**有意思的function扩展……**

[[复制]](http://bbs.51js.com/viewthread.php?tid=77397&extra=&highlight=&page=1###)

**Code:**

<script>  
Function.prototype.$bind=function(object)  
{  
    var callback = function (fn) {  
        return fn;  
    }  
    with(object)  
    {  
        return eval('callback(' + this.toString() + ')');  
    }  
}  
  
var obj={a:1,b:2};  
var f=function (){  
    a=10;  
    b=11;  
}.$bind(obj);  
f();  
alert(obj.a);  
//----------------------------------------------------------------------------------------------------  
G = {};  
G.objInstanceOf = function(obj, c){  
    if(typeof(c) == "string")  
        return typeof(obj) == c;  
    if(obj instanceof c || c.\_\_templete\_\_ && obj instanceof c.\_\_templete\_\_)  
        return true;  
    var \_interfaces = obj && obj.\_\_interfaces\_\_;  
    if(\_interfaces)  
        for(var i = 0, len = \_interfaces; i < len; i++){  
            if(\_interfafces[i] == c)  
                return true;  
        }  
    return false;  
};  
  
G.objectAsPrototype = function(obj, c){  
    c = c || function(){};  
    c.prototype = obj;  
    return c;  
};  
  
Function.prototype.getPrototypeObject = function(){  
    var p = this.\_\_templete\_\_ || (this.\_\_templete\_\_= G.objectAsPrototype(this.prototype));  
    return new p();  
};  
  
Function.prototype.$pextends = function(p){  
    var me = this.$bind({$super:p});  
    var ins = function()  
    {      
        var ret = me.apply(this, arguments);  
        return ret;  
    }  
  
    ins.prototype = p.getPrototypeObject();  
    return ins;  
};  
//--------------------------------------------------------------------------------------------------------  
function B(){};  
var A = function(){  
    alert($super);  
}.$pextends(B);  
var a = new A();  
//---------------------------------------------------------------------------------------------------------  
Function.prototype.$verify = function(){  
  
    var me = this;  
    var \_args = arguments;  
      
    //第一范式 [new] T <=> [new] R:function(){donothing, return T.apply},R.prototype = T.prototype   
    var mins = function(){  
        for(var i = 0, len = \_args.length; i < len; i++)  
        {  
            if(!G.objInstanceOf(arguments[i],\_args[i])){  
                    throw new TypeError("函数的参数类型不匹配，位置："+(i+1));  
            }  
        }  
        return me.apply(this, arguments);  
    }  
    mins.prototype = me.prototype;  
      
    return mins;      
}  
//-------------------------------------------------------------------------------------------------------  
var foo = function(x,y){  
    alert(x+y);  
}.$verify("number","number");  
  
foo(1,2);  
  
var foo2 = function(x,y){  
    x(y);  
}.$verify(Function,"number");  
//foo("error",2);  
  
foo2(function(x){alert(x)},10);  
//foo2("x","y");  
  
var Class3 = function(x,y){  
    this.x = x;  
    this.y = y;  
}.$verify("number","number");  
Class3.prototype.dist2 = function(){return this.x\*this.x + this.y\*this.y};  
var c = new Class3(10,20);  
alert(c.dist2());  
  
//--------------------------------------------------------------------------------------------------------  
Function.prototype.$staticable = function()  
{  
    var me = this;  
    var cache = [];  
    var index = 0;  
  
    var mins = function(){  
        mins.alloc = function(){  
            cache[index] = cache[index] || {};  
            return cache[index++];  
        };  
        var ret = me.apply(this, arguments);  
        mins.alloc = null;  
        index = 0;  
        return ret;  
    }  
  
    mins.prototype = me.prototype;  
  
    return mins;   
}  
//----------------------------------------------------------------------------------------------------------  
var Test = function(){  
    var x = Test.alloc();  
    x.a = x.a || 0;  
    x.a++;  
    return x.a;  
}.$staticable();  
  
