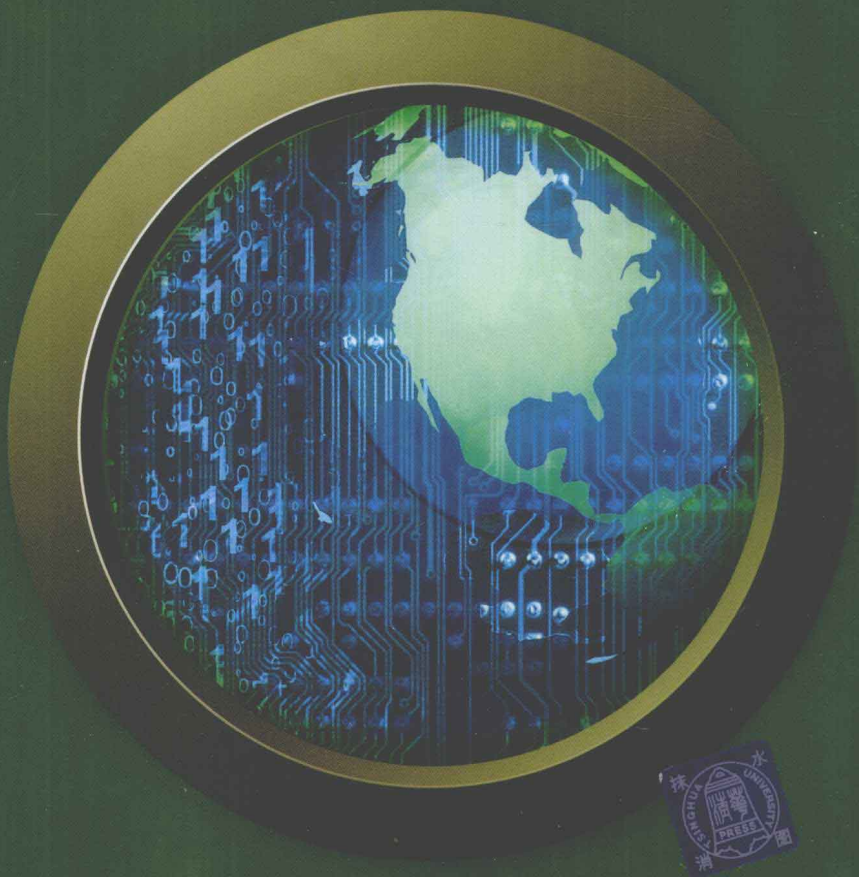
高等院校计算机应用技术系列教材

Java程序设计高级教程

赵卓君 主 编
代俊雅 魏志军 张权范 夏力前 副主编



清华大学出版社 • 北京交通大学出版社

高等院校计算机应用技术系列教材

Java 程序设计高级教程

赵卓君　主　编
代俊雅　魏志军
张权范　夏力前　副主编

清华大学出版社
北京交通大学出版社
·北京·

内 容 简 介

Java 已是目前世界最流行的高级编程语言之一。自诞生以来，Java 迅速成为开发互联网应用程序首选的编程语言。本书特针对普通高等院校计算机专业 Java 教学而编写，对 Java 的泛型和泛型接口、序列化和反序列化、GUI 编程、JDBC 数据库连接、线程和多线程、Applet 和网络媒体通信等 Java 高级知识点作了深入讲解。本书应用了大量完整的程序案例来辅助说明，基本涵盖 Sun 公司 SCJP Java 认证要考核的知识点。

