**使用说明：**

1、本文档由民哥上课笔记和PPT内容综合整理而成，如有疏漏，请自行更正。

2、带“补充”字样的内容多来自互联网，或有利于辅助理解。

3、结合导航窗格和搜索框使用更佳。开启方法：

Word：视图→显示→导航窗格

WPS：视图→导航窗格→靠左/靠右

4、本文档仅供参考。

# 08.08

学习线路：

javascript 初级

javascript 中级 全面 深度 实际 会

javascript 高级

jQuery 库

zeptojs 库

nodejs

npm/cnpm yarn/tyarn bower

commonjs 规范

es2015-2019

webpack grunt gulp

angular

react

jsx react react-router react-reudx/redux hooks ant design /pro dva umi

vue

vue vue-router vuex element ui iview

type-script

ajax promise axios fetch

--------------------------------

学习方法：

课前预习：

粗略看视频一遍

精细看视频，看一个例子，练一个例子，10遍

上课认真听讲：

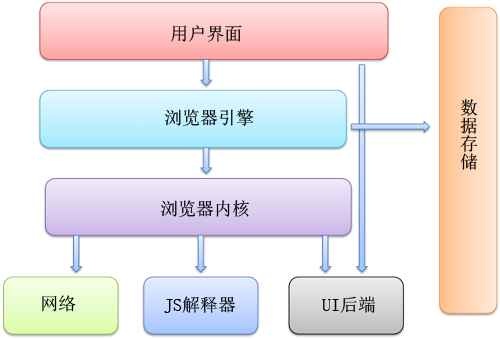
不懂上课举手提问，重讲

课后认真复习：

理解概念，多练实例，10遍

--------------------------------------------

1、浏览器架构：



浏览器主流内核：

①微软 Microsoft IE Trident

②谷歌 google Chrome webkit

③mozllia基金会 Firefox gecko

④opera ASA Opera presto→webkit

2、JavaScript概念：基于对象和事件驱动并具有安全性能的脚本语言。

JavaScript具有以下几个基本特点：

是一种脚本编写语言

基于对象的语言

安全性

动态性

跨平台性

3、JavaScript在网站中的应用：

①网站前端中的应用：

网站中特效：例如焦点图、二级菜单、放大镜等

表单验证

ajax：例如检测用户名是否被占用、登录验证、站内搜索引擎关键字提示等

html5很多功能需要js实现

②网站后台中的应用：

Node.js

4、java和javascript的区别：

|  |  |  |
| --- | --- | --- |
|  | Java | JavaScript |
| 公司 | sun-oracle | NetScape网景 |
| 前身 | oak | livescript |
| 更名 | java | javascript |
| 用途 | 软件、手机、游戏 | web web app |
| 语言 | 编程语言 | 脚本语言 |
| 声明 | 强类型 | 弱类型 |
| 嵌入方式 | <applet>  </applet> | <script>  </script> |

5、ecmascript脚本标准 w3c

javascript

jscript

actionscript

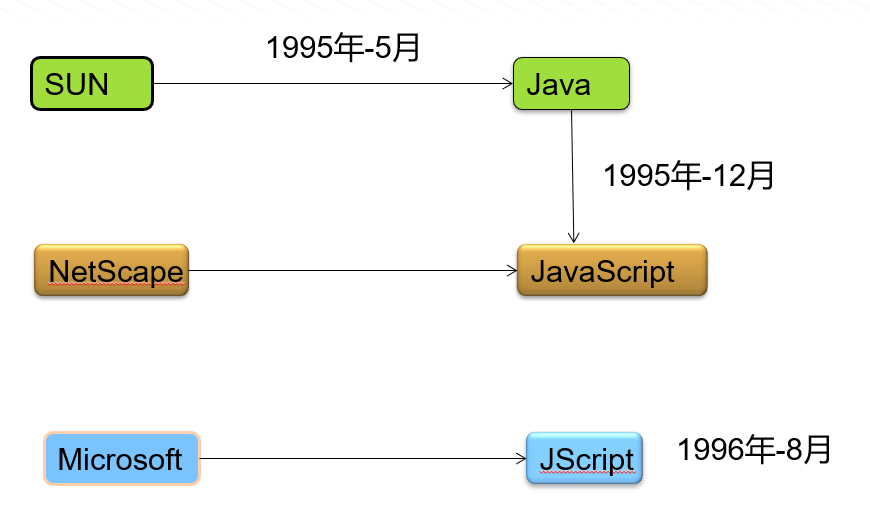
ecmascript和javascript/jscript/actionscript之间的关系：

ecmascript是javascript/jscript/actionscript的标准；

javascript/jscript/actionscript是ecmascript的具体实现

说明：

JavaScript和Jscript分属不同厂商浏览器。



ecmascript 3/5

ecmascript 6

6、JavaScript浏览器兼容性：

答：主要是IE6/7/8的私有写法造成的。

举例：绑定事件——事件监听器方法

IE6/7/8 attachEvent

IE 9+、其他浏览器 addEventListener

IE9/10 两种都支持

7、JavaScript的嵌入方式

内嵌方式：

<script type=“text/javascript”>

js代码

</script>

外联方式：

首先将js代码保存成独立的.js文件

<script type=“text/javascript” src=“js文件.js”>

</script>

8、JavaScript的注释

单行注释： //注释内容

多行注释： /\*注释内容\*/

9、JavaScript开发工具及调试工具

开发工具：

notepad 初学推荐

editplus 初学推荐

dreamweaver 初学推荐

sublime 专业开发推荐

hbuilder 专业开发推荐

webstorm 专业开发推荐

调试工具：

firebug

chrome调试器

浏览器测试：

firefox

IE

IETESTER

10、JavaScript、jquery、ajax、json简介：

JavaScript

jQuery、YUI、extjs

ajax

Asynchronous Javascript And XML（异步的javascript and XML）

javascript+css+dom+XMLHttpRequest

json 轻量级数据格式

# 08.11

变量是存取数据的内存空间。

JavaScript变量的命名规则：以字母或下划线开头，可以包含字母数字下划线，不能包含特殊字符(空格@&等)，不能是JavaScript关键字和保留字

1.关键字：break case catch continue default delete do else finally for function if in instanceof new return switch this throw try typeof var void while with

2.保留字：abstract boolean byte char class const debugger double enum export extends fimal float goto implements import int interface long mative package private protected public short static super synchronized throws transient volatile

JavaScript命名规范：

1、默认使用camel(驼峰)命名法：第一个单词的首字母小写，后面每个单词首字母都大写，比如：

addEventListener

getElementById

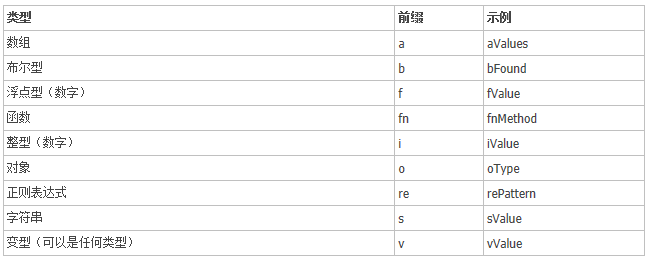
2、构造函数或类名都是用pascal(帕斯卡)命名法：所有单词的首字母都大写，比如：

new Date();

new Array();

new FormData();

3、匈牙利命名法：在以pascal命名的变量前边，附加一个小写字母(或小写字母序列)，用以说明该变量的类型。如下图：



说明：实际开发中使用的命名方式，是由公司项目的接口文档决定的。

-----------------------------------------------------------------------------------------------

