Web前端笔试题（五）

一、函数

1. 应用JavaScript函数递归打印数组到HTML页面上，数组如下：

var item = [

{

name:’Tom’,

age: 70,

child: [{

name:’Jerry’,

age: 50,

child: [

name: ’William’,

age: 20,

child: [……]

]

},

{

name: ’Jessi’,

age: 30

}

]

}

]

输出的HTML如下：

<ul>

<li>Name:Tom</li>

<li>Age:70</li>

<li>Child:

<ul>

<li>Name: Jerry</li>

<li>Age: 50</li>

<li>Child:

<ul>

<li>Name: William</li>

<li>Age: 20</li>

<li>Child: ……</li>

</ul>

</li>

</ul>

<ul>

<li>Name: Jerry</li>

<li>Age: 50</li>

</ul>

</li>

</ul>

var str = "";

function displayProp(obj){

str += "<ul>";

for( var i in obj ){

if( typeof obj[i] == "string" ){

str += "<li>" + ( i + "/" + obj[i] ) + "</li>";

}else{

str += "<li>";

displayProp( obj[i] );

str += "</li>";

}

}

str += "</ul>";

}

displayProp( item );

//alert( str );

document.getElementById('box').innerHTML = str;

二、闭包

1. 举例说明变量的作用域。举例说明如何从外部读取局部变量。

//函数内部直接读取全局变量

//var n=999;

function f1(){

//alert(n);

}

f1(); // 999

//在函数外部自然无法读取函数内的局部变量。

function f2(){

a=44;

}

f2();

alert(a); // 999

//如何从外部读取局部变量？

function f1(){

n=111; //局部变量对fa可见

function f2(){

alert(n); //f2内部的局部变量，对f1 就是不可见的

}//子对象会一级一级地向上寻找所有父对象的变量

return f2; //f1读取f2的内部变量 只要把F2作为返回值

}

//var result=f1();

//result(); // 999

1. 简述闭包的概念。

闭包的概念：定义在一个函数内部的函数

1. 简述闭包的用途。

闭包的用途：可以读取函数内部的变量，让这些变量的值始终保持在内存中

1. 使用闭包注意的注意点。

造成内存遗漏

会影响浏览器的反应速度境地用户体验或造成浏览器无响应

1. 说出下列两段代码的输出结果？解释为什么。

var name = "The Window";

 　　var object = {

 　　　　name : "My Object",

 　　　　getNameFunc : function(){

 　　　　　　return function(){

 　　　　　　　　return this.name;

 　　　　　　};

 　　　　}

 　　};

alert(object.getNameFunc()());

The Window

/\*在闭包中使用this对象也可能会导致一些问题 ,this 对象是在运行时基于函数的执行环境绑定的,如果this在全局范围就是window,如果在对象内部就指向这个对象。而闭包却在运行时指向window的,因为闭包并不属于这个对象的属性或方法。 \*/

var name = "The Window";

 　　var object = {

 　　　　name : "My Object",

 　　　　getNameFunc : function(){

 　　　　　　var that = this;

 　　　　　　return function(){

 　　　　　　　　return that.name;

 　　　　　　};

 　　　　}

 　　};

 　　alert(object.getNameFunc()());

My Object

/\*javascript是动态（或者动态类型）语言，this关键字在执行的时候才能确定是谁。所以this永远指向调用者，即对'调用对象'者的引用。第一部分通过代码：object.getNameFunc()调用返回一个函数。这是个返回的函数，它不在是object的属性或者方法，此时调用者是window。因此输出是 The Window

第二部分，当执行函数object.getNameFunc()后返回的是：

function() {

return that.name;

}

此时的that=this。而this指向object，所以that指向object。无论你执行多少次，他都是对object的引用，所以输出My Object\*/

1. 用三种方法实现打印li的索引（必须有闭包的方法）

方法一

var lis = document.getElementsByTagName("li");

for(var i=0, len = lis.length; i<len; i++){

lis[i].setAttribute("data-num",i);

}

for(var i=0, len = lis.length; i<len; i++){

lis[i].onclick = function(){

alert( this.getAttribute("data-num") );

}

}\*/

//方法二

var lis = document.getElementsByTagName("li")

//foa = 0;

