**JavaScript手册**

一、作用域链

AO链，作用域链

1.当仅附值操作时，最后交给window对象

age=18;//如果往上查找都没有var,最后得到window.age

二、词法分析

函数分析, 词法分析

1.参数

1-1.AO.参数变量名 AO.age = undefined

1-2.招收参数：

AO.参数变量名 = 实参 AO.age = 18

2.变量声明

1.查AO属性有没有该var属性，有就略过，没有就添加属性

3.函数声明

1.查AO属性有没有该变量属性，有就覆盖，没有就添加属性

函数执行

1.自上而下

函数之arguments和this

arguments

1.形参与对应的arguments单元是相互映射，相互影响的

2.arguments.length ：接收了多少个参数

3.arguments.callee ：就是当前执行的函数

this

1.this函数内使用时，如果this为null,则this指向window对象

2.this作为对象的方法调用时，this指向其调用者（母体对象）

3.

创建对象，当函数作为构造函数时，作为对象返回，返回值会被忽略

1.构造函数 function Dog(name,age){} var dog = new Dog('huzi',2);

// new过程

//i.系统创建空对象{}

//ii.把函数this指向该对象{}

//iii.执行该函数

//iv.返回该对象

2.json格式{}

函数被call,apply调用

1.call //函数名.call(对象, 参数1, 参数2, 参数n);

作用:把this指向特定对象

fn.call(func,'a','b')

function t(num){alert(this.age + num);}

var human = {age:28};

t.call(human,10); ////把this指向human对象

2.apply

fn.apply(func,['a','b']) //arguments

call与apply的作用是相同的

call传参是var1,var2,var3...

apply传参是数组[var1,var2,var3...] //或直接传入arguments

三、闭包：函数的作用域取决于声明时，而不取决于调用时！

function t1(){

var age = 20;

function t2(){alert(age);}

return t2;

}

1.在js中，t1执行过程中，又生成了t2，而从作用域上来说，t2能访问到age=20。

2.于是age=20没有消失，返回的t1函数形成了一个“环境包”。

四、面向对象

1.没有类，但有构造函数

2.只是一个“属性字典”

var ori = {};

var chicken = {leg:2,song:function(){alert("我是一只鸟");}};

3.json格式：{k1:v1,k2:v2}

4.删除属性和增加属性

delete 属性名

delete chicken.leg; //删

chicken.wear = 2; //增

5.造物主(java)与生物演化(js)

造物主：先有类再有物

生物演化：先有物再有类

6.封装(部分开放，部分封闭)

1)用构造函数创建对象

2)用闭包完成对象私有化

function Girl(a,b){

var tmp = b;

this.a = a;

this.vb = function(){

return tmp;

}

}

7.原型继承

1)通过“原型”的概念来实现继承(照猫画虎)

Tiger.prototype = new Cat();

Tiger是function(画)，new Cat()是对象

Function.prototype = new 对象();

2)继承的关键

hnhu.\_\_proto\_\_ = Tiger.prototype(即cat对象)

hnhu是对象，Tiger.prototype是对象

对象.\_\_proto\_\_ = Function.prototype(即对象)

3)\_\_proto\_\_ -》Object对象-》最终指向 NULL

8.复制继承或原型冒充

1)复制继承

function Tiger(color){

this.color = color;

this.extend = function(parent){ //复制继承

for(var key in parent){

this[key] = parent[key];

}

}

}

tiger.extend(new Cat(4,1));

9.js没有多态

多态：传一只狗进来，即可以呜呜叫，也可以旺旺叫

10.静态方法

var Hashiqi = function(){

this.bark = function(){}

}

Hashiqi.ajax = function(){

alert('ajax');

}

var h = new Hashiqi();

得出：1.ajax()方法是属于“函数”本身，和返回的对象没有关系

2.bark()要调用，必须要new Hashiqi()得到对象，且由返回对象才能调用

3.ajax()方法要调用，不需要new对象，直接用Hashiqi来调用

1)Math.random();

