# JS知识使用日记

# 闭包

## 使用闭包实现参数变量的getter和setter

function couter(n) {

return {

get count() { return n++; },

set count(m) { n = m; }

}

}

var c = couter(200);

console.log(c.count);

c.count = 2;

console.log(c.count);

## 使用闭包实现对象属性的getter和setter

/\*\*

\* 给对象中的指定的属性实现存储器方法，

\* 方法名称为get<Name>和set<Name>，例如 o.getName();

\* 如果提供了一个判断函数，setter方法就会用它来检测参数的合法性，

\* 如果验证返回值为false，setter方法抛出一个异常，否则存储它

\*

\* 这个函数有一个非同寻常之处，

\* 就是getter和sertter方法所操作的属性值并没有存储在对象o中，

\* 相反，这个值仅仅是保存在函数中的局部变量中

\* getter和setter方法同样是局部函数，因此可以访问这个局部变量，

\* 也就是说，对于两个存储器方法来说这个变量是私有的，

\* 没有办法绕过存储器方法来设置或修改这个值

\*/

function addPrivateProperty (o, name, /\*...\*/predicate) {

// 参数验证操作

if (Object.prototype.toString.call(o).slice(8,-1) !== "Object") {

throw new Error(o + " must be a object");

}

if (typeof(name) !== "string") {

throw new Error(name + " must be a string");

}

if (predicate && typeof(predicate) !== "function") {

throw new Error(predicate + " must be a function");

}

// 对name首字母做大写操作

name = name[0].toUpperCase() + name.slice(1);

var value; // 这是一个私有值

// getter 方法简单地将其返回

o["get" + name] = function () { return value; };

// setter 方法首先检测值是否合法，若不合法，抛出一个异常，否则存储它

o["set" + name] = function (val) {

if (predicate && !predicate(val)) {

throw new Error("set" + name + ": invaild value " + v);

}

value = val;

};

}

// 测试代码

var o = {};

// 要验证的属性方法

var verifyOProperty = {

name: function(val) {

return typeof(val) === "string";

},

age: function(val) {

return typeof(val) === "number";

},

sex: function(val) {

return typeof(val) === "number";

}

};

// 添加要验证的属性私有方法

for (var key in verifyOProperty) {

if (verifyOProperty.hasOwnProperty(key)) {

addPrivateProperty(o, key, verifyOProperty[key]);

}

}

// 开始测试对象属性

o.setName("bamboo");

console.log(o.getName());

o.setSex("男");

console.log(o.getSex());

## 使用闭包动态修改已有方法--“monkey-patching”

function trace(o, m) {

// 参数验证操作

if (Object.prototype.toString.call(o).slice(8,-1) !== "Object") {

throw new Error(o + " must be a object");

}

if (typeof(m) !== "string") {

throw new Error(m + " must be a string");

}

if (typeof(o[m]) !== "function") {

throw new Error(o[m] + " must be a function");

}

var original = o[m];

o[m] = function() {

console.log(new Date(), "Entering", m);

// 调用原方法

var result = original.apply(this, arguments);

console.log(new Date(), "Exiting", m);

return result;

}

}

// 测试代码

var obj = {

getMax: function(arr) {

return Math.max.apply(Math, arr);

}

};

trace(obj, "getMax");

obj.getMax([1, 3, 2, 5, 11]);

# 函数

## 不完全函数