变量的创建/声明/定义方法：

①先创建后使用

var x; //创建变量

x=10; //使用变量

②直接使用，由系统自动创建变量

y=7;

说明：这样创建的变量，不管在函数内/外创建，都是全局变量（开发的时候不可能用，因为污染顶级作用域）

创建变量同时初始化：

var z=8;

注：同时创建多个变量：var x,y,z;

同时创建多个变量并初始化：var x=10,y=7,z=8;

----------------------------------------------------------------------

JavaScript中，函数就是作用域。

变量的作用域(带var的情况)：

1.全局变量：在函数外创建的变量

函数外有效

函数内有效

2.局部变量：在函数内创建的变量

仅在函数内有效

不加var关键字，创建的变量无论在函数内外都是全局变量。

----------------------------------------------------------------------------------

JavaScript常用数据类型——不同类型数据可以做不同运算

JavaScript中基本数据类型(四种)：

1.number类型：可以做所有的算术运算

3 3.14 NaN Infinity

补充：Infinity-Infinity=NaN

2.boolean类型：布尔逻辑运算

true false

3.undefined类型：只有一个值undefined，表示变量创建，未赋值

4.string类型：凡是用""或''包裹的都是字符串

字符串型只能做字符串连接运算

----------------------------------------------------

typeof：数据类型检测运算符，得到的值是小写的字符串

"number" //4种基本数据类型

"string"

"boolean"

"undefined"

"object" //2种对象类型

"function"

说明：

instanceof：判断某个对象是否是某个类的实例，也就是判断这个对象是否是这个构造器函数创建的，换句话说：判断对象的类型。

var arr=[22,3,4];

arr instanceof Array //true

var obj={name:"tom",age:19};

obj instanceof Object //true

--------------------------------------------------

undefined和null的区别：

1.概念

undefined 代表变量创建了，但是没有初始化

null 代表对象有值，值为空

2.注意

undefined==null //true

undefined===null //false

typeof undefined //undefined

typeof null //object

----------------------------------------

在JavaScript中，以下几种情况会被当做false：

false

NaN

null

undefined

"" '' //空字符串

0

# 08.13

“+”分两种情况：

1.左右两边都是数字，那么就是加法

2.如果左右两边或一边是字符串，那么会强制将另一边转换为字符串做连接，此时的“+”是字符串连接运算符

表达式：由运算符连接操作数组成的式子，不管表达式多长，最终都是一个值。

运算符的分类：

一、按照操作数的个数分类

1.单目运算符：

-负号 ++ -- !取反

2.双目运算符（大部分运算符）：

+加或字符串连接运算符

- / \* % > <

3.三目运算符（只有一个）：条件运算符

表达式?值1:值2

说明：react vue中用条件运算符代替if-else

二、按照运算类型分类

1.算术运算符

+ - \* / % -负号 自增++ 自减--

++1：

①表达式中只有++运算，没有其他运算，就是自己加1（常用）

②表达式中除了++外，还有其他运算或输出：

(1)++在变量前面，变量先自己加1，然后再做其他运算或输出

(2)++在变量后面，变量先做其他运算或输出，然后再自己加1

说明：变量的值要想更改

赋值=

++ --

+= -= \*= /=

2.比较运算符：比较运算表达式的值是布尔值true或false

(1)自动进行数据类型转换（因为只有数据类型相同才能比较）

> >= < <= == !=

例：

undefined==null //true

"0"==0 //true

(2)不进行数据类型转换

===：类型相同，并且值相同才为true（react、vue用）

!==：类型不同，或者值不同才为true

例：

3.0===3 //true

undefined===null //false

3.布尔逻辑运算符

(1)逻辑与&&(简短与)：and，左右两边都为true才为true，有false就是false

(2)逻辑或||(简短或)：or，左右两边有一边是true，就是true，全false才是false

举例：事件对象

IE6/7/8支持 window.event

IE9+支持 e

兼容写法：var x=e||window.event;

(3)逻辑非!：取反

4.条件运算符

5.自反赋值运算符：+= -= \*= /= %=

6.位运算符：

(1)按位与&：有0就是0，全1才为1（用于消除指定位，要想消除某些位，就和0与）

例：8&3

0000 1000

& 0000 0011

————————

0000 0000

应用：IE浏览器中，Error对象的number属性，想要获取错误号。

number 高16位为机器码

低16位为错误号

做法：清除高16位，保留低16位

&0x0000ffff

注：0000ffff(16)=0000000000000000 1111111111111111(2)

(2)按位或|：有1就是1，全0才是0

例：8|3

0000 1000

| 0000 0011

————————

0000 1011

(3)按位异或^：不同是true，相同是false

例：8^3

0000 1000

^ 0000 0011

————————

0000 1011

(4)位非运算~

(5)左移位运算符<<，实现整体向左移动低位补0的功能（左移n位就相当于乘以2的n次方）

(6)右移位运算符>>，当移动的是有符号数，左边空出的位用数的符号位填充，即正数补0，负数补1。向右移动超出的位将被丢弃（右移n位相当于除以2的n次方）

(7)无符号右移位>>>，当数是无符号数时，右移后在左边空出的位上填充0

补充：

有符号数和无符号数是针对二进制来讲的。

有符号数用最高位作为符号位，“0”代表“+”，“1”代表“-”；其余数位用作数值位，代表数值。

无符号数全部二进制均代表数值，没有符号位。即第一个“0”或“1”不表示正负。

**运算符优先级：**

|  |  |
| --- | --- |
| . [] () | 字段访问、数组下标、函数调用以及表达式分组 |
| ++ — - ~ ! delete new typeof void | 一元运算符、返回数据类型、对象创建、未定义值 |
| \* / % | 乘法、除法、取模 |
| + - + | 加法、减法、字符串连接 |
| << >> >>> | 移位 |
| < <= > >= instanceof | 小于、小于等于、大于、大于等于、instanceof |
| == != === !== | 等于、不等于、严格相等、非严格相等 |
| & | 按位与 |
| ^ | 按位异或 |
| | | 按位或 |
| && | 逻辑与 |
| || | 逻辑或 |
| ?: | 条件 |
| = oP= | 赋值、运算赋值 |
| , | 多重求值 |

数制介绍：

1.二进制：0 1

2.十进制：0123456789

3.十六进制：0123456789ABCDEF

注：在JavaScript中，数值用32位2进制补码表示。

一、任何进制转换为10进制：任何进制数，按照位权展开就是10进制

补充：位权，是指数制中每一固定位置对应的单位值。例如二进制第2位的位权为2，第3位的位权为4；八进制第2位的位权为8，第3位的位权为8^2。

253(10)=2\*10^2+5\*10^1+3\*10^0

1、二进制转10进制

例1：

1000110

1\*2^6+0\*2^5+0\*2^4+0\*2^3+1\*2^2+1\*2^1+0\*2^0=64+4+2=70

即：1000110(2)=70(10)

例2：

100111

1\*2^5+1\*2^2+1\*2^1+1\*2^0=32+4+2+1=39

即：100111(2)=39(10)

2、16进制转10进制

例1：

46

4\*16^1+6\*16^0=70

即：46(16)=70(10)

例2：

71

7\*16^1+1\*16^0=113

即：71(16)=113(10)

同样的，其他进制转换为10进制，也是按照位权展开即可。

---------------------------------------------------------------------------------

二、10进制转其他进制：

0 1 2 3 4 5 6 7 8 9 10

1 2 4 8 16 32 64 128 256 512 1024

1、10进制转二进制

例1：

39=32+4+2+1

即：39(10)=100111(2)