本书既可作为普通高等院校计算机本科专业的 Java 教材，也可作为（SCJP）认证考试的辅导用书。

图书在版编目(CIP)数据

Java 程序设计高级教程/赵卓君主编. —北京：清华大学出版社；北京交通大学出版社，2011. 8

（高等院校计算机应用技术系列教材）

ISBN 978 - 7 - 5121 - 0657 - 4

Ⅰ. ① J…　Ⅱ. ①赵…　Ⅲ. ① JAVA 语言 - 程序设计 - 高等学校 - 教材　Ⅳ. ① TP312

中国版本图书馆 CIP 数据核字(2011)第 145068 号

责任编辑：谭文芳

出版发行：清 华 大 学 出 版 社　邮编：100084　电话：010 - 62776969　http://www. tup. com. cn

北京交通大学出版社　邮编：100044　电话：010 - 51686414　http://press. bjtu. edu. cn

印 刷 者：北京交大印刷厂

经　　销：全国新华书店

开　　本：185 × 260　印张：20　字数：512 千字

版　　次：2011 年 8 月第 1 版　2011 年 8 月第 1 次印刷

书　　号：ISBN 978 - 7 - 5121 - 0657 - 4/TP · 652

印　　数：1 ~ 4 000 册　定价：33. 00 元

序

Java 程序设计语言是一种完全面向对象且平台无关的高级编程语言。在面向对象技术越来越成为主流技术，网络应用编程的需求越来越扩大的今天和未来，Java 已经并也无疑将成为未来的主流程序设计语言之一。

在长时间的教学工作中，尤其在独立学院中，我们发现许多学生对于面向对象技术和面向对象程序设计语言的学习和应用颇感困难。无论他们是否曾经学习过其他程序设计语言，他们对于抽象与建模的掌握能力较差。因此，在 Java 程序设计教学中，结合大量案例，训练学生的抽象和建模能力，并应用 Java 语言进行实现显得十分重要。

本书的几位作者长期从事 Java 语言的教学工作，并且参与过许多基于 Java 和 J2EE 平台的实际软件工程项目开发，具有丰富的教学和实际应用经验。作者使用通俗易懂的语言，结合大量完整的案例阐述 Java 语言的思想及应用基础知识，还包括了实验和课后练习题，并提供所有案例的源代码、上机实验和课后练习答案。

本书可作为普通高等院校尤其是独立学院 Java 语言的教材使用，也可以作为一般 Java 语言初学者的自学参考书。相信本书的出版将有助于 Java 语言的教学和初学者的自学，也希望作者注意收集使用者的意见，并在使用过程中进一步优化该书的内容与结构。

林国璋 教授
北京理工大学珠海学院
计算机学院院长
2011 年 5 月

前　言

Java是一个完全面向对象的高级编程语言。从1995年问世以来逐步被IT界所接受。Java语言拥有丰富的类库，具有平台无关、简单安全等特点，又解决了互联网上的大型应用问题，一出现就广泛被应用在多个领域。

市面上关于Java的书用多如“恒河沙数”来形容也一点不夸张。我们之所以编写这套教材倒不是为了“凑热闹”。近年来，几乎所有普通高等院校的计算机本科专业都开设了Java程序设计语言这门课。很多学生在学过C语言程序设计有了一定程序设计基础之后再来学Java语言仍然遇到不少困难。有些学生甚至反映Java语言难学，不如C#容易入门、较易掌握。我们在几年的Java程序设计语言的实际教学中也遇到了类似的问题。在不断地解决这些问题的过程中我们积累了一定的经验，深知对于难以理解的知识点在理论课上需要深入浅出地阐述，在实验课上要细心地引导学生，从实际动手中让学生进一步理解知识点。

本书紧接着编者编写的上一本书《Java程序设计基础教程》，继续用通俗易懂的语言来讲解Java语言的高级知识，并用大量完整的案例来辅助说明这些知识点。本书共分为七章，将详细介绍Java的泛型和泛型接口、序列化和反序列化、GUI图形用户界面编程、JDBC数据库编程、线程和多线程、Applet和网络编程。在编写中，我们尽量保证Java知识体系的完整性，也尽量考虑Java教学中的逻辑顺序。为了让教师和学生更好使用本书，每章都安排了实验和课后练习。我们还将提供本书的所有案例源代码、上机实验和课后练习答案（读者可自行向出版社索取）。本书的最后还附有综合复习题以便读者进行自我练习。关于Java语言的基础知识已在《Java程序设计基础教程》中讲解。