2)$.ajax();

3)jquery插件，2种办法

3.1 通过闭包，把方法写到jquery的原型上

3.2 直接增加$对象的静态方法

$.sing = function(){alert('sing');}

五、事件(DOM高级)

1.事件发生的3步

1)点击

2)事件对象(点击瞬间的信息)

3)事件监听函数(回调函数)

2.绑定事件的3种方式

1)html属性方式

<a href="#" onclick="t()"></a>

坏处：不简洁;效率不高;不符合“行为，结构，样式相分离”

2)事件属性方式

window.onload = function(){

//好处1：行为相分离

document.getElementById('test').onclick = function(){

alert('有人点击div');

//好处2：加this便于操作当时对象

this.style.background = black;

}

//好处3：方便读取“事件对象”

document.getElementById('test').onclick = function(ev){

console.log(ev);

}

}

好处：I)行为相分离;

II)加this便于操作当时对象

III)方便读取“事件对象”,系统自动用第1个参数来传

坏处：只能绑定一个事件

如何两个人同时绑定window.onload，则后面的会覆盖前面的

3)addEventListener方式

i)确定要绑定的事件：click,load,change....

ii)绑定的函数：自定义事件函数

iii)监听执行事件函数的方式：捕捉、冒泡

语法：

\*1:var test1 = document.getElementById('test1');

\*2:test1.addEventListener('click',function(){alert('我是div');},false);

注意细节：

\*1:事件名一律不带on

\*2:绑定事件函数中的“this”指绑定该事件的对象

\*3:执行顺序，是按绑定顺序来执行的！

iv)捕捉与冒泡模型(addEventListener的第三个参数)

捕捉：false、由外到内的查找

冒泡：true、由内到外的查找

3.事件(捕捉与冒泡)的过程中，停止事件的传播(addEventListener中)

w3c: ev.stopPropagation(); //取消冒泡

ie : cancelBubble = true; //取消冒泡(ie9以下)

4.事件(捕捉与冒泡)的过程中，停止事件的效果(addEventListener中)

w3c: ev.preventDefault(); //取消效果

ie : returnValue = false; //取消效果(ie9以下)

5.解除事件绑定

w3c: obj.removeEventListener('click'/\*事件名\*/,cry/\*函数名\*/,false/\*事件模型\*/);

IE : obj.detachEvent('on事件', 监听函数, true/false);

6.IE与W3C事件模型的不同点

1) 绑定的事件函数不一样，IE就要用attachEvent()

2) 事件必须加on,比如标准中第1个参数为click, IE就要写成onclick

3) 捕捉是true,冒泡是false

4) IE6中不支持捕捉模型，只支持冒泡模型;IE7、IE8支持捕捉

注解：但在更新后的系统上，IE6支持捕捉

5) IE7、IE8还有个毛病，就是先绑定后发生(准确的说，有可能随机发生,代查)

6) 绑定的事件中，this的指向问题

w3c: this指向对象本身

IE : this指向window

7)事件对象不一样

w3c: 事件对象自动传到函数的第一个参数

IE : 事件对象放在window.event属性值

注解：实测IE6、IE7、IE8已支持自动传入event对象

8)通过自定义函数解决以上问题

i)先解决上面第1)~4)点

//obj:待绑定事件的对象

//type:事件类型 click,change,mouseover等

//event:事件对象，是一个函数

//model:捕捉/冒泡(false/true)

function addEvent(obj, type, event, model){

if(obj.attachEvent){

//需要手动判断参数的合法性

obj.attachEvent('on'+type, event, model);

}else{

obj.addEventListener(type, event, model);

}

}

ii)再解决第5)点

思维：把事件绑定放到一个数组，依次执行回调函数

function addEvent(obj, type, event, model){

/\* 插入的代码 \*/

if(obj[('\_\_'+type)]){

obj[('\_\_'+type)] = [];