例2：

49=32+16+1

即：49(10)=110001(2)

说明：这种转换方法是一个反推的过程。

2、10进制转16进制

①先将10进制转成2进制

②然后再将2进制转换成16进制，从低位到高位4位2进制对应1位16进制

例：

72(10)→1001000(2)→0100 1000(2)→48(16)

----------------------------------------------------------------

三、二进制转16进制：4位二进制对应1位16进制

例：

1000 0011

8 3

即：10000011(2)=83(16)

-----------------------------------------

JavaScript规范：十六进制0x

任何进制在JavaScript中输出都会自动转换为十进制，例如0x41→65

# 08.15

第一阶段：

1.算法 排序 7个

2.数组去重

3.变量提升

4.函数节流与去抖

5.浏览器缓存机制——缓存原理，强制缓存和协商缓存

6.http协议

7.http协议常用状态码

---------------------------------

编程：把人类思想转换为计算机能读懂的语言

基本语法：

1.变量

2.数据类型

3.运算符和表达式

4.结构

JavaScript中程序结构：C C++ Java PHP

1、顺序结构：没有结构就是顺序结构，每条语句从上到下，依次执行。

2、分支结构：

(1)单分支：要么做，要么不做

语法：如果条件为true,做if(){}所有语句，如果条件为false直接跳过if(){}

if(条件){

语句;

语句;

....;

}

(2)双分支：要么做这个，否则做那个，二选一

语法：如果条件为true就执行if(){}所有语句，否则执行else{}所有语句。

if(条件){

语句;

语句;

....;

}else{

语句;

语句;

....;

}

(3)多分支：多段条件

语法：

if(条件){

语句;

语句;

....;

}else if(条件){

语句;

语句;

....;

}else if(条件){

语句;

语句;

....;

}else{

语句;

语句;

....;

}

例1：

var score=prompt("请输入分数：",0);

if(score<60){

alert("不及格");

}else if(score<70){ //score>=60&&score<70

alert("及格");

}else if(score<80){ //score>=70&&score<80

alert("中等");

}else if(score<90){ //score>=80&&score<90

alert("良好");

}else if(score<100){ //score>=90&&score<100

alert("优秀");

}

例2：if-else嵌套

var score=prompt("请输入成绩：",0);

if(score<60){

alert("不及格");

}else{

if(score<70){

alert("及格");

}else{

if(score<80){

alert("中等");

}else{

if(score<90){

alert("良好");

}else{

alert("优秀");

}

}

}

}

补充：

①获取用户输入内容的方法prompt

window.prompt(“提示信息”,”默认值”);

获得用户输入内容（都是字符串类型数据），并返回用户输入内容

②if-else嵌套的效率要高于if-else-if，但if-else-if的可读性更好。

③if-else-if和if if的区别：

if if分支结构中的每个if判断语句都是独立的，执行完第一个if继续往下执行；

if-else-if分支结构是一个整体，尽管if-else-if结构列出了多种可能但是只能选择一个合适的执行。

if if结构的语法：

if (条件) {

如果条件为true时执行的代码

}

if (条件) {

如果条件为true时执行的代码

}

if-else-if结构的语法：

if (条件1) {

条件1为true时执行的代码块

} else if (条件2) {

条件1为false而条件2为true时执行的代码块

} else {

条件1和条件2同时为false时执行的代码块

}

3、多选一：值相等跳转到指定位置执行

switch(表达式或值){

case 值1:

语句;

语句;

....;

break;

case 值2:

语句;

语句;

....;

break;

case 值3:

语句;

语句;

....;

break;

default:

语句;

}

说明：

①break跳出当前switch-case结构。如果没有break，不会报错，只是多选一结构会产生逻辑错误。

②react的数据流管理工具redux中的reducer会使用

4、循环结构：

(1)while循环：直到型循环，直到条件为false退出循环

while(条件){

语句;

语句;

....;

}

(2)do-while：至少做一次

do{

语句;

语句;

....;

}while(条件);

(3)for循环：

for(循环变量初始化;条件;自己加1){

语句;

....;

}

------------------------------

1、break：中断循环，跳出循环结构

2、continue：中断本次循环，直接进入下次循环

3、label：使用label语句可以在代码中添加标签，以便将来使用

语法： label:statement

作用：终止外循环

例：

<script type="text/javascript">

outer:

for(var i=1;i<=3;i++){

for(var j=1;j<=2;j++){

if(j==2){

break outer;

}

console.log("i=",i,"j=",j); //i=1,j=1

}

}

</script>

说明：continue同上

4、with（很少用）

例：

with(Math){

console.log(PI);

console.log(abs(-2));

console.log(sqrt(2));

console.log(random());

}

//与下边的代码等价

console.log(Math.PI);

console.log(Math.abs(-2));

console.log(Math.sqrt(2));

console.log(Math.random());

# √08.27

前端开发三层：

html/xhtml 结构层 .html

css 表现层 .css

js 动作层 .js

JavaScript：和用户交互，和后台交互，业务逻辑编写。

JavaScript初级

JavaScript中级

JavaScript高级

JavaScript架构

jQuery

zeptojs

HTML5 api

nodejs

express

npm/cnpm yarn/tyarn bower

commonjs规范

grunt gulp webpack4

react

jsx react生命周期方法 react-router redux react-redux react-saga dva umi roadhog ant design ant design pro token react hooks

vue

vue生命周期方法 vue-router vuex elementui ant design for vue

typescript

echarts hightcharts 图形库

---------------------------------------

JavaScript数据类型分类：

一、基本数据类型 (标量类型) ：变量不论存储、赋值、比较、当做参数传递、返回值，传递的都是变量的值。

1、number：数值型

3 3.14 NaN Infinity

2、string：字符串型（文本）

"" ‘’包裹的都是字符串

3、boolean布尔型：true false

4、undefined空类型：只有一个值undefined

二、除了基本数据类型外，其他类型都是对象类型

对象类型：不论是对象类型存储、赋值、当做参数传递、比较、返回值，传递的都是对象内存块首地址。

1、用户自定义对象

{属性:值,属性:值,...}

2、内置对象：String；Math；Date；Function；Number；Error；RegExp；Array

3、浏览器对象BOM：window、location、document

4、DOM对象

5、事件对象

6、ajax对象

说明：为什么对象赋值、比较、当做参数传递，针对的都是内存块的首地址？

答：地址是一个数字，而对象通常会占用很大的内存。电脑的内存相当有限，如果拷贝是对象的内存块，那么速度会很慢，所以拷贝对象的内存块的首地址（数字），速度快。

---------------------

函数是完成指定功能的程序段，可以反复调用，减少代码冗余。

1、功能相似的程序段，不封装成函数

缺点：代码冗余；维护困难

2、功能相似的程序段，封装成函数

优点：减少代码冗余；方便维护

说明：ajax访问过程是6步，故将其封装成一个函数，可以反复调用；

事件监听器方法绑定事件，也会封装成函数，方便写兼容。

jQuery→JavaScript开发中常用的功能都封装成函数，优点：

减少代码冗余

开发方便

解决兼容性

jQuery就是用JavaScript实现的一组函数。

--------------------------------------------------------

1、创建/声明/定义一个函数

function 函数名([形参,形参,……]){

语句;

语句;

语句;

…...;

[return 表达式或值;]

}

2、函数调用

函数([实参,实参,...]);

说明1：函数不调用不会执行

函数()，“()”代表立即执行函数语句体

函数是对象，所以函数名存储的是内存块首地址

说明2：函数如果没有返回值return，那么函数的返回值就是undefined

说明3：函数的返回值和函数输出之间没有任何关系

说明4：函数的实参和形参个数可以不同

---------------------------------------------------------

1、无参函数

函数没有参数，不需要参数，一般只完成固定的功能。

2、单参函数

单参函数：只有一个参数

3、多参函数

多参函数：有多个参数

4、递归函数

递归函数：函数本身自己调用自己

function 函数名(){

.....

