***数据类型***

JavaScript中有5种简单数据类型（也称为基本数据类型）：Undefined、Null、Boolean、Number和String。还有1种复杂数据类型--Object，Object本质上是由一组无序的名值对组成的。

function getRandomColor(){

return '#' +(Math.random()\*0x1000000<<0).toString(16);

}

***JQuery***

jquery能够方便的操作html documents，绑定和控制事件，实现动态效果，然后封装了ajax操作

逐个演示jquery的基本功能

换皮肤

jquery-ui封装效果

ajax提交，返回json

***JS进阶篇***

1:任何类对象都是function对象(又称构造器对象)

例:alert(typeof(Object));//function

alert(typeof(Array));//function

alert(typeof(Function));//function

2:任何类对象的实例都是object对象,Function对象的实例除外

例:alert(typeof(new Object()));//object

alert(typeof(new Array()));//object

alert(typeof(new Function()));//function

3:Object对象

Object对象比较好理解,它是所有类的基类.

例:Object.prototype.method=function(){alert(1)};

Function.method();//1

new Function().method();//1 由此可见Function对象与其他对象的异同

new Array().method();//1

4:Function对象和其他对象的区别

我们可以粗略的认为,Function对象就相似与JAVA中的Class对象,但他们之间依然有太多异同.

例:Function.prototype.method=function(){alert(1)};

Array.method();//1

new Array().method();//错误,因为Array属于Function对象，但是Array对象的实例属于Object

5:prototype类原型

prototype(原型)的概念在其他语言中几乎没有接触,但是它的存在意义重大.

一句话，prototype中定义的成员，是属于类的成员，this中定义的是属于实例的成员，优先当然是实例的成员，如果实例中的该成员不存在则找类原型中的成员

例:function Object(){

this.a="a";

}

Object.prototype.a="b";

alert(new Object().a);//a

例:function Object(){

}

Object.prototype.a="b";

alert(new Object().a);//b

6:继承以及抽象类

//Object类的静态方法,用于将source对象的属性拷贝到childClass对象中,将childClass对象返回

Object.extend=function(childClass,source){

for(pro in source){

childClass[pro]=source[pro];

}

return childClass;

}

//定义所有对象的extend方法,实质调用的是Object的静态方法,区别在于由哪个对象调用该方法,即将传入的对象的属性拷贝到该对象

Object.prototype.extend=function(object){

return Object.extend.apply(this,[this,object]);

}

//定义一个抽象类,该类不能被实例化(实际上也是可以实例化的,弹出提示框纯属忽悠,因为js没有能力阻止一个对象(方法)的实例化)

function AbstractClass(){

alert("该类是一个抽象方法,不能被实例化");

return;

}

//为抽象类定义一个init初始方法,其中调用了一个虚方法onload,该方法等待之类去实现

AbstractClass.prototype.init=function(){

this.onload();

}

//定义子类

function ChildClass(){}

让子类继承与抽象类,并实现其onload方法

ChildClass.prototype=AbstractClass.prototype.extend({onload:function(){alert("this is init method")}});

//调用

new ChildClass().init();

7:{}对象

该种对象的名称很多,其实它就是传说中的JsonObject,关键是要理解它.

例:var Class={

create:function(){

return function(){this.init.apply(this,[arguments])}

}

}

//以上定义了一个Class对象(注意是对象而不是类),它有一个create方法,该方法返回一个Function对象,并且在new该Function对象时会 自动调用init方法，因此之前必须定义该方法

//定义一个苹果类

var Apply=Class.create();

Apply.prototype={init:function(){alert("initializing...")}}

new Apply();//initializing...

8:事件机制

事件机制存在的问题就是无法给事件方法传入参数.