for(var i = 0; i < 10; i++)  
    alert(Test());  
//-----------------------------------------------------------------------------------------------------------  
Function.prototype.$protected = function(){  
    var me = this;  
      
    var mins = function(){  
  
        var p = me.getPrototypeObject();  
  
        var ret = function(){}; //create a new object  
  
        for(var each in mins.prototype)  //clone prototypes  
        {  
            if(mins.prototype[each] instanceof Function){  
                ret.prototype[each] = function(){  
                    return mins.prototype[each].apply(p,arguments);    //call by p  
                }  
                p[each] = mins.prototype[each];  
            }  
            else{   
                p[each] = ret.prototype[each] = mins.prototype[each];  
            }  
        }  
        me.apply(p,arguments);//clone a new object by me  
  
        return new ret(); //return this object;  
    }  
    return mins;  
}  
//----------------------------------------------------------------------------------------------------------------  
//将一个类型的this域声明为全部私有  
var Test = function(x,y)  
{  
    this.x = x;  
    this.y = y;  
}.$protected();  
Test.prototype.getX = function(){  
    return this.x;  
}  
var t = new Test(5,10);  
alert(t.x);  
alert(t.getX());  
</script>

  
   提示：您可以先修改部分代码再运行

unction.prototype.$bind=function(object)  
{  
    var tmp;  
    with(object)  
    {  
        return eval('tmp=(' + this.toString() + ')');  
    }  
}  
  
我解释下这个吧 这是针对Function扩展的一个方法 将一个函数绑定到一个对象 在函数中 可以直接用变量访问相应的属性 比如 例子中的   
var obj={a:1,b:2};  
var f=function (){  
    a=10;//这里修改了obj.a  
    b=11;  
}.$bind(obj);  
f();  
alert(obj.a);  
  
不同于Prototype.js的bind 这个$bind改变了函数的作用域而非this  
  
这个$bind可能有很多用处 包括实现设计模式templete method  实现像C++和C#那样成员函数直接用名称访问成员变量 还有一些私有之类的性质都可能用它来实现

关于$bind()方法，winter的修改要好些，我看不出原来代码中声明一个callback()的理由。不过在winter的代码  
中，tmp变量也可以省掉，方法可以是这样：  
  
Function.prototype.$bind=function(object)  
{  
    with(object)  
        return eval('[' + this.toString() + '][0]');  
}  
  
如果你觉得这里构造一个数组是浪费，那么可以更为复杂一点的代码：  
Function.prototype.$bind=function(object)  
{  
    with(object)  
        return eval('(' + this.toString() + ').prototype.constructor');  
}

哈哈。对了，winter，既然提到这个话题， 不妨把书里讲到的bind()的另外一个做  
法写给你，至少~ 比现在这个用eval()来做的效率要高许多。:)  
  
Function.prototype.bind = function(obj) {  
  with (this) return function() {  
    return apply(obj, arguments);  
  }  
}  
  
在书中的"7.12 使用对象闭包来重置重写", P488。黑黑...倒觉得我好象在做推销呢~ http://bbs.51js.com/images/smilies/funk.gif

最短版本如下：  
  
Function.prototype.bindVars = function(vars) {  
        with (vars) {  
                return eval('0,' + this);  
        }  
}  
  
关于IE在eval函数表达式时的问题，请看：<http://hax.javaeye.com/blog/230182>  
关于with结构中的eval('var x=...')，请看：<http://hax.javaeye.com/blog/126544>  
  
我建议把这个函数改名叫bindVars，以便和一般的bind区分。  
  
此外，bindVars在IE5（JScript 5.0）下应该是不起作用的（虽然我没测试过）。  
  
bindVars还有一个缺点，就是所有Object.prototype上的属性都会被bind上去，比如toString。要做到完美，就还得再花不少功夫改进。  
  
最后就是，用with虽然可以干出很多有趣的事情，但是毕竟属于耍特技，是有风险的。