所有的关键字

goto 与 const　面试

在 C / C++等语言中，使用 goto 可以实现程序的跳转，从某些方面来说其提供了一定的方便性，例如，在多重嵌套的循环中，可以直接从内层循环中跳出外层循环。然而，这种跳转却没有任何限制，可以随意地进行，从而打破了正常的程序流程。如果程序中多处使用 goto，不仅降低程序的可读性，也会对程序的维护与更新造成影响。

因此，为了避免上述情况，Java 语言取消了 goto 的使用，取而代之的是使用循环标签。但是，为了避免程序员自行使用 goto 带来同样的混乱性（例如将方法或变量的名称声明为 goto），Java 语言仍将 goto 定义为一个关键字，用来限制程序员将 goto 作为一个标识符来使用，由于是一个从不使用的关键字，故也称为“保留字”。下面的程序将给予说明。

【例 1.1】goto 的说明。

```
1. package chapter1;
2.
3. public class GotoTest {
4.     public static void main(String[] args) {
5.         int[][] array = {
6.             { 1, 20, 38 },
7.             { 28, 90 },
8.             { 60, 46, 71, 100 }
9.         };
10.        //int goto = 0; 错误，不能使用goto作为标识符。
11.        int number = 0;
12.        outer:
13.        for (int i = 0; i < array.length; i++) {
14.            for (int j = 0; j < array[i].length; j++) {
15.                System.out.println(array[i][j]);
16.                number++;
17.                if (number == 5) {
18.                    //goto here; 错误，在Java中没有这种语法。
19.                    break outer; //使用循环标签代替goto。
20.                }
21.            }
22.        }
```

```
23.         here:
24.         System.out.println(number);
25.     }
26.}
```

程序运行结果如下：

```
1
20
38
28
90
5
```

该程序实现打印二维数组的元素，如果数组元素多于 5 个，则后面的数组元素将不再打印。在 Java 语言中，没有类似 C / C++中的 goto 语法，因此，如果取消程序第 18 行的注释，将会产生编译错误，我们应当使用 break 加循环标签来跳出外层循环（第 19 行）。

尽管如此，goto 依然是 Java 中的一个关键字，不允许作为标识符来使用。例如，如果试图使用 goto 作为一个变量（类、方法等）的名称，同样会产生编译错误（第 10 行）。

在 C / C++中，const 是一个关键字，用来声明一个变量的值是不可改变的（即我们通常所谓的常量），与 goto 类似，Java 语言也将 const 定义为关键字，但是却没有任何语法应用，也就是保留字。使用 const 来作为标识符也是不允许的。

true、false 与 null 面试 误区

在很多集成开发环境中，true、false、null 往往使用与关键字相同的特殊颜色给以标出，这便增加了这 3 个符号的混淆性，很多人认为这 3 个符号也是关键字。然而，这是错误的。在 Java 语言中，这 3 个符号是 3 个字面常量（也称直接量）。其中，true 与 false 是布尔类型的字面常量，null 是引用类型的字面常量。这些就好比“abc”是 String 类型的字面常量，而数值“5”是 int 类型的字面常量一样。

这么说来，这 3 个符号不是很平常了吗？似乎没有必要将其单独列出。也不是，还是有一些差别的。尽管 true、false、null 不是关键字，但是，也不能将其作为标识符使用，否则同样会产生编译错误。从这一点来说，这 3 个字面常量与关键字的表现是相似的。

【例 1.2】3 个字面常量。

```
1. package chapter1;
2.
3. public class ThreeLiteral {
4.     //byte true = 1;
5.     //char false = 'A';
```

```
6.      //int null = 232;
7.
8.      //void true() { }
9.
10.     //class false { }
11.
12.     //interface null { }
13.}
```

在本程序中，取消 4～12 行中任意一行代码的注释均会产生编译错误，这说明 true、false、null 不能当作标识符来使用。

关键字列表　面试

在 Java 中，关键字如表 1-1 所示，其中 goto 与 const 为保留字，到目前为止尚未使用。

■ 表 1-1　关键字

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

📖要点总结：

- 在 Java 中，goto 与 const 作为保留的关键字而存在，虽然未在程序中使用，但是也禁止程序员将其作为标识符来使用。
- true、false 与 null 是 3 个字面常量，并非 Java 中的关键字，在程序中也禁止作为标识符使用。

➔举一反三←

- 指出下面一系列符号中，哪些是 Java 中的关键字。

sizeof、String、volatile、NULL、then、friend、enum、true、bool、goto、end

话题 3　疑团满腹——标识符更深层的思考

抛砖引玉：

标识符，就是用来为 Java 中的类、方法、变量等命名。这样就可以通过标识符来访问相应的类（方法、变量等）。这就好比人的名字一样，每个人都有一个名字与之对应，这样，我们就可以通过人名来确定每一个人。

