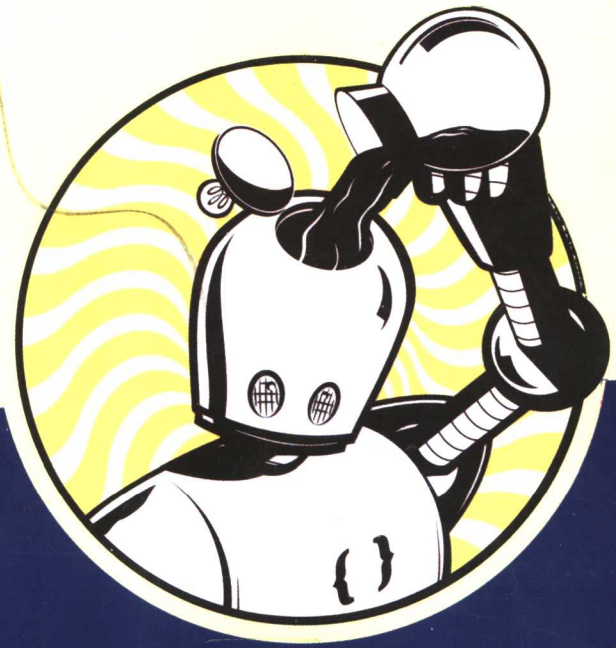
Jolt大奖入围作品！

Wicked Cool Java

code bits, open-source libraries, and project ideas

Wicked Cool Java 中文版

——代码、开源类库与项目创意



(美) Brian D. Eubanks 著

王海涛 宋丽华 译

清华大学出版社

NO STARCH PRESS

Wicked Cool Java 中文版

——代码、开源类库与项目创意

(美) Brian D. Eubanks 著

王海涛　宋丽华　译

清华大学出版社

北　京

Wicked Cool Java: Code bits, open-source libraries, and project ideas
Brian D. Eubanks
EISBN：1-59327-061-5

图书在版编目(CIP)数据

Wicked Cool Java 中文版——代码、开源类库与项目创意/ (美)尤班克斯(Eubanks, B.D.)著；

王海涛，宋丽华 译. ——北京：清华大学出版社，2007.11

书名原文：Wicked Cool Java: Code bits, open-source libraries, and project ideas

ISBN 978-7-302-16259-9

Ⅰ. W… Ⅱ. ①尤… ②王… ③宋… Ⅲ. Java 语言—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2007)第 155366 号

责任编辑：王 军 王 婷
装帧设计：孔祥丰
责任校对：成凤进
责任印制：李红英
出版发行：清华大学出版社 **地 址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编**：100084
c-service@tup.tsinghua.edu.cn
社 总 机：010-62770175 **邮购热线**：010-62786544
投稿咨询：010-62772015 **客户服务**：010-62776969
印 刷 者：清华大学印刷厂
装 订 者：北京市密云县京文制本装订厂
经 销：全国新华书店
开 本：185×230 **印 张**：15.25 **字 数**：289 千字
版 次：2007 年 11 月第 1 版 **印 次**：2007 年 11 月第1 次印刷
印 数：1～4000
定 价：29.80 元

前　言

本书主要介绍由 Sun 微系统公司创建的 Java 编程语言。本书的读者对象是不同专业知识水平的所有 Java 开发人员，以及那些希望寻找有趣的和有用的 API 或项目创意的开发人员。从最初发布以来，Java 已经经过了漫长发展。当我于 1996 年第一次接触 Java 时，当时的版本是 1.0，并且只有少数人知道 Java。当时几乎没有任何有经验的 Java 开发人员，且只有非常少的代码库，并且没有企业服务器。当时看来，它只是一种不成熟的语言，但是具有远大的前程。其实这些都是次要的，JVM 的平台无关性功能才是 Java 不断成长的关键。目前，经过 10 年的发展，Java 已经成为无所不在的成熟技术。核心 API 自身包含大量主题领域中的三千多个类。世界 500 强公司现在都围绕 Java 来构建整个公司计算策略，数以百万计的网站都在运行 servlet 和客户端的 applet。

