Java 应用程序设计
泛型

王晓东
wxd2870@163.com

中国海洋大学

November 9, 2017

参考书目

1. 张利国、刘伟 [编著], Java SE 应用程序设计, 北京理工大学出版社, 2007.10.

本章学习目标

1. 什么是泛型
2. 使用泛型
   - 集合框架中的泛型
   - 泛型的向后兼容性
3. 泛型进阶
   - 类型参数 / 定义自己的泛型类
   - 类型通配符
   - 泛型方法
   - 受限制的类型参数

大纲

泛型

使用泛型

泛型进阶

接下来…

泛型

使用泛型

泛型进阶

什么是泛型

泛型（Gernerics）机制自 JDK 5.0 开始引入，其实质是将原本确定不变的数据类型参数化。

泛型优势

作为对原有 Java 类型体系的扩充，使用泛型可以提高 Java 应用程序的类型安全、可维护性和可靠性。

泛型分析

❖ 集合框架中的数据造型问题

传统的集合为了提供广泛的适用性，会将所有加入其中的元素当作 Object 类型来处理，如 Vector 集合。基于此原因，在实际使用时，我们必须将从集合中取出的元素值再强制转换为所期望的类型。

无泛型机制

```
1 Vector v = new Vector();
2 v.addElement(new Person("Tom", 18));
3 Person p = (Person)v.elementAt(0);
4 p.showInfo();
```

泛型允许编译器实施由开发者设定的附加类型约束，将类型检查从运行时挪到编译时进行，这样类型错误就可以在编译时暴露出来、而不是在运行时才发作（抛出 ClassCastException 运行异常）。

接下来…

泛型

使用泛型

泛型进阶

集合框架中的泛型

❖ 用法

创建集合容器时规定其允许保存的元素类型，然后由编译器负责添加元素的类型合法性检查，在取用集合元素时则不必再进行造型处理。

使用泛型的列表集合

```
1  import java.util.Vector;
2  import java.util.Data;

4  public class TestVector {
5    public static void main(String[] args) {
6      Vector<String> v = new Vector<String>();
7      v.addElement("Tom");
8      v.addElement("Nancy");
9      v.addElement("new␣Data()"); //编译时出错
10     v.addElement("new␣Integer(300)"); //编译时出错

12     for(int i = 0; i < v.size(); i++) {
13       String name = v.elementAt(i); //并不需要进行强制类型转换
14       System.out.println(name);
15     }
16   }
17 }
```

集合框架中的泛型

在 Hashtable 中使用泛型

CODE ▸ Employee.java

```
1  public class Employee {
2    private final int id;
3    private String name;
4    private double salary;
5
6    public Employee(int id, String name, double salary) {
7      this.id = id;
8      this.name = name;
9      this.salary = salary;
10   }
11   // ...
12   public void showInfo() {
13     System.out.println(id + "\t" + name + "\t" + salary);
14   }
15 }
```

集合框架中的泛型

CODE ◗ TestHashtable.java

```
1  import java.util.Hashtable;
2
3  public class Testhashtable {
4    public static void main(String[] args) {
5      Hashtable<Integer, Employee> ht = new Hashtable<Integer, Employee>();
6      ht.put(101, new Employee(101, "张三", 5000));
7      ht.put(102, new Employee(102, "李四", 4800));
8
9      Employee e = ht.get(102);
10     e.showInfo();
11   }
12 }
```

程序中直接将 int 型数据当作映射“键”使用，是由于 Java 的自动封装机制——系统已自动将 int 型数据值转换为 Integer 类型对象。

泛型的向后兼容性

Java 语言中的泛型是维护向后兼容的，我们完全可以不采用泛型、而继续沿用过去的做法。这些未加改造的旧式代码将无法享用泛型带来的便利和安全性。

泛型的向后兼容性

Java 语言中的泛型向过去兼容在高版本开发环境中编译未启用泛型机制的集合框加应用代码时，会输出类似于如下形式的编译提示信息：

output

```
注意：/home/xiaodong/JavaTest/TestArrayList.java
使用了未经检查或不安全的操作。
注意：要了解详细信息，请使用 -Xlint:unchecked 重新编译。
```

可以使用 SuppressWarnings 注解关闭编译提示或警告信息：

```
1 @SuppressWarnings({"unchecked"})
2 public class TestSuppressWarnings {
3   //
4 }
```