参与本书编写的人员一共有5位老师，分别是：赵卓君、代俊雅、魏志军、张权范、夏力前，均来自北京理工大学珠海学院计算机学院。赵卓君负责本书前言、第3章GUI图形编程、第4章JDBC数据库编程的编写；代俊雅负责第1章泛型和泛型接口的编写；魏志军负责第2章序列化和反序列化的编写；夏力前负责第5章线程和多线程的编写；张权范负责第6章Applet和网络编程的编写。该书主编是赵卓君，其余编著者的署名按作者姓名的拼音排序，排名不分先后。该书的整合和校对工作由赵卓君完成。

参与本书编写的老师平时都有繁重的教学任务，但仍然坚持完成本书的编写。在这里向所有参与本书编写的老师表示衷心的感谢。没有大家的努力，就没有这本书。另外，在写作过程中也获得很多其他老师的帮助和支持，尤其是北京理工大学珠海学院计算机学院院长林国璋教授的大力支持。林院长还亲自参与本书的审核，给予了很多宝贵的建议并撰序。在这里特别鸣谢林国璋教授。

尽管我们已经尽了最大的努力，但是仍然难免有错误和疏漏之处。因此，在这里恳请所有的读者不吝指出，以便我们修改。再次感谢所有读者的鼓励和支持。

编　者

2011年4月

目　录

第 1 章　泛型和泛型接口

本章要点：

- ☑ 类型安全问题
- ☑ 什么是泛型
- ☑ 泛型进阶：泛型类、泛型方法和泛型接口
- ☑ 受限类型参数
- ☑ 类型通配符

在编程过程中，人们经常遇到这样的情形：某些程序除了所处理的数据类型之外，程序代码和程序功能完全相同，但为了实现它们，却不得不编写多个与具体数据类型紧密结合的程序。能不能将数据类型参数化，使得代码重用成为可能呢？

JDK5.0 提出了泛型的概念，通过使用泛型可以写出更为通用的程序代码。本章将对 Java 中的泛型技术进行介绍。

1.1　类型安全问题

Java 集合框架允许在 Java 的集合类对象中存储各种类型的元素，因为这些类型的基类是 Object。但是，当访问这些对象时，通常需要进行强制类型转换。此时，程序员必须明确地了解数据在此集合中的存放情况。否则，不合适的强制类型转换会发生运行时异常。

【例 1-1】 使用传统方法访问集合类。

```
/**
 * 文件名:AccessCollection.java
 * 使用普通(非泛型)方式访问集合框架
 */
import java.util.*;
public class AccessCollection {
    public static void main(String[] args) {
        ArrayList lst = new ArrayList();
        lst.add(0,"Ann");
        lst.add(1,"Bill");
        lst.add(2,"List");
        lst.add(3,new Integer("1"));
        lst.add(4,"Smith");
        lst.add(5,"Tom");
```

```
            Iterator it = lst. iterator( ) ;
            while ( it. hasNext( ) ) {
                String s = ( String) it. next( ) ;
                System. out. println( s) ;
            }
        }
    }
```

对该程序进行编译，没有产生编译错误，接下来，运行该程序，结果如下：

```
Ann
Bill
List
Exception in thread " main" java. lang. ClassCastException: java. lang. Integer cannot be cast to java. lang. String
at AccessCollection. main( AccessCollection. java:15)
```

很显然，程序在将 Integer 型数据转换为 String 类型时发生了运行时错误。这是因为编译器并不做类型检查，这种检查是在运行时进行的。从软件工程的角度来看，这对软件的开发是非常不利的。