}

obj[('\_\_'+type)].push(event);

if(!obj[('on'+type)]){

obj[('on'+type)] = function(){

for(var len=obj[('\_\_'+type)].length,i=0;i<len;i++){

obj[('\_\_'+type)][i]();

}

}

}

/\* 代码结束 \*/

if(obj.attachEvent){

//需要手动判断参数的合法性

obj.attachEvent('on'+type, event, model);

}else{

obj.addEventListener(type, event, model);

}

}

//html代码

window.onload = function(){

var test1 = document.getElementById('test1');

addEvent(test1,'click',function(){alert('haha');},false);

addEvent(test1,'click',function(){alert('hehe');},false);

}

iii)再解决第6)点

把代码 obj[('\_\_'+type)][i]();

改为 obj[('\_\_'+type)][i].apply(obj);

7.事件委托

1)五子棋

思路：i)造棋盘 table+背景图片

ii)开发下棋功能

iii)判断胜负

2)利用冒泡原理，冒泡到父元素时再执行的特点===称为事件委托

即把子元素的事件委托给父元素

3)利用事件对象中的srcElement即可知道点了哪个td

4)判断是否有棋

ev.srcElement.style.background.indexOf('image') >=0

5)判断胜负

var curr = {x:this.cellIndex,y:this.parentElement.rowIndex}

jquery

1. window.onload()

2. $(document).ready()

区别：i) onload()只能有一个，ready()可以有多个

ii)onload()加载完资源才会显示，ready()只要加载完dom树就会显示

iii)ready()可以简写 $(function(){});

3.DOM转jquery

$(DOM对象)即可转换成jquery对象

4.jquery与其它js共存

1)jQuery.noConflict();

jquery 的 jQuery('#a').val() 与 prototype.js 的 $F('a')

2)与引入js顺序也有关系

5.基本选择器

1) div>a ：强制在div下的a标签才能用这个css

$('div>a').css('color','red');

2) jQuery容错机制，即便标签写错也不会报错

3) ID选择器

$('#p1').css('color','#f90');

4) Class选择器

$('.p1').css('color','#f90');

5) 标签选择器

$('p').css('color','#f90');

6) \*号选择器

$('\*').css('color','#f90');

$('body \*').css('color','#f90');

7) 多重选择器

$('p,div').css('color','#f90');

8) 层次选择器

E F :选择E元素所有的后代元素

E>F :选择E元素的子元素，子选择器

E+next :选择E元素后的紧临的兄弟元素，等价于.next()

$('div+p').css('color','red'); //取后面一个p标签

$('div').next().css('color','red');

E~siblings :选择E元素后的所有兄弟元素，等价于.nextAll()

$('div~p').css('color','red'); //取后面所有p标签

$('div').nextAll().css('color','red');

6.过滤选择器

1) 基本

:first 选取第一个元素

$('p:first').css('background','red');

:last 选取最后一个元素

$('p:last').css('background','red');

:not() 选择除指定之外的元素

$('p:not(".aaa")').css('background','red');

$('p:not(\'.aaa\')').css('background','red');

:event 偶数，重0开始计算

$('p:event').css('background','red');

:odd 奇数，重0开始计算

$('p:odd').css('background','red');

:eq() 选择指定索引的元素

$('p:eq(3)').css('background','red');

:gt() 选取大于指定索引的元素

$('p:gt(2)').css('background','red');

:lt() 选取小于指定索引的元素

$('p:lt(2)').css('background','red');

:header 选取h1,h2,h3,h4,h5,h6标签

$(':header').css('background','red');

:animated 选取正在执行动画的元素

$('#go').animate({'left':'1000'},10000);

$(':animated').css('background','red');

:lang() 语言选择器，没什么用

:root 根元素选择器，没什么用

:target 选择由文档URI格式化识别码表示的目标元素，没用

2) 内容

:contains() 选取含有文本内容的元素

$('p:contains("LiveScript")').css('background','red');