但许多开发人员认为这就是使用该语言可以做的所有工作，我对此感到非常惊讶。除了核心内容外，Java 还有许多免费的财富，即开放源代码的库。本书就是为了介绍这些库，以及介绍一些 Java 可以实现的有趣的和有用的内容而编写的。而不是为了编写一本 Java 编程参考书籍——已有许多这样的优秀书籍可帮助人们学习编写 Java 代码。相反，我希望读者可以通过发现使用 Java 可以完成许多有趣的事来发现乐趣，而不是按部就班地学习。本书主要为对 Java 有一定了解(从基本了解到中等程度的了解)以及正在寻找改进其代码方法的程序员而编写。

本书中有许多示例代码和各种 API 的“Hello World”程序。其中的一些是介绍性的示例，而其他的一些示例则较为高级。书中有时会给出一些代码提示，有时则提出关于新项目的想法，如果某个人曾经试着实现这些想法，则这些项目就可能成为成功的开放源代码项目！如果你实现了任何有关这些项目的一个想法，那么我鼓励你加入本书的网站，和其他读者一同协作来完成开发以避免犯同样的错误。开发人员最应该做的事情是，协同工作并和其他人共享工作成果。

0.1　本书的结构

本书分为 8 章，每一章介绍了 Java 的特定领域。下面给出了每一章的简短描述。

第 1 章

在第 1 章中，我们讨论了一些核心 API 特性——一些存在较长时间但却很少有人知道的特性，而其他一些特性则是 Java 5 的新特性。我们对新的 for 循环、枚举、泛型、匿名类和断言展开了讨论。

第 2 章

在第 2 章中，我们讨论了字符串处理技术。首先介绍正则表达式，随后讨论随机文本、数组、二进制字符串以及消息格式化。Regex 是功能非常强大的工具，可用来搜索、拆分和替换文本。从 Java 1.4 开始出现该工具，但许多 Java 程序员新手仍然不熟悉该工具。正则表达式是分析复杂文档的好起点。

第 3 章

在本章中，我们处理 XML 和 HTML 文档以及其他类型的结构化文本。我们介绍了分析器生成器并且给出了一些使用该工具的示例代码。

第 4 章

第 4 章介绍了语义网(Semantic Web)，这是下一代的网络，我们将从概念上而不是用包含文本的文档实现事物的真正关联。我们介绍了一些使用 RDF 和 RSS 的 API。

第 5 章

本章介绍了 Java 中的科学和数学应用程序。我们链接了大量开放源代码的项目，这些项目使用了不同方面的科学和数学知识。

第 6 章

本章中讨论了能简化图形应用程序开发及方便添加新特性的 API。

第 7 章

本章处理声音和音乐 API，其中还介绍了高级的线程同步。

第 8 章

本书的最后一章介绍了其他开放源代码的项目，并且讨论了有关创建自己的项目

或创建集成前面章节中代码的项目的想法。

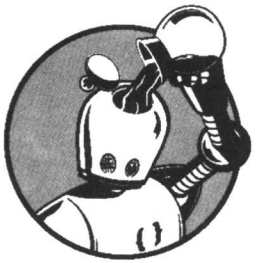
0.2 本书的网站

Java 非常有趣，我们在本书中介绍了一些优秀的项目，也介绍了一些可用于你自己的项目的有用工具和技术。为了最有效地发挥 Java 的作用，你需要在某些情况下利用开放源代码的项目，这可以在本书的配套网站 http://wickedcooljava.com 中找到。该网站提供了正文、勘误表和代码示例中提及的项目的链接。此外，该网站中还提供了方便读者讨论本书内容及组织新项目的论坛。我希望能以此鼓励你学习 Java 核心 API 以外的内容并继续更深层次的学习。

目　录

1

JAVA 语言及核心 API



本书假定你对 Java 有所了解并且熟悉 Java 语言及它的核心应用编程接口(Application Programming Interface，API)的基本知识。随着 Java 5 的发布(也称为 Java 1.5 或 Tiger)，Java 语言及核心 API 中添加了一些重要的新特性。本章将通过示例来介绍一些新特性以及 Java 开发人员不了解的一些有用的类和方法。Java API 非常庞大，它的类超过 3000 个，要想了解所有的内容非常困难。在核心软件包中，有一些类甚至连有经验的 Java 开发人员都可能忽视，或许是因为他们以前从未接触过这些类或者不需要使用它们。