这看似是十分简单的概念，就算初学者也不会陌生。不过，却有超过 80%的人对标识符定义规则的理解是不准确的，这个结果似乎有些令人吃惊。本节将对标识符进行一系列深入的说明，包括标识符的定义规则、标识符集合、相关使用及标识符的最大长度。

重点摘要：

- 标识符的定义规则。
- 标识符首字符允许使用的字符。
- 标识符非首字符允许使用的字符。
- 勿用“$”。
- 标识符的最大长度。

标识符定义规则　面试　误区

在大多数人的理解中，Java 标识符的定义规则如下。

1．标识符由字母、数字、货币符号（￥、$等）、连接符号（_等）组成。（这里的字母为 Unicode 字符集，而不再局限于传统的 26 个英文字母。）

2．标识符的首字符可以是字母、货币符号与连接符号，但不能是数字。

3．标识符不能与 Java 中的关键字相同。

4．标识符不能和 Java 中预定义的字面常量名称相同（true、false、null）。

以上的定义正确吗？

恩，基本上是正确的，在使用过程中通常也不会遇到问题。但是，更准确地来说，还是存在一定的错误成分。

首先，Unicode 字符集的取值范围为 U+0000～U+10FFFF，但是，并非范围内每一个 Unicode 值（代码点）都与一个字符相对应（部分代码点尚未使用），这也就意味着，并非整个 Unicode 字符集在 Java 中都可以作为标识符。其次，不能作为 Java 标识符首字符的字符集，也不仅仅是简单的 0～9 这 10 个数字字符而已。

可以使用 Character 类中如下两个静态方法来判断标识符的合法性：

```
public static boolean isJavaIdentifierStart(char ch)
public static boolean isJavaIdentifierPart(char ch)
```

后来，由于 Unicode 字符集的扩展，为了能够处理增补字符（U+10000～U+10FFFF），在 JDK1.5 中，新增了两个重载方法：

```
public static boolean isJavaIdentifierStart(int codePoint)
public static boolean isJavaIdentifierPart(int codePoint)
```

其中 isJavaIdentifierStart 用来判断代码点（codePoint）对应的字符是否可以作为 Java 标识符的首字符，而 isJavaIdentifierPart 用来判断代码点对应的字符是否可以作为 Java 标识符的一部分（首字符或首字符后面的字符）。

【例 1.3】合法的标识符个数。

```
1. package chapter1;
2.
3. public class ValidIdentifier {
4.     public static void main(String[] args) {
5.         int startNumber = 0;
6.         int partNumber = 0;
7.         for (int i = 0x0000; i <= 0x10ffff; i++) {
8.             if (Character.isJavaIdentifierStart(i)) {
9.                 startNumber++;
10.            }
11.            if (Character.isJavaIdentifierPart(i)) {
12.                partNumber++;
13.            }
14.        }
15.        System.out.println("Unicode字符集个数：" + (0x10ffff + 1));
16.        System.out.println("可作为标识符首字符的字符个数：" + startNumber);
17.        System.out.println("可作为标识符一部分的字符个数：" + partNumber);
18.        System.out.println("二者之差：" + (partNumber - startNumber));
19.    }
20.}
```

程序的目的是在整个 Unicode 区间（第 7 行），输出可作为标识符的字符个数，运行结果如下：

```
Unicode字符集个数：1114112
可作为标识符首字符的字符个数：100801
可作为标识符一部分的字符个数：102903
二者之差：2102
```

从运行结果可知，可以作为标识符使用的字符只是 Unicode 集合中的一小部分，而且可以作为标识符一部分的字符总数与可以作为标识符首字符的字符总数相差 2102 个，绝不是仅仅

0～9 这 10 个数字而已。

现在，让我们来重新定义一下标识符的定义规则。

1. 标识符的首字符所对应的代码点必须使得 Character 类的 isJavaIdentifierStart 方法返回值为 true，后续字符所对应的代码点（如果存在后续字符的话）必须使得 Character 类的 isJavaIdentifierPart 方法返回值为 true。

2. 标识符不能与 Java 中的关键字相同。

3. 标识符不能和 Java 中预定义的字面常量名称相同（true、false、null）。

4. 标识符的长度必须在系统所支持的范围内（这点是 Java 虚拟机要求的）。

“$”惹的祸 扩展

尽管$可以作为标识符使用，然而我们应该尽量避免对其使用。因为$通常在编译器生成的标识符名称中使用，如果我们也使用这个符号，可能会有一些意想不到的错误发生……

【例 1.4】重复的类型。

```
1. package chapter1;
2.
3. public class User$VIP {
4.     public static void main(String[] args) {
5.         User user = new User();
6.         User.VIP vip = user.new VIP();
7.         vip.print();
8.     }
9. }
10.
11.class User {
12.    class VIP {
13.        void print() {
14.            System.out.println("成员类");
15.        }
16.    }
17.}
```