例:function exec\_onload(name){//定义一个方法

alert(name)}

window.onload=exec\_onload;//将该方法赋予onload事件

//加载时弹出undefind,因为你没有传入参数name,并且不能写成window.onload=exec\_onload('moumou'),解释器将解释为你将exec\_onload('moumou')该方法的调用结果赋予window.onload事件

那么,我们可以使用一种经典巧妙的方法来实现传参事件

例:function createFunction(obj,func){

if(!obj){

obj=window;

}

var args=new Array();

for(var i=2;i<arguments.length;i++){

args["push"](arguments[i]);

}

return function(){

obj[func].apply(obj,args);

}

}

function execOnload(name,time){

alert(name+"先生/小姐 您好,你在"+time+"加载了这个页面");

}

window.onload=createFunction(null,"execOnload","YangYu",new Date().toLocaleString());

//声明一个createFunction方法，其中要至少要传入2个参数(即你将调用obj对象的func方法),后面都为隐式参数.

这样就实现了其传参事件。

9:Cookie保存客户端数据

Cookie的用途非常广泛,它的优点是很多的,但是它也有缺点,而最大的缺点就是安全性和隐私性,因为客户端用户可以屏蔽浏览器Cookie.

但是它依然很有价值,有些电影网站记录用户的登陆,你下次再登陆,直接进入登陆后的界面,这就是Cookie的作用.

javascript对Cookie也有支持.

function CookieManager(){}

CookieManager.addCookie=function (name,value,expire,path){

var cookieStr=name+"="+escape(value);

if(expire&&expire>0){

var date = new Date();

date.setTime(date.getTime()+expire\*24\*3600\*1000);

cookieStr+=";expire="+date.toGMTString();

}

if(path){

cookieStr+=";path="+escape(path);

}

document.cookie=cookieStr;

}

10:popup对象

<script>

var popup=window.createPopup();

var tipBody=popup.document.body;

tipBody.style.margin=5;

tipBody.style.border='solid black 1px';

tipBody.style.backgroundColor='#e5f1f4';

function showTip(obj){

tipBody.innerHTML=obj.options[obj.selectedIndex].innerText;

popup.show(event.screenX,event.screenY,obj.options[obj.selectedIndex].innerText.length\*10,30);

}

</script>

11:apply方法

例子：function A(){

this.a="aaa";

alert(this.b);

}

function B(){

this.b="bbb";

}

var obj=new B()；

A.apply(obj);

表示obj对象调用A方法,结果为bbb

var Clazz=function(){

this.method=function(){

alert(1);

}

}

Clazz.prototype.method=function(){

alert(2);

}

当new Clazz()的时候。.

返回一个{}对象,js解析器,首先找到Clazz这个function对象,然后将它的prototype里面的属性全部copy到{},再将function自己的属性copy到{}.

这么一来,alert(2)的这个函数自然被alert(1)这个函数覆盖了。

正则表达式

<script>

var testStr="<姓名>冯元洪<姓名>冯元洪1<性别>男<民族>汉族<出生日期>19870516<住址>广州市萝岗区青年路102号<公民身份号码>441702198705160013<签发机关>广州市公安局萝岗分局<有效期限>2011092620210926";

document.writeln(testStr+"<br>");

var re =/<姓名>([^<]\*)<性别>([^<]\*)/g;

re.test(testStr);

alert(RegExp.$1+','+RegExp.$2);

</script>

**兼容FF和IE、Chrom、360等本地图片预览功能代码**

sender = input.file;

if( sender.files && sender.files[0] ){

objPreview.style.display = 'block';

objPreview.style.width = 'auto';

objPreview.style.height = 'auto';

var reader = new FileReader();

reader.onload = function(evt){

objPreview.src = evt.target.result;

targetPreviewFake.src = evt.target.result;

targetPreviewSizeFake.src = evt.target.result;

targetPreviewFake.setAttribute('IMG\_SRC',evt.target.result);

}

reader.readAsDataURL(sender.files[0]);

}else if( objPreviewFake.filters ){

sender.select();

var imgSrc = document.selection.createRange().text;

objPreviewSizeFake.src = imgSrc;

objPreviewFake.filters.item(

'DXImageTransform.Microsoft.AlphaImageLoader').src = imgSrc;

targetPreviewSizeFake.src = imgSrc;

targetPreviewFake.setAttribute('IMG\_SRC',imgSrc);

targetPreviewFake.filters.item(

'DXImageTransform.Microsoft.AlphaImageLoader').src = imgSrc;