很多编程人员把 Java 仅作为另一种语言和语法来学习。但是高效的 Java 编程人员是从对象层次结构、行为和关系的角度来审视 Java，而不是像在过程语言中那样从函数调用和过程的角度来学习 Java。在介绍性讨论中，将强调 Java 的以下三个相关的部分。

面向对象的语言

高效地使用 Java 意味着要理解面向对象编程(object-oriented programming，OOP)，而不仅仅是 Java 的语法。如果你对过程语言，如 C 或 Pascal 语言，比较熟悉，那么在

编写 Java 代码之前亲自熟悉 OOP 是非常重要的。本书假定你已经在某种程度上使用过 Java。如果你不熟悉 OOP，可以在网站 http://java.sun.com/docs/books/tutorial/java/concepts 上找到有关 OOP 的介绍。此外，本章还会讨论一些最新的语法变化。

Java 虚拟机(Java Virtual Machine，JVM)

当 Java 发布时，它最引人注目的方面就是它的 Java 虚拟机。不同于其他语言中需要为每个目标平台重新编译或修改源代码，Java 中只需要编译源代码一次。然后可以在已安装 JVM 的任何目标系统上运行可执行代码，或字节码(bytecode)。字节码就像一种跨平台的机器语言。除了 Java 外，其他语言也可以编译成 JVM 字节码，在后面的章节中会看到这方面的内容。

核心 API

Java 核心库的规模有时对开发人员来说过于庞大。它包括针对网络、图形、声音、多线程、I/O、安全、加密、数据库、XML 以及很多其他方面的软件包。在后续章节中，将会探讨一些核心 API 的高级特性。此外，还会介绍一些开放源代码的 Java 项目，这些项目能提供核心 API 未提供的有用功能。本章中，我们会使用少量核心类和一些源自 Java 5 的新语法。

1.1 在 Java 中没有 for：使用增强的 for 循环

JAVA 5+

在一些编程语言中，通过列表或数组可以非常方便地进行遍历，通过一个循环即可逐个遍历项并将该项赋值一个局部变量从而实现自动循环。我曾经告诉过一个同事我认为 Java 中的“for”循环功能是不完全的，因为它没有“for-each”。我的朋友也是一个有经验的 Java 开发人员，他的回答是“你疯了吗？，在 Java 中当然有 for!”在此之后很长一段时间里他为此而不断地嘲笑我，并定期地提醒我在 Java 中存在 for(为了防止我遗忘此事)。但是我有一个好消息要告诉他和所有 Java 开发人员：目前在 Java 中有了真正的 for!

考虑这样一种情况，你希望对一个整型对象集合(如 java.util.ArrayList)中的所有数值求和。你很可能编写过类似于下面这样的代码：

```
ArrayList theList = new ArrayList();
theList.add(new Integer(2));
theList.add(new Integer(3));
```

```
theList.add(new Integer(5));
theList.add(new Integer(7));
int sum = 0;
// The old way to iterate
for (Iterator iter = theList.iterator(); iter.hasNext(); ) {
     Integer x = (Integer) iter.next();
     sum = sum + x.intValue();
}
System.out.println("The sum is " + sum);
```

这段代码多麻烦啊，难道编译器不应该知道你正在进行迭代吗?毕竟这是一个 for 循环，不是吗？而在 Java 5 中，增强的 for 循环现已支持集合对象。因此不再需要使用迭代器。在下面的修订的代码中，一个 for 循环通过列表进行迭代并显示出每个值：

```
ArrayList<Integer> theList = new ArrayList<Integer>();
theList.add(2);
theList.add(3);
theList.add(5);
theList.add(7);
int sum = 0;
// new Java 5 iteration syntax
for (Integer item : theList) {
     sum = sum + item;
}
System.out.println("The sum is " + sum);
```

for 循环定义了一个叫做 item 的局部变量，在每次迭代过程中，它将得到列表中的下一个值。除了完美的 for 循环语法外，此代码在以下两个方面也不同于过去的 Java 代码。

