1、

var a="s1";  
 function fun1(){  
 var b="s2";  
 alert(a);//s1  
 }  
 fun1();  
 alert(a);//s1  
 alert(b);//报错  
  
 var a="s1";  
 function fun1(){  
 b="s2";//var省略之后 该变量自动变为全局变量  
 alert(a);//s1  
 }  
 fun1();  
 alert(a);//s1  
 alert(b);//s2  
  
 var a="s1";  
 function fun1(){  
 b="s2";  
 alert(a);  
 }  
 alert(a);//s1  
 alert(b);//报错  
 fun1();  
  
 var a="s1";  
 function fun1(){  
 a="s2";  
 alert(a);//s2  
 }  
 fun1();  
 alert(a);//s2  
  
 var a="s1";  
 function fun1(){  
 a="s2";  
 alert(a);//s2  
 }  
 alert(a);//s1  
 fun1();  
  
 var a="s1";  
 function fun1(){  
 var a="s2";  
 alert(a);//s2  
 }  
 fun1();  
 alert(a);//s1  
  
 var a="s1";  
 function fun1(){  
 var a="s2";  
 alert(a);//s2  
 }  
 alert(a);//s1  
 fun1();  
  
// 预编译  
 var a="s1";  
 function fun1(){  
 alert(a);//undefined  
 var a="s2";  
 alert(a);//s2  
 }  
 fun1();  
 alert(a);//s1

1. BOM:browser object Model 浏览器对象模型  
   window--/-navigator  
    /-frames  
    /-history  
    /-location  
    /-screen  
    /-document  
   一、window  
   1）window对象是客户端最高层对象  
   2）window对象代表当前打开的浏览器窗口  
   3）window是全局对象，它所在的属性与方法可以省略window window.alert("hello");  
   4）打开新窗口 open  
   // o1.close();//子窗口关闭  
   // window.close();//当前窗口关闭  
   var o1=window.open("test01.html","新窗口","width=300px,height=300");  
   function fun1(){

//火狐早期版本不支持innerHTML  
 //o1.document.getElementById("sel\_div").innerContent="你好，子窗口";   
 o1.document.getElementById("sel\_div").innerHTML="你好，子窗口";  
}  
if(o1==null){  
 alert("新窗口被拦截，请打开");  
}

function fun2(){  
 //子向父传  
 window.opener.document.getElementById("fa\_div").innerHTML="你好，父窗口";  
}

5）对话框  
消息提示框 window.alert()  
消息输入框 prompt()  
消息确认框 confirm()  
  
6）

setInterval() 间歇调用，指定时间间隔重复执行代码  
问题：有时后一个间歇调用，会在前一个间歇调用之前启动  
setTimeout() 超时调用，指定时间过后执行代码

var n=0;  
var timer;  
function fun1(){  
 n++;  
 if(n==5){  
 clearInterval(timer);  
 }  
 alert(n);  
}  
timer=setInterval(fun1,1000);  
  
var n=0;  
function fun1(){  
 n++;  
 if(n<5){  
 setTimeout(fun1,1000)  
 }  
 alert(n)  
}  
setTimeout(fun1,1000);

7）window.status 状态栏

var flag=0;  
function fun1(){  
 if(flag==0){  
 window.status="hello 校招";  
 flag=1;  
 }else{  
 window.status="";  
 flag=0;  
 }  
}

3、location  
function fun1(){  
 //location 既是window对象属性，也是document对象属性  
 //window.location与document.location引用的是同一个对象  
  
 //跳转页面  
 window.location.href="test01.html";  
 window.location.assign("test01.html");  
 window.location="test01.html";  
 window.documentIsHTML.location="test01.html";  
  
 //获取url属性  
 //http://localhost:63342/jqueryProject\_43/7.19/test21.html  
 alert(location.host); //localhost:63342 服务器名称以及端口  
 alert(location.hostname); //localhost 服务器名称  
 alert(location.port); //63342 端口  
 alert(location.protocol); //http 使用协议  
 alert(location.pathname); //jqueryProject\_43/7.19/test21.html 路径  
 alert(location.search); //？uname=aaa&pwd=123  
  
 //页面的重新加载  
 location.reload();  
  
  
 location.replace("test01.html");//不会再历史记录中生成新的记录  
 window.location.href="test01.html";//会再历史记录中生成新的记录  
  
}

<a href="test22.html">test22</a>  
<form action="" onsubmit="return fun1()">  
 <input type="text" name="uname"><br>  
 <input type="text" name="pwd"><br>  
 <button onclick="fun1()">确定</button>  
</form>

1. function fun1(){  
    //history保存用户上网的历史记录

history.back();//返回前一页  
 history.forward();//下一页  
 history.go(0);//当前页  
 history.go(-1);//前一页  
 history.go(-2);//前两页  
 history.go(1);//下一页  
 history.go(2);//下两页  
 /\*  
 \* screen 客户端显示信息  
 \* 屏幕像素宽高：  
 \* alert(screen.width);  
 \* alert(screen.height);  
 \* 浏览器与显示器边界的距离：  
 \* alert(window.screenTop);  
 \* alert(window.screenLeft);  
 \* 浏览器的宽高改变为：  
 \* window.resizeTo(500,600);  
 \* 浏览器的宽高每次改变为：  
 \* window.resizeBy(100,100);  
 \* \*/  
}

<button onclick="fun1()">确定</button>

5、

1）navigator 浏览器内核  
alert(navigator.appCodeName);//Mozilla 浏览器代码名称  
alert(navigator.appName);//Netscape 浏览器名称  
alert(navigator.appVersion);//5.0(window NT 6.1 ;WOW64..........)//火狐：5.0(window) 浏览器版本号  
alert(navigator.platform);//Win32 操作系统平台  
  
2）操作系统平台：  
win---window系统  
mac---苹果系统，macintonsh  
linux---linux 系统  
xll--unix系统  
  
3）document.write(navigator.userAgent);//用户代理 Mozilla(window NT 6.1 ;WOW64..........)  
ie: Mozilla/5.0 (compatible; MSIE 9.0; qdesk 2.4.1266.203; Windows NT 6.1; WOW64; Trident/6.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E)  
谷歌：Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36  
火狐：Mozilla/5.0 (Windows NT 6.1; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0//浏览器名称/版本号  
欧朋：Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.106 Safari/537.36 OPR/38.0.2220.41

4）navigator.mimeTypes//获取浏览器资源媒体类型，返回数组  
var i=3;  
document.write(navigator.mimeTypes[i].type+"<br>");//媒体类型  
document.write(navigator.mimeTypes[i].description+"<br>");//描述  
document.write(navigator.mimeTypes[i].suffixes+"<br>");//扩展名  
document.write(navigator.mimeTypes[i].enabledPlugin.name+"<br>");//对该类型配置好的plugin对象的引用

6、  
\* 历史：  
\* 1、1993 美国NCSA国家计算机中心发布 Mosaic  
\* 2、Netscape 介入浏览器领域 ，Mozilla (Mosaic killer 的简写)  
\* Mozilla/版本号[语言][平台；加密]......  
\* 3、微软 发布第一个浏览器 internet Explorer 3  
\* Mozilla/2.0(compatible...........)  
\*  
\* 浏览器内核：  
\* 也叫排版引擎或渲染引擎，主要功能是提取网页的内容（html\xml\图像等）  
\* 整理信息--加入css，js，计算网页的显示方式，输出到显示器或打印机上  
\*  
\* 1、Trident--ie内核  
\* html引擎：Trident  
\* js引擎：低版本：JScript  
\* 高版本：chakra 查克拉  
\*  
\* 2、Gecko --火狐内核  
\* html引擎：Gecko  
\* js引擎：低版本：SpiderMonkey  
\* 高版本: JagerMonkey  
\*  
\* 3、webkit--苹果开源内核，基于该内核的浏览器主要用于移动safari chrome  
\* html引擎：KHTML  
\* js引擎：safari: 低版本：javascript core  
\* 高版本: Nitro  
\* chrome: v8  
\*  
\* 4、presto--opera早期内核  
\* 5、blink-- 是webkit分支，早期opera/chrome内核  
\*  
\* 6、konqueror--只能在Linux使用的内核  
\* \*/

1. /\*  
   \* DOM：文档对象模型 document object Model  
   \*  
   \* 每个节点都有一个childNodes集合属性  
   \* 节点属性：  
   \* node.nodeName 获取节点名称  
   \* node.nodeType 节点类型  
   \* 元素节点 返回1  
   \* 文本节点 返回3  
   \* node.nodeValue 节点值  
   \* 元素节点 返回 null  
   \* 文本节点 返回文本内容  
   \*

\* 判断指定节点是否有子节点：  
\* node.hasChildNodes true有 false无

\*

\* 删除子节点：  
\* p.removeChild(子节点)  
\*   
\* \*/

window.onload=function(){  
 alert(document.childNodes[1]);//object DocumentType  
 alert(document.childNodes[1].childNodes[2].childNodes[0].childNodes[0]);  
}

8、DOM:  
\* \*/  
window.onload=function(){  
 var v=document.getElementById("s1").parentNode.nodeName;//DIV  
  
 var v=document.getElementById("s1").nextSibling;//object Text  
 var v=document.getElementById("s1").nextElementSibling;//object HTNLSpanElement忽略元素之间的文本节点  
 var v=document.getElementById("s1").previousSibling;//object Text  
 var v=document.getElementById("s1").previousElementSibling;//object HTNLSpanElement忽略元素之间的文本节点  
  
 var v=document.getElementById("s1").parentNode.firstChild;//object Text  
 var v=document.getElementById("s1").parentNode.firstElementChild;  
 var v=document.getElementById("s1").parentNode.lastChild;//object Text  
 var v=document.getElementById("s1").parentNode.lastElementChild;  
  
 var v=document.getElementById("s1").parentNode.children[2];  
 alert(v);  
 childNodes 获取所有的节点  
 children 只获取所有的元素节点  
  
 alert(document.body.children[0].offsetWidth);  
 alert(document.body.children[0].offsetHeight);  
  
 var e1=document.body.firstChild;  
 if(e1.nodeType==1){//元素节点  
 alert(e1);  
 }else if(e1.nodeType==3){//文本节点  
 var reg=/^\S+$/g;  
 var r1=reg.exec(e1.nodeValue);//文本节点 返回文本内容  
 if(r1==null){  
 alert(e1.nextSibling);  
 }else{  
 alert(e1.nodeValue);  
 }  
 }  
  
  
 function first\_child(e){  
 var e1=e.firstChild;  
 if(e1 && e1.nodeType==3){//文本节点  
 var reg=/\S/g;  
 var r1=reg.exec(e1.nodeValue);//文本节点 返回文本内容  
 if(r1==null){  
 e1=e1.nextSibling;  
 }  
 }  
 return e1;  
 }  
 var v1=first\_child(document.body);  
 if(v1.nodeType==3){  
 alert(v1.nodeValue);  
 }else if(v1.nodeType==1){  
 alert(v1.nodeName);  
 }  
  