}

***性能优化***

**避免全局查找**

在一个函数中会用到全局对象存储为局部变量来减少全局查找，因为访问局部变量的速度要比访问全局变量的速度更快些

function search() {

//当我要使用当前页面地址和主机域名

alert(window.location.href + window.location.host);

}

//最好的方式是如下这样 先用一个简单变量保存起来

function search() {

var location = window.location;

alert(location.href + location.host);

}

**定时器**

如果针对的是不断运行的代码，不应该使用setTimeout，而应该是用setInterval，因为setTimeout每一次都会初始化一个定时器，而setInterval只会在开始的时候初始化一个定时器

var timeoutTimes = 0;

function timeout() {

timeoutTimes++;

if (timeoutTimes < 10) {

setTimeout(timeout, 10);

}

}

timeout();

//可以替换为：

var intervalTimes = 0;

function interval() {

intervalTimes++;

if (intervalTimes >= 10) {

clearInterval(interv);

}

}

var interv = setInterval(interval, 10);

C/S client/server LOL .net framework

前端（B/S） browser/server

html+css+javascript+h5

应用服务器

java、php、C#、nodejs

数据库服务器

mysql（sql）

**字符串连接**

如果要连接多个字符串，应该少使用+=，如

//undefined string number boolean object function

s+=a;//s = s + a;

s+=b;

s+=c;

应该写成s+=a + b + c；

而如果是收集字符串，比如多次对同一个字符串进行+=操作的话，最好使用一个缓存，使用JavaScript数组来收集，最后使用join方法连接起来

var buf = [];

for (var i = 0; i < 100; i++) {

buf.push(i.toString());

}

var all = buf.join("");

**数字转换成字符串**

般最好用”" + 1来将数字转换成字符串，虽然看起来比较丑一点，但事实上这个效率是最高的，性能上来说：

(“” +) > String() > .toString() > new String()

**浮点数转换成整型**

很多人喜欢使用parseInt()，其实parseInt()是用于将字符串转换成数字，而不是浮点数和整型之间的转换，我们应该使用Math.floor()或者Math.round()

**各种类型转换**

var myVar = "3.14159",

str = "" + myVar, // to string

i\_int = ~ ~myVar, // to integer

f\_float = 1 \* myVar, // to float

b\_bool = !!myVar, /\* to boolean - any string with length

and any number except 0 are true \*/

array = [myVar]; // to array

如果定义了toString()方法来进行类型转换的话，推荐显式调用toString()，因为内部的操作在尝试所有可能性之后，会尝试对象的toString()方法尝试能否转化为String，所以直接调用这个方法效率会更高

**多个类型声明**

在JavaScript中所有变量都可以使用单个var语句来声明，这样就是组合在一起的语句，以减少整个脚本的执行时间，就如上面代码一样，上面代码格式也挺规范，让人一看就明了。

**插入迭代器**

如var name=values[i]; i++;前面两条语句可以写成var name=values[i++]

**使用直接量**

var aTest = new Array(); //替换为

var aTest = [];

var aTest = new Object; //替换为

var aTest = {};

var reg = new RegExp(); //替换为

var reg = /../;

//如果要创建具有一些特性的一般对象，也可以使用字面量，如下：

var oFruit = new O;

oFruit.color = "red";

oFruit.name = "apple";

//前面的代码可用对象字面量来改写成这样：

var oFruit = { color: "red", name: "apple" };

使用DocumentFragment优化多次append

一旦需要更新DOM,请考虑使用文档碎片来构建DOM结构，然后再将其添加到现存的文档中。

for (var i = 0; i < 1000; i++) {

var el = document.createElement('p');

el.innerHTML = i;

document.body.appendChild(el);

}

//可以替换为：

var frag = document.createDocumentFragment();

for (var i = 0; i < 1000; i++) {

var el = document.createElement('p');

el.innerHTML = i;

frag.appendChild(el);