函数名();

}

5、匿名函数

匿名函数：没有函数名称的函数

function([形参,形参,...]){

语句;

语句;

…...;

[return 表达式或值;]

}

为了方便调用，一般写成这样：

var test=function([形参,形参,...]){

语句;

语句;

…...;

[return 表达式或值;]

}

6、带返回值的函数

带返回值的函数：函数定义中有return语句

**7、构造函数**

构造函数：用来创建对象的函数（JavaScript中，用 new 关键字来调用的函数）

例：

function Student(name,age,sex){

this.name=x;

this.age=y;

this.sex=z;

this.study=function(){

alert("持之以恒~");

}

}

构造器函数的使用：

var s1=new Student("Tom",22,"男");

补充1：当一个函数创建好以后，我们并不知道它是不是构造函数，即使函数名为大写，我们也不能确定。只有当一个函数以 new 关键字来调用的时候，我们才能说它是一个构造函数。

补充2：使用new关键字，那么可以少做四件事情：

①不用创建临时对象，因为new会帮你做（使用this就可以访问到临时对象）；

②不用绑定原型，因为new会帮你做（new为了知道原型在哪，所以指定原型的名字为prototype）；

③不用return临时对象，因为new会帮你做；

④不要给原型想名字了，因为new指定名字为prototype。

8、回调函数

回调函数：被当做实参传递的函数。

当一个函数的返回值传递给另一个函数的时候使用回调函数。

function test(a,b){

return a()+b();

}

function one(){ //回调函数（被当做实参传递的函数）

return 1;

}

function two(){ //回调函数

retrun 2;

}

test(one,two); //函数调用

匿名回调函数：回调函数是匿名的

function test(x,y){

return x()+y();

}

test(function(){return 1;},function(){return 2;});

9、自调函数

匿名函数自己调用自己，函数定义完就执行。

**自调函数，和普通函数的区别是编译到该函数时立即执行，所以也叫立即执行函数。它执行过后会销毁自己的执行期上下文。**

当我们给匿名函数外面加上一层（）的时候，匿名函数就变成了函数表达式，在JavaScript编译的时候就会被立即执行；

除了用（）把匿名函数包起来之外，后面还需要再跟一个（），可以通过此（）给匿名函数传参。

1.

(

function(){

alert("红红火火恍恍惚惚");

})

();

2.

(

function(name){

alert('hello'+name+'!');

})

("GaoYuanYuan");

说明：多个人写的代码合并到一起，极有可能出现命名冲突，所以要使用匿名自调函数，进行作用域隔离。

那么，使用匿名自调函数，解决了变量或对象命名冲突问题，但是如果有些对象或函数需要多人共享怎么办？

答：把局部变量$升级成全局变量↓

function ajax(){}

$.ajax=ajax;

window.$=$;

10、内部（私有）函数

内部（私有）函数：位于函数内部的函数。

例：

//私有函数：函数内部的函数

function test(){

var x=100;

//私有函数a

function a(){

alert(x);

}

a();

}

test();

11、返回函数的函数

返回函数的函数：函数的返回值是一个函数。

①//执行函数a，返回一个函数

function a(){

alert("a函数");

return function(){

alert("返回函数");

};

}

②//执行返回的函数

var test=a();

③//如果想让返回的函数立即执行

test();

12、能重写自己的函数

由于一个函数可以返回另一个函数，因此我们可以用新的函数来覆盖旧的函数。

例：

function test(){

test=function(){

alert("新函数");

}

alert("旧函数");

}

test(); //旧函数

test(); //新函数

# √08.28

系统内置函数：

内置函数也叫系统函数，前面不需要加任何对象名，就可以在任何对方直接使用。

1、encodeURI("字符串"); //对字符串进行URI编码

应用场景：存储cookie，cookie变量的值要求是URI编码（详见第13章）

2、decodeURI("字符串"); //对encodeURI()函数编码过的URI进行解码

应用场景：取cookie变量的值，取到之后是URI编码，得转码后才能识别

3、parseInt(); //解析一个字符串，并返回一个整数。转换失败返回NaN。

4、parseFloat(); //解析一个字符串，并返回一个浮点数。转换失败返回NaN。

5、isNaN(x)： //用于检查其参数是否是非数字值。

说明：如果x是特殊的非数字值NaN（或者能被转换为这样的值(数字类型字符串)），返回的值就是true。如果x是其他值,则返回 false。

补充：isNaN() 函数通常用于检测parseFloat()和parseInt()的结果，以判断它们表示的是否是合法的数字。当然也可以用isNaN()函数来检测算数错误，比如用0作除数的情况。

NaN：number类型，代表所有非数值

说明：NaN!=NaN；NaN!==NaN

6、eval(字符串); //注意：eval里的内容要加双引号

eval函数功能：将字符串当做js执行。

应用场景：前后端分离开发，ajax进行前后端交互，后台给前端json字符串，前端需要将json（JavaScript Object Notation，JavaScript对象表示法）字符串转换为js对象。

eval优点：兼容性好

eval缺点：能够将字符串直接当成js执行，如果不控制，容易出现安全隐患

json字符串：

'{"name":"tom","age":19,"sex":"男"}'

json字符串转换为js对象：

eval("("+json字符串+")");

补充：

为什么eval这里要写成("("+data+")")呢？

原因在于：eval本身的问题。由于json是以“{}”的方式来开始以及结束的，在JS中，它会被当成一个语句块来处理，所以必须强制性的将它转换成一种表达式。

加上圆括号的目的是迫使eval函数在处理JavaScript代码的时候强制将括号内的表达式（expression）转化为对象，而不是作为语句（statement）来执行。举一个例子，例如对象字面量{}，如若不加外层的括号，那么eval会将大括号识别为JavaScript代码块的开始和结束标记，那么{}将会被认为是执行了一句空语句。所以下面两个执行结果是不同的：

alert(eval("{}"); //return undefined

alert(eval("({})"); //return object[object]

------------------------------------------------------------

函数的作用域

前提：这里指全部都通过var创建的变量或对象（**没有用var关键字，使用直接赋值方式声明的都是全局变量）**

1.[全局变量](变量作用域-全局变量.html)：在函数外创建的变量，称为全局变量，在函数内外均有效。

var x=10;

function test(){

alert("全局变量在test函数中"+x); //10

function a() {

alert("全局变量在私有函数a中"+x); //10

}

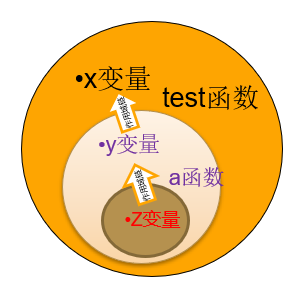
a();

}

2.局部变量：在函数内部创建的变量，称为局部变量，仅在当前函数内及子函数内有效，函数外无效。

var *x*=10;

function *test*(){

var y=20;

alert("全局变量在test函数中"+*x*); //10

alert("局部变量在test函数中"+y); //20

//alert("局部变量在test函数中"+z); //报错z未定义

function *a*(){

var z=30;

alert("局部变量在函数a中"+y); //20

}

*a*();