 /\*  
 \* 原型扩展  
 \* DOM文档中的所有元素都继承于HTMLElement类（HTMLElement上级是Element上级是node）  
 \* 在HTMLElement类的原型对象上定义自己的方法，DOM文档中的所有元素都可访问  
 \* \*/  
 //调用原型封装  
 var v1=document.body.first\_child();  
 if(v1.nodeType==3){  
 alert(v1.nodeValue);  
 }else if(v1.nodeType==1){  
 alert(v1.nodeName);  
 }  
}

HTMLElement.prototype.first\_child=function(){  
 var e1=this.firstChild;  
 if(e1 && e1.nodeType==3){//文本节点  
 var reg=/\S/g;  
 var r1=reg.exec(e1.nodeValue);//文本节点 返回文本内容  
 if(r1==null){  
 e1=e1.nextSibling;  
 }  
 }  
 return e1;  
};

9、

window.onload=function(){

/\*  
\* 元素查找  
\* 1、getElementById("id01") 按id查找  
\* 2、getElementByTagName("div") 按标签查找，返回数组  
\* 3、document.getElementsByClassName("d1");按class属性查找，返回数组  
\* 4、getElementsByName("r1") 按name查找，返回数组  
\*  
\* 属性设置:  
\* setAttribute（属性名，属性值） 设置属性  
\* this.className="c1"  
\* this.title="c1"  
\*  
\* getAttribute（属性名） 获取属性  
\*  
\* removeAttribute（属性名） 删除属性  
\* his.className=null  
\* \*/

document.getElementById("id01");  
  
var v1=document.getElementsByTagName("div");  
var v2=document.getElementsByClassName("d1");  
v2[1].style.color="red";  
  
var v3=document.getElementById("id02").getElementsByClassName("d1");  
  
var v4=document.getElementsByName("r1");  
for(var i=0;i<v4.length;i++){  
 v4[i].style.backgroundColor="red";  
}  
  
document.getElementById("id01").setAttribute("title","hello.....");  
alert(document.getElementById("id01").getAttribute("class"));

//鼠标悬停添加属性，鼠标移出去掉属性  
 document.getElementById("id01").onmouseover=function(){  
// this.setAttribute("class","c1");//<div class="c1">  
// this.className="c1";//设置属性  
// this.title="c1";  
// alert(this.id);//获取属性  
 };  
 document.getElementById("id01").onmouseout=function(){  
// this.removeAttribute("class");  
// this.className=null;//移除属性  
 };

}

10、DOM元素创建

function fun1(){  
 //创建元素节点  
 var vli=document.createElement("li");//<li></li>  
 //创建文本节点  
 var tli=document.createTextNode("hello");//hello  
 //1  
 //将文本节点加入到元素节点中  
 vli.appendChild(tli);//<li>hello</li>  
 document.getElementById("u01").appendChild(vli);  
 //2  
 vli.innerText="world";  
 vli.title="aaaaaa";  
 document.getElementById("u01").appendChild(vli);  
  
 var vdiv=document.createElement("div");  
 var text=document.createTextNode("hello");  
 vdiv.appendChild(text);  
 document.body.appendChild(vdiv);  
  
 var vli=document.createElement("li");  
 var tli=document.createTextNode("hello");  
 vli.appendChild(tli);  
 //appendChild 将当前节点插入到所有节点之后 --追加  
 document.getElementById("u01").appendChild(vli);  
 //insertBefore(a1,a2) 将a1插入到a2之前，a1--新节点 a2--旧节点  
 var vold=document.getElementById("u01").firstElementChild;  
 document.getElementById("u01").insertBefore(vli,vold)  
}  
function fun2(){  
 var vli=document.createElement("li");  
 var tli=document.createTextNode("world");  
 vli.appendChild(tli);  
 var vold=document.getElementById("u01").firstElementChild;  
 //替换--replaceChild(newnode,oldnode);  
 document.getElementById("u01").replaceChild(vli,vold)  
  
 //克隆（负值）节点  
 //cloneNode(true) true 复制的节点将包含所有子节点的内容  
 // false 复制的节点只包含指定对象本身，不包含任何子节点  
 var c1=document.getElementById("u01").cloneNode(true);  
 document.body.appendChild(c1);  
}  
function fun3(){  
 //删除  
 var v1=document.getElementById("u01").firstElementChild  
 document.getElementById("u01").removeChild(v1);  
}  
  
/\*  
\* 文本节点操作  
\* textnode.appendData(text)向文本节点中追加text内容  
\* textnode.deleteData(offset,count) 从offset指定位置开始删除count个字符  
\* textnode.insertData(offset,text) 从offset指定位置插入text  
\* textnode.replaceData(offset,count,text) 用text替换从指定的位置开始到offset+count为止处的文本  
\* textnode.splitText(offset)从offset指定位置将文本节点分成两个文本节点  
\* textnode.subStringData(offset,count) 提取从offset指定位置开始到offset+count为止处的文本  
\* \*/  
function fun4(){  
 var text=document.getElementById("u01").firstChild;  
 text.appendData("hello");  
 text.deleteData(9,2);  
 text.insertData(9,"text");  
 text.replaceData(9,2,"zz");  
 text.splitText(9);  
 var v1=text.subStringData(9,3);  
 alert(v1);  
}  
function fun5(){  
 var li=document.getElementById("u01").getElementsByTagName("li");  
 for(var i=0;i<li.length;i++){  
 li[i].style.cssFloat="left";  
 li[i].style.color="red";  
 }  
  
 //DOM2样式对象 cssText：直接写行内样式  
 var li=document.getElementById("u01").getElementsByTagName("li")[0];  
 li.style.cssText="color:red;font-size:30px;width:100px;";  
 for(var i=0;i<li.style.length;i++){  
 //li.style[i]等价于li.style.item(i) 获取属性名  
 alert(li.style[i]);//color font-size width  
 alert(li.style.item(i));//color font-size width  
  
 li.style.getPropertyValue(s); //获取属性值  
 var s=li.style[i];  
 alert(li.style.getPropertyValue(s));//red 30px 100px  
 }  
  
 //获取属性值 也可以不在循环内  
 alert(li.style.getPropertyValue("color"));  
}  
function fun6(){  
 var li=document.getElementById("u01").getElementsByTagName("li")[0];  
 //删除样式属性  
 li.style.removeProperty("color");  
 //设置样式属性  
 li.style.setProperty("background-color","green");  
}  
  
  
//js获取外部css属性（只能获取、不能设置）  
function fun7(){  
 var u1=document.getElementById("u01");  
 var v1=document.defaultView.getComputedStyle(u1,null);//高级浏览器  
 alert(v1.width);  
  
 var v1=u1.currentStyle//ie低版本  
 alert(v1.width);  
  
 var v1;  
 if(window.getComputedStyle){  
 v1=document.defaultView.getComputedStyle(u1,null);  
 }else{  
 v1=u1.currentStyle;  
 }  
 alert(v1.width);//200px  
 var v2=parseInt(v1.width);  
 u1.style.width=v2\*2+"px";  
}  
  
  
//操作样式表  
//只针对 link 或 <style> 方式定义的样式表  
function fun8(){  
 for(var i=0;i<document.styleSheets.length;i++){  
 alert(document.styleSheets[i].href);  
 }  
  
 //css规则  
 var link=document.styleSheets[0];  
 var r1=link.cssRules || link.rules;//兼容，取得规则列表  
  
 alert(r1[0].selectorText);//.d1  
 alert(r1[0].style.cssText);//color:red;font-size: 30px;  
 alert(r1[0].style.color);//red  
  
 alert(r1[1].selectorText);//.d2  
 alert(r1[1].style.cssText);//font-size: 40px  
 alert(r1[1].style.color);//空  
  
 //查找规则  
 r1[1].style.color  
  
 //修改规则  
 r1[0].style.color="green";  
  
 //插入规则  
 if(link.insertRule){  
 link.insertRule("body{background:red;}",0);//高级浏览器 0表示插入规则的位置  
 }else{  
 link.addRule("body","background:red",0);//ie低版本  
 }  
  
 //删除规则  
 link.deleteRule(1);//高级浏览器 1表示删除规则的位置  
 link.removeRule(1);//ie低版本  
}

<link href="../css/css1.css" rel="stylesheet">  
<link href="../css/css2.css" rel="stylesheet">  
<style>  
 #u01{  
 width: 200px;  
 height: 200px;  
 background: red;  
 }  
</style>

<div class="d1">hello</div>  
<div class="d2">world</div>  
<ul id="u01">  
 <!--abcdefgh-->  
 <li>aaaaaa</li>  
 <li>bbbbbb</li>  
 <li>cccccc</li>  
</ul>  
<button onclick="fun8()">操作样式表</button>  
<button onclick="fun7()">js获取外部css属性</button>  
<button onclick="fun6()">删除样式</button>  
<button onclick="fun5()">样式</button>  
<button onclick="fun4()">追加文本</button>  
<button onclick="fun1()">添加</button>  
<button onclick="fun2()">替换</button>  
<button onclick="fun3()">删除</button>

11、文档信息：document

alert(document.title);//文档标题 index...  
alert(document.url);//域名地址 undefined  
alert(document.domain);//域名 localhost  
alert(document.referrer);//取得来源网页的url地址 空  
  
