Java中for的几种用法

J2SE 1.5提供了另一种形式的for循环。借助这种形式的for循环，可以用更简单地方式来遍历数组和Collection等类型的对象。本文介绍使用这种循环的具体方式，说明如何自行定义能被这样遍历的类，并解释和这一机制的一些常见问题。

在Java程序中，要“逐一处理”――或者说，“遍历”――某一个数组或Collection中的元素的时候，一般会使用一个for循环来实现（当然，用其它种类的循环也不是不可以，只是不知道是因为for这个词的长度比较短，还是因为for这个词的含义和这种操作比较配，在这种时候for循环比其它循环常用得多）。

对于遍历数组，这个循环一般是采取这样的写法：

清单1：遍历数组的传统方式

/\* 建立一个数组 \*/

int[] integers = {1， 2， 3， 4};

/\* 开始遍历 \*/

for (int j = 0; j<integers.length; j++){

int i = integers[j];

System.out.println(i);

}

而对于遍历Collection对象，这个循环则通常是采用这样的形式：

清单2：遍历Collection对象的传统方式

/\* 建立一个Collection \*/

String[] strings = {"A"， "B"， "C"， "D"};

Collection stringList = java.util.Arrays.asList(strings);

/\* 开始遍历 \*/

for (Iterator itr = stringList.iterator(); itr.hasNext();) {

Object str = itr.next();

System.out.println(str);

}

而在Java语言的最新版本――J2SE 1.5中，引入了另一种形式的for循环。借助这种形式的for循环，现在可以用一种更简单地方式来进行遍历的工作。

1. 第二种for循环

不严格的说，Java的第二种for循环基本是这样的格式：

for (循环变量类型 循环变量名称 : 要被遍历的对象) 循环体

借助这种语法，遍历一个数组的操作就可以采取这样的写法：

清单3：遍历数组的简单方式

/\* 建立一个数组 \*/

int[] integers = {1， 2， 3， 4};

/\* 开始遍历 \*/

for (int i : integers) {

System.out.println(i);/\* 依次输出“1”、“2”、“3”、“4” \*/

}

这里所用的for循环，会在编译期间被看成是这样的形式：

清单4：遍历数组的简单方式的等价代码

/\* 建立一个数组 \*/

int[] integers = {1， 2， 3， 4};

/\* 开始遍历 \*/

for (int 变量名甲 = 0; 变量名甲<integers.length; 变量名甲++)="" {

System.out.println(integers[变量名甲]);/\* 依次输出“1”、“2”、“3”、“4” \*/

}

这里的“变量名甲”是一个由编译器自动生成的不会造成混乱的名字。

而遍历一个Collection的操作也就可以采用这样的写法：

清单5：遍历Collection的简单方式

/\* 建立一个Collection \*/

String[] strings = {"A"， "B"， "C"， "D"};

Collection list = java.util.Arrays.asList(strings);

/\* 开始遍历 \*/

for (Object str : list) {

System.out.println(str);/\* 依次输出“A”、“B”、“C”、“D” \*/

}

这里所用的for循环，则会在编译期间被看成是这样的形式：

清单6：遍历Collection的简单方式的等价代码

/\* 建立一个Collection \*/

String[] strings = {"A"， "B"， "C"， "D"};

Collection stringList = java.util.Arrays.asList(strings);

/\* 开始遍历 \*/

for (Iterator 变量名乙 = list.iterator(); 变量名乙.hasNext();) {

Object str = 变量名乙.next();

System.out.println(str);/\* 依次输出“A”、“B”、“C”、“D” \*/

}

这里的“变量名乙”也是一个由编译器自动生成的不会造成混乱的名字。

因为在编译期间，J2SE 1.5的编译器会把这种形式的for循环，看成是对应的传统形式，所以不必担心出现性能方面的问题。

不用“foreach”和“in”的原因

Java采用“for”（而不是意义更明确的“foreach”）来引导这种一般被叫做“for-each循环”的循环，并使用“:”（而不是意义更明确的“in”）来分割循环变量名称和要被遍历的对象。这样作的主要原因，是为了避免因为引入新的关键字，造成兼容性方面的问题――在Java语言中，不允许把关键字当作变量名来使用，虽然使用“foreach”这名字的情况并不是非常多，但是“in”却是一个经常用来表示输入流的名字（例如java.lang.System类里，就有一个名字叫做“in”的static属性，表示“标准输入流”）。

