链滴

019 泛型

作者：pzs233
原文链接：https://ld246.com/article/1537287819743
来源网站：链滴

<blockquote>
本文为《Java 语言程序设计》第十版 章节笔记
</blockquote>
<h2 id="19-1-引言">19.1 引言</h2>
主要优点：泛型可以使我们在编译时而不是运行时检测出错误。
泛型（generic）可以参数化类型，我们可以定义带泛型类型的类或方法，随后编译器会用具体类型来替换它。
泛型类或方法允许用户指定可以和这些类或方法一起工作的对象类型。如果试图使用一个不相容对象，编译器就会检测出这个错误。
<h2 id="19-2-动机和优点">19.2 动机和优点</h2>
动机：在编译时检测出错误。
从 JDK 1.5 开始，Java 允许定义泛型类、泛型接口和泛型方法。
JDK 1.5 之前：
<pre><code class="language-java highlight-chroma">
package java.lang;

public interface Comparable {
    public int compareTo(Objec o);
}
</code></pre>
JDK 1.5 之后：
<pre><code class="language-java highlight-chroma">
package java.lang;

public interface Comparable<T> {
    public int compareTo(T o);
}
</code></pre>
这里的 < T > 表示 形式泛型类型（formal generic type），随后可以用一个 实际具体类型 actual concrete type）来替换它。替换泛型类型称为 泛型实例化（generic instantiation）。按照例，像 E 或 T 这样的单个大写字母用于表示 形式泛型类型。
<h3 id="一个例子来显示我们为什么要引入泛型-">一个例子来显示我们为什么要引入泛型：</h3>
JDK 1.5 之前：
<pre><code class="language-java highlight-chroma">
Compaarable c = new Date();
</code></pre>

.println(c.compareTo("red"));
</code></pre>
这里 c 的实际类型是 Date，而 “red” 则是 字符串类型，实际是不能比较的。也确实，这里的码可以通过编译（因为 接口 comparable 的参数为 object ，能容纳任何类型，二者都符合，没发现个比较的对象实际类型不同，没报错），但它运行时会产生错误（直到编译器开始比较两个对象时才现 实际类型 不同，才报错）。
也就是，在 JDK 1.5 之前，这样的代码，写的时候是不会提示有错误的，但是等到你把源代码编后运行时，错误才会暴露出来。此时再回头去改源代码就浪费了时间，导致效率降低了。
而 JDK 1.5 之后：
<pre><code class="language-java highlight-chroma">Comparable<Date> c = new Date();
System.ou
.println(c.compareTo("red"));
</code></pre>
c 声明为一个引用变量时，它的声明类型是 Comparable< Date >,规定了此实例接收的参必须为 Date 类型，然后调用 compareTo 方法来比较 Date 对象 和一个字符串，因为此时传递给 coareTo 方法的参数必须是 Date 类型，而字符串显然不是，此时的代码就会产生编译错误。
这样，就能在写代码时就发现参数的实际类型不一致，而立马改正，提前发现错误，将节省时间提高了代码的可靠性。
泛型类型必须是 引用类型。不能使用 int、double 或 char 这样的基本类型来替换泛型类型，但以使用其 包装类 integer 来创建泛型类型。
<h2 id="19-3-定义泛型类和接口">19.3 定义泛型类和接口</h2>
可以为类或接口定义泛型。但使用类来创建对象，或者使用类或接口来声明引用变量时，必须制具体的类型。
<h2 id="19-4-定义泛型方法">19.4 定义泛型方法</h2>
可以为静态方法定义泛型类型。

为了声明泛型方法，将泛型类型 < E > 置于方法头关键字 static 之后，eg：<code>public static <E> print (E[] list)</code>。
为了调用泛型方法，需要将实际类型放在尖括号内作为方法名的前缀，eg：<code>className.lt;Integer>print(integers);</code>。

可以将泛型指定为另一种类型的子类型。这样的泛型类型称为 受限的（bounded），<code>&lt E extends className></code>。非受限泛型类型 < E > 等同于 。
注意：
定义一个类为泛型类型，是将泛型类型放在类名之后，eg：<code>className<E></code>
定义一个方法为泛型方法，是将泛型类型放在方法返回类型之前，eg：<code><E> void max (o1, E o2)</code>。
<h2 id="19-6-原始类型和向后兼容">19.6 原始类型和向后兼容</h2>
没有指定具体类型的泛型类和泛型接口被称为 原始类型，用于和早期的 Java 版本向后兼容。</>
<h2 id="19-7-通配泛型">19.7 通配泛型</h2>

Java 编程的逻辑 (36) - 泛型 (中) 解析通配符
<h2 id="19-8-消除泛型和对泛型的限制">19.8 消除泛型和对泛型的限制</h2>
类型消除（type erasure）：编译器使用泛型类型信息来编译代码，但随后会消除它，因此泛型息在运行时是不可用的。
泛型存在于编译时。一旦编译器确认泛型类型是安全使用的，就会将它转换为原始类型。
当编译泛型类、接口和方法时，编译器用 Object 类型代替泛型类型。
如果一个泛型类型是受限的，那么编译器就会用该受限类型来替换它。
由于泛型类在运行时被消除，因此，对于如何使用泛型类型是有一些限制的：

不能使用 <code>new E()</code>，即不能使用泛型类型参数创建实例。
不能使用 <code>new E[]</code>，不能使用泛型类型创建数组。
在静态上下文中不允许类的参数是泛型类型
异常类不能是泛型类

END