//特殊集合（返回数组）  
document.all  
document.images  
document.forms  
document.links //返回所有带href属性的a属性  
document.anchors //返回所有不带href属性的a属性  
  
var a1=document.all;  
for(var i=0;i<a1.length;i++){  
 alert(a1[i])  
}  
  
window.onload=function(){  
 var a2=document.images;  
 for(var i=0;i<a2.length;i++){  
 a2[i].style.border="10px solid red";  
 }  
  
 var a2=document.links;  
 var a2=document.anchors;  
 for(var i=0;i<a2.length;i++){  
 a2[i].style.border="10px solid red";  
 }  
}

12、html5的DOM扩展  
1）getElementsByClassName(可包含一个或多个类名)？？？  
2）classList  
 classList.remove("a3") 移除class属性  
 classList.add("a4") 增加class属性  
 classList.toggle("a3") class属性有则移除，无则增加  
 classList.contains("a3") 判断class列表中是否存在指定的值 true false  
3）  
 querySelector(复合选择器) 返回第一个匹配元素，无--返回null  
 querySelectorAll() 返回所有匹配元素（返回数组）  
4）自定义数据属性  
html5规定可为元素添加非标准的属性，前缀data-  
用dataset来访问  
5）插入标记  
insertAdjacentHTML(插入位置,要插入的文本)  
插入位置：beforebegin在当前元素前插入  
 afterbegin在当前元素内部前面插入（做第一个子元素）  
 beforeend在当前元素内部后面插入（做最后一个子元素）  
 afterend在当前元素后插入  
\* \*/  
function fun1(){  
 //1  
 var v1=document.getElementById("id01").getElementsByClassName(".s1 .s2");  
 for(var i=0;i<v1.length;i++){  
 v1[i].style.color="red";  
 }  
  
 //2  
 var v2=document.getElementById("id02").classList.remove("a3");  
 var v2=document.getElementById("id02").classList.add("a4");  
 var v2=document.getElementById("id02").classList.toggle("a4");  
 var v2=document.getElementById("id02").classList.contains("a3");  
 alert(v2)//true  
  
 //3  
 document.body.querySelector("#id01 div").style.color="red";  
 document.getElementById("id01").querySelector("div span").style.color="red";  
  
 var v1=document.getElementById("id01").querySelectorAll("span");  
 for(var i=0;i<v1.length;i++){  
 v1[i].style.color="red";  
 }  
  
 //4  
 var v1=document.getElementById("id03");  
 //获取自定义属性  
 alert(v1.dataset.mypid);//123  
 alert(v1.dataset.myname);//zs  
 //设置自定义属性  
 var v2=v1.dataset.myname="ls";  
 alert(v2);//ls  
  
 innerText//覆盖元素中的文本  
 innerHTML  
 outerText//覆盖元素及文本  
 outerHTML  
 var v2=document.getElementById("id04");  
 v2.innerText="aaa";  
 v2.innerHTML="bbb";  
 v2.outerText="ccc";  
 v2.outerHTML="ddd";  
  
 //5  
 //beforebegin在当前元素前插入  
 document.getElementById("id04").insertAdjacentHTML("beforebegin","<p>kkkk</p>");  
 //afterbegin在当前元素内部前面插入  
 document.getElementById("id04").insertAdjacentHTML("afterbegin","<p>kkkk</p>");  
 //beforeend在当前元素内部后面插入  
 document.getElementById("id04").insertAdjacentHTML("beforeend","<p>kkkk</p>");  
 //afterend在当前元素后插入  
 document.getElementById("id04").insertAdjacentHTML("afterend","<p>kkkk</p>");  
}

<style>  
 .a1{  
 color:red;  
 }  
 .a2{  
 background: blue;  
 }  
 .a3{  
 font-size: 30px;  
 }  
 .a4{  
 margin-left: 30px;  
 }  
</style>

<!--4-->  
<div id="id03" data-mypid="123" data-myname="zs"></div>  
<div id="id04" style="color:red;">222eeeesss33333</div>  
<!--3-->  
<div id="id01">  
 <div>ssss  
 <span>mmmm111</span>  
 <span>mmmmm222</span>  
 </div>  
 <span>dddd111</span>  
 <span>dddd222</span>  
</div>  
<!--2-->  
<div class="a1 a2 a3" id="id02">hello</div>  
<!--1-->  
<div id="id01">  
 <div class="s1">ssss</div>  
 <span class="s2">dddd</span>  
</div>  
<a name="r2" class="s1">qqqqq</a>  
<button onclick="fun1()">确定</button>  
  
<a href="">ssss</a>  
<a href="">wwwww</a>  
<a name="r1">eeee</a>  
<a name="r2">qqqqq</a>  
  
<img src="../images/zy01.jpg">  
<img src="../images/zy02.jpg">

13、html DOM 加载过程  
1）解析html结构  
2）加载外部脚本、样式文件（js css）  
3）解析并执行脚本代码  
4）构造html dom 模型  
5）加载图片等外部文件  
6）页面加载完毕

1. 事件处理  
   \* 1、事件的传播过程  
   \* div(hello)->div->body->html 冒泡事件流  
   \* html->body->div->div(hello) 捕获事件流  
   \* 冒泡事件流：  
   \* 指事件由下至上传播，传播过程像气泡一样，不断上升到顶端，所有叫冒泡事件流  
   \* 捕获事件流：  
   \* 捕获事件与冒泡事件流相反，由上至下传播，由开发人员制定  
   \* 1）默认情况，事件使用冒泡事件流  
   \* 2）ie8及以下版本只支持冒泡事件流  
   \* 3）ie9及以上版本、现在高级浏览器同时支持以上两种事件流  
   \* 2、事件模型  
   \* dom0->dom1->dom2  
   \* （1）dom0事件模型  
   \* <div onclick="fun1()">hello</div>  
   \* document.getElementById("id01").onmouseover=function(){....};  
   \* 【1】该方式不支持对同一元素的同一事件注册多个事件监听  
   \* 【2】该方式只支持冒泡事件传播  
   \*  
   \* （2）dom2事件模型（ie8及其以下版本不支持）  
   \* 1）addEventListener(arg1,arg2,arg3)  
   \* arg1:要绑定的事件类型  
   \* arg2:回调方式  
   \* arg3:事件的传播方式  
   \* true:事件捕获  
   \* false:事件冒泡  
   \* 2）removeEventListener（参数同上）  
   \* 移出事件可释放系统资源  
   \* 移出事件的参数一定要与绑定事件参数一致  
   \* （3）IE事件模型 ie8以下 有多个输出时从后到前  
   \* attachEvent(arg1,arg2)  
   \* arg1:要绑定事件的类型  
   \* arg2:回调函数  
   \* detachEvent(参数同上) 移出事件  
   window.onload=function(){  
    document.getElementById("id01").attachEvent("onclick",function(){  
    alert("aaaa");//后  
    });  
    document.getElementById("id01").attachEvent("onclick",function(){  
    alert("bbbb");//先  
    });  
     
    function fun1(){  
    alert("bbbb");  
    }  
    document.getElementById("id01").attachEvent("onclick",fun1);  
    function fun2(){  
    document.getElementById("id01").detachEvent("onclick",fun1);  
    }  
   };

<div id="id01">hello</div>  
<button onclick="fun2()">确定</button>

\* 3、键盘事件  
\* onkeydown 键盘按下某个键时触发，不区分大小写（字母键都以大写方式显示）  
\* onkeyup 释放某个键时触发，不区分大小写（字母键都以大写方式显示）  
\* onkeypress 按下某个键并释放时触发，区分大小写（字母键都以大、小写方式显示）  
\* \*/

15、

window.onload=function(){  
 document.getElementById("id01").onclick=function(){  
 alert("aaaa");  
 };  
 document.getElementById("id01").onclick=function(){  
 alert("bbbb");//输出  
 };  
  
 function f1(){  
 alert("aaaa");  
 }  
 document.getElementById("id01").addEventListener("click",f1,false);  
 document.getElementById("id01").addEventListener("click",function(){  
 alert("bbbb");  
 },false);  
  
 //移除事件监听  
 document.getElementById("but01").onclick=function(){  
 document.getElementById("id01").removeEventListener("click",f1,false);  
 };  
  
 //键盘事件  
 //onkeyup onkeydown onkeypress  
 document.getElementById("inp01").onkeyup=function(e){  
 alert(e.keyCode);  
 }  
  
 document.addEventListener("keydown",function(e){  
 /\*  
 \* window.event ie低版本  
 \* 1、当事件发生时，事件消息会以event对象形式，在文档结构中传播  
 \* 2、ie低版本：当事件发生时，会将事件对象存储在window对象中  
 \* \*/  
 e=event||window.event;//兼容性处理  
  
 e.ctrlKey  
 e.altKey  
 e.shiftKey  
 if(e.shiftKey && e.keyCode==65){  
 alert("按到此键");  
 }  
  
 //歌曲播放和暂停  
 var v1=document.getElementById("id01");  
 if(e.keyCode==39){  
 v1.style.left=v1.offsetLeft+50+"px";  
 var au=document.getElementById("au01");  
 if(au.pause()){  
 au.play();  
 }else{  
 au.pause();  
 }  
 }  
 },false)  
};

<div id="id01" style="width:100px;height:100px;background: red;position: absolute;"></div>  
<audio src="123.mp3" id="au01"></audio>  
  
<input type="text" id="inp01">  
  
<div id="id01">hello</div>  
<button id="but01">确定</button>  
  
<div onclick="alert('fun111111');">hello</div>  
  
this.value当前元素值  
<input type="text" id="inp01" onblur="fun2(this.value)"><!--失去焦点-->  
  
<input type="text" id="inp01">  
<button onclick="fun2(document.getElementById('inp01').value)">确定</button>

