**Java 协变式覆盖(Override)和泛型重载(Overload)**

 @author ixenos

**1.协变式覆盖(Override)**

在JDK 1.4及以前，子类方法如果要覆盖超类的某个方法，必须具有完全相同的方法签名，包括返回值也必须完全一样。

**JDK 5开始，只要子类方法与超类方法具有相同的方法签名，或者子类方法的返回值是超类方法的子类型(增加了对协变返回值的支持)，就可以覆盖**。这样有什么好处呢？以Object类的clone方法为例：

class Object {

...

public Object clone() { ... }

}

在5.0以前，如果子类需要重载clone方法，必须像下面这样写代码：

class Point {

public int x;

public int y;

public Point(int x, int y) { this.x=x; this.y=y; }

public Object clone() { return new Point(x,y); }

}

虽然在我们的Point类里，clone方法总是返回一个Point类型的对象，但却必须把返回类型写成Object，在外部使用clone方法时也必须使用恼人的强制类型转换。

在Java5.0以后，我们就可以利用新的覆盖规则，像下面这样编写代码：

class Point {

public int x;

public int y;

public Point(int x, int y) { this.x=x; this.y=y; }

public Point clone() { return new Point(x,y); }

}

 这样，我们就可以直接使用Point p2 = p1.clone(); 而不用强制类型转换了。

**2．泛型重载(overload)**

       Java的**方法重载**一般指在同一个类中的两个同名方法，规则是：两个方法必须具有**不同的方法签名**。因此**形式参数**必须不相同，使得编译器能够区分开这两个重载的方法。由于编译器不能仅仅通过方法的返回值类型来区分重载方法，所以如果两个方法只有返回类型不同，其它完全一样，**编译是不能通过**的。（但是JVM可以，因为**JVM是根据参数类型和返回类型来确定一个方法的**）

在**泛型方法的重载**时，这个规则稍微有一点变化。先看下面代码：

[复制代码](javascript:void(0);)

class Overloaded {

public static int sum(List<Integer> ints) {

int sum = 0;

for (int i : ints) sum += i;

return sum;

}

public static String sum(List<String> strings) {

StringBuffer sum = new StringBuffer();

for (String s : strings) sum.append(s);

return sum.toString();

}

}

[复制代码](javascript:void(0);)

**上面是两个泛型方法的重载例子，由于Java的泛型采用擦除法实现，List<Integer>和List<String>在运行时是完全一样的，都是List类型。也就是，擦除后的方法签名如下：**

int sum(List)

String sum(List)

**Java允许这两个方法进行重载，虽然它们的方法签名相同(形参)，只有返回值类型不同。**这在两个普通方法的重载中是不允许的。当然了，如果两个泛型方法的参数在擦除后相同，而且返回值类型也完全一样，那编译肯定是不能通过的。

类似地，一个类不能同时实现两个具有相同擦除的接口。如Class A implements Comparable<Integer>, Comparable<Long>。

总结一下，两个泛型方法在擦除泛型信息后，如果具有相同的参数类型，而返回值不一样，是可以进行重载的。

**泛型、重载是java语言级别的，但“擦除”技术是关于实现的，它关系到合法class文件的生成，而合法的class文件才能被jvm接受，**j**vm本来就支持签名相同，但返回类型不同的方法存在**。

     在java语言角度的添加这种限制也是自然的。比如两个方法：

　　void test(int i);

　　int test(int i);

　　编译器不能确定到底应该调用哪个方法，所以这种情况在java中不允许存在。但是，对于这两个方法test(ArrayList<String> list)和test(ArrayList<Integer> list),在java语言的级别，这是合法的重载，因为**编译器**可以通过参数类型信息来确定调用哪个版本。再加上返回类型不同，经过编译和类型擦除得到的两个方法是可以在class文件中共存的。这样问题就解决了。