**Java 协变性、逆变性**

@author ixenos

   **协变性、逆变性和无关性**

在面向对象的计算机程序语言中，经常涉及到类型之间的转换，例如从具体类小猫到动物之间的类型转换（上行转换），或者从形状向三角形之间的转换（下行转换）。

协变性（covariance）、逆变性(contravariance)和无关性（invariant）。他们都是用来描述类型转换的性质的术语，形式化的描述如下：

如果A和B是类型，f表示类型转换，≤表示子类型关系，（例如A≤B，表示A是B的子类）那么：

如果A≤B 则f(A) ≤ f(B)，**小推大**，那么f是**协变**的；如果A≤B 则f(B) ≤ f(A)，**大推小**，那么f是**逆变**的；如果上面两种都不成立，那么f是**无关**的。

 **Java语言有一些语法可以用协变性和逆变性来理解**

* **Java泛型**

class DataHolder<T>{

}

　　　　假如类型 A ≤ B, 但是**直接使用DataHolder<A> 和DataHolder<B>是不可协变的**，之前我们已经叙述过。

　　　　但是利用Java提供的**通配符语法**，却**可以提供一个协变的类型转换**。  
　　　　　　　DataHolder<A> ≤DataHolder<? Extends B>  // ?是通配符，但**这样协变是类型不安全的**

　　　　例如：

static boolean find(Iterable<? extends Object> where, Object what){ //通配符?赋给泛型变量，可以有一个协变的类型转换

return false;

}

　　　　协变的类型转换：可使用Iterable<String> 来调用find函数。

  **数组**

**协变数组是有漏洞的，具体分析请看：数组协变带来的静态类型漏洞 rednaxelafx.iteye.com/blog/379703**

　　注意：  
例如 A = String, B = Object 我们这里有A≤B  
f(A) = String[], f(B) = Object[]. 因为Object[] objects = new String[n].（赋值） 所以我们可以认为数组具有**协变性**。

  **协变返回值（在子类重写方法的时候）**

在面向对象语言中，一个协变返回值方法是一个**在子类覆盖(Override)该方法的时候**，方法的返回值可以被一个“更窄”的类型所替代。（Java JDK5.0）  
例如：

[复制代码](javascript:void(0);)

public static class Super {

Object getSomething() {

return null;

}

}

public static class SubClass extends Super {

@Override

String getSomething() {

return null;

}

}

[复制代码](javascript:void(0);)

这段话主要的思想就是协同返回值使用在子类需要一个不同于父类覆盖方法的返回值，但是**返回值却有继承关系**的情况下。  
一个可能更好的例子如下所示：

[复制代码](javascript:void(0);)

class Collection {

Iterator iterator() { ... }

}

class List extends Collection {

@Override

ListIterator iterator() { ... }

}

[复制代码](javascript:void(0);)

Iterator函数获得到当前集合的迭代器，在子类中，迭代器有着更确切的表示，所以使用了Iterator的子类ListIterator作为新的返回值。

  **逆变参数(形参)**  
先举个例子，可以一下子明白逆变参数的例子，其实逆变参数使用环境**依然是在继承体系下的函数覆盖**。

[复制代码](javascript:void(0);)

class Super {

void doSomething(String parameter) {

}

}

class Sub extends Super {

void doSomething(Object parameter) { //逆变了

}

}

[复制代码](javascript:void(0);)

为了更好的理解这个问题，首先引入Liskov Substitution Principle的概念。这个原则的内容就是说，**子类的对象可以在任何需要父类对象的地方使用**。  
这个原则和参数以及返回值有什么关系呢？  
假如我们定义类如下：

[复制代码](javascript:void(0);)

class Super {

ReturnType doSomething(ParameterType a) {

return null;

}

}

class Sub extends Super {

ReturnType doSomething(ParameterType a) {

return null;

}

}

[复制代码](javascript:void(0);)

假如我有Super sup;  
首先考虑参数，现在要调用sup.doSomething(o);  
如果两个函数需要的参数类型一致，没有什么值得讨论的。假如在一些情况下，sub需要的参数类型变成了和ParameterType**有继承关系**的另一个类NewParameterType。如下：

class Sub extends Super {

ReturnType doSomething(NewParameterType a) {

return null;

}

}

**如果NewParameterType是ParameterType的子类，那么sup.doSomething(o)调用中，如果o是指向NewParameterType的对象，则不会被通过。只有NewParameterType是ParameterType的父类的时候，才不会发生任何问题**(形参要变只能逆变)。所以在提出逆变参数的概念。  
同样地，利用这个原则也可以用来理解。  
假如在某使用环境中有如下语句：  
ReturnType rt = sup.doSomething(o);  
假如设计变更，子类需要返回一个与ReturnType有继承关系的类NewReturnType，如果返回的是ReturnType的父类，那么则上面语句则不能通过编译，只有NewReturnType是ReturnType的子类的时候，才不会违反Liskov子类原则。