Java 集合框架 Java 序列化

Java 泛型

Java 泛型（generics）是 JDK 5 中引入的一个新特性, 泛型提供了编译时类型安全检测机制，该机制允许程序员在编译时检测到非法的类型。
泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。

> *假定我们有这样一个需求：写一个排序方法，能够对整型数组、字符串数组甚至其他任何类型的数组进行排序，该如何实现？*
>
> *答案是可以使用 **Java 泛型**。*
>
> *使用 Java 泛型的概念，我们可以写一个泛型方法来对一个对象数组排序。然后，调用该泛型方法来对整型数组、浮点数数组、字符串数组等进行排序。*

泛型方法

你可以写一个泛型方法，该方法在调用时可以接收不同类型的参数。根据传递给泛型方法的参数类型，编译器适当地处理每一个方法调用。

下面是定义泛型方法的规则：

- 所有泛型方法声明都有一个类型参数声明部分（由尖括号分隔），该类型参数声明部分在方法返回类型之前（在下面例子中的<E>）。
- 每一个类型参数声明部分包含一个或多个类型参数，参数间用逗号隔开。一个泛型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。
- 类型参数能被用来声明返回值类型，并且能作为泛型方法得到的实际参数类型的占位符。
- 泛型方法体的声明和其他方法一样。注意类型参数只能代表引用型类型，不能是原始类型（像int,double,char的等）。

实例

下面的例子演示了如何使用泛型方法打印不同字符串的元素：

实例

```
public class GenericMethodTest
{
// 泛型方法 printArray
public static < E > void printArray( E[] inputArray )
{
// 输出数组元素
for ( E element : inputArray ){
System.out.printf( "%s ", element );
}
System.out.println();
}
public static void main( String args[] )
{
// 创建不同类型数组： Integer, Double 和 Character
```

```
Integer[] intArray = { 1, 2, 3, 4, 5 };
Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
System.out.println( "整型数组元素为:" );
printArray( intArray ); // 传递一个整型数组
System.out.println( "\n双精度型数组元素为:" );
printArray( doubleArray ); // 传递一个双精度型数组
System.out.println( "\n字符型数组元素为:" );
printArray( charArray ); // 传递一个字符型数组
}
}
```

编译以上代码，运行结果如下所示：

```
整型数组元素为:
1 2 3 4 5

双精度型数组元素为:
1.1 2.2 3.3 4.4

字符型数组元素为:
H E L L O
```

有界的类型参数:

可能有时候，你会想限制那些被允许传递到一个类型参数的类型种类范围。例如，一个操作数字的方法可能只希望接受Number或者Number子类的实例。这就是有界类型参数的目的。

要声明一个有界的类型参数，首先列出类型参数的名称，后跟extends关键字，最后紧跟它的上界。

实例

下面的例子演示了"extends"如何使用在一般意义上的意思"extends"（类）或者"implements"（接口）。该例子中的泛型方法返回三个可比较对象的最大值。

实例

```
public class MaximumTest
{
// 比较三个值并返回最大值
public static <T extends Comparable<T>> T maximum(T x, T y, T z)
{
T max = x; // 假设x是初始最大值
if ( y.compareTo( max ) > 0 ){
max = y; //y 更大
}
if ( z.compareTo( max ) > 0 ){
max = z; // 现在 z 更大
}
return max; // 返回最大对象
}
public static void main( String args[] )
```

```
{
System.out.printf( "%d, %d 和 %d 中最大的数为 %d\n\n",
3, 4, 5, maximum( 3, 4, 5 ) );
System.out.printf( "%.1f, %.1f 和 %.1f 中最大的数为 %.1f\n\n",
6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );
System.out.printf( "%s, %s 和 %s 中最大的数为 %s\n","pear",
"apple", "orange", maximum( "pear", "apple", "orange" ) );
}
}
```

编译以上代码，运行结果如下所示：

```
3, 4 和 5 中最大的数为 5

6.6, 8.8 和 7.7 中最大的数为 8.8

pear, apple 和 orange 中最大的数为 pear
```

泛型类

泛型类的声明和非泛型类的声明类似，除了在类名后面添加了类型参数声明部分。

和泛型方法一样，泛型类的类型参数声明部分也包含一个或多个类型参数，参数间用逗号隔开。一个泛型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。因为他们接受一个或多个参数，这些类被称为参数化的类或参数化的类型。

实例

如下实例演示了我们如何定义一个泛型类:

实例

```
public class Box<T> {
private T t;
public void add(T t) {
this.t = t;
}
public T get() {
return t;
}
public static void main(String[] args) {
Box<Integer> integerBox = new Box<Integer>();
Box<String> stringBox = new Box<String>();
integerBox.add(new Integer(10));
stringBox.add(new String("菜鸟教程"));
System.out.printf("整型值为 :%d\n\n", integerBox.get());
System.out.printf("字符串为 :%s\n", stringBox.get());
}
}
```

编译以上代码，运行结果如下所示：

```
整型值为 :10

字符串为 :菜鸟教程
```

类型通配符

1、类型通配符一般是使用?代替具体的类型参数。例如 **List<?>** 在逻辑上是**List<String>,List<Integer>** 等所有List<具体类型实参>的父类。

实例

```
import java.util.*;
public class GenericTest {
public static void main(String[] args) {
List<String> name = new ArrayList<String>();
List<Integer> age = new ArrayList<Integer>();
List<Number> number = new ArrayList<Number>();
name.add("icon");
age.add(18);
number.add(314);
getData(name);
getData(age);
getData(number);
}
public static void getData(List<?> data) {
System.out.println("data :" + data.get(0));
}
}
```

输出结果为：

```
data :icon
data :18
data :314
```

解析： 因为getData()方法的参数是List类型的，所以name，age，number都可以作为这个方法的实参，这就是通配符的作用

2、类型通配符上限通过形如List来定义，如此定义就是通配符泛型值接受Number及其下层子类类型。

实例

```
import java.util.*;
public class GenericTest {
public static void main(String[] args) {
List<String> name = new ArrayList<String>();
List<Integer> age = new ArrayList<Integer>();
List<Number> number = new ArrayList<Number>();
name.add("icon");
```

```
age.add(18);
number.add(314);
//getUperNumber(name);//1
getUperNumber(age);//2
getUperNumber(number);//3
}
public static void getData(List<?> data) {
System.out.println("data :" + data.get(0));
}
public static void getUperNumber(List<? extends Number> data) {
System.out.println("data :" + data.get(0));
}
}
```

输出结果：

```
data :18
data :314
```

解析： 在(//1)处会出现错误，因为getUperNumber()方法中的参数已经限定了参数泛型上限为Number，所以泛型为String是不在这个范围之内，所以会报错

3、类型通配符下限通过形如 **List<? super Number>**来定义，表示类型只能接受Number及其三层父类类型，如Objec类型的实例。

← Java 集合框架 Java 序列化 →

3 篇笔记 写笔记