;

for(var i=0, len = lis.length; i<len; i++){

lis[i].index = i;

lis[i].onclick = function(){

alert( this.index );

}

}

方法三：

for(var i=0, len = lis.length; i<len; i++){

(function(i){

lis[i].onclick = function(){

alert( lis[i] );

}

})(i);

}

三、面向对象

1. 完成下列JS面向对象案例
2. 定义父类：Shape（形状）类，Shape只用一个属性color,并有相应的getColor和setColor方法
3. Shape类有两个子类：Rectangle(矩形)类和Circle(圆形)类，子类继承了父类的color属性和getColor、setColor方法。
4. 为两个子类增加相应的属性和getArea方法，可调用getArea方法获得矩形和圆形的面积。

/\*//定义形状类

function Shape( color ){

this.color = color;

}

Shape.prototype.getColor = function(){

return this.color;

}

Shape.prototype.setColor = function( newColor ){

this.color = newColor;

}

//定义矩形子类

function Rectangle( color){

Shape.call( this, color );

}

//实现继承

Rectangle.prototype = new Shape();

Rectangle.prototype.getArea=function( width, height ){

return width\*height;

}

//定义圆形子类

function Circle(){

Shape.call( this, color );

}

//实现继承

Circle.prototype = new Shape();

Circle.prototype.getArea = function( radius ){

return Math.PI\*Math.pow(radius,2);

}

var r1 = new Rectangle( "blue" );

alert( r1.getColor() );

alert( r1.getArea( 30, 40 ) );

\*/

/\*var s1 = new Shape( "yellow" );

alert( s1.getColor() );

s1.setColor( "red" );

alert( s1.getColor() );\*/

四、ajax 与 json

1. 应用原生JS实现ajax封装

var createAjax = function() {

var xhr = null;

try {

//IE系列浏览器

xhr = new ActiveXObject("microsoft.xmlhttp");

} catch (e1) {

try {

//非IE浏览器

xhr = new XMLHttpRequest();

} catch (e2) {

window.alert("您的浏览器不支持ajax，请更换！");

}

}

return xhr;

};

//核心函数。

var ajax = function(conf) {// 初始化

var type = conf.type; //pe参数,可选

var url = conf.url;//url参数，必填

var data = conf.data;//data参数可选，只有在post请求时需要

var dataType = conf.dataType;//datatype参数可选

var success = conf.success;//回调函数可选

if (type == null){

type = "get";//pe参数可选，默认为get

}

if (dataType == null){

dataType = "text";//dataType参数可选，默认为text

}

var xhr = createAjax();// 创建ajax引擎对象

xhr.open(type, url, true);// 打开

if (type == "GET" || type == "get") {// 发送

xhr.send(null);

} else if (type == "POST" || type == "post") {

xhr.setRequestHeader("content-type",

"application/x-www-form-urlencoded");

xhr.send(data);

}

xhr.onreadystatechange = function() {

if (xhr.readyState == 4 && xhr.status == 200) {

if(dataType == "text"||dataType=="TEXT") {

if (success != null){

success(xhr.responseText);//普通文本

}

}else if(dataType=="xml"||dataType=="XML") {

if (success != null){

success(xhr.responseXML);//接收xml文档

}

}else if(dataType=="json"||dataType=="JSON") {

if (success != null){

success(eval("("+xhr.responseText+")"));//将json字符串转换为js对象

}

}

}

};

};

//此函数的用法。

ajax({

type:"post",

url:"test.jsp",

data:"name=dipoo&info=good",

dataType:"json",

success:function(data){

alert(data.name);

}

});

1. 应用原生JS实现JSON的封装（兼容浏览器）

function ShowJsonString(){

response = ("[{

name: 'Joe',

age: '30',

gender: 'M'

},{

name: 'Chandler',

age: '32',

gender: 'M'

},{

name: 'Rose',

age: '31',

gender: 'M'

}]"//字符串形式

);

var response1 = "({

name: 'Vicson',

age: '30',

gender: 'M'

})";//字符串形式,这里的小括号不能少

json = eval(response);

json1 = eval(response1);

alert(json[0].name + "," + json[1].age + "," + json[2].gender);

alert(json1.name);

}

ShowJsonString();\*/