$('p:contains("LiveScript")').show().siblings().hide(); //只显示包含内容的元素

:has() 选择指定元素的元素

$('p:has("strong")').css('background','red');

:empty 选择不包含子元素或者文本空元素

$('p:empty').css('background','red');

:parent 选取含有子元素或者文本的元素

$('p:parent').css('background','red');

3) 可见性

:hidden 选取所有不可见元素，display:none

alert($('div:hidden').html());

:visible 选取所有可见元素

alert($('div:visible').html());

4) 属性

[attr] 选择拥有此属性的选择器

$('div[title]').css('background','red');

[attr=value] 指定属性值的选择器

<div title="js"></div>

$('div[title=js]').css('background','red');

[attr1][attr2][attr3] 复合属性选择器

$('div[id][title=js]').css('background','red');

5) 子元素

:first-child 选择父元素第一个子元素

$('div p:first-child').css('background','red');

:last-child 选择父元素最后一个子元素

$('div p:last-child').css('background','red');

:only-child 选择元素中只包含一个子元素的元素

$('div p:only-child').css('background','red');

:nth-child(表达式) 选取父元素中的第X个元素，X由表达式决定

$('div p:nth-child(2)').css('background','red'); //由1开始

2n,2n-1,odd 奇,even 偶

:first-of-type 选取所有相册元素的第一个兄弟元素 1.9新增

:last-of-type 选取所有相同元素的最后一个兄弟元素 1.9新增

:nth-last-child(表达式) 选择所有他们父元素的第n个子元素，计数从最后一个元素开始到第一个，1.9+

:only-of-type

:nth-of-type(表达式)

:nth-last-of-type(表达式)

6) 表单

:input

alert($(':input').length);

:text 文本框

$(':text').css('background','red');

:password 密码框

$(':password').css('background','red');

:radio 单选框

$(':radio').css('background','red');

:checkbox

$(':checkbox').css('background','red');

alert($('input[type=checkbox]').val());

:submit

:image

:reset

:button

:file

:hidden display:none可用 (opacity:0,visibility:hidden不可用)

alert($('input:hidden').val());

注：肉眼要在网页上看不到的元素，这个元素不占有网页上的位置

:enabled 选择所有可用元素

alert($('input:enabled').val());

:disabled 选择所有不可用元素

alert($('input:disabled').val());

alert($('input[disabled=disabled]').val());

:checked 选择可有选中元素

alert($(':checked').val());

:selected 选取所有被选中的选项元素

alert($(':selected').val());

alert($('select option:selected').val());

alert($('select option[selected=selected]').val());

:focus 选择当前获取焦点的元素 1.6添加,基本上没用

7. 筛选,面对DOM操作

eq() 选择指定索引的元素

$('p').eq(0).css('background','red');

filter(表达式) 筛选指定表达式的元素

$('p').filter(':odd').css('background','red'); //可传选择器

first() 选择第一个元素

$('p').first().css('background','red');

last() 选择最后一个元素

$('p').last().css('background','red');

is() 检测是否元素返回布尔值

$('p').is('.first')

has() 保留包含特定后代的元素，去掉那些不含有指定后代的元素

$('p').has('.first')

not() 从匹配的元素集合中移除指定的元素

$('p').not('.last')

map() 将一组元素转换成其他数组

$('input').map(function(){alert($(this).val());}).get().join(',');

slice() 根据指定的下标范围，选取匹配的元素集合

对象级插件开发

;(function($){

$.fn.plugin = function(options){

var defaults = {

//各种参数，各种属性

};

var options = $.extend(defaults, options);

this.each(function(){

//实现功能

});

return this;

}

})(jQuery);

$(function(){

$('#tab').tab({

//参数

参数名1:'参数值1',

参数名2:'参数值2'

});

});