}

注：x是全局变量，在整个作用域有效

y是局部变量，在test函数中有效，在a函数中有效

z是局部变量，只在a函数中有效

问题：如果想在函数外，使用函数内的变量或数据怎么办？

答：通过闭包函数实现

----------------------------------

1.闭包（closure）的概念：是指有权访问一个函数作用域中的变量的函数。创建闭包常见的方式，就是在一个函数内部创建另一个内部（私有）函数。

2.闭包是一个函数，能将创建的变量的值始终保持在内存中，以供本地环境使用。

3.使用闭包的主要作用是间接访问函数的内部数据。

闭包缺点：该回收的内存没有回收，造成内存浪费。

-------------------------------------------------------------------------

页面加载流程分为两个阶段：

第一个阶段：从上到下依次创建每个标记对应的DOM对象，遇到js执行js。

第二个阶段：从上到下依次渲染每个标记，之后触发window.onload事件。

--------------------------------------------------------------------------------------------------

JavaScript执行也分为两个阶段：

第一个阶段：预编译阶段（变量提升）

1.创建所有带var的变量和有名函数形参，并且赋值为undefined，但不会执行任何赋值语句。

2.创建所有有名函数【注意匿名函数不会被创建】，并赋值（方法体），但不会调用函数。

说明：赋值语句不参加预编译，匿名函数不参加预编译。

第二个阶段：执行阶段

从上到下依次执行语句

实例1：

alert(x); //undefined

var x=111;

alert(x); //111

实例2：

var x=2;

function test(){

alert(x); //undefined

var x=10;

}

test();

实例3：

test();

function test(){

var x=10;

alert(x); //10

}

实例4：

test(); //test is not a function(匿名函数不参加预编译)

var test=function(){

var x=10;

alert(x);

}

综合实例：

var a=0,b=0;

function A(a){

A=function(){ //注意没有var不参加预编译

alert(a+b++);

}

alert(a++);

}

A(1); //1传给形参，局部变量a弹出的值是1，自己加1后a的值为2，由于闭包局部a并没有销毁

A(3); //函数体已改为新的匿名函数，且传入的3形参接收不到，弹出a+b++=2，b++为1

# √08.29

**闭包实例：**

1、闭包-1

在函数外部直接调用局部变量x，报错

function *test*(){

var x=10;

return function(){

return x;

};

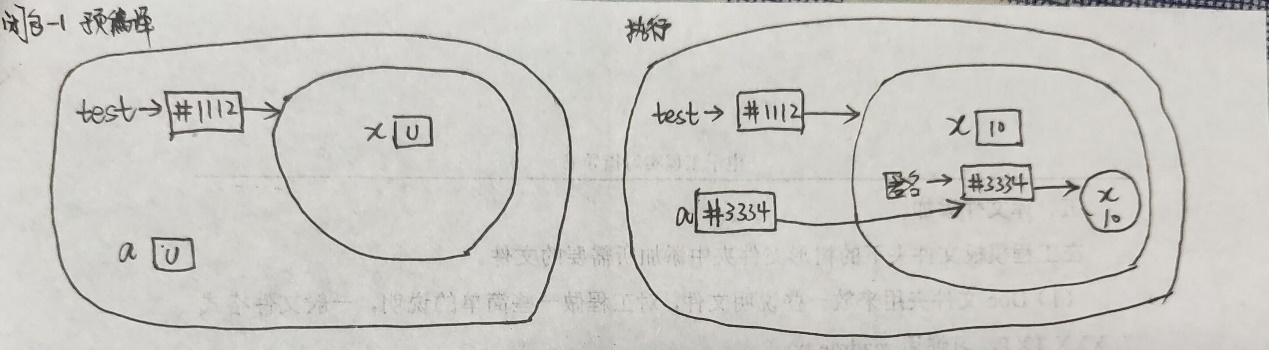
}

//alert(x); //调用局部变量x，报错未定义

var a=test(); //[调用](闭包1.html)

alert(a());

说明：a实际上就是闭包匿名函数。函数test中的局部变量x一直保存在内存中，并没有在test调用后被自动清除。



2、闭包-2

var y;

function *test*(){

var x=10;

y= function(){

return x;

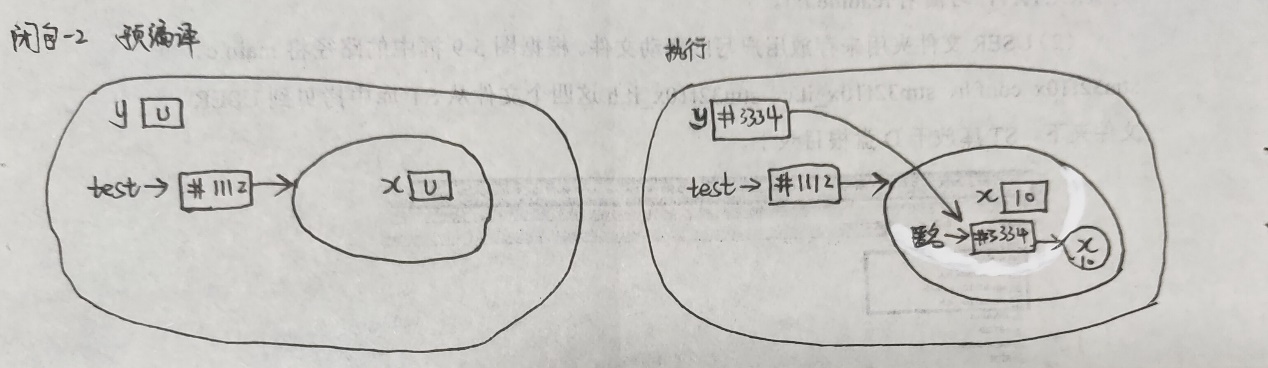
}

}

test(); //调用函数

alert(y()); //[调用](闭包2.html)

说明：y实际上就是闭包y函数。函数test中的局部变量x，并没有在test函数调用后删除，x一直保留在内存中。y函数依赖于test函数都一直保存在内存中，不会被垃圾回收机制（garbage collection）回收。



3、闭包-3

function *test*(*arg*){

var *y*=function(){

return *arg*;

}

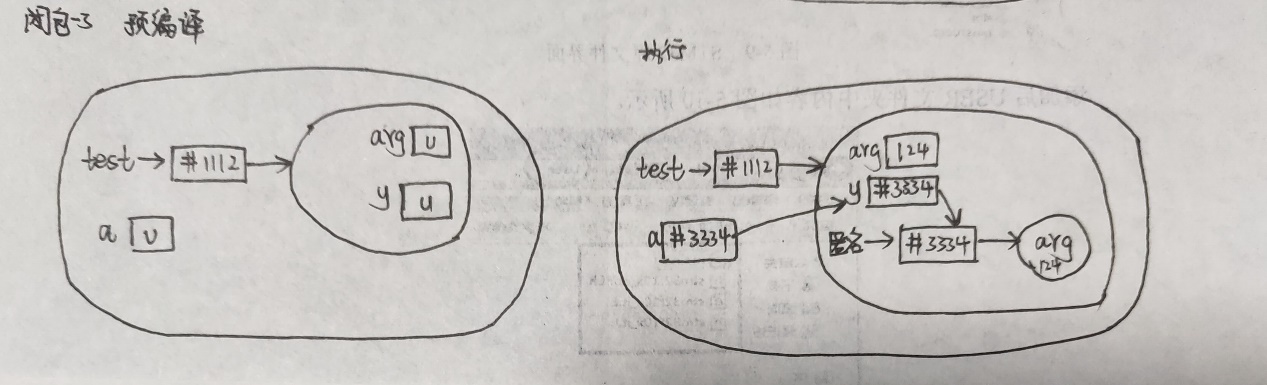
*arg*++;

return *y*;