接下来…

泛型

使用泛型

泛型进阶

类型参数

如 Vector<String>，其中，尖括号括起来的部分被称为**类型参数**，而这种由类型参数修饰的类型则被称为**泛型类**，我们可以将泛型类理解为一种新的构造类型，当然也一定属于引用类型。
注意：应在声明泛型类变量和创建对象时均给出类型参数，且两者应保持一致。

类型参数

使用类型参数 E 进行泛型化处理的 java.util.Vector 类的定义代码摘要如下：

```
1 public class Vector<E> --- {
2   public void addElement(E obj) { --- }
3   public E elementAt(int index) { --- }
4 }
```

这里的 E 也称为“形式类型”参数。在实际使用该泛型类时，我们需要指定相应的具体类型，即实际类型参数。

```
1 Vector<String> v = new Vector<String>();
```

编译器遇到 Vector<String> 类型变量时，即知道此 Vector 变量/对象的类型参数 E 已经被绑定为 String 类型，进而也就确定其 addElement() 方法的参数和 elementAt() 方法的返回值均为 String 类型。

类型参数

使用自己的泛型类

CODE ▸ Person.java

```
1  public class Person<T> {
2    private final int id;
3    private T secrecy;
4    public Person(int id) {
5      this.id = id;
6    }
7    public getId() {
8      return id;
9    }
10   public void setSecrecy(T secrecy) {
11     this.secrecy = secrecy;
12   }
13   public T getSecrecy() {
14     return secrecy;
15   }
16 }
```

类型参数

CODE ⏵ Test.java

```
1  public class Test {
2    public static void main(String[] args) {
3      Person<String> p1 = new Person<String>(101);
4      p1.setSecrecy("芝麻开门");
5      String s = p1.getSecrecy();
6      System.out.println(p1.getId() + "\t␣密码是: " + s);
7
8      Person<Double> p2 = new Person<Double>(102);
9      p2.setSecrecy("8700.85");
10     double money = p2.getSecrecy();
11     System.out.println(p2.getId() + "\t␣秘密资金数额是: " + money);
12 }
```

上述代码示例中的泛型类 Person 可以在使用时通过类型参数 T 指定其属性 secrecy 的具体类型（以及该属性相应存/取方法的参数和返回值类型），进而提供了通用的信息存储能力。

类型参数

形式类型参数的编程惯例。

❖ Java 编程惯例

符号	意义
K	键，比如映射的键
V	值，比如 List 和 Set 的内容，或者 Map 中的值
E	元素，比如 Vector<E>
T	泛型

类型通配符

❖ 对泛型的理解

- 泛型类可以理解为具有广泛适用性、尚未最终定型的类型。
- Person<String> 和 Person<Double> 属于同一个类，但确是不同的类型。
- 同一个泛型类与不同的类型参数复合而成的类型间并不存在继承关系，即使是类型参数间存在继承关系时也是如此。如：Vector<String> 不是 Vector<Object> 的子类。

泛型类型的处理

遍历泛型 Vector 集合

```
1 public void overview(Vector<String> v) {
2   for(String o: v) {
3     String.out.println(o);
4   }
5 }
```

上述方法用于遍历 Vector<String> 类型集合。但，我们可能还需要定义多个类似的方法来遍历其他类型的 Vector 集合元素。例如，定义 overview(Vector<Person> v)、overview(Vector<Integer> v) 等，显然过于繁琐，引用泛型机制后代码的通用性似乎不如从前。

泛型类型的处理

一种可能的处理方法：将遍历方法的形参定义为不带任何类型参数的原型类型 Vector，但这样会破坏已有的类型安全性。

```
1 public void overview(Vector v) {
2   for(Object o: v) {
3     String.out.println(o);
4   }
5 }
```

上述代码能够编译和运行，但编译时会出现“unchecked”提示信息。

泛型类型的处理

为了解决类似泛型遍历的问题，Java 泛型机制中引入了通配符“?”。

使用类型通配符

```
1  import java.utils.Vector;
2
3  public class Test {
4    public static void main(String[] args) {
5      Test t = new Test();
6      Vector<String> vs = new Vector<String>();
7      vs.add("Tom");
8      vs.add("Kessey");
9      Vector<Integer> vi = new Vector<Integer>();
10     vi.add(300);
11     vi.add(500);
12     t.overview(vi);
13   }
14   public void overview(Vector<?> v) {
15     for(Object o : v) {
16       System.out.println(o);
17     }
18   }
19 }
```

