1、简介；2、对象和基本数据类型；3、函数；4、逻辑运算；5、异常处理；6、DOM对象；7、事件；8、WINDOW对象和他的一些比较重要的属性；9、一些没有讲到的常用对象属性和方法 10、ajax

## 1、简介

JavaScript(简称JS，后面都将用JS代替)是脚本语言

是一种轻量级的编程语言，可插入HTML页码中的编程代码，并可由现在的浏览器执行

它可以对HTML进行编辑来展示出想要得到的结果

浏览器会在读取代码时，逐行地执行脚本代码。而对于传统编程来说，会在执行前对所有代码进行编译

简单实现

可以直接在HTML页面中插入JS代码，使用<script></script>标签即可，也可以将代码写在JS文件中，然后用<script src="XXX.js"></script>来进行引用，如下：

<script>

    document.write("<h1>This is a heading</h1>");

    document.write("<p>This is a paragraph</p>");

</script>

JS每句的结尾可以不加分号，但是还是建议尽量加上

如果按照上面这么写，那么在页面加载的时候就会执行，但是我们更多的时候是需要在某个事件发生的时候执行代码，比如：鼠标点击，鼠标移动到上面等操作。

这时候我们就需要用到函数，将代码放入特定的函数中，在事件发生时触发此函数，如下：

<script>

    function popup(){

        alert("test");

    }

</script>

注释

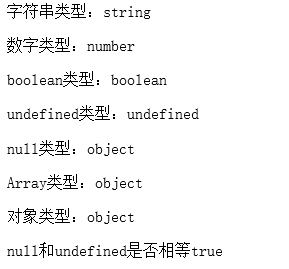
单行注释用//，多行注释用/\*   \*/, 实际测试在HTML中写的时候用<!-- -->也是可以的，但是写在JS文件中的时候不能用这种写法

## 2、对象和基本数据类型

2、基本的数据类型

基本类型有：字符串（String）、数字(Number)、未初始化（Undefined）、空（Null）、布尔（Boolean）

使用typeof获取类型时，如下图所示：



虽然null和undefined属于不同的类型，但是他们是相等的

要判断对象类型，需要使用instanceof，和java类似，在不同的window或iframe下，相同的对象也会返回false

JS中定义对象只需要用var就行了，例如：    （如果不用var而直接定义，那么这个变量会变成全局变量）

var a = 11;（数字前面加0表示8进制，加0X表示16进制，所以不要轻易在数字前加0）

通过isNaN()函数来判断一个对象是否是数字

var b = "abc";(定义字符串时，可以用双引号也可以用单引号将类容引起来，当字符串中包含与引起来的引号相同的引号时，需要用\进行转义)

"ab\"c" === 'ab"c';

var c; (此时c为undefined)

var d = null;

var e = true;

对象声明：

var a=new String;

var b= new Number;

var c= new Boolean;

var d= new Array;

var e= new Object;

所有的事务都是对象，比如上面所说的字符串、数字等，对象都会拥有属性和方法，內建对象一般都提供好了对应的属性和方法

## 3、函数

前面已经讲到过函数了，JS中函数就是包裹在花括号中的代码块，前面使用了关键词 function，可以接受参数：

function functionname(argument1,argument2){

//需要执行的代码

}

对象和函数名，需要注意，不要使用：关键字，保留字，内置对象、属性、方法，window保留关键字，HTML事件句柄

函数重载：JS的函数重载和一般的面向对象有所区别，比如我们用的JAVA，.net，在方法名称相同，参数不同或者参数类型不同的时候，可以正确找到方法，但是JS不能，JS会以最后一个函数为有效函数

想要实现重载，需要手动在方法中判断参数的个数，如下所示,通过下面两个函数，可以完成对参数个数以及参数类型的判断，从而来实现重载：

function overLoad(){

    //获取参数的数量

    var paramCount = arguments.length;

    var type;

    if(paramCount == 0){

        alert("没有参数");

    }

    if(paramCount == 1){

        alert("一个参数执行这个,类型分别为：" + getType(arguments[0]));

    }

    if(paramCount == 2){

        alert("两个参数执行这个,类型分别为：" + getType(arguments[0]) + "," + getType(arguments[1]));

    }

    if(paramCount == 3){

        alert("三个参数执行这个,类型分别为：" + getType(arguments[0]) + "," + getType(arguments[1]) + "," + getType(arguments[2]));

    }

}

function getType(param){

    var paramType = param.constructor;

    if(paramType == String){

        return "字符串"

    }

    if(paramType == Number){

        return "数字"

    }

    if(paramType == Function){

        return "函数"