}

var ***a***=*test*(123); //[调用](闭包3.html)

alert(***a***());



4、闭包-4

function *test*(){

var x=[];

var i;

for(i=0;i<3;i++){

x[i]=function(){

return i;

}

}

return x; //返回了数组，所以a可以直接调用

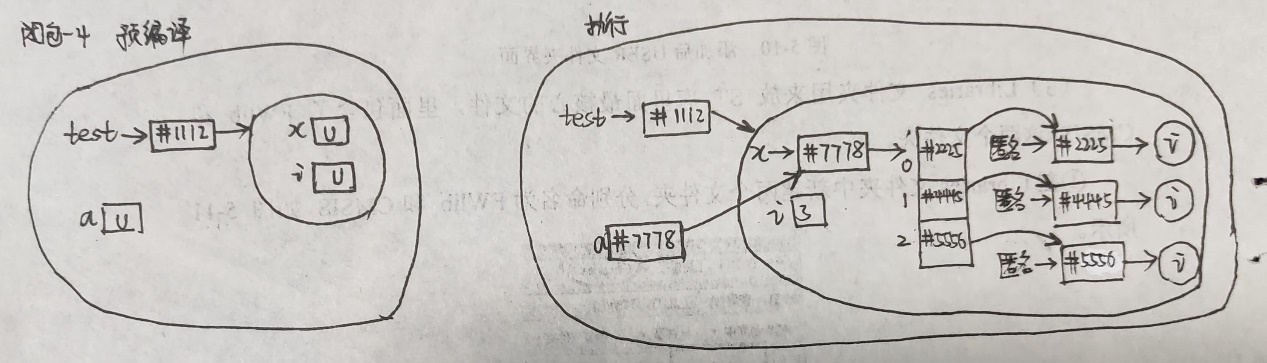
}

var ***a***=*test*(); //[调用](闭包4-问题.html)

alert(***a***[0]()); //3，到了这一步才执行的函数

alert(***a***[1]()); //3

alert(***a***[2]()); //3



说明：输出结果都是3，因为循环完毕i的值为3，这三个函数都指向这一个共同的值，如何解决这个问题?

①使用自调函数来实现

function *test*(){

var x=[];

var i;

for(i=0;i<3;i++){

x[i]=(function(*a*){ //返回函数的函数

return function(){

return a;

}

})(i);

}

return x;

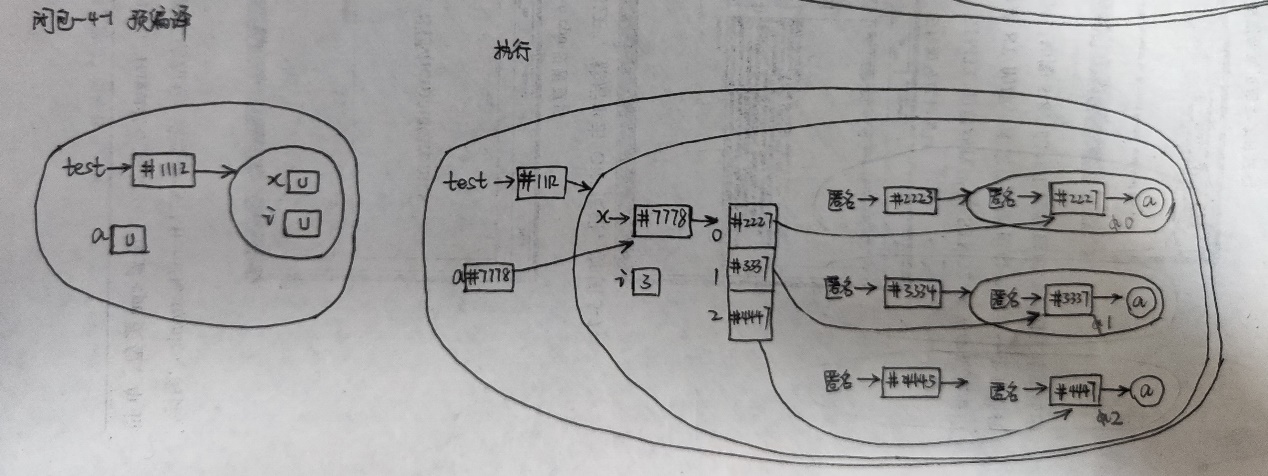
}

var ***a***=*test*(); //[调用](闭包4.html)

alert(***a***[0]());

alert(***a***[1]());

alert(***a***[2]());



注：与灰色代码相同↓。

(function test(){

var x=[];

var i;

for(i=0;i<3;i++){

x[i]=(function(){

return i;

})();

alert(x[i]); //运行结果：0 1 2

}

})();

②不使用自调函数来实现

function *test*(){

function *makeClosure*(*x*){

return function(){

return *x*;

}

}

var a=[];

var i;

for(i=0;i<3;i++){

a[i]=*makeClosure*(i);

}

return a;

}

var ***b***=*test*(); //[调用](闭包4-2.html)

alert(***b***[0]());

alert(***b***[1]());

alert(***b***[2]());

**闭包应用实例：getter与setter**

实现了封装

var *getValue*,*setValue*;

(function(){

var secret=0;

*getValue*=function(){

return secret;

};

*setValue*=function(*v*){

secret=*v*;

};

})();

alert(*getValue*()); //查看内部局部变量值

*setValue*(3); //设置内部局部变量的值

alert(*getValue*()); //输出修改后的值

**迭代器实例：**

function *setup*(*x*){

var i=0;

return function(){

return x[i++];

};

}

var *next*=*setup*(["a","b","c"]);

alert(*next*()); //a

alert(*next*()); //b

alert(*next*()); //c

# √08.30

面向对象编程：用户只关注对象提供的功能（接口），不关注对象实现的细节。开发人员分为两类：1.写对象；2.用对象

面向对象编程（oop）特点：

抽象：从现实应用中抓住事物重点

封装：不考虑内部实现，只考虑提供的功能

继承：从已有对象继承出新对象

多重继承

多态

对象的组成：

属性——变量（静态特性）

方法——函数（动态行为）

---------------------------------------

ecmascript脚本标准：

javascript、jscript和actionscript是ecmascript的具体实现

说明：在JavaScript中级和高级学的是ecmascript版本3和5，后面还有一个ecmascript6（2015年）→ecmascript2015

--------------------------------------------------------

1、ecmascript3/5不像其他面向对象编程语言那样有类的概念，JavaScript没有类（构造函数）的概念，只有对象的概念。用构造器函数（用来创建对象的函数）当做类。

//创建一个类/构造器函数

function Student(name,age,sex){

this.name=name;

this.age=age;

this.sex=sex;

}

Student.prototype.study=function(){

alert("学无止境~");

}

//使用构造器函数创建一个对象/通过类实例化一个对象

var s1=new Student("Obama",22,"男");

2、ecmascript2015是面向对象编程语言，有类。

class Student{ //类

//构造器函数：通过类创建对象的时候，js解释器自动调用

//作用：初始化对象的属性

constructor(name,age,sex){

this.name=name;

this.age=age;

this.sex=sex;

}

study(){

alert("持之以恒");