的确可以通过巧妙的设计语法，让关键字只在特定的上下文中有特殊的含义，来允许它们也作为普通的标识符来使用。不过这种会使语法变复杂的策略，并没有得到广泛的采用。

“for-each循环”的悠久历史

“for-each循环”并不是一个最近才出现的控制结构。在1979正式发布的Bourne shell（第一个成熟的UNIX命令解释器）里就已经包含了这种控制结构（循环用“for”和“in”来引导，循环体则用“do”和“done”来标识）。

2. 防止在循环体里修改循环变量

在默认情况下，编译器是允许在第二种for循环的循环体里，对循环变量重新赋值的。不过，因为这种做法对循环体外面的情况丝毫没有影响，又容易造成理解代码时的困难，所以一般并不推荐使用。

Java提供了一种机制，可以在编译期间就把这样的操作封杀。具体的方法，是在循环变量类型前面加上一个“final”修饰符。这样一来，在循环体里对循环变量进行赋值，就会导致一个编译错误。借助这一机制，就可以有效的杜绝有意或无意的进行“在循环体里修改循环变量”的操作了。

清单7：禁止重新赋值

int[] integers = {1， 2， 3， 4};

for (final int i : integers) {

i = i / 2; /\* 编译时出错 \*/

}

注意，这只是禁止了对循环变量进行重新赋值。给循环变量的属性赋值，或者调用能让循环变量的内容变化的方法，是不被禁止的。

清单8：允许修改状态

Random[] randoms = new Random[]{new Random(1)， new Random(2)， new Random(3)};

for (final Random r : randoms) {

r.setSeed(4);/\* 将所有Random对象设成使用相同的种子 \*/

System.out.println(r.nextLong());/\* 种子相同，第一个结果也相同 \*/

}

3. 类型相容问题

为了保证循环变量能在每次循环开始的时候，都被安全的赋值，J2SE 1.5对循环变量的类型有一定的限制。这些限制之下，循环变量的类型可以有这样一些选择：

循环变量的类型可以和要被遍历的对象中的元素的类型相同。例如，用int型的循环变量来遍历一个int[]型的数组，用Object型的循环变量来遍历一个Collection等。

清单9：使用和要被遍历的数组中的元素相同类型的循环变量

int[] integers = {1， 2， 3， 4};

for (int i : integers) {

System.out.println(i);/\* 依次输出“1”、“2”、“3”、“4” \*/

}

清单10：使用和要被遍历的Collection中的元素相同类型的循环变量

Collection< String> strings = new ArrayList< String>();

strings.add("A");

strings.add("B");

strings.add("C");

strings.add("D");

for (String str : integers) {

System.out.println(str);/\* 依次输出“A”、“B”、“C”、“D” \*/

}

循环变量的类型可以是要被遍历的对象中的元素的上级类型。例如，用int型的循环变量来遍历一个byte[]型的数组，用Object型的循环变量来遍历一个Collection< String>（全部元素都是String的Collection）等。

清单11：使用要被遍历的对象中的元素的上级类型的循环变量

String[] strings = {"A"， "B"， "C"， "D"};

Collection< String> list = java.util.Arrays.asList(strings);

for (Object str : list) {

System.out.println(str);/\* 依次输出“A”、“B”、“C”、“D” \*/

}

循环变量的类型可以和要被遍历的对象中的元素的类型之间存在能自动转换的关系。J2SE 1.5中包含了“Autoboxing/Auto-Unboxing”的机制，允许编译器在必要的时候，自动在基本类型和它们的包裹类（Wrapper Classes）之间进行转换。因此，用Integer型的循环变量来遍历一个int[]型的数组，或者用byte型的循环变量来遍历一个Collection< Byte>，也是可行的。

清单12：使用能和要被遍历的对象中的元素的类型自动转换的类型的循环变量

int[] integers = {1， 2， 3， 4};

for (Integer i : integers) {

System.out.println(i);/\* 依次输出“1”、“2”、“3”、“4” \*/

}

注意，这里说的“元素的类型”，是由要被遍历的对象的决定的――如果它是一个Object[]型的数组，那么元素的类型就是Object，即使里面装的都是String对象也是如此。

可以限定元素类型的Collection

截至到J2SE 1.4为止，始终无法在Java程序里限定Collection中所能保存的对象的类型――它们全部被看成是最一般的Object对象。一直到J2SE 1.5中，引入了“泛型（Generics）”机制之后，这个问题才得到了解决。现在可以用Collection< T>来表示全部元素类型都是T的Collection。