16、

//等待文档加载完毕之后就执行，不包括图片等(DOM)  
//$(function(){});  
//等待文档、图片等加载完毕之后执行  
//window.onload=function(){}  
  
window.onload=function(){  
 /\*  
 \* mouseover:鼠标指针穿过被选元素与子元素触发 支持冒泡  
 \* mouseenter:鼠标指针穿过被选元素触发 不支持冒泡（先显示父级，再显示子级）  
 \*  
 \* mouseout:鼠标指针无论离开子元素还是被选元素都触发 支持冒泡  
 \* mouseleave:鼠标指针只离开被选元素时触发 不支持冒泡  
 \* \*/  
  
 var v1=document.getElementById("d01");  
 v1.onmouseover=function(){  
 alert("hello");  
 };  
 v1.onmouseenter=function(){  
 alert("hello11111");  
 };  
 v1.onmouseout=function(){  
 alert("hello22222");  
 };  
 v1.onmouseleave=function(){  
 alert("hell333333");  
 }  
  
 //不支持冒泡事件  
 v1.focus;  
 v1.blur;  
 v1.load;//加载  
 v1.unload;//卸载  
 v1.mouseenter  
 v1.mouseleave  
 };  
  
function f1(){  
 alert("1111");  
}  
function f2(){  
 alert("2222");  
}  
  
//监听dom加载完成事件  
//方法一：先弹出，后出图片  
//DOMContentLoaded：形成完整的DOM树之后就会触发，不理会图片、文件等其他多媒体是否已经加载完毕，该事件会在onload之前触发。  
document.addEventListener('DOMContentLoaded',function(){  
 alert("hello");  
},false);  
//方法二：  
function fun1(){  
 //检测一些函数和元素是否已经可访问  
 if(document && document.getElementByTagName && document.getElementById && document.body){  
 clearInterval(timer);  
 alert("hello1111");  
 }  
}  
var timer=setInterval(fun1,10);

<body onload="alert('aaa')" onunload="alert('bbb')">  
<div id="d01" onmouseenter="f2()">  
 <div id="d02" onmouseenter="f1()"></div>  
</div>  
<img src="../images/zy01.jpg">  
</body>

17、

window.onload=function(){  
 //鼠标事件  
 //onclick ondbclick onmouseover onmouseout onmousemove onmousedown onmouseup  
  
 //当点击事件发生时的响应顺序  
 //onmousedown->onmouseup->onclick->ondbclick onmouseover -> onmousemove -> onmouseout  
  
 var v1=document.getElementById("id01");  
 v1.onclick=function(){  
 alert("onclick");  
 };  
 v1.ondblclick=function(){  
 alert("ondbclick");  
 };  
 v1.onmouseover=function(){  
 alert("onmouseover");  
 };  
 v1.onmouseout=function(){  
 alert("onmouseout");  
 };  
 v1.onmousemove=function(){  
 alert("onmousemove");  
 };  
 v1.onmousedown=function(){  
 alert("onmousedown");  
 };  
 v1.onmouseup=function(){  
 alert("onmouseup");  
 };  
  
 //文字随鼠标移动  
 document.getElementById("show").innerText="x:"+e.pageX+" y:"+e.pageY;  
 var v1=document.getElementById("id01");  
 document.onmousemove=function(event){  
 var e=event||window.event;  
 //距离页面的距离 e.pageX e.pageY  
 v1.style.position="absolute";  
 v1.style.left=e.pageX+10+"px";  
 v1.style.top=e.pageY+10+"px";  
 }  
}

<div id="show"></div>  
<div id="id01">hello</div>

18、/\*  
\* 阻止浏览器默认动作：  
\* 1、return false; 表示退出执行，其之后所有代码不会被执行 在行内阻止默认行为  
\* 2、e.preventDefault(); 阻止浏览器默认动作,其之后所有代码会被执行  
\* e.returnValue=false; ie低版本  
\*  
\* 阻止冒泡传播事件：  
\* e.stopPropagation；  
\* e.cancelBubble=true;ie低版本  
\* \*/  
  
 window.onload=function(){  
 document.getElementById("a1").onclick=function(event){  
 var e=event||window.event;  
 alert("hello");  
 e.preventDefault();  
 e.returnValue=false;//ie低版本  
 alert("world");  
 return false;  
 alert("!!!!!!");  
 }  
};  
  
function fun1(event){  
 alert("11111");  
 var e=event||window.event;  
 //兼容性 阻止冒泡传播事件  
 if(e.stopPropagation){  
 e.stopPropagation();  
 }else{  
 e.cancelBubble=true;//ie低版本  
 }  
}  
function fun2(){  
 alert("22222");  
}  
  
//屏蔽右键  
document.oncontextmenu=function(){  
 return false;  
};  
//禁止粘贴  
document.onpaste=function(){  
 return false;  
};  
//禁止拷贝  
document.oncopy=function(){  
 return false;  
};  
//禁止剪切  
document.oncut=function(){  
 return false;  
};  
//屏蔽滚轮  
document.onmousewheel=function(){  
 return false;  
};  
//禁止选择  
document.onselectstart=function(){  
 return false;  
};  
  
window.onload=function(){  
 document.getElementById("id01").onclick=function(event){  
 //事件对象  
 var e=event||window.event;  
  
 //事件源  
 //火狐 e.target ers.textcontent  
 var esrc=e.srcElement || e.target;  
 alert(esrc.innerText);//esrc.textcontent  
 alert(esrc.tagName);//标签名  
  
 //获取当前事件的相邻节点  
 //其他浏览器 e.relatedTarget  
 //IE e.fromElement  
 var rsrc= e.relatedTarget || e.fromElement;  
 alert(rsrc.innerText);  
 alert(rsrc.tagName);  
 }  
};

19、封装

window.onload=function(){  
 //跨浏览（兼容）器事件处理程序  
 var EventUtil={  
 //元素 事件类型 执行函数 事件传播方式  
 addHandle:function(element,type,handle,capture){  
 if(element.addEventListener){//dom2事件模型  
 element.addEventListener(type,handle,capture);  
 }else if(element.attachEvent){//ie事件模型  
 element.attachEvent("on"+type,handle);  
 }else{//dom0事件模型  
 element["on"+type]=handle;  
 }  
 },  
 removeHandle:function(element,type,handle,capture){  
 if(element.removeEventListener){//dom2事件模型  
 element.removeEventListener(type,handle,capture);

}else if(element.detachEvent){//ie事件模型  
 element.detachEvent("on"+type,handle);  
 }else{//dom0事件模型  
 element["on"+type]=null;  
 }  
 },  
 //事件对象  
 getEvent:function(event){  
 return event?event:window.event;  
 },  
 //事件源  
 getsrc:function(event){  
 // return event.srcElement || event.target;  
 return event.srcElement?event.srcElement:event.target;  
 },  
 //阻止冒泡事件  
 stopmp:function(event){  
 if(event.stopPropagation){  
 event.stopPropagation();  
 }else{  
 event.cancelBubble=true;  
 }  
 },  
 //阻止默认事件  
 preventDefault:function(event){  
 if(event.preventDefault){  
 event.preventDefault()  
 }else{  
 event.returnValue=false;  
 }  
 }  
 };  
 var v1=document.getElementById("id01");  
 function fun1(){  
 alert("dddd");  
 }  
 EventUtil.removeHandle(v1,'click',fun1,false);  
 };

1. 事件代理  
    \* 事件代理：又叫事件委托，事件委派，即将事件处理加载到父级元素上，可

\* 避免将事件处理添加到多个子级元素上。  
 \* 涉及特性：事件冒泡和目标元素  
 \* 优点：1、提高性能  
 \* 2、对于新增的子元素，前面的事件依然有效  
 \* \*/  
 window.onload=function(){  
 //1  
 var ali=document.getElementById("ul01").getElementsByTagName("li")  
 for(var i=0;i<ali.length;i++){  
 ali[i].onmouseover=function(){  
 this.style.background="red";  
 };  
 ali[i].onmouseout=function(){  
 this.style.background="";  
 };  
 }  
  
 //2  
 document.getElementById("but01").onclick=function(){  
 var v\_ul=document.getElementById("ul01");  
 var v\_li=document.createElement("li");  
 v\_li.innerHTML="a555555555555555";  
 v\_ul.appendChild(v\_li);  
 };  
  
 //事件代理  
 //方法一：  
 var v\_ul=document.getElementById("ul01");  
 v\_ul.onmouseover=function(event){  
 var e=EventUtil.getEvent(event);  
 var esrc=EventUtil.getESrc(e);  
 if(esrc.nodeName.toLowerCase()=="li"){  
 esrc.style.background="red";  
 }  
 };  
 v\_ul.onmouseout=function(event){  
 var e=EventUtil.getEvent(event);  
 var esrc=EventUtil.getESrc(e);  
 if(esrc.nodeName.toLowerCase()=="li"){  
 esrc.style.background="";  
 }  
 };  
 //方法二：  
 var v\_ul=document.getElementById("ul01");  
 v\_ul.onmouseover=function(event){  
 var e=EventUtil.getEvent(event);  
 var esrc=EventUtil.getESrc(e);  
 var child=document.getElementById("ul01").children;  
 for(var i=0;i<child.length;i++){  
 if(child[i]==esrc){  
 alert("index:"+i)  
 }  
 }  
 };  
}

<ul id="ul01">  
 <li>a111111111111111</li>  
 <li>a222222222222222</li>  
 <li>a333333333333333</li>  
 <li>a444444444444444</li>  
</ul>  
<button id="but01">新增</button>

//鼠标拖拽 onmousedown onmousemove onmouseup

window.onload=function(){  
 var pagex\_o,pagey\_o,offast\_l,offast\_t;  
 var v1=document.getElementById("id01");  
 v1.onmousedown=function(event){  
 var e=EventUtil.getEvent(event);  
 pagex\_o= e.pageX;  
 pagey\_o= e.pageY;  
 offast\_l=this.offsetLeft;  
 offast\_t=this.offsetTop;  
 v1.onmousemove=function(event){  
 var e=EventUtil.getEvent(event);  
 this.style.left=e.pageX-(pagex\_o-offast\_l)+"px";  
 this.style.top=e.pageY-(pagey\_o-offast\_t)+"px";  
 };  
 };  
 v1.onmouseup=function(event){  
 var e=EventUtil.getEvent(event);  
 this.style.left=e.pageX-(pagex\_o-offast\_l)+"px";  
 this.style.top=e.pageY-(pagey\_o-offast\_t)+"px";  
 v1.onmousemove=null;  
 }  
}

