



Java泛型详解，通俗易懂只需5分钟

<上一节

下一节>

分享到：

QQ空间

新浪微博

腾讯微博

豆瓣

人人网

Google Pixel 3

Designed to be the
smartest, most hel
device in your life. f
now.

教程目录

- 1 Java概述
- 2 Java语法基础
- 3 Java类与对象
- 4 Java继承和多态
- 5 面向对象高级特性
- 5.1 Java内部类及其实例化
- 5.2 内部类的分类
- 5.3 抽象类的概念和使用
- 5.4 接口的概念和使用
- 5.5 接口和抽象类的区别
- 5.6 Java 泛型
- 5.7 泛型通配符和类型参数的范围
- 6 异常处理
- 7 多线程编程
- 8 输入输出(IO)操作
- 9 常用类库、向量与哈希
- 10 图形界面(GUI)设计
- 11 图形、图像与多媒体
- 12 网络与数据库编程

在线一对一师徒制，让学生在互联网上拜一位具有丰富开发与教学经验的师父，由师父根据徒弟的基础，学习能力，学习时间，学习习惯，为徒弟量身打造专属学习计划，一对一监督学习。[猛击这里查看学习体验。](#)

我们知道，使用变量之前要定义，定义一个变量时必须指明它的数据类型，什么样的数据类型赋给什么样的值。

假如我们现在要定义一个类来表示坐标，要求坐标的数据类型可以是整数、小数和字符串，例如：

- x = 10、y = 10
- x = 12.88、y = 129.65
- x = "东京180度"、y = "北纬210度"

针对不同的数据类型，除了借助方法重载，还可以借助自动装箱和向上转型。我们知道，基本数据类型可以自动装箱，被转换成对应的包装类；Object 是所有类的祖先类，任何一个类的实例都可以向上转型为 Object 类型，例如：

- int --> Integer --> Object
- double --> Double --> Object
- String --> Object

这样，只需要定义一个方法，就可以接收所有类型的数据。请看下面的代码：

```
01. public class Demo {
02.     public static void main(String[] args){
03.         Point p = new Point();
04.
05.         p.setX(10); // int -> Integer -> Object
06.         p.setY(20);
07.         int x = (Integer)p.getX(); // 必须向下转型
08.         int y = (Integer)p.getY();
09.         System.out.println("This point is: " + x + ", " + y);
10.
11.         p.setX(25.4); // double -> Integer -> Object
12.         p.setY("东京180度");
13.         double m = (Double)p.getX(); // 必须向下转型
14.         double n = (Double)p.getY(); // 运行期间抛出异常
15.         System.out.println("This point is: " + m + ", " + n);
16.     }
17. }
18.
19. class Point{
20.     Object x = 0;
21.     Object y = 0;
22.
23.     public Object getX() {
24.         return x;
25.     }
26.     public void setX(Object x) {
```

✖

纯文本

新窗口

```
27.         this.x = x;
28.     }
29.     public Object getY() {
30.         return y;
31.     }
32.     public void setY(Object y) {
33.         this.y = y;
34.     }
35. }
```

上面的代码中，生成坐标时不会有任何问题，但是取出坐标时，要向下转型，在 [Java多态对象的类型转换](#) 一文中我们讲到，向下转型存在着风险，而且编译期间不容易发现，只有在运行期间才会抛出异常，所以要尽量避免使用向下转型。运行上面的代码，第12行会抛出 `java.lang.ClassCastException` 异常。

那么，有没有更好的办法，既可以不使用重载（有重复代码），又能把风险降到最低呢？

有，可以使用**泛型类(Java Class)**，它可以接受任意类型的数据。所谓“泛型”，就是“宽泛的数据类型”，任意的数据类型。

更改上面的代码，使用泛型类：

```
01. public class Demo {
02.     public static void main(String[] args){
03.         // 实例化泛型类
04.         Point<Integer, Integer> p1 = new Point<Integer, Integer>();
05.         p1.setX(10);
06.         p1.setY(20);
07.         int x = p1.getX();
08.         int y = p1.getY();
09.         System.out.println("This point is: " + x + ", " + y);
10.
11.         Point<Double, String> p2 = new Point<Double, String>();
12.         p2.setX(25.4);
13.         p2.setY("东京180度");
14.         double m = p2.getX();
15.         String n = p2.getY();
16.         System.out.println("This point is: " + m + ", " + n);
17.     }
18. }
19.
20. // 定义泛型类
21. class Point<T1, T2>{
22.     T1 x;
23.     T2 y;
24.     public T1 getX() {
25.         return x;
26.     }
27.     public void setX(T1 x) {
28.         this.x = x;
29.     }
30.     public T2 getY() {
31.         return y;
32.     }
33.     public void setY(T2 y) {
34.         this.y = y;
35.     }
36. }
```

