[JavaScript Patterns 6.2 Expected Outcome When Using Classical Inheritance](http://www.cnblogs.com/haokaibo/p/Expected-Outcome-When-Using-Classical-Inheritance.html)

2014-07-16 17:05 by 小郝（Kaibo Hao）, 27 阅读, 0 评论, [收藏](http://www.cnblogs.com/haokaibo/p/Expected-Outcome-When-Using-Classical-Inheritance.html), [编辑](http://i.cnblogs.com/EditPosts.aspx?postid=3848796)

[复制代码](javascript:void(0);)

// the parent constructor

function Parent(name) {

this.name = name || 'Adam';

}

// adding functionality to the prototype

Parent.prototype.say = function () {

return this.name;

};

// empty child constructor

function Child(name) {}

**inherit(Child, Parent);**

[复制代码](javascript:void(0);)

A method say() added to the parent constructor’s prototype, and a call to a function called inherit() that takes care of the inheritance. The  inherit() function is not provided by the language, so you have to implement it yourself.

**Classical Pattern #1—The Default Pattern**

Create an object using the  **Parent() constructor** and assign this object to the **Child()’s prototype**.

[复制代码](javascript:void(0);)

function inherit(C, P) {

C.prototype = **new** P();

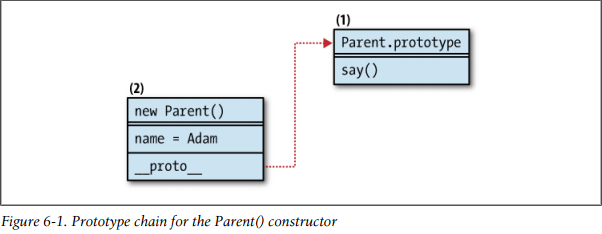
}

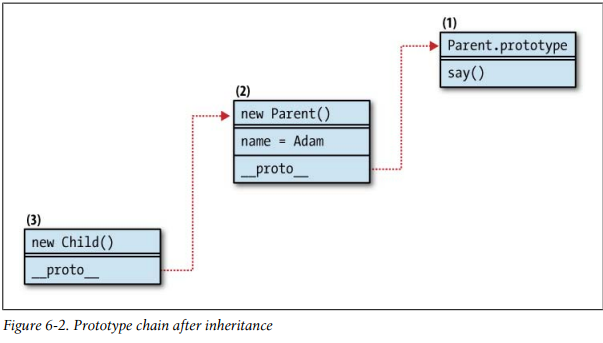
var kid = new Child();

kid.say(); // "Adam"

[复制代码](javascript:void(0);)

1. **Following the Prototype Chain**





1. **Drawbacks when Using Pattern #1**
2. Inherit both own properties added to this and prototype properties.

Note: reusable members should be added to the prototype.

1. It doesn't enable you to pass parameters to child constructor, which the child then passes to the parent.

**Classical Pattern #2 -- Rent-a-Constructor**

Passing arguments from the child to the parent.

[复制代码](javascript:void(0);)

function Child(a, c, b, d) {

**Parent.apply(this, arguments);**

}

// a parent constructor

function Article() {

this.tags = ['js', 'css'];

}

var article = new Article();

// a blog post inherits from an article object

// via the classical pattern #1

function BlogPost() {}

BlogPost.prototype = article;

var blog = new BlogPost();

// note that above you didn't need `new Article()`

// because you already had an instance available

// a static page inherits from article

// via the rented constructor pattern

function StaticPage() {

**Article.call(this);**

}

var page = new StaticPage();

alert(article.hasOwnProperty('tags')); // true

alert(blog.hasOwnProperty('tags')); // false

alert(page.hasOwnProperty('tags')); // true

[复制代码](javascript:void(0);)

1. **The Prototype Chain**

The inheritance was a one-off action that copied parent’s own properties as child’s own properties and that was about it; no \_\_proto\_\_ links were kept.