<div id="id01" style="width: 100px;height: 100px;background: red;position: absolute"></div>

22、移动端

<meta charset="UTF-8">  
<!--  
viewport:可视区域，视口；  
user-scalable=no 用户不可以手动缩放  
-->  
<meta name="viewport" content="initial-scale=1.0;user-scalable=no">

/\*  
\* 触摸事件--html5移动端  
\*  
\* 方法：  
\* ontouchstart：当手指触摸屏幕时触发，即使已经有一个手指放到屏幕了也会触发  
\* ontouchmove：当手指在屏幕上滑动时连续触发  
\* ontouchend：当手指从屏幕上离开时触发  
\*  
\* 属性：  
\* touches：表示当前跟踪的触摸操作touch对象的数组（几个手指头）  
\* changedTouches：表示自上次触摸以来发生了改变的touch对象数组  
\*  
\* 注意：手指在活动屏幕时，浏览器会有默认行为，如缩放和滚动，可用来禁止  
\* event.preventDefault();  
\*  
\* 手势事件：  
\* gesturestart：当一个手指按在屏幕上，另一个手指触摸屏幕时触发  
\* gesturechange：当触摸屏幕的任何一个手指发生变化时触发  
\* gestureend：当任何一个手指从屏幕上移开时触发  
\*  
\* 屏幕旋转事件：  
\* onorientationchange：  
\* \*/

window.onload=function(){  
 document.getElementById("id01").onclick=function(){  
 alert("1111");  
 };  
 document.getElementById("id01").ontouchstart=function(event){  
 alert("2222");  
 alert("hello");  
 alert(event.touches[0].clientX+" "+event.touches[0].clientY);  
 };  
 document.getElementById("id01").ontouchmove=function(event){  
 alert("3333");  
 alert("aaaaa");  
 alert(event.changedTouches[0].clientX+"

"+event.changedTouches[0].clientY);  
 event.preventDefault();  
 };  
 document.getElementById("id01").ontouchend=function(event){  
 alert("4444");  
 alert("hello");  
 };  
  
 //事件触发顺序：touchstart->touchmove->touchend->click  
  
  
 //手势事件  
 document.getElementById("id01").ongesturestart=function(event){  
 alert("aaa");  
 };  
 document.getElementById("id01").ongesturechange=function(event){  
 alert("bbb");  
 };  
 document.getElementById("id01").ongestureend=function(event){  
 alert("ccc");  
 };  
   
 window.onorientationchange=function(){  
 switch(window.orientation){  
 case 0:alert(screen.width+" "+screen.height);break;  
 case 90:alert(screen.width+" "+screen.height);break;  
 case -90:  
 case 180:  
  
 }  
 }  
};

<div id="id01" style="width: 200px;height:200px;background: red;"></div>

23、

<script>

/\*  
 \* js函数  
 \* \*/  
 //一、创建函数方式  
 //1、普通方式（声明式函数）  
 function sum(a,b){  
 var s=a+b;  
 return s;  
 }  
 var v=sum(2,3);  
 alert(v);  
 //2、函数表达式（赋值式函数）  
 var sum=function(a,b){  
 var s=a+b;  
 return s;  
 }  
 var v=sum(2,3);  
 alert(v);  
 //3、直接调用（只能调用一次）  
 (function(a,b){  
 var s=a+b;  
 alert(s);  
 })(2,3);  
 //4、函数对象创建  
 function与Function的区别？  
 var sum=new Function('a','b','var s=a+b;return s;');  
 alert(sum(2,3));  
  
 /\*  
 \* js内容运行机制---Event Loop  
 \* \*/  
 alert(1);  
 setTimeout(function(){alert(2)},1000);  
 alert(3);//1 3 2  
  
 alert(1);  
 setTimeout(function(){alert(2)},0);  
 alert(3);//1 3 2 为啥不是123？  
 //setTimeout 中的回调函数不一定会在setTimeout指定的时间内执行  
 /\*  
 \* 1、js单线程  
 \* 原因：js主要用途是与用户交互及操作dom，这决定了它只能是单线程的  
 \* 2、任务队列 Event Loop  
 \* js将所有任务分为两种：  
 \* 1）同步任务：synchronous 在主线层上必须等待前一个任务执行完，才能执行后一个任务  
 \* 2）异步任务：asynchronous 不进主线程，而是进入任务队列，主线程只有在得到“任务队列”的通知  
 \* 某个异步任务可以执行了，该异步任务才会进入主线程  
 \*  
 \* 任务队列：（事件队列）i/o输入输出 鼠标事件 键盘事件 页面滚动 onload 定时器 ajax等  
 \* 注意：只要指定过回调函数，这些事件发生时就会进入任务队列  
 \*  
 \* 主线程从“任务队列”中读取文本的过程是不断循环的----此机制称之为“Event Loop”  
 \*  
 \* 3、js代码块  
 \* （1）是以<script>或文件的方式分割  
 \* （2）js按代码块进行编译和执行、代码块间相互独立、但变量和方法共享  
 \*  
 \* 4、js预编译期与执行期（只预编译声明式函数）  
 \* 预编译期js会对本“代码块”中的所有声明的变量和“声明式函数”进行预编译  
 \* （声明式函数:后面的同名函数会覆盖前面的）  
 \* 变量只声明不赋值  
 \*  
 \* \*/  
  
 f1();//输出bbbb  
 function f1(){  
 alert("aaaa");  
 }  
 function f1(){  
 alert("bbbb");  
 }  
  
 f1();//赋值式函数:报错 声明式函数：aaaa （只预编译声明式函数）  
 function f1(){//声明式函数  
 alert("aaaa");  
 }  
 var f1=function(){//赋值式函数  
 alert("bbbb");  
 }  
  
</script>  
  
<script>  
 alert(i);//报错，代码块间相互独立  
</script>  
<script>  
 alert(i);//undefined  
 var i="a";  
</script>  
  
  
<script>  
 //js预处理的只是执行到代码块的声明式函数和变量，对于未加载的代码块，是没法进行与处理的  
 f1();//报错，未定义  
</script>  
<script>  
 function f1(){  
 alert("bbbb");  
 }  
</script>  
  
  
<!--块1-->  
<script>  
 alert(v1);//报错  
 alert("aaaaa");//不执行  
 var v2="cccc";  
</script>  
<!--块2-->  
<script>  
 alert("bbbbb");//执行  
 alert(v2);//undefined 把v1注销掉，输出cccc  
</script>  
<!--1、块1运行报错，但不影响块2的运行，这就是代码块的独立运行-->  
<!--2、块2中可以调用块1中的变量，即块间的共享性-->  
  
<script>  
 function f1(){  
 alert("hello");  
 }  
</script>

<body onload="f1()">  
<script>  
 alert("world");  
</script>  
</body>

24、

<script>  
 /\*  
 \* js函数  
 \* 二、函数返回值  
 \* 函数参数个数不限，但返回值只能有一个  
 \* \*/  
  
 //1、如有多个返回值---数组  
 //练习：输入两个数，用一个函数实现他们的加、减、乘、除，并返回四个结果  
 function f1(a,b){  
 return [a+b,a-b,a\*b,a/b];  
 }  
 alert(f1(2,4));  
  
 //2、返回函数  
 function f2(a,b){  
// function f3(){  
// return a+b;  
// }  
 return function(){return a+b;};  
 }  
 alert(f2(2,3)());

//3、返回函数自身  
var i=1;  
function fun1(){  
 i++;  
 return fun1;  
}  
fun1()()()();  
alert(i);//5

function fun2(a,b){  
 return a%b+ a++ + --a + ++b - --a%b++;  
}  
alert(fun2(fun2(12,7),fun2(5,9)));  
// 34 21 92  
  
function f(){  
 return 1;  
}  
function f(){  
 function f(){  
 function n(){  
 return 3;  
 }  
 return n();  
 }  
 function n(){  
 return 4;  
 }  
 return f();  
}  
alert(f());//3

//测试  
 function f1(a,b){  
 function f2(){  
 s=a\*3;  
 function f3(){  
 return s+b;  
 }  
 return f3;  
 }  
 return f2;  
 }  
 alert(f1(2,3)()());  
  
 var tt = 'aa';  
 function test(tt){  
 alert(tt);  
 var tt = 'dd';  
 alert(tt);//aa  
 }  
 test(tt);  
  
 var tt = 'aa';  
 function test(tt){  
 alert(tt);  
 tt = 'dd';  
 alert(tt);  
 }  
 test(tt);  
 alert(tt);  
  
  
 var x = 2, y = z = 1;  
 function add(n){  
 n = n+1;  
 return n;  
 }  
 y = add(x);  
 function add(n){  
 n = n + 3;  
 return n;  
 }  
 z = add(x);  
   
 </script>  
 <script>  
 function add(n) {  
 return n = n+1;  
 }  
 alert(add(1));  
 </script>  
  
 <script>  
 function add(n) {  
 return n = n+3;  
 }  
 alert(add(1));  
 </script>

25、

<script>  
 f1();//7  
 function f1(){  
 alert(1);  
 }  
 var f1=function(){  
 alert(2);  
 };  
 f1();//2  
 function f1(){  
 alert(3);  
 }  
 f1();//2  
 var f1=new Function("alert(4)");  
 f1();//4  
 function f1(){  
 alert(5);  
 }  
 function f1(){  
 alert(6);  
 }  
 f1();//4  
 var f1=function(){  
 alert(8);  
 };  
 f1();//8  
 function f1(){  
 alert(7);  
 }  
 var f1=new Function("alert(9)");  
 f1();//9  
 //7 2 2 4 4 8 9  
 </script>  
  
