JDK1.5 新特性

JDK1.5 一个重要主题就是通过新增一些特性来简化开发，这些特性包括泛型，for-each 循环，自动装包/拆包，枚举，可变参数, 静态导入 。使用这些特性有助于我们编写更加清晰，精悍，安全的代码。

下面我们简单介绍一下这些新特性。

1.泛型(Generic)

C++通过模板技术可以指定集合的元素类型，而 Java 在 1.5 之前一直没有相对应的功能。一个集合可以放任何类型的对象，相应地从集合里面拿对象的时候我们也不得不对他们进行强制得类型转换。猛虎引入了泛型，它允许指定集合里元素的类型，这样你可以得到强类型在编译时刻进行类型检查的好处。

```
Collection<String> c = new ArrayList();
c.add(new Date());
```

编译器会给出一个错误，

add(java.lang.String) in java.util.Collection<java.lang.String> cannot be applied to (java.util.Date)

2.For-Each 循环

For-Each 循环得加入简化了集合的遍历。假设我们要遍历一个集合对其中的元素进行一些处理。典型的代码为：

```
void processAll(Collection c){
  for(Iterator i=c.iterator(); i.hasNext();){
    MyClass myObject = (MyClass)i.next();
    myObject.process();
  }
}
```

使用 For-Each 循环，我们可以把代码改写成，

```
void processAll(Collection<MyClass> c){
for (MyClass    myObject :c)
     myObject.process();
}
```

这段代码要比上面清晰许多，并且避免了强制类型转换。

3.自动装包/拆包(Autoboxing/unboxing)

自动装包/拆包大大方便了基本类型数据和它们包装类地使用。

自动装包：基本类型自动转为包装类.(int >> Integer)

自动拆包：包装类自动转为基本类型.(Integer >> int)

在 JDK1.5 之前，我们总是对集合不能存放基本类型而耿耿于怀，现在自动转换机制解决了我们的问题。

```
int a = 3;
Collection c = new ArrayList();
c.add(a);//自动转换成 Integer.
Integer b = new Integer(2);
c.add(b + 2);
```

这里 Integer 先自动转换为 int 进行加法运算，然后 int 再次转换为 Integer.

4.枚举(Enums)

JDK1.5加入了一个全新类型的“类”－枚举类型。为此JDK1.5引入了一个新关键字enmu. 我们可以这样来定义一个枚举类型。

```
public enum Color
{Red,
White,
Blue}
```

然后可以这样来使用 Color myColor = Color.Red.

枚举类型还提供了两个有用的静态方法 values()和 valueOf(). 我们可以很方便地使用它们，例如

```
for (Color c : Color.values())
System.out.println(c);
```

5.可变参数(Varargs)

可变参数使程序员可以声明一个接受可变数目参数的方法。注意，可变参数必须是函数声明中的最后一个参数。假设我们要写一个简单的方法打印一些对象，

```
util.write(obj1);
util.write(obj1,obj2);
util.write(obj1,obj2,obj3);
…
```

在 JDK1.5 之前，我们可以用重载来实现，但是这样就需要写很多的重载函数，显得不是很有效。如果使用可变参数的话我们只需要一个函数就行了

```
public void write(Object... objs) {
for (Object obj: objs)
System.out.println(obj);
}
```

在引入可变参数以后，Java 的反射包也更加方便使用了。对于 c.getMethod("test", new Object[0]).invoke(c.newInstance(), new Object[0])，现在我们可以这样写了 c.getMethod("test").invoke(c.newInstance())，这样的代码比原来清楚了很多。

6.静态导入(Static Imports)

要使用用静态成员（方法和变量）我们必须给出提供这个方法的类。使用静态导入可以使被导入类的所有静态变量和静态方法在当前类直接可见，使用这些静态成员无需再给出他们的类名。

```
import static java.lang.Math.*;
…….
r = sin(PI * 2); //无需再写 r = Math.sin(Math.PI);
```

不过，过度使用这个特性也会一定程度上降低代码地可读性。

7. Covariant Return Types overriding

5.0

```
public Employee clone(){...}
...
Employee cloned = e.clone();
```

1.4

```
public Object clone(){...}
...
Employee cloned = (Employee )e.clone();
```