JavaScript继承基础讲解(原型链、借用构造函数、混合模式、原型式继承、寄生式继承、寄生组合式继承)

**原型链**

　　JavaScript中实现继承最简单的方式就是使用原型链，将子类型的原型指向父类型的实例即可，即“子类型.prototype = new 父类型();”，实现方法如下：

// 为父类型创建构造函数

function SuperType() {

this.name = ['wuyuchang', 'Jack', 'Tim'];

this.property = true;

}

// 为父类型添加方法

SuperType.prototype.getSuerperValue = function() {

return this.property;

}

// 为子类型创建构造函数

function SubType() {

this.test = ['h1', 'h2', 'h3', 'h4'];

this.subproperty = false;

}

// 实现继承的关键步骤，子类型的原型指向父类型的实例

SubType.prototype = new SuperType();

// 在此处给子类型添加方法，一定要在实现继承之后，否则会在将指针指向父类型的实例，则方法为空

SubType.prototype.getSubValue = function() {

return this.subproperty;

}

/\* 以下为测试代码示例 \*/

var instance1 = new SubType();

instance1.name.push('wyc');

instance1.test.push('h5');

alert(instance1.getSuerperValue()); // true

alert(instance1.getSubValue()); // false

alert(instance1.name); // wuyuchang,Jack,Tim,wyc

alert(instance1.test); // h1,h2,h3,h4,h5

var instance2 = new SubType();

alert(instance2.name); // wuyuchang,Jack,Tim,wyc

alert(instance2.test); // h1,h2,h3,h4

可以看到如上的代码就是通过原型链实现的一个简单的继承，但看到测试代码示例中还是存在些问题。相信看了我的博文《[面向对象JS基础讲解，工厂模式、构造函数模式、原型模式、混合模式、动态原型模式](http://www.jb51.net/article/53823.htm" \t "_blank)》的童鞋一定知道原型链代码存在的**第一个问题是由于子类型的原型是父类型的实例，也就是子类型的原型中包含的父类型的属性，从而导致引用类型值的原型属性会被所有实例所共享**。以上代码的instance1.name.push('wyc');就可以证明此问题的存在。而原型链的第二个问题就是：**在创建子类型的实例时，不能向超类型的构造函数中传递参数**。因此我们在实际开发中，很少单独使用原型链。

**借用构造函数**

　　为了解决原型链中存在的两个问题，开发人员开始使用一种叫做借用构造函数的技术来解决原型链中存在的问题。这种技术的实现思路也挺简单，只需要在子类型的构造函数内调用父类型的构造函数即可。别忘了，函数只不过是在特定环境中执行代码的对象，因此可以**通过apply()或call()方法执行构造函数**。代码如下：

// 为父类型创建构造函数

function SuperType(name) {

this.name = name;

this.color = ['pink', 'yellow'];

this.property = true;

this.testFun = function() {

alert('http://tools.jb51.net/');

}

}

// 为父类型添加方法

SuperType.prototype.getSuerperValue = function() {

return this.property;

}

// 为子类型创建构造函数

function SubType(name) {

SuperType.call(this, name);

this.test = ['h1', 'h2', 'h3', 'h4'];

this.subproperty = false;

}

// 在此处给子类型添加方法，一定要在实现继承之后，否则会在将指针指向父类型的实例，则方法为空

SubType.prototype.getSubValue = function() {

return this.subproperty;

}

/\* 以下为测试代码示例 \*/

var instance1 = new SubType(['wuyuchang', 'Jack', 'Nick']);

instance1.name.push('hello');

instance1.test.push('h5');

instance1.color.push('blue');

instance1.testFun(); // http://tools.jb51.net/

alert(instance1.name); // wuyuchang,Jack,Nick,hello

// alert(instance1.getSuerperValue()); // error 报错

alert(instance1.test); // h1,h2,h3,h4,h5

alert(instance1.getSubValue()); // false

alert(instance1.color); // pink,yellow,blue

var instance2 = new SubType('wyc');

instance2.testFun(); // http://tools.jb51.net/

alert(instance2.name); // wyc

// alert(instance2.getSuerperValue()); // error 报错

alert(instance2.test); // h1,h2,h3,h4

alert(instance2.getSubValue()); // false

alert(instance2.color); // pink,yellow

可以看到以上代码中子类型SubType的构造函数内通过调用父类型"SuperType.call(this, name);"，从而实现了属性的继承，也可以在子类型创建实例的时候为父类型传递参数了，但新的问题又来了。可以看到我在父类型的构造函数中定义了一个 方法：testFun，在父类型的原型中定义了一个方法：getSuperValue。可是在**实例化子类型后仍然是无法调用父类型的原型中定义的方法**getSuperValue，只能调用父类型中构造函数的方法：testFun。这就同创建对象中只使用构造函数模式一样，使得函数没有复用性可言。考虑到这些问题，借用构造函数的技术也是很少单独使用的。