}

}

//通过类实例化一个对象

const s1=new Student("Obama",22,"男");

------------------------------------------------------------

JavaScript中对象的概念：无序的属性值对

{属性:值,属性:值,属性:值,.....}

说明1：无序，就是指属性在前或在后都可以。

说明2：值可以是js中任何类型。为了模拟面向对象编程语言，我们把值是function的属性称为方法（模拟java叫法）。

-----------------------------------------------

**重点，重点，重点！**

JavaScript中对象的创建方式：

对象创建的方式：

一、直接创建js对象

1、先创建一个空对象：

var 对象名=new Object();

等价写法：var 对象名={};

然后再向空对象中添加属性和方法：

对象名.属性名=值;

对象名.方法名=function(){};

例：

var obj=new Object();

obj.name="tom";

obj.age=19;

obj.sex="男";

obj.study=function(){};

对象创建好之后，如何使用对象的属性或方法呢？

对象属性的使用：

(1)对象名.属性名

(2)对象名["属性名"]

对象方法的使用：

对象名.方法名();

2、创建对象，一次添加所有属性和方法

var 对象名={属性:值,属性:值,……};

例：

var obj={name:"tom",age:19,sex:"男",study:function(){}};

二、通过构造器函数创建js对象（重点）

1、不使用构造器函数创建对象

分析：三个对象，属性名相同，方法名和方法体相同，只有值不同，写了三份。

缺点：代码冗余

维护困难

方法无法共享，浪费内存

2、工厂模式：使用构造器函数创建对象

优点：解决了代码冗余、维护困难的问题

缺点：没有new

方法无法共享，浪费内存

例：

//工厂模式创建对象

//原料入厂

function Student(name,age,sex){

//空罐子

var obj=new Object();

obj.name=name;

obj.age=age;

obj.sex=sex;

obj.study=function(){

alert("持之以恒~");

}

return obj; //成品出厂

}

//通过构造器函数创建对象

var s1=Student("Obama",22,"男");

var s2=Student("Putin",25,"男");

alert(s1.study===s2.study); //false

-------------------------------------------------------------

this指向当前环境对象：当前的方法属于哪个对象，this就代表哪个对象。

1.函数中没有指向，那么this就是window

2.如果函数前面加一个关键字new，即：new函数名();

new 的含义是新创建一个Object对象

new等价于new Object()→{}

new Object()对象后，调用函数初始化对象的属性的值

new 函数名(); 构造函数里面的this指向了new出来的那个对象

补充：

①函数在执行时，会在函数体内部自动生成一个this指针。谁**直接调用**产生这个this指针的函数，this就指向那个对象。

②使用new操作符的时候，js先创建一个新的空对象，然后，js传入指定的参数并将这个新对象当做this的值来调用一个指定的函数。这个函数可以使用this来初始化这个新创建对象的属性。所以当使用new操作符之后，函数中的this指向的是新创建的对象。所以构造函数中的this就是指向new出来的那个对象。

-------------------------------------------------------

3、改进版的工厂模式（带new）

优点：解决了new的问题

缺点：方法无法共享

例：

function Student(name,age,sex){

this.name=name;

this.age=age;

this.sex=sex;

this.study=function(){

alert("持之以恒~");

}

}

//通过构造器函数创建对象

var s1=new Student("Obama",22,"男");

var s2=new Student("Putin",25,"男");

alert(s1.study===s2.study); //还是false

4、混合模式创建对象：

预备知识：

函数是不是对象?答：当然是对象，因为函数也是由属性和方法组成的。

1、对象的分类（按照typeof分）：

(1)function类型：

每个function类型对象都有一个内置属性prototype(本身也是个js对象)，也叫原型对象。

{constructor:function(){}}

如何共享属性或方法?

答：把需要共享的属性或方法，挂到prototype属性上即可。

(2)object类型：

每个object类型对象，都有一个内置属性\_\_proto\_\_(在有些浏览器里面，比如IE，是私有属性)。

说明：object对象的\_\_proto\_\_属性始终等于它的构造器函数对象的prototype。

换言之，普通对象.\_\_proto\_\_===函数对象.prototype，即s1.\_\_proto\_\_===Student.prototype。[详情点击这里](https://blog.csdn.net/cc18868876837/article/details/81211729)

2、对象的属性分类：

用户自定义属性（优先级高）

原型属性——构造器函数.prototype上的属性

-------------------------------------------------------------

分析：通过同一个构造器函数创建的多个对象，这些对象的方法和方法体完全相同，属性值不同。

混合模式：

(1)工厂模式创建属性：因为同一个构造器函数创建的多个对象，属性值是不同的。

(2)原型模式创建方法：因为同一个构造器函数创建的多个对象，方法和方法体是一样的。（原型模式，就是把方法添加到构造器函数的prototype上）

//混合模式

//工厂模式创建对象的属性

function Student(name,age,sex){

this.name=name;

this.age=age;

this.sex=sex;

}

//原型模式创建对象的方法

//添加前：

//prototype：{constructor:function(){}}

Student.prototype.study=function(){

alert("自律和用心！");

}

//添加后：

/\*prototype：{constructor:function(){},study:function(){alert("自律和用心！");}\*/

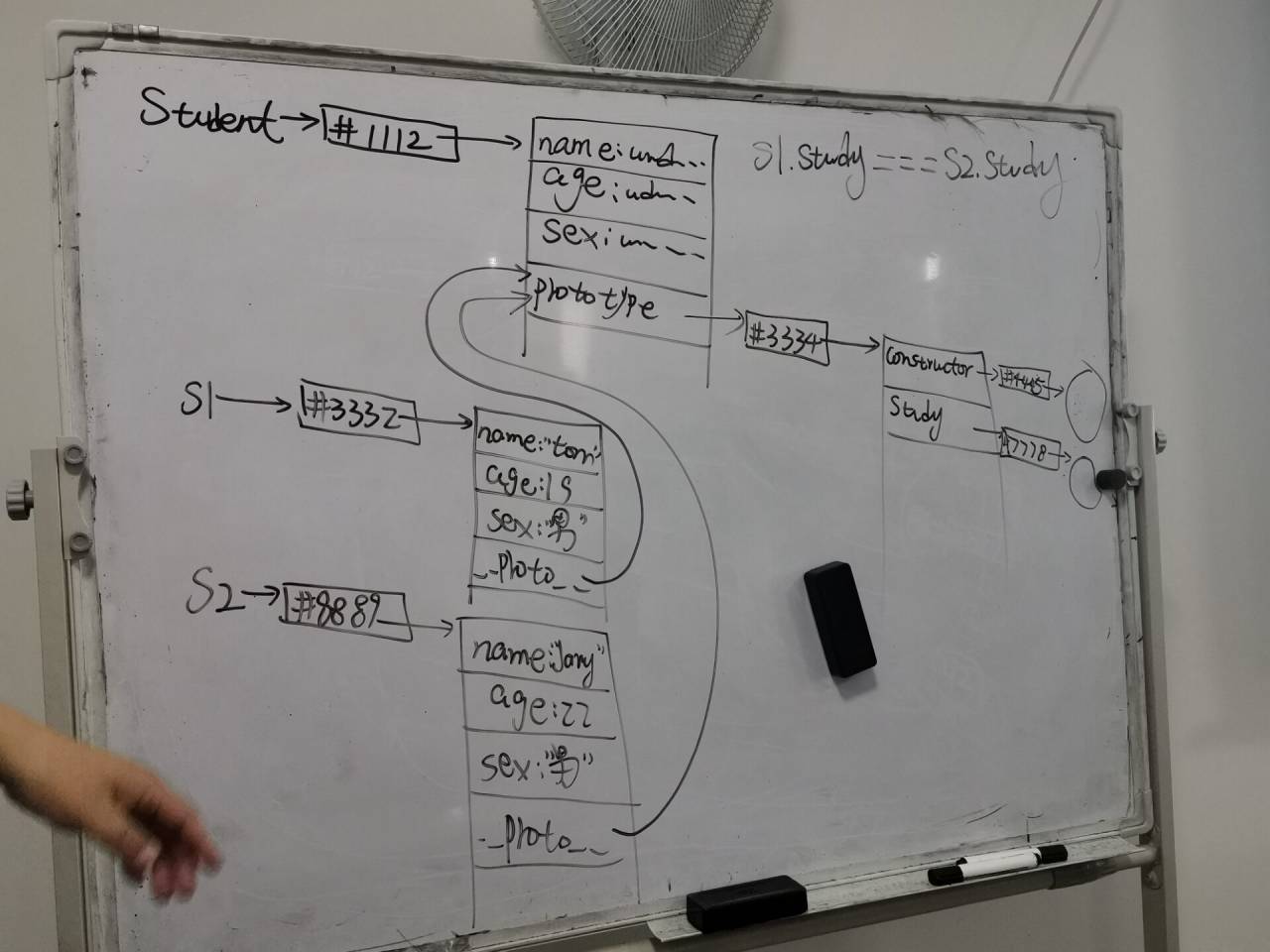
var s1=new Student("Putin",22,"男");