    }

    if(paramType == Array){

        return "数组"

    }

    if(paramType == Boolean){

        return "boolean"

    }

    if(paramType == Objcet){

        return "Objcet"

    }

}

## 4、逻辑运算

4、判断和运算

运算符

算术运算符 + - \* / % ++ --

浮点算术运算是不准确的，比如：0.2 + 0.1算出来的值显示的就可能不是0.3

赋值运算符 = += -= \*= /= %=

x+=y ---> x=x+y

当对字符串用+时，会拼接两个字符串，数字和字符串相加时，依然是字符串的拼接，但是数字和字符串相减时，就变成算术运算了

比较

比较运算 == === != > < >= <=

逻辑运算符 && || !

三元运算 ：

c = (a > b) ? a : b (表示将a和b中的较大者赋给c)

条件判断 if/else

选择判断 switch

循环 for while do/while

使用break来终止循环，使用continue来跳过此次循环

for循环中有一个特殊的for/in循环可用来遍历对象的属性

var a = {x:"a", y:"b", z:"c"}

var i;

for(i in a){

document.write(i);

}

## 5、异常处理

try catch throw

    try{

        document.write(abc);

    }catch(err){

        try{

            if(confirm(err.message + "\n是否抛出一个自定义的异常")){

                var err = new Object();

                err.message = "抛出异常了";

                throw err;

            }

            else{

            }

        }catch(err){

            alert(err.message);

        }

    }

## 6、DOM对象（element对象）

通过HTML DOM，可访问HTML文档的所有元素，能访问到元素就能对元素进行各种操作

查看元素有四个方法：通过ID，通过签名，通过类名，通过名称查找

分别是

document.getElementById(id);

document.getElementsByTagName("p");(此例可以查出HTML中所有P标签元素)

document.getElementsByClassName(classname);(IE9开始支持)

document.getElementsByName();

<p class="a">Hello World!</p>

<div id="main">

<p>The DOM is very useful.</p>

<p>本例演示 <b>getElementsByTagName</b> 方法。</p>

</div>

<script>

var x=document.getElementById("main");

var y=x.getElementsByTagName("p");

var z = document.getElementsByClassName("a");

x.style.display="none"; //隐藏标签，隐藏后不占位置，为block时可见

x.style.visibility="hidden"; //隐藏标签，隐藏后依然占用位置，为visible时可见

x.style.color="blue"; //改变样式

x.innerHTML="改变内容"; // 改变内容

for(var i = 0; i < z.length; i++){

 document.write('Class为 "a" 的第'+ (i+1) +'段文本是：' + z[i].innerHTML + '<br/>');

}

for(var i = 0; i < y.length; i++){

 document.write('id 为 "main" 的 div 中的第'+ (i+1) +'段文本是：' + y[i].innerHTML + '<br/>');

}

</script>

document.getElementBy(id).attribute = new value(改变属性)

对DOM元素进行增加删除操作

var para = document.createElement(); //创建元素节点

var node = document.createTextNode(); //创建文本元素节点

para.appendChild(node); //向元素节点追加文本节点

上述的操作能建立一个节点，建立之后你还需要用document.getElementById(id).appenChild(para)将此节点追加到原有的HTML元素中

如下，会在id为div1的标签下面创建一个p标签，并且标签里的内容是‘这是新段落’

var para=document.createElement("p");

var node=document.createTextNode("这是新段落。");

para.appendChild(node);

var element=document.getElementById("div1");

elemet.appendChild(para);

使用appendChild(element)进行元素的插入（默认插入在最后）

使用insertBeforer(element1,element2)进行插入，可以将元素插入到element2上面

使用replaceChilde(element1,element2)可以用element1来替换element2

删除元素：删除元素必须通过父元素删除子元素,如果希望直接删除元素，可用parentNode来直接找到他的父元素如下：

var child = document.getElementById(id);

child.parentNode.removeChild(child);

## 7、事件（通过事件来触发对应的函数来获得需要的结果）（event对象）

常见的事件，比如：onclick(点击)，onmouseover(鼠标移动到元素上)，onmouseout(鼠标移开)，onblur(元素失去焦点)，onfocus(元素获得焦点)，onload(加载完成后)，等等

此处不列出所有事件，需要了解可以直接上网查询<http://www.w3school.com.cn/jsref/dom_obj_event.asp>

事件冒泡、事件捕获和事件委托

事件冒泡时，子元素先触发，父级元素后触发（使用return false来阻止事件冒泡，或者使用window.event.stopPropagation(),IE下使用window.event.cancelBubble=true）