**组合继承（原型链+借用构造函数）**

　　顾名思义，组合继承就是结合使用原型链与借用构造函数的优点，组合而成的一个模式。实现也很简单，既然是结合，那当然结合了两方的优点，即**原型链继承方法，而在构造函数继承属性**。具体代码实现如下：

// 为父类型创建构造函数

function SuperType(name) {

this.name = name;

this.color = ['pink', 'yellow'];

this.property = true;

this.testFun = function() {

alert('http://tools.jb51.net/');

}

}

// 为父类型添加方法

SuperType.prototype.getSuerperValue = function() {

return this.property;

}

// 为子类型创建构造函数

function SubType(name) {

SuperType.call(this, name);

this.test = ['h1', 'h2', 'h3', 'h4'];

this.subproperty = false;

}

SubType.prototype = new SuperType();

// 在此处给子类型添加方法，一定要在实现继承之后，否则会在将指针指向父类型的实例，则方法为空

SubType.prototype.getSubValue = function() {

return this.subproperty;

}

/\* 以下为测试代码示例 \*/

var instance1 = new SubType(['wuyuchang', 'Jack', 'Nick']);

instance1.name.push('hello');

instance1.test.push('h5');

instance1.color.push('blue');

instance1.testFun(); // http://tools.jb51.net/

alert(instance1.name); // wuyuchang,Jack,Nick,hello

alert(instance1.getSuerperValue()); // true

alert(instance1.test); // h1,h2,h3,h4,h5

alert(instance1.getSubValue()); // false

alert(instance1.color); // pink,yellow,blue

var instance2 = new SubType('wyc');

instance2.testFun(); // http://tools.jb51.net/

alert(instance2.name); // wyc

alert(instance2.getSuerperValue()); // true

alert(instance2.test); // h1,h2,h3,h4

alert(instance2.getSubValue()); // false

alert(instance2.color); // pink,yellow

以上代码通过SuperType.call(this, name);继承父类型的属性，通过SubType.prototype = new SuperType();继承父类型的方法。以上代码很方便的解决了原型链与借用构造函数所遇到的问题，成为了JavaScript中最为常用的实例继承 的方法。但混合模式也并非没有缺点，可以看到在以上代码中在继承方法的时候实际已经继承了父类型的属性，只不过此时对于引用类型属于共享的，因此在子类型 的构造函数内在次调用父类型的构造函数从而继承了父类型的属性而去覆盖了原型中所继承的属性，这样调用两次构造函数显然没有必要，但有什么方法可以解决 呢？在解决此问题时先看以下两个模式。

**原型式继承**

　　原型式继承的的实现方法与普通继承的实现方法不同，原型式继承并没有使用严格意义上的构造函数，而是借助原型可以基于已有的对象创建新对象，同时还不必因此创建自定义类型。具体代码如下：

function object(o) {

function F() {}

F.prototype = o;

return new F();

}

代码示例：

/\* 原型式继承 \*/

function object(o) {

function F() {}

F.prototype = o;

return new F();

}

var person = {

name : 'wuyuchang',

friends : ['wyc', 'Nicholas', 'Tim']

}

var anotherPerson = object(person);

anotherPerson.name = 'Greg';

anotherPerson.friends.push('Bob');

var anotherPerson2 = object(person);

anotherPerson2.name = 'Jack';

anotherPerson2.friends.push('Rose');

alert(person.friends); // wyc,Nicholas,Tim,Bob,Rose

**寄生式继承**

/\* 寄生式继承 \*/

function createAnother(original) {

var clone = object(original);

clone.sayHi = function() {

alert('hi');

}

return clone;

}

使用示例：

/\* 原型式继承 \*/

function object(o) {

function F() {}

F.prototype = o;

return new F();

}

/\* 寄生式继承 \*/

function createAnother(original) {

var clone = object(original);

clone.sayHi = function() {

alert('hi');

}

return clone;

}

var person = {

name : 'wuyuchang',

friends : ['wyc', 'Nicholas', 'Rose']

}

var anotherPerson = createAnother(person);

anotherPerson.sayHi();

**寄生组合式继承**

　　前面说过了JavaScrip中组合模式实现继承的缺点，现在我们就来解决它的缺点，实现思路是，对于构造函数继承属性，而原型链的混成形式继承方法，即不用在继承方法的时候实例化父类型的构造函数。代码如下：

function object(o) {

function F() {}

F.prototype = o;

return new F();

}

/\* 寄生组合式继承 \*/

function inheritPrototype(subType, superType) {

var prototype = object(superType.prototype);

prototype.constructor = subType;

subType.prototype = prototype;

}

而在使用时只需要将组合模式中的“SubType.prototype = new SuperType();”这行代码替换成inheritPrototype(subType, superType);即可。寄生组合式继承的高效率体现在它只调用了一次父类型构造函数，避免了创建不必要的或多余的属性。与此同时，原型链还能保持不 变，因此，还能够正常使用instanceof和isPrototypeof()。这也是目前来说最理想的继承方式了，目前也在向这种模式转型。（YUI 也使用了这种模式。）