[复制代码](javascript:void(0);)

// the parent constructor

function Parent(name) {

this.name = name || 'Adam';

}

// adding functionality to the prototype

Parent.prototype.say = function () {

return this.name;

};

// child constructor

function Child(name) {

Parent.apply(this, arguments);

}

function showMsg(msg) {

$('#msg').append(msg).append('<br/>');

}

$(function () {

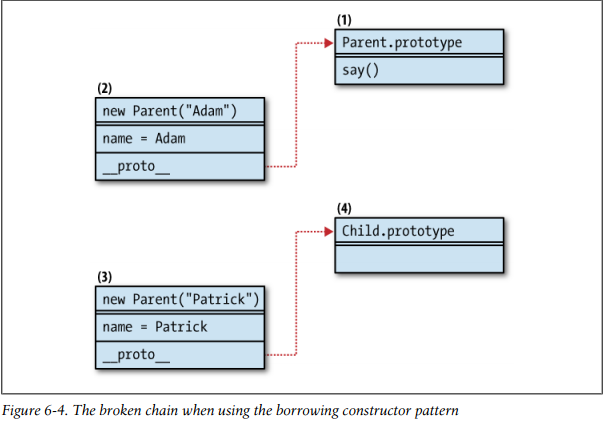
var kid = new Child("Patrick");

showMsg(kid.name); // "Patrick"

showMsg(typeof kid.say); // "undefined"

});

[复制代码](javascript:void(0);)



1. **Multiple Inheritance by Borrowing constructors**

Implement multiple inheritance simply by borrowing from more than one constructor

[复制代码](javascript:void(0);)

function Cat() {

this.legs = 4;

this.say = function () {

return "meaowww";

}

}

function Bird() {

this.wings = 2;

this.fly = true;

}

function CatWings() {

Cat.apply(this);

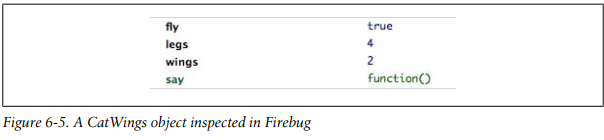
Bird.apply(this);

}

var jane = new CatWings();

console.dir(jane);

[复制代码](javascript:void(0);)



1. Pros and Cons of the Borrowing Constructor Pattern

Pros: Get true copies of the parent's own members and there's no risk that a child can accidentally overwrite a parent's property.

Cons: Nothing from the prototype gets inherited.

**Classical Pattern #3—Rent and Set Prototype**

[复制代码](javascript:void(0);)

function Child(a, c, b, d) {

**Parent.apply(this, arguments);**

}

**Child.prototype = new Parent();**

[复制代码](javascript:void(0);)

The benefit is that the result objects get copies of the parent’s own members and references to the parent’s reusable functionality (implemented as members of the prototype). The child can also pass any arguments to the parent constructor. This behavior is probably the closest to what you’d expect in Java; you inherit everything there is in the parent, and at the same time it’s safe to modify own properties without the risk of modifying the parent.

A drawback is that the parent constructor is called twice, so it could be inefficient. At the end, the own properties (such as name in our case) get inherited twice.

[复制代码](javascript:void(0);)

// the parent constructor

function Parent(name) {

this.name = name || 'Adam';

}

// adding functionality to the prototype

Parent.prototype.say = function () {

return this.name;

};

// child constructor

function Child(name) {

Parent.apply(this, arguments);

}

Child.prototype = new Parent();

function showMsg(msg) {

$('#msg').append(msg).append('<br/>');

}

var kid = new Child("Patrick");

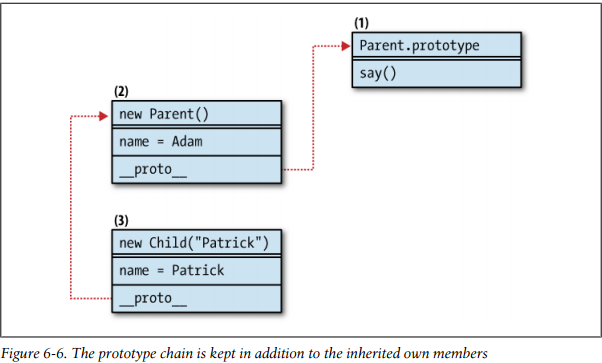
kid.name; // "Patrick"

kid.say(); // "Patrick"

delete kid.name;

kid.say(); // "Adam"