事件捕获时，父级元素先触发，子元素后触发

事件委托：将事件委托给父级，进入函数后再去根据具体的子集做出相应的操作

如下所示：先通过父级来触发事件，然后通过event.target来获取到具体触发事件的元素，然后就可以对元素来进行操作

<html>

<head>

<script type="text/javascript">

    function getEventTrigger(event)

    {

          x=event.target;

          x.style.color="blue";

          alert("点击的是: " + x.innerHTML);

    }

</script>

</head>

<body >

<div id="p1" onmousedown="getEventTrigger(event)">

<p>这是第一行</p>

<p>这是第二行</p>

</div>

</body>

</html>

## 8、window对象

window对象表示浏览器窗口（还有框架，每个frame或iframe标签都会被创建一个window对象）,在JS中window对象是全局对象，所有的表达式都在当前的环境中，可以当成全局变量来使用，所以一般我们使用window对象时，不用加上window前缀，比如：

我们所使用的document也是window对象的属性之一，但是我们都是直接用的document而不是window.document,再比如使用弹窗alert()时，也不用写成window.alert();

window对象的方法基本都是对浏览器窗口或框架进行某种操作。

通过window对象，我们可以获取浏览器的一些属性，比如浏览器窗口的尺寸（在获取尺寸上面，不同的浏览器不同的版本，有时会有不同的方法，特别是老版本的IE）

window.innerHeight - 浏览器窗口内部高度 老版本IE需要使用 document.documentElement.clientHeight或document.body.clientHeight

window.innerWidth - 浏览器窗口内部宽度 老版本IE需要使用 document.documentElement.clientWidth或document.body.clientWidth

9、Location对象

也是window对象的属性，包含的是URL的信息,主要的属性和方法

location.hostname //web主机域名

location.pathname //当前页面的路径和文件名

location.port //web主机端口号

location.protocol //使用的web协议

location.search //返回从问号开始的URL

方法

assign(url) //加载新文档

reload() //重新加载当前文档

replace(url) //用新文档替换当前文档

10、screen对象

也是window对象的属性，里面包含的有关用户屏幕的信息，比如：显示屏幕的高度、宽度、颜色分辨率、屏幕刷新率等。

11、history对象

也属于window，包含用户访问过的URL

属性有： length //浏览器历史列表中URL的数量

方法： back() //加载history列表前一个URL（相当于后退操作）

forward() //加载后一个URL（相当于前进操作）

go() //加载某个具体的页面（使用go(-2)相当于后退两次操作）

12、navigator对象

也属于window，包含有关访问者浏览器的信息

比如浏览器名称，版本，语言，运行的操作系统，是否启动cookie等浏览器的信息

方法： javaEnabled() //检测浏览器是否启用JAVA

taintEnabled() //检测浏览器是否启用了data tainting(当允许数据污点时，一个窗口中的JS可以观看另一个窗口的属性)

## 9、一些没有讲到的常用对象属性和方法

13、一些可能用的到的函数操作和对象

alert(text) //警告框

confirm(text) //确认框

prompt(text,default) //提示框

var t = setTimeout(function,time) //在time毫秒后执行function

clearTimeout（t） //取消setTimeout();

document.cookie //存储和获取cookie;

MATH对象，进行算术操作，比如max(),min(),random(),E,PI,abs()等

Data对象，获取时间，Data(),getFullYear(),getHours(),getMinutes(),getSeconds()

RegExp对象，对字符串进行匹配

## 10、ajax

14、ajax

使用JS实现AJAX，需要使用对象XMLHttpRequest，此对象新的浏览器都支持，在window对象中，但是老板IE可能不支持，所以用如下方法拿到

var xmlhttp;

if (window.XMLHttpRequest)

{// code for IE7+, Firefox, Chrome, Opera, Safari

xmlhttp=new XMLHttpRequest();

}

else

{// code for IE6, IE5

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

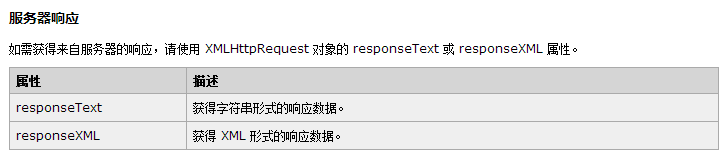
拿到XMLHttpRequest对象后，就可以使用它的方法进行异步请求

open(method,url, async)        //method:请求的类型，比如：GET、POST等，url:访问路径，async:true(异步）、false(同步）

send(String)        //发送请求，String仅用于POST请求(可以是JSON对象)

