**js十进制转二进制的函数:**

parseInt(数, 进制) var i=parseInt(100,2);

**前端页面有哪三层构成?**

**网页的结构层**（structural layer）由 HTML 或 XHTML 之类的标记语言负责创建。标签，也就是那些出现在尖括号里的单词，对网页内容的语义含义做出了描述，但这些标签不包含任何关于如何显示有关内容的信息。例如，P 标签表达了这样一种语义：“这是一个文本段。”

**网页的表示层**（presentation layer） 由 CSS 负责创建。 CSS 对“如何显示有关内容”的问题做出了回答。

**网页的行为层**（behavior layer）负责回答“内容应该如何对事件做出反应”这一问题。这是 Javascript 语言和 DOM 主宰的领域。

**jquery的$.extend和$.fn.extend作用及区别**

前者是用于添加jquery的类方法（静态），只能直接调用，如$.ajax，实例对象不能调用。

后者是添加jquery的成员函数，是给具体的对象调用的。

**csrf（跨站请求伪造）与xss（跨站脚本注入）网站攻击**

csrf:黑客诱使用户点进黑客网站的链接，然后利用用户用户的cookie信息，在用户不知情的情况下，向服务器发起类似转账等请求。

防御：使用浏览器的referer，判断请求来源的url。在请求中加入token验证。

XSS:在用户页面（如评论）注入JavaScript脚本，如搜集发送用户的cookie的信息，死循环弹窗等。

防御：Ajax中禁止跨域（url不是服务器url）请求，过滤和转义用户输入的文本。

**float：left与display:inline的用法区别**

**垂直对齐的实现方式：**

1. display:table-cell

vertical-align:middle

2．在需要对齐的元素后，加入一个空的span，设置它的height为100%，width为0.1px。然后分别在空span与需对齐的元素中，加入vertical-align: middle属性。目的是让行内元素的基线，相对于该元素所在行的基线垂直对齐。

**Jquery trigger事件：触发所选元素绑定的事件**

**事件捕获与事件冒泡**

**CSS BFC（Block formatting context）：**

作用：

1. 自适应两栏布局（多栏）
2. 清除内部浮动（了解另一种方法，通过clear实现方式！！）
3. 防止垂直margin重叠

**获取整个页面代码/内容**

var html =document.getElementsByTagName('html')[0]. innerHTML;

var text1=document.getElementsByTagName('html')[0].textContent;

var text2=document.getElementsByTagName('html')[0]. innerText;

**js on事件中的event对象（事件句柄）及可用属性：**

<script type="text/javascript">

$(document).ready(function(){

$(document).mousemove(function(e){

$("span").text(e.pageX + ", " + e.pageY);

});

});

</script>

<p>鼠标位于坐标： <span></span>.</p>

**Js prototype属性用法**

function Employee(name, salary){

this.name=name;

this.salary=salary;

}

Employee.prototype.getSalary=function getSalaryFunction(){

return this.salary;

}

Employee.prototype.addSalary=function addSalaryFunction(addition){

this.salary=this.salary+addition;

}

Prototype可以为js中类的对象，添加属性和方法（为一个特定类声明通用的变量或者函数）。因为对象中有一个内部属性\_\_proto\_\_，每次调用对象的prototype方法/属性时，会从这个内部属性找到。Prototype还可以用作克隆另外一个类的方法属性（继承？？？？？？），如：

Function baseClass{

/\* xxxxxxxx\*/

}

Function extendClass{}

extendClass.prototype = new baseClass();

若extend类有与base类同名的方法，将会取代。

**console.log()方法（window.console.log() ）**

window对象中会注册一个名为console的成员变量, 可用于在控制台打印出变量/对象…

与alert相比，它不会打断程序的执行，在调试的时候更值得推荐。同时能类似C语言传参数：console.log("%s is %d years old.", people, years);

**其它日志级别：**

console.log("Log level");  
console.debug("Debug level");  
console.info("Info level");  
console.warn("Warn level");  
console.error("Error level");

**html高亮标签: mark**

**js确认提示框：**

<button id="confirm">confirm</button>

$('#confirm').click(function(event){

var \_confirm = confirm("跳转页面？");

if (\_confirm == true){

event.returnValue = true; //return true;

window.location.href="../blog/blog.html";

}

else{

event.returnValue = false; //return false;

}

});

**Jsonp跨域请求（另一种方式：XMLHttpRequest level 2）**

<scipt>标签的src属性是跨域的，在url后加入参数，包括回调函数的名字。服务端动态生成js文件，里面包含一个对回调函数的实例调用。如：

客户端：

<script type="text/javascript">

function onCustomerLoaded(result, methodName) {

var html = '<ul>';

for (var i = 0; i < result.length; i++) {

html += '<li>' + result[i] + '</li>';

