**Commonalities**

• There’s a convention on how to name a method, which is to be considered the constructor of the class.

• Classes inherit from other classes.

• There’s access to the parent class (superclass) from within the child class.

The function takes two parameters: a parent class to be inherited and implementation of the new class provided by an object literal.

1. A  Child()  constructor function is created. This is the function that will be returned at the end and will be used as a class. In this function the \_\_construct method is called if it exists. Also before that the parent’s  \_\_construct is called (again, if it exists)  using  the  static  uber property.  There  might  be  cases  when uber is  not defined—when  you  inherit  from  Object for  example,  as  the  case  was  with  the Man class definition.

2. The second part takes care of the inheritance bit. It’s simply using the classical inheritance’s Holy Grail pattern. There’s only one new thing: setting the Parent to Object if no  Parent was passed to inherit from.

3. The  final  section  is  looping  through  all  the  implementation  methods  (such  as \_\_construct and getNam ein the examples), which are the actual definition of the class and adding them to the prototype of Child.

[复制代码](javascript:void(0);)

var klass = function (Parent, props) {

var Child, F, i;

// 1. new constructor

Child = function () {

if (Child.uber && Child.uber.hasOwnProperty("\_\_construct")) {

Child.uber.\_\_construct.apply(this, arguments);

}

if (Child.prototype.hasOwnProperty("\_\_construct")) {

Child.prototype.\_\_construct.apply(this, arguments);

}

};

// 2. inherit

Parent = Parent || Object;

F = function () {};

F.prototype = Parent.prototype;

Child.prototype = new F();

Child.uber = Parent.prototype;

Child.prototype.constructor = Child;

// 3. add implementation methods

for (i in props) {

if (props.hasOwnProperty(i)) {

Child.prototype[i] = props[i];

}

}

// return the "class"

return Child;

};

// The demo for the 6.2 Klass

var Man = klass(null, {

\_\_construct: function (what) {

showMsg("Man's constructor");

this.name = what;

},

getName: function () {

return this.name;

}

});

var SuperMan = klass(Man, {

\_\_construct: function (what) {

showMsg("SuperMan's constructor");

},

getName: function () {

var name;

if (SuperMan.uber.hasOwnProperty("getName")) {

name = SuperMan.uber.getName.call(this);

}

return "I am " + name;

}

});

function showMsg(msg) {

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

}

$(function () {

var clark = new SuperMan('Clark Kent');

showMsg(clark.getName());

// "I am Clark Kent"

});

[复制代码](javascript:void(0);)

**Advantage**

This pattern allows you to forget about the prototypes completely, and the good thing is you can tweak the syntax and the conventions

to resemble another of your favorite languages.

**Disadvantage**

It brings the whole confusing notion of classes, which don’t technically exist in the language. It adds new syntax and new rules to learn and remember.

**References*:***

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