setRequestHeader(header, value);            //在HTTP头中加入参数

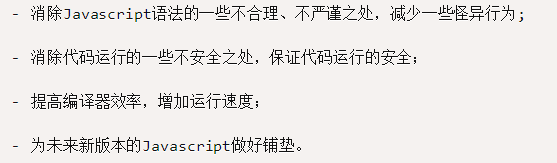




这部分没有讲

14、上面未写出的东西

严格模式：使用"use strict"启动



类型检测：

typeof（只能判断基本类型）        instanceof        Object.prototype.toString（不能判断自定义类型，不能判断基本类型，会自动转为包装类型）        constructor            duck type

call/apply方法，代替对象调用方法

包装类：  
String和string，Boolean和boolean,Number和number        和JAVA类似，但是基本类型可以用包装类的属性和方法，JS会自动将其转换为包装类进行执行，如下：

var a = "string";

var b = new String("string");

console.log(a.length);        //6

console.log(b.length);        //6

a.t = "abc";

b.t = "abc";

console.log(a.t);        //undefined

console.log(b.t);        //abc

prototype属性（对象类型的原型引用），里面包含\_proto\_和constructor

原型链：Object.prototype-->Function.prototype-->Object

Object.\_proto\_ === Function.prototype                Function.\_proto\_ === Function.prototype        Function.prototype.\_proto\_ === Object.prototype

普通对象没有prototype属性，函数对象才有prototype,prototype里面存的就是函数的实例对象，即func.prototype = new func();

get/set方法，以及有get/set方法的原型链属性查询

对象内设置隐性属性，前面加$即可

属性的各种标签，获取：Object.getOwnPropertyDescriptor(对象名，属性名)

使用Object.defineProperty(对象名,属性名，{参数}来为对象添加属性，参数有：{value：值，writable:是否可写，enumerable:是否可枚举, configurable:是否可修改，get: function(){}, set: function(){}}  
也可以使用Object.defineProperties(对象名,{ 属性名 : {参数}， 属性名 : {参数} });

对象的标签：\_proto\_（原型标签），class（对象类型标签），extensible(对象是否可扩展)

Object.isExtensible(obj)对象是否可扩展        Object.preventExtensions(obj)将对象设置为不可扩展

Object.seal(obj);将对象设置为不可扩展，同时将所有属性的configurable设置为false

Object.freeze(obj);将对象设置为不可扩展，同时将将对象所有属性设置为writable:false,configurable:false

序列化

JSON.stringify(obj);//属性值为undefined时，不会出现在序列化中，NaN,Infinity都会变成null，Date（）会成字符串格式，

反序列化JSON.parse(json);

自定义序列化

    var obj = {

        x:1,

        y:2,

        o:{

            o1:1,

            o2:2,

            toJSON:function(){

                return this.o1 + this.o2;

            }

        }

    };

JSON.stringify(obj);        //"{"x":1,"y":2,"o":3}"

重写toString之类的对象方法

数组：每个元素都可以是不同类型。最大2^23-1长度

new Array(100)        //长度为100

new Array（true,false,null,1,2）;        //创建元素

new可以省略

delete arr[0]，仅删除值，arr[0] = undefined;

arr.push(3)        在数组尾部添加一个元素3,

arr.unshift(3)    在数组头部添加一个元素3

arr.length -=1;        可以删除数组尾部的一个元素

arr.pop()            删除尾部元素

arr.shift()            删除头部元素

遍历：for        for/in(不保证顺序)

稀疏数组：有些有值了有些没有值

数组的原型Array.prototype

join()将数组转成字符串，默认用‘，’隔开，带参数可以指定分隔符

new Array(3).join("a");            //会返回aa,因为数组为空，就会把a当做分隔符，长度为3，有2个分隔符

reverse()        //将数组逆序

sort()            //排序（数组本身会变化），默认按照字母，数组也是按照字符串比较而不是按照数字大小（需要自定义sort）

arr.sort(function(a,b){ return a -b } )

concat()        //合并数组（不修改原数组）

slice(起始位置，结束位置)            //返回数组片段，可以用正数或负数，-1是最后一个元素，-2是倒数第二元素

splice(起始位置，删除元素个数，添加参数值，添加参数值...)        //数组拼接（删除和添加元素），删除元素可以在返回值中拿到，原数组是操作后的值

forEach()            //迭代 （IE9开始支持）

map()        //映射

filter()        //过滤

every()数组每个元素都要符合条件/some()只要一个符合条件就可以        //数组判断

reduce()（从左往右）/reduceRight()（从右往左）            //数组元素两两进行操作

如下操作求最大值

var arr = [3,9,6]