}

html += '</ul>';

document.getElementById('divCustomers').innerHTML = html;

}

</script>

<script type="text/javascript" src="http://www.yiwuku.com/myService.aspx?jsonp=onCustomerLoaded"></script>

服务端：

jsonpcallback({"Email":"zhww@outlook.com","Remark":"我来自遥远的东方"})

//函数的参数引入一个json对象

**Js闭包**

function a(){

var i=0;

function b(){

alert(++i);

}

return b;

}

var c = a();

c(); //1

c(); //2

闭包的作用就是在a执行完并返回后，闭包使得Javascript的垃圾回收机制GC不会收回a所占用的资源，因为a的内部函数b的执行需要依赖a中的变量。所以每次调用C()，i都会递增上去。

闭包作用：

1、保护函数内的变量安全。以最开始的例子为例，函数a中i只有函数b才能访问，而无法通过其他途径访问到，因此保护了i的安全性。

2、在内存中维持一个变量。依然如前例，由于闭包，函数a中i的一直存在于内存中，因此每次执行c()，都会给i自加1。

**Js回调函数**

function a(callback)

{

alert("我是parent函数a！");

alert("调用回调函数");

callback();

}

function b(){

alert("我是回调函数b");

}

function c(){

alert("我是回调函数c");

}

function test()

{

a(b);

a(c);

}

**Js深复制对象**

(function () {

//深复制对象方法

var cloneObj = function (obj) {

var newO = {};

if (obj instanceof Array) {

newO = [];

}

for (var key in obj) {

var val = obj[key];

newO[key] = typeof val === 'object' ? arguments.callee(val) : val;

}

return newO;

};

//测试

var a = {a:{a:123}},

b = cloneObj(a);

b.a.a = 456;

alert(a.a.a);

})();

I, 数组的深复制可以直接用slice与concat方法

II, js arguments.callee & caller的用法及区别

在函数内部，arguments.callee该属性是一个指针，指向拥有这个arguments对象的函数;

而函数对象的另一个属性：caller，这个属性保存着调用当前函数的函数的引用，如果是在全局作用域中调用当前函数，它的值为null。

function inner(){

alert(arguments.callee);//指向拥有这个arguments对象的函数，即inner()

alert(arguments.callee.caller);//这个属性保存着调用当前函数的函数的引用,即outer()

}

function outer(){

inner();

}

**getElementsByClassName实现**

function getElementsByClassName(node,classname) {

if (node.getElementsByClassName) { // use native implementation if available

return node.getElementsByClassName(classname);

} else {

return (function getElementsByClass(searchClass,node) {

if ( node == null )

node = document;

var classElements = [],

els = node.getElementsByTagName("\*"),

elsLen = els.length,

pattern = new RegExp("(^|\\s)"+searchClass+"(\\s|$)"), i, j;

for (i = 0, j = 0; i < elsLen; i++) {

if ( pattern.test(els[i].className) ) {

classElements[j] = els[i];

j++;

}

}

return classElements;

})(classname, node);

}

}

**else部分还能这样改写**

else{

var tags = document.getElementByTagName(tagName); //或者是”\*”

var returnArr = [];

for (var i=0 ; in<tags.length ;i++){  
 if (tags[i].className.match(‘searchClass’) != null)

returnArr.push(tags[i]);

}

return returnArr;

}

**JSON的parse(), stringfy() , eval()方法** (eval\_r() )

**js作用域链：**

js的作用域为函数作用域，没有类似C语言的块级作用域。

**setTimeout递归调用与setInterval的区别**

setTimeout( function() {

/\* 代码块... \*/

setTimeout( arguments.callee, 10 );

}, 10 );

setInterval( function() {

/\*代码块... \*/

}, 10 );

假设js线程中任务队列为空，若代码块的执行时间为0ms，则两者没有区别。若代码执行时间为一定的ms，setInterval将无视代码执行时间，按照间隔重复执行。setTimeout则需等代码块执行完，才再次调用自身的setTimeout，导致间隔时间增长。

**js 中call , apply ,bind方法**

这几个方法都可以显示制定函数的作用域，也就是设置函数体内this对象的值。call与apply都是对函数直接调用，而bind创建一个函数的实例，返回的是一个函数，需要加上()调用。this值绑定到 call/apply/ bind中参数。

var name = "The Window";

　　var object = {

　　　　name : "My Object",

kind : "closure" ,

　　　　getNameFunc : function(){

　　　　　　return function(){

　　　　　　　　return this.name;

　　　　　};

　　　　}

};

alert(object.getNameFunc()()); //The Window

alert(object.getNameFunc().call(object)); //My Object

alert(object.getNameFunc().bind(object)**()** ); //My Object