仔细阅读一下代码，有什么问题吗？

真的没有什么问题，代码也比较简单，可是，当编译这段程序的时候，编译器无情地报告了如下的错误：

```
D:\Test\src\chapter1\User$VIP.java:12: 错误: 类重复: chapter1.User.VIP
    class VIP {
    ^
1 个错误
```

提示 User.VIP 类重复声明了，可是，真的是独一无二的 VIP，没有重复声明啊！

确实，并没有重复定义，归根结底，都是“$”惹的祸。因为“$”被编译器所使用，在源文件（.java 文件）编译成字节码（.class 文件）后，会成为顶层类型与嵌套类型之间的连接符。例如，如果存在一个顶层类 A，在其内声明一个成员类 B，那么编译之后就会产生两个 class 文件，分别为 A.class 与 A$B.class。就本程序来说，会生成 3 个 class 文件，分别是 User$VIP.class（顶层类）、User.class 与 User$VIP.class（User 类的成员类，也就是类 VIP）。由于试图存在两个 User$VIP.class，因此才会产生如上的编译错误。

这个问题说明，我们不要图新鲜使用符号“$”，因为 Java 中的标识符真的有很多，完全没有必要去跟编译器争用这个字符。

标识符的最大长度　扩展

在 Java 语言规范中，标识符的长度是任意的。但是，在 Java 虚拟机规范中，标识符是有长度限制的。在 class 文件中，代表标识符的常量字符串存储在 CONSTANT_Utf8_info 表中，而该表使用两个字节（length 项）来表示字符串的长度，由于 length 是无符号类型，因此最大长度为 $2^{16}-1$，即 65535。这也就是标识符的最大长度。但是，这个最大长度仅限于除了空字符 null 以外的 ASCII 字符（‘\u0001’ ～ ‘\u007f’），如果标识符中含有这个范围以外的字符，最大长度将会减少。具体请参考【话题 20 追本溯源——String 字面常量的“极限”】。

📖要点总结：

- 在 Java 语言中，标识符是区分大小写的，仅当两个标识符的 Unicode 字符序列完全相同时，这两个标识符才是相同的。
- 标识符的首字符所对应的代码点必须使得 Character 类的 isJavaIdentifierStart 方法返回值为 true，后续字符（如果存在后续字符的话）所对应的代码点必须使得 Character 类的 isJavaIdentifierPart 方法返回值为 true。并且不能与关键字、布尔字面常量（true、false）和引用字面常量（null）相同。
- 应该避免在标识符中使用“$”，尽管“$”可以在标识符中使用，但可能会与编译器的命名相冲突。
- 当标识符的所有字符都是除了 null 之外的 ASCII 字符时，其最大长度可以达到 65535，如果超过了这个长度，编译器将产生错误信息。

➔举一反三←

- 参考【话题 20 追本溯源——String 字面常量的“极限”】，编写程序，用来测试并且输出标识符的最大长度。

话题 4　鞭长莫及——我的特殊字符，你不能用！

抛砖引玉：

还有不能正常使用的字符吗？这貌似很奇怪，不过，的确如此。在 Java 程序中，某些字符在程序中有着特殊的含义。例如，双引号用来界定一个字符串常量，如果需要在字符串中使用双引号，就需要将双引号转义，否则编译器会当作界定符来处理，从而产生编译错误。另外，增补字符需要使用两个字符才能表示，那么，如果遇到两个字符，该怎样来区分是两个单独的字符还是一个增补字符呢？

下面就请看这些“特殊”的字符吧。

重点摘要：

- 转义序列符的使用。
- 八进制转义的使用。
- Unicode 转义的使用。
- 3 种转义字符的区别与联系。
- Unicode 转义字符的特殊性。
- 增补字符的使用。

转义字符介绍　面试

首先，请看下面的一个转义示例。

【例 1.5】转义字符。

```
1. package chapter1;
2.
3. public class Escape {
4.     //char c1 = '\u0027';
5.     //char c2 = '\u005c';
6.     //String s = "\u0022";
7.     //char c3 = '\400';
8.     //char c4 = '\28';
9. }
```

在这个程序中，如果取消第 4～8 行的任意一行代码的注释，都会产生编译错误。对于第 4～6 行，3 个 Unicode 转义所表示的字符分别为“'”（单引号），“\”（反斜杠）与“"”（双引号），