运行结果：

This point is: 10, 20

This point is: 25.4, 东京180度

与普通类的定义相比，上面的代码在类名后面多出了 `<T1, T2>`，`T1, T2` 是自定义的标识符，也是参数，用来传递数据的类型，而不是数据的值，我们称之为**类型参数**。在泛型中，不但数据的值可以通过参数传递，数据的类型也可以通过

参数传递。T1, T2 只是数据类型的占位符, 运行时会被替换为真正的数据类型。

传值参数 (我们通常所说的参数) 由小括号包围, 如 (int x, double y), 类型参数 (泛型参数) 由尖括号包围, 多个参数由逗号分隔, 如 <T> 或 <T, E>。

类型参数需要在类名后面给出。一旦给出了类型参数, 就可以在类中使用了。类型参数必须是一个合法的标识符, 习惯上使用单个大写字母, 通常情况下, K 表示键, V 表示值, E 表示异常或错误, T 表示一般意义上的数据类型。

泛型类在实例化时必须指出具体的类型, 也就是向类型参数传值, 格式为:

```
className variable<dataType1, dataType2> = new className<dataType1, dataType2>();
```

也可以省略等号右边的数据类型, 但是会产生警告, 即:

```
className variable<dataType1, dataType2> = new className();
```

因为在使用泛型类时指明了数据类型, 赋给其他类型的值会抛出异常, 既不需要向下转型, 也没有潜在的风险, 比本文一开始介绍的自动装箱和向上转型要更加实用。

注意:

- 泛型是 Java 1.5 的新增特性, 它以C++模板为参照, 本质是参数化类型(Parameterized Type)的应用。
- 类型参数只能用来表示引用类型, 不能用来表示基本类型, 如 int、double、char 等。但是传递基本类型不会报错, 因为它们会自动装箱成对应的包装类。

泛型方法

除了定义泛型类, 还可以定义泛型方法, 例如, 定义一个打印坐标的泛型方法:

```
01. public class Demo {
02.     public static void main(String[] args){
03.         // 实例化泛型类
04.         Point<Integer, Integer> p1 = new Point<Integer, Integer>();
05.         p1.setX(10);
06.         p1.setY(20);
07.         p1.printPoint(p1.getX(), p1.getY());
08.
09.         Point<Double, String> p2 = new Point<Double, String>();
10.         p2.setX(25.4);
11.         p2.setY("东京180度");
12.         p2.printPoint(p2.getX(), p2.getY());
13.     }
14. }
15.
16. // 定义泛型类
17. class Point<T1, T2>{
18.     T1 x;
19.     T2 y;
20.     public T1 getX() {
21.         return x;
22.     }
23.     public void setX(T1 x) {
24.         this.x = x;
25.     }
26.     public T2 getY() {
27.         return y;
28.     }
29.     public void setY(T2 y) {
30.         this.y = y;
31.     }
32.
33.     // 定义泛型方法
34.     public <T1, T2> void printPoint(T1 x, T2 y){
35.         T1 m = x;
36.         T2 n = y;
37.         System.out.println("This point is: " + m + ", " + n);
38.     }
39. }
```

运行结果：

This point is: 10, 20

This point is: 25.4, 东京180度

上面的代码中定义了一个泛型方法 `printPoint()`，既有普通参数，也有类型参数，类型参数需要放在修饰符后面、返回值类型前面。一旦定义了类型参数，就可以在参数列表、方法体和返回值类型中使用了。

与使用泛型类不同，使用泛型方法时不必指明参数类型，编译器会根据传递的参数自动查找出具体的类型。泛型方法除了定义不同，调用就像普通方法一样。

注意：泛型方法与泛型类没有必然的联系，泛型方法有自己的类型参数，在普通类中也可以定义泛型方法。泛型方法 `printPoint()` 中的类型参数 `T1, T2` 与泛型类 `Point` 中的 `T1, T2` 没有必然的联系，也可以使用其他的标识符代替：

```
01. public static <V1, V2> void printPoint(V1 x, V2 y){
02.     V1 m = x;
03.     V2 n = y;
04.     System.out.println("This point is: " + m + ", " + n);
05. }
```