实际上，集合的典型应用是存储同种类型的元素。为了达到此目的，要求定义集合时，规定向集合中添加的所有元素均为同一种类型，但是这个类型又不能被指定为特定的类型，也就是能明确地让系统理解定义的集合中元素的数据类型位置只是一个占位符，或者说只是一个“类型的参数”，以便将来使用这个集合时，集合中的元素只能用指定的某种类型替换。为了解决这个问题，从 JDK 5.0 开始，引入了泛型的概念。

1.2 什么是泛型

所谓泛型，就是在创建集合对象时就规定其允许保存的元素类型，然后由编译器负责检查所要添加元素的合法性，在取用元素时就不必再进行强制性的类型转换处理。其实就是将原本确定不变的类型参数化。通过使用泛型，保证了集合类对象中存储的元素数据类型相同。将例 1-1 进行修改，加上泛型机制，代码如例 1-2 所示。

【例 1-2】 使用泛型技术访问集合。

```
/**
 * 文件名:AccessCollectionPro. java
 * 使用泛型访问集合框架
 */
import java. util. *;
public class AccessCollectionPro{
```

```
    public static void main(String[ ] args){
        List < String > lst = new ArrayList < String > ( );
        lst. add( "Ann" );
        lst. add( "Bill" );
        lst. add( "List" );
        lst. add( new Integer( "1" ) ); //编译错误
        lst. add( "Smith" );
        lst. add( "Tom" );

        Iterator < String > it = lst. iterator( );
        while ( it. hasNext( ) ) {
            String s = it. next( ); //不需要强制类型转换
            System. out. println( s );
        }
    }
}
```

该程序编译时，发生错误。错误信息如下：

```
E:\java 高级教程\ch01\ AccessCollectionPro. java:8: 找不到符号
符号: 方法 add(java. lang. Integer)
位置: 接口 java. util. List < java. lang. String >
        lst. add( new Integer( "1" ) );
             ^
1 错误
```

可以看出，该程序创建 List 类的对象时，在 List 之后加了 < String > 类型，用于指定向该集合类中添加的对象必须是 String 类型。new Integer("1")对象不是 String 类对象，所以编译器给出了错误信息。

实际上，例 1-2 中 ArrayList < String > ，即为泛型类。从 JDK5. 0 版本开始，集合框架中的接口和类均定义成泛型。使用这些类和接口的方法是：在类名后紧跟尖括号 <> ，括号内指定允许添加的元素类型。例如，要创建 HashSet，并准备向其中添加 String 类的对象，则方法如下：

```
HashSet < String > hs = new HashSet < String > ( );
```

通过在 HashSet 类名后加尖括号 <> ，并在 <> 中指定 String，限制了该集合中只能添加 String 类的对象作为集合元素。使用 JDK5. 0 以上的版本时，要创建集合类的对象，最好采用如上的方法。将例 1-2 程序中发生编译错误的第 8 行 lst. add （new Integer("1")）；删除，程序将正常编译运行。

细心的读者应该已经注意到，在使用迭代器的 next( )方法访问集合中数据时，无需再进行强制类型转换。

综上所述，使用泛型类具有如下优点：

- 保证了集合中所有元素的数据类型相同；
- 访问集合类中的元素时，无需进行强制类型转换；
- 向集合中添加不符合指定类型的数据，编译时会发生错误信息，避免了运行时错误的发生。

在这里需要说明的是，Java 编译器向后兼容。对于已经在 JDK 5.0 以下版本的应用程序，高版本的 Java 编译器也提供支持。但是，在编译时，会出现一个警告："使用了未经检查或不安全的操作"。例如，例 1-1 中的程序在 JDK6.0 环境下编译时，会给出如下警告：

```
注意:① E:\java 高级教程\ch01\ AccessCollection.java 使用了未经检查或不安全的操作。
      ② 要了解详细信息,请使用 -Xlint:unchecked 重新编译。
```

要想避免该警告，可以在类的前面使用@ SuppressWarnings({"unchecked"})的注解。

```
import java.util.*;
@SuppressWarnings({"unchecked"})
public class Ch1_1{
    ...
}
```

