S.O.L.I.D五大原则之里氏替换原则LSP**前言**

英文原文：http://freshbrewedcode.com/derekgreer/2011/12/31/solid-javascript-the-liskov-substitution-principle/

开闭原则的描述是：

Subtypes must be substitutable for their base types.  
派生类型必须可以替换它的基类型。

在面向对象编程里，继承提供了一个机制让子类和共享基类的代码，这是通过在基类型里封装通用的数据和行为来实现的，然后已经及类型来声明更详细的子类型，为了应用里氏替换原则，继承子类型需要在语义上等价于基类型里的期望行为。

为了来更好的理解，请参考如下代码：

function Vehicle(my) {  
 var my = my || {};  
 my.speed = 0;  
 my.running = false;  
  
 this.speed = function() {  
 return my.speed;  
 };  
 this.start = function() {  
 my.running = true;  
 };  
 this.stop = function() {  
 my.running = false;  
 };  
 this.accelerate = function() {  
 my.speed++;  
 };  
 this.decelerate = function() {  
 my.speed--;  
 }, this.state = function() {  
 if (!my.running) {  
 return "parked";  
 }  
 else if (my.running && my.speed) {  
 return "moving";  
 }  
 else if (my.running) {  
 return "idle";  
 }  
 };  
}

上述代码我们定义了一个Vehicle函数，其构造函数为vehicle对象提供了一些基本的操作，我们来想想如果当前函数当前正运行在服务客户的产品环境上，如果现在需要添加一个新的构造函数来实现加快移动的vehicle。思考以后，我们写出了如下代码：

function FastVehicle(my) {  
 var my = my || {};  
  
 var that = new Vehicle(my);  
 that.accelerate = function() {  
 my.speed += 3;  
 };  
 return that;  
}

在浏览器的控制台我们都测试了，所有的功能都是我们的预期，没有问题，FastVehicle的速度增快了3倍，而且继承他的方法也是按照我们的预期工作。此后，我们开始部署这个新版本的类库到产品环境上，可是我们却接到了新的构造函数导致现有的代码不能支持执行了，下面的代码段揭示了这个问题：

var maneuver = function(vehicle) {  
 write(vehicle.state());  
 vehicle.start();  
 write(vehicle.state());  
 vehicle.accelerate();  
 write(vehicle.state());  
 write(vehicle.speed());  
 vehicle.decelerate();  
 write(vehicle.speed());  
 if (vehicle.state() != "idle") {  
 throw "The vehicle is still moving!";  
 }  
 vehicle.stop();  
 write(vehicle.state());  
};

根据上面的代码，我们看到抛出的异常是“The vehicle is still moving!”，这是因为写这段代码的作者一直认为加速（accelerate）和减速（decelerate）的数字是一样的。但FastVehicle的代码和Vehicle的代码并不是完全能够替换掉的。因此，FastVehicle违反了里氏替换原则。

在这点上，你可能会想：“但，客户端不能老假定vehicle都是按照这样的规则来做”，里氏替换原则(LSP)的妨碍（译者注：就是妨碍实现LSP的代码）不是基于我们所想的继承子类应该在行为里确保更新代码，而是这样的更新是否能在当前的期望中得到实现。

上述代码这个case，解决这个不兼容的问题需要在vehicle类库或者客户端调用代码上进行一点重新设计，或者两者都要改。

**减少LSP妨碍**

那么，我们如何避免LSP妨碍？不幸的话，并不是一直都是可以做到的。我们这里有几个策略我们处理这个事情。

**契约（Contracts）**

处理LSP过分妨碍的一个策略是使用契约，契约清单有2种形式：执行说明书（executable specifications）和错误处理，在执行说明书里，一个详细类库的契约也包括一组自动化测试，而错误处理是在代码里直接处理的，例如在前置条件，后置条件，常量检查等，可以从Bertrand Miller的大作《[契约设计](http://en.wikipedia.org/wiki/Design_by_contract)》中查看这个技术。虽然自动化测试和契约设计不在本篇文字的范围内，但当我们用的时候我还是推荐如下内容：

1. 检查使用测试驱动开发（Test-Driven Development）来指导你代码的设计
2. 设计可重用类库的时候可随意使用契约设计技术

对于你自己要维护和实现的代码，使用契约设计趋向于添加很多不必要的代码，如果你要控制输入，添加测试是非常有必要的，如果你是类库作者，使用契约设计，你要注意不正确的使用方法以及让你的用户使之作为一个测试工具。

**避免继承**

避免LSP妨碍的另外一个测试是：如果可能的话，尽量不用继承，在Gamma的大作《[Design Patterns – Elements of Reusable Object-Orineted Software](http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612)》中，我们可以看到如下建议：

Favor object composition over class inheritance  
尽量使用对象组合而不是类继承

有些书里讨论了组合比继承好的唯一作用是静态类型，基于类的语言（例如，在运行时可以改变行为），与JavaScript相关的一个问题是耦合，当使用继承的时候，继承子类型和他们的基类型耦合在一起了，就是说及类型的改变会影响到继承子类型。组合倾向于对象更小化，更容易想静态和动态语言语言维护。

**与行为有关，而不是继承**

到现在，我们讨论了和继承上下文在内的里氏替换原则，指示出JavaScript的面向对象实。不过，里氏替换原则（LSP）的本质不是真的和继承有关，而是行为兼容性。JavaScript是一个动态语言，一个对象的契约行为不是对象的类型决定的，而是对象期望的功能决定的。里氏替换原则的初始构想是作为继承的一个原则指南，等价于对象设计中的隐式接口。

举例来说，让我们来看一下Robert C. Martin的大作《[敏捷软件开发 原则、模式与实践](http://www.amazon.com/Software-Development-Principles-Patterns-Practices/dp/0135974445)》中的一个矩形类型：

**矩形例子**

考虑我们有一个程序用到下面这样的一个矩形对象:

var rectangle = {  
 length: 0,  
 width: 0  
};

过后，程序有需要一个正方形，由于正方形就是一个长(length)和宽(width)都一样的特殊矩形，所以我们觉得创建一个正方形代替矩形。我们添加了length和width属性来匹配矩形的声明，但我们觉得使用属性的getters/setters一般我们可以让length和width保存同步，确保声明的是一个正方形：

var square = {};  
(function() {  
 var length = 0, width = 0;  
 // 注意defineProperty方式是262-5版的新特性  
 Object.defineProperty(square, "length", {  
 get: function() { return length; },  
 set: function(value) { length = width = value; }  
 });  
 Object.defineProperty(square, "width", {  
 get: function() { return width; },  
 set: function(value) { length = width = value; }  
 });  
})();

不幸的是，当我们使用正方形代替矩形执行代码的时候发现了问题，其中一个计算矩形面积的方法如下：

var g = function(rectangle) {  
 rectangle.length = 3;  
 rectangle.width = 4;  
 write(rectangle.length);  
 write(rectangle.width);  
 write(rectangle.length \* rectangle.width);  
};

该方法在调用的时候，结果是16，而不是期望的12，我们的正方形square对象违反了LSP原则，square的长度和宽度属性暗示着并不是和矩形100%兼容，但我们并不总是这样明确的暗示。解决这个问题，我们可以重新设计一个shape对象来实现程序，依据多边形的概念，我们声明rectangle和square，relevant。不管怎么说，我们的目的是要说里氏替换原则并不只是继承，而是任何方法（其中的行为可以另外的行为）。

**总结**

里氏替换原则（LSP）表达的意思不是继承的关系，而是任何方法（只要该方法的行为能体会另外的行为就行）。