var max = arr.reduce(function(x,y){    return x > y ? x : y;} );    //max==9

indexOf(检索数字，检索开始位置),lastIndexOf() （从右到左找）           //数组检索

Array.isArray(参数)        //ES5提供，判断是否是数组

函数   作用域

创建

函数声明            function functionname(){}

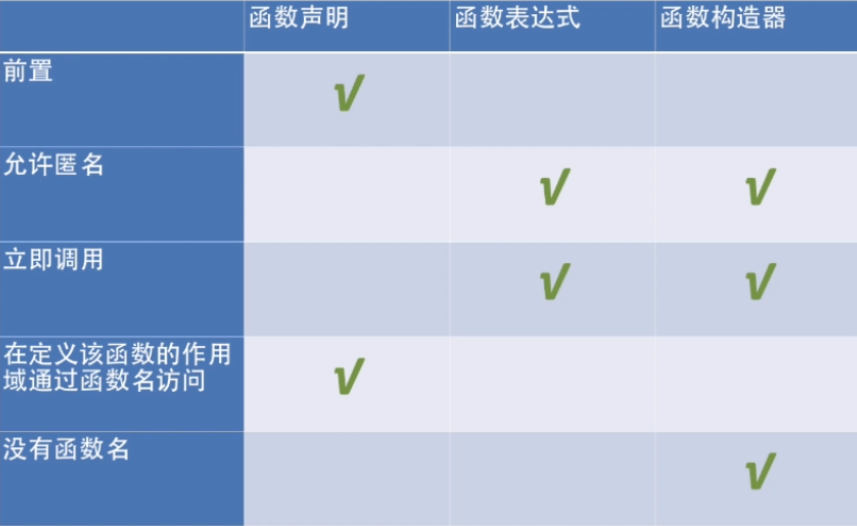
函数表达式        var a = function(){}            (function(){})();//立即执行的函数表达式        return function(){}            var a = function functionname(){}（调试的时候可以使用，其他匿名就可以了）

函数声明会被前置，函数表达式不会

使用构造器创建（不常见）

var func = new Function('a','b','console.log(a+b)');    (拿不到外部的局部变量，而且他的变量外部无法访问)，后面加上（）可以立即执行

func(1,2);        //3



this

全局中this就是window

函数里面this也是window,严格模式下函数里面是undefined

对象里方法的this，会指向对象

原型链上的this，可以指向原型链也可以指向对象本身

构造器中的this，new创建时会指向原型对象

call()/apply()        //一般模式传入null或undefined时，this会是window，严格模式下传什么就是什么

bind()        //IE9支持，依然是传入值代替this,但是可以获得一个代替后的方法，可以一直使用，还有一个功能，将函数颗粒化（返回的函数没有prototype属性）

如下：

柯里化函数

function add(a,b,c){

    return a + b + c;

}

var func = add.bind(undefined,100);

func(1,2);//103

函数属性和参数

func.length；    拿到形参的个数

func.name;        拿到函数名称

实际参数，在函数里用arguments来拿到，arguments.length//实际参数的个数，arguments[i]对应参数的值，可以改变值，但是不能增加值（严格模式下不能修改值）

严格模式下还禁止使用arguments.callee === func;//true

闭包(简单理解就是函数里面的函数可以访问外面函数的变量)        灵活方便，可以用来封装        容易造成的问题：空间浪费，内存泄露，性能消耗

闭包例子如下：

function student(){

        var name = "abc";

        var age = 20;

        return function(){

                retrun name;            //里面可以访问外面的变量

        }

}

var test = student();

test();        //abc            student()方法已经结束了，但是test()依然可以拿到student()里面的值

作用域：全局、函数、eval;

使用！表示是函数表达式而不是函数声明下面两种写法都是直接进行执行，而不是声明一个函数

！function(){}();                (function(){})();

ES3执行上下文

变量对象和执行上下文            变量函数声明会前置，名称有冲突时，函数直接覆盖，变量直接忽略

var a = 123;

function a(){};

console.log(a);        //123

实际执行过程（声明都被提前了）

声明函数a，忽略变量a声明，初始化完成后执行赋值语句a=123,所以最后a的值是123

OOP

继承(Student继承Person)

function Person(){}

function Student(){}

Student.prototype = Object.create(Pserson.prototype);    //ES5开始支持

Student.prototype.constructor = Student;

考虑到兼容问题的话，要用create方法需要这样

if(!Object.create){

    Object.create = function(proto){

        function F(){}

        F.prototype = proto;

        return new F

    };

}