1.3 泛型进阶：泛型类、泛型方法和泛型接口

1.3.1 泛型类

在前面的讲述中，读者已经能够使用 Java 系统中提供的泛型类。实际上，Java 也允许用户定义自己的泛型类。下面通过例 1-3 说明如何自己定义泛型类。

【例 1-3】 定义泛型类。

```
/**
 * 文件名:Container1.java
 * 定义泛型类 Container1
 */
class Person{
    private String name;
    public Person(String name){
        this.name = name;
    }
}
//定义泛类 Container1 <T>
```

```
public class Container1 <T> {
    private T elem;
    public Container1(T elem) {
        this.elem = elem;
    }
    public T get() {
        return this.elem;
    }
    public void set(T elem) {
        this.elem = elem;
    }
    public static void main(String[] args) {
        //创建泛类的对象,泛类中的类型参数用 Student 类代替
        Container1 <Person> c1 = new Container1 <Person> (new Person("Ann"));
        Person s1 = c1.get(); //不需要类型转换
        c1.set("Bill");//将发生编译错误
    }
}
```

程序中定义了两个类：Person 类和 Container 类，其中后者为泛型类。可以看出，与普通的 Java 类定义不同的只是：泛型类的表示方法是在类名后面用尖括号括起一个类型参数 T。

注意：这种由类型参数修饰的类被称为泛型类。

通过这样定义类的头部，在类体内，用户既可以定义 T 类型的数据成员，也可以在成员方法（静态方法除外）中使用 T 类型的形式参数或内部变量。也就是说，类型参数的作用域是定义这个类型参数的整个类，但是不包括静态成员方法。这是因为当访问同一个静态成员方法时，同一个类的不同实例可能有不同的类型参数，所以类型参数的作用域不应该包括这些静态方法，否则就会引起混乱。

在创建泛类的对象时，通过将 T 的位置指定为特定的引用类型时保证了类中所有出现 T 的位置均用指定类型代替。

注意：

泛型本质上是提供类型的“类型参数”，也就是说，在创建一个类时，类中部分成员或方法中参数的类型可以暂时不指定成特定类型，而是用一个符号代替。所以，泛型也被称为参数化类型或参量多态。

Java 允许对类中的多个类型参数化。方法是在类名之后的尖括号内 <> 写多个类型参数，并用逗号“,”隔开。

【例 1-4】 定义两个参数的泛型类。

```
/**
 * 文件名:Container2.java
```

```
 * 定义泛类 Container2 < S,T >
 */
public class Container2 < S,T > {
    private S elem1;
    private T elem2;
    public Container2(S elem1,T elem2) {
        this. elem1 = elem1;
        this. elem2 = elem2;
    }
    public void set2(S elem1,T elem2) {
        this. elem1 = elem1;
        this. elem2 = elem2;
    }
    public static void main(String[ ] args) {
        //创建泛类的对象,泛类中的类型参数 S 用 Integer 代替,类型参数 T 用 String 代替
        Container2 < Integer,String > c1 = new Container2 < Integer,String > (1,"Ann");
    }
}
```

注意：

类型参数名可以任意取名，只要符合标识符的取名规则即可，在实际使用中，习惯用一个大写字母命名。

1.3.2 泛型方法

迄今为止，我们已经可以通过定义泛型类，使类的数据成员的类型和非静态成员方法中参数和局部变量的数据类型参数化。Java 也允许对普通的类（非泛型类）中的成员方法、静态方法（无论是否为泛型类）的数据类型参数化，参数化后的方法即为泛型方法。