  
 <script>  
 var a=1;  
 function f1(){  
 alert(a); //undefined  
 alert(window.a); //1  
 a=2;  
 alert(a); //2  
 var a=3;  
 alert(a); //3  
 }  
 f1();  
 alert(a);//1  
 //1 2 3 1  
  
  
 f=function(){  
 return true;  
 };  
 g=function(){  
 return false;  
 };  
 (function(){  
 if(g() && []==![]){  
 f=function f(){return false;}  
 function g(){  
 return true;  
 }  
 }  
 })();  
 alert(f());  
  
 /\*  
 \* js变量及变量作用域  
 \* 1、全局变量  
 \* （1）var+变量并在函数function外部声明  
 \* （2）省略var，隐示全局变量  
 \* （3）window.v1 window对象声明全局  
 \*  
 \* 2、显示声明与隐示声明全局变量区别  
 \* （1）函数体外var声明的全局变量不能删除  
 \* （2）省略var定义的变量。无论函数体内或体外，都能被删除  
 \* （3）window对象定义的全局变量，ie8及以下无法删除，其他浏览器可删除  
 \*  
 \* 3、js变量的作用域（scope）是根据函数快来划分的，而for、if、while等不是作用域的划分标准  
 \*  
 \* 4、函数声明和  
 \* \*/  
 var v1=1;//全局变量  
 function f1(){  
 v1=8;  
 alert(v1);//8  
 }  
 f1();  
 alert(v1);//8  
  
 var v1=1;//全局变量  
 function f1(){  
 var v1=8;//局部变量  
 alert(v1);//8  
 }  
 f1();  
 alert(v1);//1  
  
 v1=1;//隐示全局变量  
 function f1(){  
 v2=2;//隐示全局变量  
 alert(v1);//1  
 }  
 f1();  
 alert(v1);//1  
 alert(v2);//2  
  
 v1=1;  
 window.v2=3;  
 function f1(){  
 alert(window.v1);//1  
 alert(v1);//undefined  
 var v1=2;  
 alert(v1);//2  
 alert(v2);//3  
 }  
 f1();  
 alert(v1);//1  
  
 var v1=1;  
 v2=2;  
 window.v3=3;  
 function f2(){  
 v4=4;  
 }  
 delete v1;//flase  
 delete v2;//true  
 delete v3;//true  
 delete v4;//true  
 alert(v1);//1  
 alert(v2);//报错  
 alert(v3);//报错  
 alert(v4);//报错  
  
 function f3(){  
 for(var i=0;i<3;i++){  
 var sum=i;  
 }  
 alert(i);//3  
 alert(sum);//2  
  
 var v1=3;  
 if(v1>0){  
 var m=5;  
 }  
 alert(m);//5  
  
 while(true){  
 var n=3;  
 break;  
 }  
 alert(n);//3  
 }  
 f3();  
 alert(m);//报错 之后没有执行  
 alert("hello");  
 alert(n);  
  
  
 //重点  
 for(var i=0;i<3;i++){  
 console.log(j+" "+k);  
 for(var j=0;j<3;j++){  
 var k=j+1;  
 }  
 }  
 console.log(i);  
 //undefined undefined  
 //3 3  
 //3  
  
 //如何进行异步情况下局部变量当前值的传递？  
 //解决：闭包  
 for(var i=0;i<3;i++){  
 //1  
 setTimeout(function(){alert(i)},1);//3 //3 //3  
 //2 闭包  
 setTimeout((  
 function(m){  
 return function(){  
 alert(m);//0 //1 //2  
 }  
 })(i),1);  
 }  
  
  
 //声明的变量声明的优先级高于函数声明的优先级  
 //未变量声明的优先级低于函数声明的优先级  
 var a=2;  
 function a(){}  
 alert(typeof a);//number  
 var a;//undefined  
 function a(){}  
 alert(typeof a);//function  
  
  
 //特例：函数声明的优先级高于变量的优先级  
 var a=1;  
 function b(){  
 a=3;  
 return;  
 }  
 b();  
 alert(a);//3  
  
 var a=1;？？？？？？？？  
 function b(){  
 a=3;  
 return;  
 function a(){}  
 }  
 b();  
 alert(a);//1  
  
 </script>  
<body>  
先看看下一段函数的返回结果  
f = function() {return true;};  
g = function() {return false;};  
(function() {  
if (g() && [] == ![]) {  
f = function f() {return false;};  
function g() {return true;}  
}  
})();  
请计算 f()和g()的结果  
f()=false g()=true  
结果是出乎意料的，原因是IE/Chrome/Safari 和 Firefox不一致  
if(true) {  
function fn(){ return 1; }  
} else {  
function fn(){ return 2; }  
}  
Firefox 效果等同（IE/Chrome/Safari/Firefox一致）  
if(true) {  
var fn = function(){ return 1; }  
} else {  
var fn = function(){ return 2; }  
}  
绿皮书解释：  
但是从本质上来说，这只是因为SpiderMonkey Javascript支持在运行期对语句中声明的函数作解析。而if语句后的这个函数声明被理解为“直接量表达式语句”。座位语句的语法元素时，SpiderMonkey Javascript就能按照函数声明的语法效果进行理解了。同样的，即使是在if语句之后，如果把它作为表达式中的运算元，也是达不到声明函数的效果的  
</body>

26、函数参数  
\* 1、f1.length获取形参参数个数  
\* 2、如果形参个数多于实参个数，多余出来的形参 undefined  
\* 如果形参个数小于实参个数，多余出来的实参忽略  
\* 3、参数管理器arguments  
\* （1）arguments.length获取实参参数个数  
\* （2）arguments该对象仅能在函数体内使用  
\* （3）arguments[0]获取实参值，可以不用指定形参  
\* （4）arguments该对象可用数组形式来调用实参值，但它不是数组  
\* （5）可以修改参数值 arguments[0]=6;  
\* \*/  
  
 function f1(a,b){//形参  
 return f1.length;  
 return d;  
  
 return f1.length;//2  
 return arguments.length;//3  
  
 return arguments[0];//1  
 return arguments[1];//2  
 return arguments[2];//3  
 return arguments[0]+arguments[1]+arguments[2];  
  
 return(arguments instanceof Array);//false  
 return(arguments instanceof Object);//true  
  
 var arr1=[];  
 for(var i=0;i<arguments.length;i++){  
 arr1.push(arguments[i]);  
 alert(arguments[i])  
 }  
 return arr1;  
  
 arguments[0]=6;  
 return arguments[0]+arguments[1]+arguments[2];//11  
}  
alert(f1(1,2,3));//实参  
  
//异常处理  
function f2(a,b){  
 var v1=f2.length;  
 var v2=arguments.length;  
 if(v1!=v2){  
 //异常处理  
 var error=new Error();//创建异常对象  
 error.msg="参数不符，请重新输入";  
 error.num=10001;  
 throw error;//抛出异常  
 }  
}  
//捕获异常  
try{  
 f2(1,2,3);  
}catch(err){  
 console.log("异常信息："+err.msg+" 错误码："+err.num);  
}

27、callee与caller  
//callee 该属性是一个指针，指向拥有这个arguments参数对象的函数（返回当前arguments所属函数的引用）  
function f3(a,b){  
 return f3.length  
 return arguments.callee.length;  
}  
alert(f3(1,2,3));  
  
var f1=function(n){  
 if(n>0){  
 return n+f1(n-1);  
 return n+arguments.callee(n-1);  
 }  
 return 0;  
};  
alert(f1(5));  
  
//caller 保存着调用当前函数的函数引用--返回调用者  
//如在全局作用域中调用当前函数，它的值为null  
function f1(){  
 f2();  
}  
function f2(){  
 alert(f2.caller);  
}  
alert(f1())  
  
function f3(){  
 alert(f3.caller);  
}  
function f4(){  
 f3();  
}  
alert(f3());

28、apply 与 call  
 /\*  
 \* 1、apply 和 call都可以“扩充函数赖以生存的作用域”  
 \* 即能改变函数内部指针的指向，从而实现属性方法的继承  
 \* 改变函数体中this对象的指向  
 \* 2、将函数绑定到另外一个对象上去运行  
 \* 3、apply与call的区别就是传递参数的方式不同  
 \* call：参数分开传递  
 \* apply：参数以数组或arguments参数对象的形式传递  
 \* \*/  
 //1  
 var v1='a';  
 function f1(){  
 v1='b';  
 }  
 function f2(){  
 alert(v1);  
 }  
 f2();//a  
 f2.call(f1());//b  
 f2.apply(f1());//b  
  
 var v1='c';  
 function f1(){  
 this.v1='d';  
 }  
 function f2(){  
 this.v1='e';  
 }  
 function f3(){  
 alert(v1);  
 }  
 function f4(){  
 var v1='f';  
 alert(this.v1);//c this指向window对象  
 }  
 f4();// c this指向window对象  
 f4.call(f1());//d 将f4中的this指针指向函数f1  
 f4.call(f2());//e  
 f4.call(f3());//c  
   
 window.color="red";  
 var c1={color:"yello"};  
 function f1(){  
 alert(this.color);  
 }  
 f1();//red  
 f1.call(window);//red  
 f1.call(c1);  
  
 //2  
 function sum(a,b){  
 return a+b;  
 }  
 function f1(x,y){  
// var s=sum.call(this,x,y);  
// var s=sum.apply(this,[x,y]);  
// var s=sum.apply(this,arguments);  
 alert(s);//5  
 }  
 f1(2,3);  
   
 function person(name,age){  
 this.name=name;  
 this.age=age;  
 }  
 function stu(name,age,grade){  
 person.apply(this,[name,age]);  
 this.grade=grade;  
 }  
 var s=new stu("zs",23,"2年级");  
 alert(s.name+" "+ s.age+" "+ s.grade);

29、闭包（closure）  
\*  
\* js链式作用域，子对象会一级一级向上寻找父对象变量，直到找到为止  
\* 所以，父对象的所有变量对子对象是可见的，而子对象对父对象是不可见的  
\*  
\* 解决：闭包  
\* 1、闭包就是能够读取其他函数内部变量的函数  
\* 2、闭包是定义在一个函数内部的函数  
\* 3、闭包就是将函数内部与函数外部连接起来的桥梁  
\*  
\* 特点：变量始终保存在内存中  
\* 确定：由于闭包会使用函数中的变量保存在内存中，内存消耗大，要慎用，避免造成性能问题  
\*  
\* \*/  
  
 var v1=10;  
function f1(){  
 var v2=30;  
 function f1\_1(){  
 var m=40;  
 alert(v2);  
 }  
 alert(m);  
}  
f1();  
alert(v2);  
  