泛型接口

在Java中也可以定义泛型接口，这里不再赘述，仅仅给出示例代码：

```
01. public class Demo {
02.     public static void main(String args[]) {
03.         Info<String> obj = new InfoImp<String>("www.weixueyuan.net");
04.         System.out.println("Length Of String: " + obj.getVar().length());
05.     }
06. }
07.
08. //定义泛型接口
09. interface Info<T> {
10.     public T getVar();
11. }
12.
13. //实现接口
14. class InfoImp<T> implements Info<T> {
15.     private T var;
16.
17.     // 定义泛型构造方法
18.     public InfoImp(T var) {
19.         this.setVar(var);
20.     }
21.
22.     public void setVar(T var) {
23.         this.var = var;
24.     }
25.
26.     public T getVar() {
27.         return this.var;
28.     }
29. }
```

运行结果：

Length Of String: 18

类型擦除

如果在使用泛型时没有指明数据类型，那么就会擦除泛型类型，请看下面的代码：

```
01. public class Demo {
02.     public static void main(String[] args){
03.         Point p = new Point(); // 类型擦除
04.         p.setX(10);
05.         p.setY(20.8);
06.         int x = (Integer)p.getX(); // 向下转型
```

```
07.         double y = (Double)p.getY();
08.         System.out.println("This point is: " + x + ", " + y);
09.     }
10. }
11.
12. class Point<T1, T2>{
13.     T1 x;
14.     T2 y;
15.     public T1 getX() {
16.         return x;
17.     }
18.     public void setX(T1 x) {
19.         this.x = x;
20.     }
21.     public T2 getY() {
22.         return y;
23.     }
24.     public void setY(T2 y) {
25.         this.y = y;
26.     }
27. }
```

运行结果:

This point is: 10, 20.8

因为在使用泛型时没有指明数据类型, 为了不出现错误, 编译器会将所有数据向上转型为 Object, 所以在取出坐标使用时要向下转型, 这与本文一开始不使用泛型没什么两样。

限制泛型的可用类型

在上面的代码中, 类型参数可以接受任意的数据类型, 只要它是被定义过的。但是, 很多时候我们只需要一部分数据类型就够了, 用户传递其他数据类型可能会引起错误。例如, 编写一个泛型函数用于返回不同类型数组 (Integer 数组、Double 数组、Character 数组等) 中的最大值:

```
01. public <T> T getMax(T array[]){
02.     T max = null;
03.     for(T element : array){
04.         max = element.doubleValue() > max.doubleValue() ? element : max;
05.     }
06.     return max;
07. }
```

上面的代码会报错, doubleValue() 是 Number 类的方法, 不是所有的类都有该方法, 所以我们要限制类型参数 T, 让它只能接受 Number 及其子类 (Integer、Double、Character 等)。

通过 extends 关键字可以限制泛型的类型, 改进上面的代码:

```
01. public <T extends Number> T getMax(T array[]){
02.     T max = null;
03.     for(T element : array){
04.         max = element.doubleValue() > max.doubleValue() ? element : max;
05.     }
06.     return max;
07. }
```

<T extends Number> 表示 T 只接受 Number 及其子类, 传入其他类型的数据会报错。这里的限定使用关键字 extends, 后面可以是类也可以是接口。但这里的 extends 已经不是继承的含义了, 应该理解为 T 是继承自 Number 类的类型, 或者 T 是实现了 XX 接口的类型。

注意: 一般的应用开发中泛型使用较少, 多用在框架或者库的设计中, 这里不再深入讲解, 主要让大家对泛型有所认识, 为后面的教程做铺垫。



别再乱花冤枉钱！Java从菜鸟到架构师自学系列课程下载 点我免费领取...

<上一节 下一节> 分享到： QQ空间 新浪微博 腾讯微博 豆瓣 人人网

关于我们 | 联系我们 | 业务合作 | 发布你自己的教程

精美而实用的网站，关注编程技术，追求极致，让您轻松愉快的学习。
Copyright ©2011-2015 www.weixueyuan.net, All Rights Reserved, 粤ICP备15014638号

www.weixueyuan.net