## 函数：

1、在js中一切皆对象

2、如果在js中声明了一个function,该function是由Function产生的

3、任何一个对象都有可能成为任何一个对象的属性

4、this可以代表任意对象，所以js的this具有多态性

案例1

function Person(){

alert("person");

}

function Student(){

alert("student");

}

function SupStudent(){

alert("sup student");

}

SupStudent.aa = Person;

SupStudent.bb = Student;

Person();

alert(Person.constructor);

//a是一个整形的变量，而不是一个对象

Person.a = 5;//给Person对象动态的添加了一个属性a,值为5

Person.b = Student;//把Student对象赋值给属性b

Person.b();

//错误的写法

//Person.a.c = Student;

//Person.a.c();

Person.b.c = SupStudent;

Person.b.c();

//因为d是undefined，所以d后面是不能添加属性的

//Person.d.c = SupStudent;

//Person.d.c();

/\*\*

\* 作业：

\* a.b.c.d.f.e.g()

\* 输出"i am g"

\*/

答案:

**function** g(){

alert("i am g");

}

**function** a(){

}

//g();

a.b = g;

a.b.c = g;

a.b.c.d = g;

a.b.c.d.e = g;

a.b.c.d.e.f = g;

a.b.c.d.e.f.g = g;

a.b.c.d.e.f.g();

案例2

function Person(){

alert(this);

}

function Student(){

}

Student.aa = Person;

Person("aa");//window.Person() this window

Student.aa();//this Student

var json = {

a:Person

};

json.a();//this json

## js的原型模式：

1、在js中没有类的概念

2、在js中创建对象也是使用new关键字

3、function也是一个构造器函数

4、prototype也是一个对象，是任意一个对象都拥有的属性

function Person(){

}

/\*\*

\* 利用Person创建对象

\* Person是一个构造器函数

\*/

Person.prototype.a = 5;

Person.prototype.b = function(){

alert("b");

}

var p = new Person();

alert(p.a);

p.b();

//Person.b();

Person.prototype.setName = function(name){

this.name = name;

}

Person.prototype.getName = function(){

return this.name;

}

p.setName("aaa");

alert(p.getName());

function Student(){

}

//Student.prototype = Person.prototype;

Student.prototype = p;

var s = new Student();

s.setName("stuent");

alert(s.getName());

createClass

function createClass(jsonClass){

function F(){

}

for(var i in jsonClass){

F.prototype[i] = jsonClass[i];

}

return F;

}

var A = createClass({

a:function(){

alert("a");

},

b:function(){

alert("b");

}

});

var aa = new A();

aa.a();

aa.b();

//var json = {

// a:function(){

// alert("a");

// }

//};

//json.a();

//json['a']();

Extends

function heimaExtends(jsonClass,prop){

/\*\*

\* 定义了一个类的模型

\*/

function F(){

}

/\*\*

\* 第一次走if语句

\*/

if(typeof jsonClass=="object"){

for(var i in jsonClass){

F.prototype[i] = jsonClass[i];

}

}

/\*\*

\* 当第二次调用时

\*/

if(typeof jsonClass=="function"){

F.prototype = jsonClass.prototype;

for(var i in prop){

F.prototype[i] = prop[i];

}

}

return F;

}

/\*\*

\* 创建一个类Person

\* json对象中的内容就是类中的内容

\* @param {Object} name

\*/

var Person = heimaExtends({

setName:function(name){

this.name = name;

},

getName:function(){

return this.name;

}

});

var Student = heimaExtends(Person,{

setId:function(id){

this.id = id;

},

getId:function(){

return this.id;

}

});

var s = new Student();

s.setName("student");

alert(s.getName());

s.setId("1");

alert(s.getId());

## 回调函数

window.onload = function(){

document.getElementById("mybutton").onclick = function(){

post({

method:'post',

url:'../helloServlet',

data:null,

callback:function(data){

alert(data);

}

});

}

}

function ajaxFunction(){

var xmlHttp;

try { // Firefox, Opera 8.0+, Safari

xmlHttp = new XMLHttpRequest();

}

catch (e) {

try {// Internet Explorer

xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");

}

catch (e) {

try {

xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");

}

catch (e) {

}

}

}

return xmlHttp;

}

function post(json){

var xmlHttp = ajaxFunction();

/\*\*

\* 该函数在什么时候执行

\* 该函数是由服务器端触发的

\* 如果客户端的js代码要用到服务器段回调过来的数据，js代码必须放在回调函数中写

\* 回调函数：

\* 1、回调函数的参数只在被调用的方法中起作用

\* 2、回调函数很少有返回值

\* 3、jQuery中post方法的第三个参数是回调函数，所以第三个参数和其他的代码是异步的过程

\*/

var a = 5;

xmlHttp.onreadystatechange = function(){

if (xmlHttp.readyState == 4) {

if (xmlHttp.status == 200) {

var message = xmlHttp.responseText;

a = message;

json.callback(message);

alert("aaa");

}

}

}

xmlHttp.open(json.method, json.url, true);

xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xmlHttp.send(json.data);

alert(a);

alert("bbb");

}

## 事件

$().ready(function(){

for (var i = 0; i < 300; i++) {

$("input[type='button']").unbind("click");

$("input[type='button']").bind("click",function(){

/\*\*

\* 触发自定义的事件

\*/

$(this).trigger("云三很牛",{

a:5,

b:6

});

});

}

$("input[type='button']").unbind("云三很牛");

$("input[type='button']").bind("云三很牛",function(event,json){

alert(json.a+json.b);

});

});

/\*\*

\* 自定义事件

\* 1、自定义一个事件，该事件的名称为"云三很牛"

\* 2、把该事件绑定在button上

\* 3、该事件在点击button的时候触发

\*/

/\*\*

\* 作业：

\* 自定义一个事件：

\* 在页面上有一个下拉列表框，给下拉列表框添加一个自定义的事件，当点击下拉列表框的某一个元素时触发该事件

\* 该事件传递一个参数，把选中的下拉列表框的值传递过来，在自定义的事件中输出

\*/

function Person(){

}

window.Person();

## 闭包

/\*\*

\* 闭包

\* 在函数中定义的函数，在外部使用

\* 闭包范式

\*

\* setName,getName,setId,getId

\* setName和getName成为API

\* setId和getId成为私有的方法

\*/

(function(window){

function Person(){

return {

setName:setName,

getName:getName

};

}

function setName(){

alert("set name");

}

function getName(){

alert("get name");

}

function setId(){

}

function getId(){

}

//动态的给window创建一个属性Person,把Person函数赋值给该属性

window.Person = Person;

})(window);

var p = window.Person();

p.setName();

p.getName();

## jQuery的内核分析：

(function( window, undefined ) {

var jQuery = function( selector, context ) {

return new jQuery.fn.init( selector, context );

}

//利用jQuery的选择器创建的对象就相当于利用jQuery构造器创建出来的对象

jQuery.fn = jQuery.prototype = {

size: function() {},

each: function( callback, args ) {

return jQuery.each( this, callback, args );

},

ready:function(){}

}

//jQuery就是window的一个属性

//$等于jQuery

window.jQuery = window.$ = jQuery;

$.fn=jQuery.prototype=window.jQuery.fn=window.$.fn=window.$.prototype

jQuery.a = 5; == $.a = 5;

jQuery.prototype.b = 6; ==$.prototype.b = 6;

})();