//解决  
var v1=10;  
function f1(){  
 var v2=30;  
 //闭包  
 function f1\_1(){  
 return v2;  
 }  
 return f1\_1;  
}  
var m=f1();  
alert(m());  
  
var v1=10;  
function f1(){  
 var v2=30;  
 function f1\_1(){  
 v2++;  
 return v2;  
 }  
 return f1\_1;  
}  
var m=f1();  
alert(m());//31  
alert(m());//32  
alert(m());//33  
alert(m());//34  
var n=f1();  
alert(n());//31  
alert(n());//32  
alert(n());//33  
alert(n());//34  
  
//闭包只能取得包含函数中任何变量的“最后”一个值  
//匿名函数强制闭包符合预期  
function f1(){  
 var arr1=new Array();  
 for(var i=0;i<10;i++){  
 arr1[i]=function(n){  
 alert(n);  
 return function(){return n;}  
 }(i);  
 }  
 return arr1;  
}  
var v=f1();  
for(var i=0;i<v.length;i++){  
 document.write(v[i]()+" ");//0 1 2 3 4 5 6 7 8 9  
}

30、

/\*  
\* 面向对象技术  
\* \*/  
//对象的创建函数：  
  
//1、普通方式创建  
var person=new Object();  
//属性  
person.name='zs';  
person.age=34;  
//方法  
person.say=function(){  
 alert("我在说话。。。"+this.name+this.age);  
};  
person.eat=function(){  
 alert("我在吃。。。");  
};  
//对象调用（属性、方法调用）  
alert(person.name);  
person.say();  
  
//2、直接量方式  
var person={name:'zs',  
 age:34,  
 say:function(){alert("我在说话。。。"+this.name+this.age);},  
 eat:function(){alert("我在吃。。。");}  
};  
alert(person.name);  
person.say();  
  
//3、构造函数方式  
/\*  
\* 1、构造函数一般首字母大写，用来与普通函数进行区别  
\* 2、一般通过this来指定构造函数成员  
\* 3、构造函数由new进行创建  
\* 4、构造函数一般情况下没有返回值  
\* 如果构造函数返回的是一个对象，则返回的对象会覆盖构造函数的对象  
\* \*/  
function Person(){  
 this.name='zs';  
 this.age='34';  
 this.say=function(){  
 alert("say...."+this.name);  
 }  
}  
var p=new Person();  
p.say();  
  
//特例  
//1  
function F1(){  
 this.a=1;  
 return true;  
}  
var f=new F1();  
alert(f.a);//1  
alert(f);//object Object  
alert(F1());  
//2  
function F1(){  
 this.a=1;  
 return {b:3,a:5};  
}  
var f=new F1();  
alert(f.a);  
  
//4、函数对象方式  
var person=new Function("this.name='zs';this.age='34';this.say=function(){alert('say....'+this.name);}");  
var p=new person();  
p.say();

31、//构造函数模式  
//对象  
function Person(name,age){  
 this.name=name;  
 this.age=age;  
 this.say=function(){  
 alert("say........"+this.name);  
 }  
}  
//创建实例  
var p1=new Person("zs",34);  
p1.say();  
var p2=new Person("ls",23);  
p2.say();  
alert(p1.say==p2.say);//false  
  
//原型模式  
function person(){}  
person.prototype.name="zs";  
person.prototype.say=function(){  
 alert("say......"+this.name);  
};  
var p1=new person();  
p1.say();  
var p2=new person();  
p2.say();  
alert(p1.say==p2.say);//true  
  
//原型构造函数模式  
function Person(name,age){  
 this.name=name;  
 this.age=age;  
}  
Person.prototype.say=function(){  
 alert("say......"+this.name+" "+this.age);  
};  
var p1=new Person("zs",34);  
p1.say();  
var p2=new Person("ls",23);  
p2.say();  
alert(p1.say==p2.say);//true  
  
//原型链的继承  
function Father(x){  
 this.x=x;  
 this.get=function(){  
 return "父："+this.x;  
 }  
}  
function Son(y){  
 this.y=y;  
 this.getson=function(){  
 return "子："+this.y;  
 }  
}  
Son.prototype=new Father(2);//原型链继承  
var s=new Son(3);  
alert(s.y);  
alert(s.x);  
alert(s.get());  
  
function Person(name,age){  
 this.name=name;  
 this.age=age;  
 this.say=function(){  
 return "say......"+this.name+" "+this.age;  
 };  
}  
function Stu(num){  
 this.num=num;  
 this.study=function(){  
 return "study...... "+this.num;  
 };  
}  
function Gr(n){  
 this.gl=n;  
 this.work=function(){  
 return "work...... "+this.gl;  
 };  
}  
Stu.prototype=new Person("zs",34);  
var s=new Stu(20136511);  
alert("学生姓名："+ s.name+",年龄："+ s.age+",学号："+ s.num+",说："+ s.say());  
Gr.prototype=new Person("ls",24);  
var t=new Gr("2年");  
alert("工人姓名："+ t.name+",年龄："+ t.age+",工龄："+ t.gl+",工作："+ t.work());  
  
  
//层层指向父原型的关系--原型链（原型链先找构造函数内，再找原型）  
function a(x){  
 this.q=x;  
}  
a.prototype.x=0;  
  
function b(x){  
 this.z=x;  
}  
b.prototype=new a(1);  
  
function c(x){  
 this.y=x;  
}  
c.prototype=new b(2);  
  
var m=new c(3);  
delete m.x;  
alert(m.x);

32、面向对象 prototype \_proto\_ constructor

<script>  
 /\*  
 \* 1、每个函数都有一个prototype属性，该属性指向了原型对象  
 \* 2、原型对象中，有constructor属性，该属性指向函数本身  
 \* 3、new创建的对象实例没有prototype属性  
 \* 4、每个对象都有\_\_proto\_\_属性，指向它所对应的原型对象  
 \* 原型对象正是基于\_\_proto\_\_属性，才得以形成（而不是基于prototype）  
 \* \*/  
 function Animal(){}  
 Animal.prototype.name="dw01";  
 Animal.prototype.dosoming=function(){  
 alert(thus.name);  
 };  
   
 alert(Animal.prototype);//object Object  
 alert(typeof Animal.prototype);//object  
 alert(Animal.prototype.constructor);//function Animal(){}  
  
 var a1=new Animal();  
 alert(a1.prototype);//undefined  
  
 //4  
 function Person(){}  
 Person.prototype.age=23;  
 Person.prototype.method01=function(){  
 alert(this.age);  
 };  
  
 Person.prototype=new Animal();//继承  
 var p1=new Person();  
 var p2=new Person();  
  
 alert(Person.prototype.\_\_proto\_\_==Animal.prototype);//true  
 alert(p1.\_\_proto\_\_.\_\_proto\_\_==Animal.prototype);//true 原型链  
  
 alert(Animal.prototype.\_\_proto\_\_==Object.prototype);//true  
  
 alert(Object.prototype.\_\_proto\_\_);//null 所以到这里原型链结束了  
  
 alert(p1.\_\_proto\_\_==Person.prototype);//true  
 alert(p2.\_\_proto\_\_==Person.prototype);//true  
  
 //p1本身没有constructor属性，但p1.\_\_proto\_\_=Person.prototype，则，  
 //Person.prototype中的所有属性和方法都可以被p1访问  
 alert(p1.constructor);//function Person(){}  
 alert(p1.\_\_proto\_\_.constructor);//function Person(){}  
  
   
 //直接量方式  
 function Person(){}  
 Person.prototype=new Object({  
 constructor:Person,  
 age:23,  
 method01:function(){  
 alert(this.age);  
 }  
 });  
 var p1=new Person();  
 p1.method01();  
 alert(Person.prototype.constructor);//function Person(){}  
   
   
 //扩展  
 var o1={};  
 var o2=new Object();  
  
 var f1=function(){};  
 function f2(){}  
 var f3=new Function();  
  
 alert(typeof o1);//object  
 alert(typeof o2);//object  
 alert(typeof f1);//function  
 alert(typeof f2);//function  
 alert(typeof f3);//function  
  
 alert(typeof Object);//function  
 alert(typeof Function);//function  
 //1、函数也是对象，由Function创建，f1，f2最终也是由Function创建的  
 //2、Function与Object本身也是函数对象  
  
 function Person(){}  
 Person.prototype.age=23;  
 Person.prototype.method01=function(){  
 alert(this.age);  
 };  
  
 alert(Person.\_\_proto\_\_);//function (){}  
 alert(Person.\_\_proto\_\_==Function.prototype);//true  
  
 alert(Function.prototype.constructor==Function);//true  
 alert(Object.prototype.constructor==Object);//true  
 alert(Object.constructor);//Function  
  
 //Function原型对象中没有prototype属性  
 alert(Function.prototype.prototype);//undefined  
</script>

33、html5 canvas 绘图步骤

<script>  
 window.onload=function(){  
 //html5canvas 绘图步骤  
 //获取画布  
 var v1=document.getElementById("canvas01");  
 v1.width=300;  
 v1.height=300;  
 //获取上下文对象  
 var context=v1.getContext("2d");  
 //绘图  
  
 //1实心  
 //矩形  
 context.fillStyle="#223344";//设置填充样式  
 context.fillRect(20,20,100,80);//绘制矩形（坐标X，坐标Y，宽，高）  
 //清空  
 context.clearRect(0,0,300,300);  
  
 //2空心  
 context.strokeStyle="red";//填充边框样式  
 context.lineWidth=3;  
 context.strokeRect(20,20,100,80);//绘制线  
  