类型通配符

❖ 使用类型通配符的好处

1. Vector<?> 是任何泛型 Vector 的父类型，因此可以将 Vector<String>、Vector<Integer>、Vector<Object> 等作为实参传给 overview(Vector<?> v) 方法处理；
2. Vector<?> 类型的变量在调用方法时是受到限制的——凡是必须知道具体类型参数才能进行的操作均被禁止。

```
1 Vector<String> vs = new Vector<String>();
2 vs.add("Tom");
3 Vector<?> v = vs;
4 v.add("Billy"); //非法，编译器不知道具体类型参数
5 Object e = v.elementAt(0); //合法，允许检索元素，此时读取的元素均当作 Object 类型处理
6 System.out.println(e);
```

类型通配符

上述限制不等同于将 Vector<?> 变为“只读”，在不需要编译器确定类型参数的情况下也是可以修改集合内容的，例如：

```
1 Vector<String> vs = new Vector<String>();
2 vs.add("Tom");
3 vs.add("Billy");
4 Vector<?> v = vs;
5 v.remove(new Integer(200)); //形参为 Object 类型，运行不受影响
6 v.clear(); //不需要参数，运行不受影响
```

泛型方法

与泛型类的情况类似，方法也可以被泛型化，且无论其所属的类是否为泛型类。

使用泛型方法

```
1  import java.utils.Vector;
2
3  public class Test {
4    public static void main(String[] args) {
5      Test t = new Test();
6      String valid = t.evaluate("tiger", "tiger")
7      Integer i = t.evaluate(new Integer(300), new Integer(350));
8      System.out.println(valid);
9      System.out.println(i);
10   }
11
12   public <T>T evaluate(T a, T b) {
13     if(a.equals(b))
14       return a;
15     else
16       return null;
17   }
18 }
```

泛型方法

上述代码中，方法 evaluate() 声明中的“<T>”用于标明这是一个**泛型方法**——类型 T 是可变的，不必显示的告知编译器 T 具体取何值，但出现了多处（两个形参、一个返回值类型）的这些值必须都相同。

泛型方法

方法 evaluate() 中的类型参数 T 也可以添加到其所在的类定义中，此时类就变成了泛型类。

```
1  import java.utils.Vector;
2  public class Test<T> {
3    public static void main(String[] args) {
4      // ----
5    }

7    public T evaluate(T a, T b) {
8      // ----
9    }
10 }
```

❖ 使用泛型方法，而不是定义泛型类的原则

1. 不涉及到类中的其他方法时，则应将之定义为泛型方法，因为泛型方法的类型参数是局部性的，这样可以简化其所在类型的声明和处理开销；
2. 要施加类型约束的方法为静态方法时，只能将之定义为泛型方法，因为静态方法不能使用其所在类的类型参数。

受限制的类型参数

泛型机制允许开发者对类型参数进行附加约束。

使用受限制的类型参数

```
1  import java.utils.Number;
2
3  public class Point<T extends Number> {
4    private T x;
5    private T y;
6    public Point() {}
7    public Point(T x, T y) {
8      this.x = x;
9      this.y = y;
10   }
11   public T getX() {
12     return x;
13   }
14   public T getY() {
15     return y;
16   }
17   public void setX(T x) {
18     this.x = x;
19   }
20   public void setY(T y) {
21     this.y = y;
22   }
23   public void showInfo() {
24     System.out.println("x=" + x + ",␣y=" + y);
25   }
26 }
```

受限制的类型参数

```
1  public class Test {
2    Point<Integer> pi = new Point<Integer>(20, 40);
3    pi.setX(pi.getX() + 100);
4    pi.showInfo();
5    Point<Double> pd = new Point<Double>();
6    pd.setX(3.45);
7    pd.setY(6.78);
8    pd.showInfo();
9    Point<String> ps = new Point<String>(); //非法
10 }
```

上述程序中，Point 类用于描述平面直角坐标系中点的坐标，其封装的应为数值型信息（如 int、float、double），考虑到类型参数不能为基本数据类型，而 java.lang.Number 是所有数值型封装类（如 Integer、Float、Double 等）的父类型，于是决定限制泛型类 Point 的类型参数必须为 Number 或其子类类型，并使用 extends 关键字来标明这种继承层次上限制。

THE END

wxd2870@163.com