使用了泛型

上面带有尖括号的语法是 Java 5 新增加的泛型特性。泛型允许为一些具体类型的对象定义类，但是直到创建该类的一个实例时才能知道具体的类型。编译器将会强迫执行类型检查。在这个示例中，ArrayList 是一个特殊的类，对于 add 方法它只接受整数(并只从它的 Iterator 的 next 方法中返回整数)。这意味着当从列表中检索对象时不需要强制类型转换，可以立即将它们作为 Integer 实例来对待。不使用泛型时，仍可以使用新的 for 循环语法，但需要将 Object 强制转换成 Integer。在 1.4 节中将我们更详细地介绍泛型。

整型对象到整型数值的自动转换

在 Java 5 中，可以将 Integer 对象作为 int 来对待。编译器将自动执行从 int 到 Integer 对象的转换(反之亦然)，此过程称为自动装箱(autoboxing)。当循环中得到一个 Integer 对象时，可以将它与一个 int 值相加而不需要执行显式的转换。

新的 for 语言也适用于数组：

```
int[] theList = new int[] {2,3,5,7};
int sum = 0;
for (int x : theList) {
     sum = sum + x;
}
System.out.println("The sum is " + sum);
```

这种新的语法的确使代码变得更加易读和紧凑。但还不能完全放弃迭代器，至少暂时是这样，因为还有很多开发人员没有将他们的 JDK 升级到版本 5。

1.2 计数：使用枚举

JAVA 5+

大多数应用程序需要记录一个值的有限集——即应用程序中表示一组选择或状态的常量。一种常见的 Java 编程惯例是使用 static int 变量来表示这些值。然后让程序通过比较这些值和其他变量的值来做出决定。尽管核心 Java API 本身也采用这种惯例，但它很可能导致严重的问题！下面是一个有关水果信息的示例。该示例给出了使用 int 变量来表示枚举数据时会出现的一些问题。

```
public class FruitConstants {
   // this is not such a good practice
   public static final int APPLE = 1;
   public static final int ORANGE = 2;
   public static final int GRAPEFRUIT = 3;
   public static final int BANANA = 4;
   public static final int DURIAN = 5;
   public static final int CITRUS = 6;
   public static final int SWEET = 7;
   public static final int SMELLY = 8;
   public static final int UNKNOWN = 9;

   public static int getCategory(int fruit) {
```

```
        switch(fruit) {
            case APPLE: case BANANA:
              return SWEET;
            case ORANGE: case GRAPEFRUIT:
              return CITRUS;
            case DURIAN:
              return SMELLY;
        }
        return UNKNOWN;
    }
}
```

其中，来自东南亚的水果榴莲(Durian)具备“Sweet”和“Smelly”，但是这里的水果只能返回一个特点，也就是说，所有的值只能是对应单一的值而不是对应多个值。另外一个主要的问题是任何 int 值都可以传递给 getCategory 方法，无论它是否代表一个有效的水果。这可能导致细微的错误，因为编译器不关心是调用 getCategory(SWEET)还是调用 getCategory(42)。并且如果整型常量的值发生了变化，getCategory(3)显示的将不再是正确的信息！

另一个问题是使用水果和类别的 int 值并无区别——它们都只是普通的 int 值。通过简单地将类别常量置于一个不同的类中可以部分地解决水果和类别分离的问题，但是它们仍只是 int 值而不是类型安全(typesafe)的，也就是没能将 getCategory 的参数限制到一个固定的值集合。

在 Java 5 中，有一个优雅的解决方法：可以像在 C 语言中那样创建枚举类型。这是一个新特性，它创建一个类，该类包含一个所有它允许的实例清单。除了 enum 内定义的实例外，不允许其他的实例。下面看一些 enum 的示例：

```
enum Fruit {APPLE, ORANGE, GRAPEFRUIT, BANANA, DURIAN}
enum FruitCategory {SWEET, CITRUS, SMELLY, UNKNOWN}
enum Dessert {PIE, CAKE, ICECREAM, BROWNIE}
```