}

document.body.appendChild(frag);

使用一次innerHTML赋值代替构建dom元素

对于大的DOM更改，使用innerHTML要比使用标准的DOM方法创建同样的DOM结构快得多。

var frag = document.createDocumentFragment();

for (var i = 0; i < 1000; i++) {

var el = document.createElement('p');

el.innerHTML = i;

frag.appendChild(el);

}

document.body.appendChild(frag);

//可以替换为：

var html = [];

for (var i = 0; i < 1000; i++) {

html.push('<p>' + i + '</p>');

}

document.body.innerHTML = html.join('');

通过模板元素clone，替代createElement

很多人喜欢在JavaScript中使用document.write来给页面生成内容。事实上这样的效率较低，如果需要直接插入HTML，可以找一个容器元素，比如指定一个div或者span，并设置他们的innerHTML来将自己的HTML代码插入到页面中。通常我们可能会使用字符串直接写HTML来创建节点，其实这样做，1无法保证代码的有效性2字符串操作效率低，所以应该是用document.createElement()方法，而如果文档中存在现成的样板节点，应该是用cloneNode()方法，因为使用createElement()方法之后，你需要设置多次元素的属性，使用cloneNode()则可以减少属性的设置次数--同样如果需要创建很多元素，应该先准备一个样板节点

var frag = document.createDocumentFragment();

for (var i = 0; i < 1000; i++) {

var el = document.createElement('p');

el.innerHTML = i;

frag.appendChild(el);

}

document.body.appendChild(frag);

//替换为：

var frag = document.createDocumentFragment();

var pEl = document.getElementsByTagName('p')[0];

for (var i = 0; i < 1000; i++) {

var el = pEl.cloneNode(false);

el.innerHTML = i;

frag.appendChild(el);

}

document.body.appendChild(frag);

使用firstChild和nextSibling代替childNodes遍历dom元素

var nodes = element.childNodes;

for (var i = 0, l = nodes.length; i < l; i++) {

var node = nodes[i];

//……

}

//可以替换为：

var node = element.firstChild;

while (node) {

//……

node = node.nextSibling;

***利用defineSetter实现hiJacking（IE不支持）***

***<html>***

***<head>***

***<script type="text/javascript">***

***Object.prototype.\_\_defineSetter\_\_('Money', function(obj) {***

***alert(obj);***

***});***

***function sendMsg(){***

***this.Money = 1000;***

***alert(1);***

***}***

***</script>***

***</head>***

***<body>***

***<button onclick = "sendMsg()">aaa</button>***

***</body>***

***</html>***

***function hovertreebrowser() {***

***var ua = window.navigator.userAgent,***

***ret = "";***

***if (/Firefox/g.test(ua)) {***

***ua = ua.split(" ");***

***ret = "Firefox|" + ua[ua.length - 1].split("/")[1];***

***} else if (/MSIE/g.test(ua)) {***

***ua = ua.split(";");***

***ret = "IE|" + ua[1].split(" ")[2];***

***}***

***else if (!!window.ActiveXObject || "ActiveXObject" in window) {***

***if (/rv:11.0/g.test(ua))***

***ret = "IE|11";***

***else if (/rv:10.0/g.test(ua))***

***ret = "IE|10";***

***else if (/rv:9.0/g.test(ua))***

***ret = "IE|9";***

***else {***

***ret = "HoverTree:IE|未知版本";***

***}***

***}***

***else if (/Opera/g.test(ua)) {***

***ua = ua.split(" ");***

***ret = "Opera|" + ua[ua.length - 1].split("/")[1];***

***} else if (/Chrome/g.test(ua)) {***

***ua = ua.split(" ");***

***ret = "Chrome|" + ua[ua.length - 2].split("/")[1];***

***} else if (/^apple\s+/i.test(navigator.vendor)) {***

***ua = ua.split(" ");***

***ret = "Safair|" + ua[ua.length - 2].split("/")[1];***

***} else {***

***ret = "HoverTree:未知浏览器";***

***}***

***return ret.split("|");***

***}***