注意：实际上，泛型方法在泛型类和普通类中均可以定义。

要定义泛型方法，只需要在这个方法的返回类型之前加上一个类型参数“T”，并用尖括号“ <> ”将它括起来。

【例 1-5】 定义和使用泛型方法。

```
/**
 * 文件名 GjMethod. java
 * 定义和使用泛型方法
 */
import java. util. *;
public class GjMethod {
    public < T > void displayType(T a) {
```

```
        System.out.println(a.getClass().getName());
    }
    public <T> T getFirst(List<T> lst){
        return lst.get(0);
    }
    public static void main(String[] args) {
        //创建 GjMethod 类的对象
        GjMethod gm = new GjMethod();
        List<Integer> lst1 = new ArrayList<Integer>();
        lst1.add(1);
        lst1.add(2);
        Integer i = gm.getFirst(lst1);//调用泛型方法 getFirst
        gm.displayType(i);//调用泛型方法 displayType
        gm.displayType(100);
    }
}
```

运行结果如下：

```
java.lang.Integer
java.lang.Integer
```

例 1-5 中 GjMethod 类没有被参数化，是普通的 Java 类，但是其中的方法 getFirst( )的返回类型和形式参数都使用参数化类型，displayType( )方法的形参也为类型参数 T，这两个类方法均为泛型方法。

调用泛型方法时，一般情况下，和类的其他方法一样，不需要指定类型参数，因为编译器能够根据实际参数的类型自动将类型参数替换为实际参数类型。例 1-5 中，通过将 List<Interger>类型的实参 lst1 传入泛型方法 getFirst，编译器自动识别到类型参数为 Integer；通过将 Integer 类型的实参 i 传入泛型方法 displayType，编译器自动识别到类型参数为 Integer。程序中还有一行：gm.displayType（100），传入的实参属于基本数据类型 int，程序执行后，显示 Integer 类型，这是因为 java 的自动装箱机制会对基本数据类型包装。

实际上，调用泛型方法时，方法名前也可以加入实际的类型参数，以显式说明实际参数的类型。例如：gm.displayType（100）可以写成 gm.<Integer>displayType（100）。

注意：

使用泛型类和泛型方法的不同之处在于：使用泛型类创建对象时必须指定参数类型，但是调用泛型方法时虽然可以在方法名前指定类型参数，但通常情况下可以不指定。

泛型方法适用于以下场合：

① 添加类型约束只作用于一个方法的多个参数之间、而不涉及类中的其他方法时，使用泛型方法；

② 施加类型约束的方法为静态方法时，只能将其定义为泛型方法，因为静态方法不能

使用其所在类的类型参数。

1.3.3 泛型接口

泛型同样适用于接口。定义接口时，接口名之后也可以加类型参数，这样定义的接口称为泛型接口。例如，Java 中 Comparable 接口定义如下：

```
public interface Comparable <T> {
    public int compareTo(T that);
}
```

接口 Comparable 名之后加了类型参数 <T>，此接口为泛型接口。下面以 Java 系统中定义的两个用于对象比较的接口 Comparable 和 Comparator 为例介绍如何一个类如何实现泛型接口。

1. Comparable <T> 接口

Comparable 接口强行对实现它的每个类的对象进行整体排序，这种排序被称为类的自然排序。接口中的 compareTo( )方法用于比较此对象与指定对象的顺序。如果该对象小于、等于或大于指定对象，则分别返回负整数、零或正整数。

注意：

① 实现此接口的对象列表（和数组）可以通过 Collections. sort（和 Arrays. sort）进行自动排序。

② 实现此接口的对象可以用作有序映射中的键或有序集合中的元素，无需指定比较器。

③ Java 系统中很多类实现了该接口。例如，我们学过的 String 类、Number 类的所有子类。

下面的程序定义了实现 Comparable 接口的类 Student，在该类中通过比较 age 属性值的大小重写 compareTo( )方法。

【例 1-6】 定义实现 Comparable 接口的类。

```
/**
 * 文件名:ComparableDemo. java
 * 定义实现 Comparable 接口的类
 */
import java. util. *;
class Student implements Comparable <Student> {
    String name;
    int age;
    Student( ) { }
    Student(String name,int age) {
        this. name = name;
        this. age = age; }
    void printInfo( ) {
        System. out. println (name + " \t" + age);
```