以上每个例子均定义了一组不同的枚举元素(选择)。这样做的好处是不会将 Fruit 值与其他类型的相混合或相混淆。对待每个 enum 就好像它是一个不同的类一样。不能将 FruitCategory 作为一个参数传递给一个期望 Dessert 的值方法，也不能传递一个 int 值。下面扩展 Fruit enum 以包含最初的 FruitConstants 类具有的功能：

```
public enum Fruit {
   APPLE, ORANGE, GRAPEFRUIT, BANANA, DURIAN;
```

```
    public static FruitCategory getCategory(Fruit fruit) {
        switch(fruit) {
            case APPLE: case BANANA:
                return FruitCategory.SWEET;
            case ORANGE: case GRAPEFRUIT:
                return FruitCategory.CITRUS;
            case DURIAN:
                return FruitCategory.SMELLY;
        }
        return FruitCategory.UNKNOWN;
    }
}
```

可以看出一个 enmu 也可以像一个类那样定义方法。现在的 getCategory 方法采用 Fruit 作为参数，并且只允许使用 enum 中定义的值。接下来这个代码段将引起编译错误，而不会出现调用最初的未受保护的 getCategory 方法时出现的运行时异常：

```
Fruit.getCategory(Dessert.PIE); // compile error
Fruit.getCategory(10); // compile error
```

如果每种水果管理它自己的类别将会更好，因此为了完善 Fruit 类，将删除 getCategory 的水果参数并使该方法为每个 enum 状态返回不同的值。通过创建一个适用于所有值的抽象的 getCategory 方法并对每个 enum 以不同的方式重写它，可以完成此操作。它非常类似于为每个枚举值编写一个不同的子类，并让每个这样的子类重写抽象的方法。

```
public enum Fruit {
  APPLE
    { FruitCategory getCategory() {return FruitCategory.SWEET;} },
  ORANGE
    { FruitCategory getCategory() {return FruitCategory.CITRUS;} },
  GRAPEFRUIT
    { FruitCategory getCategory() {return FruitCategory.CITRUS;} },
  BANANA
    { FruitCategory getCategory() {return FruitCategory.SWEET;} },
  DURIAN
    { FruitCategory getCategory() {return FruitCategory.SMELLY;} };

  abstract FruitCategory getCategory();
}
```

一旦创建一个类似这样的 enum，就可以像对待其他对象那样来对待 APPLE 值(使

用静态的 Fruit.APPLE 引用)，并且可以调用它的 getCategory 方法来得到它的相应类别。现在可以为上面的类添加一个 main 方法来说明如何使用新的 Fruit enum：

```
public static void main(String[] args) {
    Fruit a = Fruit.APPLE;
    // toString() returns "APPLE"
    System.out.println ("The toString() for a: " + a);
    // getCategory() returns "SWEET"
    System.out.println ("a.getCategory() is: " + a.getCategory());
    for (Fruit f : Fruit.values()) {
      System.out.println ("Fruit is: " + f);
    }
}
```

正如代码中所示的那样，可以使用 values 方法来迭代 enum 内的所有值。enum 的 toString 方法将会返回一个 String，它具有和值相同的名字。使用 enum 而不是 int 来表示状态，可以使代码具有更好的可读性和更强的防错性。它明确地定义了一个特殊的枚举状态的所有值并可以防止有人使用不正确的值。

1.3 执行安全存放：使用类型安全映射

JAVA 5+

正如前面讨论 for 循环时看到的那样，使用泛型有助于简化代码并降低出错概率。for 循环会假定 ArrayList 仅包含 Integer 对象，因为 ArrayList 严格地被定义为由 Integer 对象组成。因而当从列表检索项目时可以避免从 Object 到 Integer 的强制类型转换。

Java 5 对核心 API 做出了很多利用泛型的更改。查看相关文档你会发现已重新定义了很多类以允许使用泛型。如果愿意的话，仍可以按照以前的方式构造和使用这些类，例如使用 new ArrayList()。这样做的原因是为了兼容性，以便仍可以在旧版本的编译器下运行代码。当然，这样会失去泛型提供的类型检查带来的便利性和安全性。

一个得到很好修订的类是 java.util.Map (和 HashMap)。我们知道，映射操作就像查表一样，每个值都存储在一个唯一的键标下。在早期的 Java 版本中，当在映射表中放置表项时，它们是作为 Object 项存放的。当从映射表中检索表项时，它被作为标准的 Object 引用来对待，即被强制转换成正确的子类以便能够识别为它的实际类型。这与 List 中存在的危险相同。要么对象不正确，要么出现 ClassCastException 异常，这样的情况太常见了。