[复制代码](javascript:void(0);)



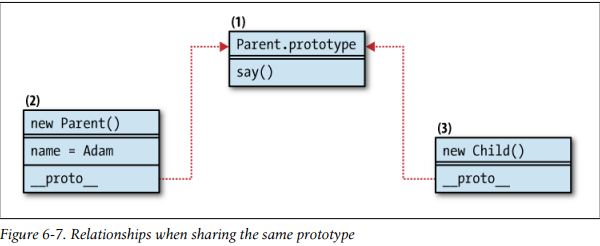
**Classical Pattern #4—Share the Prototype**

This gives you short and fast prototype chain lookups because all objects actually share the  same  prototype.

function inherit(C, P) {

C.prototype = P.prototype;

}



**Drawback**

if  one  child  or  grandchild somewhere down the inheritance chain modifies the prototype, it affects all parents and grandparents.

**Classical Pattern #5—A Temporary Constructor**

An empty function  F(), which serves as a proxy between the child and the parent. F()’s  prototype property points to the prototype of the parent. The prototype of the child is an instance of the blank function:

[复制代码](javascript:void(0);)

function inherit(C, P) {

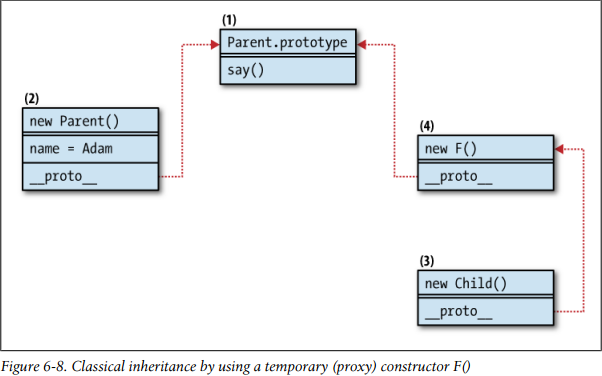
var F = function () {};

F.prototype = P.prototype;

C.prototype = new F();

}

[复制代码](javascript:void(0);)



In this pattern, any members that the parent constructor adds to this are not inherited.

1. Storing the Superclass

The property is called uber because “super” is a reserved word and “superclass” may

lead the unsuspecting developer down the path of thinking that JavaScript has classes.

Here’s an improved implementation of this classical pattern:

[复制代码](javascript:void(0);)

function inherit(C, P) {

var F = function () {};

F.prototype = P.prototype;

C.prototype = new F();

C.uber = P.prototype;

}

[复制代码](javascript:void(0);)

1. Resetting the Constructor Pointer

If you don’t reset the pointer to the constructor, then all children objects will report that Parent() was their constructor, which is not useful.

[复制代码](javascript:void(0);)

// parent, child, inheritance

function Parent() {}

function Child() {}

inherit(Child, Parent);

// testing the waters

var kid = new Child();

kid.constructor.name; // "Parent"

kid.constructor === Parent; // true

function inherit(C, P) {

var F = function () {};

F.prototype = P.prototype;

C.prototype = new F();

C.uber = P.prototype;

C.prototype.constructor = C;

}

[复制代码](javascript:void(0);)

Create temporary (proxy) constructor once and only change its prototype. You can use an immediate function and store the proxy function in its closure:

[复制代码](javascript:void(0);)

var inherit = (function () {

// This will only be executed once which means only one function object is created for every inheritance.

var F = function () {};

return function (C, P) {

F.prototype = P.prototype; // F.prototype.constructor is pointed to Parent.

C.prototype = new F();

C.uber = P.prototype;

C.prototype.constructor = C;

}

}());

[复制代码](javascript:void(0);)

**References*:***

*JavaScript Patterns -*by Stoyan Stefanov (O`Reilly)