 //3画线  
 //moveTo(起始点横坐标，起始点纵坐标)  
 //lineTo(终止横坐标，终止点纵坐标)  
 context.moveTo(10,10);  
 context.lineTo(80,80);  
 context.strokeStyle="green";//填充边框样式  
 context.lineWidth=5;  
 context.stroke();//绘制线  
  
 var x,y;  
 var z=Math.PI/15\*11;  
 context.beginPath();  
 context.moveTo(80,90);  
 for(var i=0;i<30;i++){  
 x=Math.sin(i\*z);  
 y=Math.cos(i\*z);  
 context.lineTo(150+x\*100,150+y\*100);  
 }  
 context.closePath();  
 context.strokeStyle="blue";//填充边框样式  
 context.lineWidth=1;  
 context.stroke();//绘制线  
  
 document.onmousedown=function(e){  
 e.offsetX  
 e.offsetY  
 }  
 }  
</script>

<!--画布-->  
<canvas id="canvas01" style="border: 1px solid #000000"></canvas>

1. html5 canvas 画布中加载图片

<script>

window.onload=function(){  
 var c1=document.createElement("canvas");  
 c1.style.width="500px";  
 c1.style.height="500px";  
 c1.style.border="1px solid black";  
 document.body.appendChild(c1);  
 var context=c1.getContext("2d");  
 //加载图片  
 var img=new Image();  
 img.src="../images/2.jpg";  
 //onload事件  
 img.onload=function(){  
 //context.drawImage(img,0,0);//图片起始位置  
 //context.drawImage(img,0,0,200,100);//图片宽高  
 /\*  
 \* context.drawImage(img,sw,sy,sw,sh,dx,dy,dw,dh);  
 \* 将画布中已经绘制好的图像全部或部分区域复制到另一个位置上  
 \* sx，sy：源图像被复制区在画布中的起始横纵坐标  
 \* sw，sh：被复制区的宽度与高度  
 \* dx，dy：复制后的目标图像在画布中的起始横纵坐标  
 \* dw，dh：复制后的目标图像的宽度与高度  
 \* \*/  
 context.drawImage(img,30,40,50,50,150,150,100,100);  
 }  
}

</script>

35、

//父

<script>  
 //iframe 内联框架  
 //父框架向子框架传递数据  
 window.onload=function(){  
 document.getElementById("id01").addEventListener("click",function(){

iframe01.document.getElementById("subframe").innerHTML="你好，我是父框架。";  
 },false);  
 }  
</script>

<iframe src="http://www.baidu.com" style="background:gainsboro;width: 300px;height:300px;border: 1px solid #000000"></iframe><br>

<iframe src="test46\_1.html" name="iframe01" style="background:gainsboro;width: 300px;height:300px;border: 1px solid #000000"></iframe><br>

<iframe src="test46\_2.html" name="iframe02" style="background:gainsboro;width: 300px;height:300px;border: 1px solid #000000"></iframe>

<button id="id01">确定</button><br>

<div id="fatherdiv"></div>

//子一：

function fun1(){  
 //子框架向父框架传递数据

window.parent.window.document.getElementById("fatherdiv").innerHTML="你好,子窗口！";  
  
}  
function fun2(){  
 window.parent.window.iframe02.document.getElementById("bortherdiv").innerHTML="你好，兄弟子窗口！";  
}

子框架  
<div id="subframe"></div><br>  
<button onclick="fun1()">确定</button>  
<button onclick="fun2()">兄弟内联框架</button>

//子二：

<div id="bortherdiv"></div>

36、

/\*  
\* js跨域：  
\* 同源策略：阻止从一个域上加载脚本获取或操作另一个域上的文档属性  
\* 意味着浏览器隔离来自不同源的内容，防止他们之间的操作  
\*  
\* jsonp:json with padding 非官方协议  
\* 1、它能够通过在当前文档（客户端）中生成脚本标记<script>标签，  
\* 来调用跨域脚本（服务器脚本文件）时使用的约定  
\* 2、jsonp是一种可以绕狗同源策略的方法，从任何服务端直接返回可执行的  
\* js函数调用或js对象  
\* \*/

<div id="show"></div>  
  
<input type="text" id="inp01">  
<button onclick="fun3()">发送至其他服务器</button>

<script>  
function fun1(v1){  
 alert(v1);  
}  
</script>  
<script src="http://192.168.1.69:81/javascriptProject\_43/js/ky01.js" type="text/javascript"></script>

<script>  
 function fun3(){  
 var v1=document.getElementById("inp01").value;  
 //动态创建脚本  
 var s=document.createElement("script");  
 s.setAttribute("type","text/javascript");  
 s.setAttribute("src","http://192.168.1.69:81/javascriptProject\_43/test62\_php.php?str="+v1+"&fname=fun4");  
 document.body.appendChild(s);  
 }  
</script>  
<script>  
 function fun4(v){  
 alert("来自qq端的信息："+v);  
 }  
</script>

<?php  
$v2=$\_GET["str"];  
$v3=$\_GET["fname"];  
  
//向新浪发送数据  
//fun4("dddd")  
echo $v3."('$v2')"  
?>

37、

//1

<script>  
 /\*  
 \* html5跨域通信----postMessage  
 \* \*/  
 function fun1(){  
 //window对象的postMessage:表示向其他窗口发送信息  
 //postMessage（所发送的消息文本，接收消息的对象窗口URL地址）  
 window.frames[0].postMessage("hi,我是zxy。"

,"http://192.168.1.74:81/javascriptProject\_43/test64.html")  
 }  
 //监听message事件  
 window.addEventListener("message",function(e){  
 document.getElementById("show").innerText=e.data;  
 },false);  
</script>

<iframe src="http://192.168.1.74:81/php-js/test48\_1.html" onload="fun1()" style="width:200px;height: 200px;border:1px solid #000000;"></iframe>  
<div id="show"></div>

//2

<script>  
 //监听message事件  
 window.addEventListener("message",function(e){  
 /\*  
 \* e.data 远程发送来的数据  
 \* e.source 当前的地址源  
 \* e.origin消息源（谁发来的消息）  
 \* \*/  
 document.getElementById("show").innerText=e.source+" "+e.data+" "+e.origin;  
 //向48.html发数据  
 e.source.postMessage("hi....wo shi 48\_1",e.origin+"/php-js/test48\_1.html");  
 },false);  
</script>

<div id="show" ></div>

38、js模块化编程  
模块化标准：  
commonJs：是服务端模块的规范，nodejs采用了该标准  
AMD：浏览器端的模块要采用异步加载的函数，所以产生了AMD  
 Asynchronous Module Denfinition 异步方式加载模块  
 即模块的加载不影响后面语句的运行  
 实现该规范的js库：reqire.js 和 curl.js  
CMD：Common Module Definition  
 实现该规范的js库：sea.js（淘宝公司--玉伯老师 开发的）  
 define  
 exports  
 require  
  
一、commonJs  
语法：  
require():模块引入（引入外部模块）  
exports:模块定义（该对象用于导出当前模块的方法和变量）  
module:模块标识（该对象代表模块本身）  
  
commonJs规范不适合浏览器端环境，由于采用同步加载  
  
二、AMD  
语法：  
require([module],callback):加载模块  
module:要加载的模块，以数组的形式表示  
callback:加载成功后的回调函数  
  
define(id,dependencies,factory):定义模块  
id:可选项，模块标识  
dependencies：当前的依赖模块，以数组形式表示，如无依赖可省略模块  
factory：必写项，需要调用的函数或对象  
1、可进行异步加载  
2、解决命名冲突  
3、管理模块间的依赖，便于代码间的编写与维护

<!--1-->  
 <script src="../js/mathtest.js"></script>  
 <script src="../js/testcom.js"></script>  
  
 <!--2-->  
 <script src="../js/require2.2.js"></script>  
 <script>  
 //加载模块  
 require(["../js/lib/a"],function(a1){  
 a1.hello();//后出  
 });  
 alert("aaaaa");//先出  
 </script>  
  
 <script src="../js/require2.1.js"></script>  
 <script>  
 //依赖 test\_a要使用test\_b中的b1方法  
 require(["../js/lib/test\_a"],function(a1){  
 a1.f1();//后出  
 });  
 </script>  
  
 <script src="../js/require2.2.js"></script>  
 <script>  
 //加载模块  
 require(["../js/lib/b"],function(a1){  
 a1.sum(7,2,3);  
 });  
 </script>  
  
 <script src="../js/lib/test\_a.js"></script>  
 <script src="../js/lib/test\_b.js"></script>  
 <script src="../js/require2.2.js"></script>  
 <script>  
 //命名冲突（js文件里函数名相同，命名冲突）  
 require(["../js/lib/test\_a","../js/lib/test\_b"],function(a,b){  
 a.f1();  
 b.f1();  
 });  
 </script>  
  
 <script src="../js/require2.2.js"></script>  
 <script>  
 //第二种引用方式  
 //require.config 配置模块加载位置  
 require.config({  
// baseUrl:"../js/lib/",//基础路径  
 paths:{  
// "a1":"test\_a",  
// "a2":"test\_b",  
  
 //第三方框架加载  
 "jquery":"../js/lib/jquery-1.8.2"  
 }  
 });  
 require(["a1","a2","jquery"],function(a,b,$){  
 a.hello();  
 b.f1();  
 });  
 require(["jquery"],function($){  
 $(function(){  
 $("#but01").click(function(){  
 alert("hello");  
 });  
 });  
 });  
 //主模块加载  
 </script>  
 <script src="../js/require2.2.js" data-main="main"></script>

mathtest.js

//封装成模块  
exports.add1=function(){  
 var sum=0;  
 for(var i=0;i<arguments.length;i++){  
 sum+=arguments[i];  
 }  
 return sum;  
};

testcom.js

//引用模块  
//var mk\_math=require("mathtest");  
//调用方法  
//mk\_math.add1(2,3,4);  
  
var mk\_math=require("./mathtest");  
console.log(mk\_math.add1(2,3,4));  
console.log("dddddd");

main.js

require.config({  
 paths:{  
 "jquery":"../js/lib/jquery-1.8.2"  
 }  
});  
require(["jquery"],function($){  
 $(function(){  
 $("#but01").click(function(){  
 alert("hello");  
 });  
 });  
});

<button id="but01">确定</button>