var s2=new Student("Obama",25,"男");

alert(s1.\_\_proto\_\_.study===Student.prototype.study); //true

alert(s2.\_\_proto\_\_.study===Student.prototype.study); //true

alert(s1.study===s2.study); //true



**补充1：**

***工厂模式：***

考虑到在ECMAScript中无法创建类，开发人员就发明了一种函数，用函数来封装以特定接口创建对象的细节，如下面的例子所示：

*function createPerson(name,age,job){*

*var o = new Object();*

*o.name = name;*

*o.age = age;*

*o.job = job;*

*o.sayName = function(){*

*alert(this.name);*

*}*

*return o;*

*}*

*var person1 = createPerson('Grey',27,'Doctor');*

函数createPerson()能够根据接受的参数来构建一个包含所有必要信息的 Person对象。可以无数次地调用这个函数，而每次它都会返回一个包含三个属性一个方法的对象。工厂模式虽然解决了创建多个相似对象的问题，但却没有解决对象识别的问题（即怎样知道一个对象的类型）。

　　主要好处就是可以消除对象间的耦合，通过使用工程方法而不是new关键字。将所有实例化的代码集中在一个位置防止代码重复。

　　工厂模式解决了重复实例化的问题 ，但还有一个问题,那就是识别问题，因为根本无法搞清楚他们到底是哪个对象的实例。

[***原型模式***](原型模式引入2.html)***：***

我们创建的每个函数都有一个prototype（原型）属性，这个属性是一个指针，指向一个对象，而这个对象的用途是包含可以由特定类型的所有实例共享的属性和方法。

　　如果按字面意思来理解，那么prototype就是通过调用构造函数而创建的那个对象实例的原型对象。使用原型对象的好处是可以让所有对象实例共享它所包含的属性和方法。换句话说，不必在构造函数中定义对象实例的信息，而是可以将这些信息直接添加到原型对象中。例如：

*function Person(){};*

*Person.prototype.name = 'Nicholas';*

*Person.prototype.age = 29;*

*Person.prototype.job = 'Software Engineer';*

*Person.prototype.sayName = function(){*

*alert(this.name);*

*}*

*var person1 = new Person();*

*person1.sayName(); //"Nicholas"*

*var person2 = new Person();*

*person2.sayName(); //"Nicholas"*

*alert(person1.sayName == person2.sayName); //true*

　　在此，我们将sayName()方法和所有属性直接添加到了Person的prototype属性中，构造函数变成了空函数。

即使如此，也仍然可以通过调用构造函数来创建新对象，而且新对象还会具有相同的属性和方法。但与构造函数模式不同的是，新对象的这些属性和方法是由所有实例共享的。换句话说，person1和person2访问的都是同一组属性和同一个sayName()函数。要理解原型模式的工作原理，必须先理解ECMAScript中原型对象的性质。

**补充2：构造函数（改进版工厂模式）**

ECMAScript中的构造函数可用来创建特定类型的对象，像Array和Object这样的原生构造函数，在运行时会自动出现在执行环境中。此外，也可以创建自定义的构造函数，从而定义自定义对象的属性和方法。使用构造函数的方法，既解决了重复实例化的问题，又解决了对象识别的问题。例如，可以使用构造函数模式将P14的例子重写如下：

*function Person(name,age,job){*

*this.name = name;*

*this.age = age;*

*this.job = job;*

*this.sayName = function(){*

*alert(this.name);*

*}*

*}*

*var person1 = new Person("Nicholas", 29, "Software Engineer");*

*var person2 = new Person('Grey',27,'Doctor');*

　　Person()中的代码除了与createPerson()中相同的部分外，还存在以下不同之处：

　　1.没有显式地创建对象；

　　2.直接将属性和方法赋给了this对象；

　　3.没有return语句。

　按照惯例，构造函数始终都应该以一个大写字母开头，而非构造函数则应该以一个小写字母开头。

　　要创建 Person的新实例，必须使用new操作符。以这种方式调用构造函数实际上会经历以下4个步骤：

　　(1)创建一个新对象；

　 (2)将构造函数的作用域赋给新对象（因此this就指向了这个新对象）；

　　(3)执行构造函数中的代码（为这个新对象添加属性）；

　　(4)返回新对象。

在前面例子的最后，person1和person2分别保存着Person的一个不同的实例。这两个对象都有一个constructor（构造函数）属性，该属性指向Person，如下所示。

　　alert(person1.constructor == Person); //true

　　alert(person2.constructor == Person); //true

　　对象的 constructor 属性最初是用来标识对象类型的。但是，提到检测对象类型，还是instanceof操作符要更可靠一些。我们在这个例子中创建的所有对象既是Object的实例，同时也是Person的实例，这一点通过instanceof操作符可以得到验证。

　　alert(person1 instanceof Object); //true

　　alert(person1 instanceof Person); //true

　　alert(person2 instanceof Object); //true

　　alert(person2 instanceof Person); //true

　　创建自定义的构造函数意味着将来可以将它的实例标识为一种特定的类型；而这正是构造函数模式胜过工厂模式的地方。

　　在这个例子中，person1和person2之所以同时是Object的实例，是因为所有对象均继承自Object。

前面例子中定义的 Person()函数可以通过下列任何一种方式来调用。

　　// 当作构造函数使用

　　var person=new Person("Nicholas",29,"Software Engineer");

　　person.sayName(); //"Nicholas"

　　// 作为普通函数调用

　　Person("Greg",27,"Doctor"); // 添加到 window

　　window.sayName(); //"Greg"

　　// 在另一个对象的作用域中调用

　　var o=new Object();

　　Person.call(o, "Kristen", 25, "Nurse");

　　o.sayName(); //"Kristen"

　　这个例子中的前两行代码展示了构造函数的典型用法，即使用new操作符来创建一个新对象。

　　接下来的两行代码展示了不使用new操作符调用Person()会出现什么结果：属性和方法都被添加给window对象了。

当在全局作用域中调用一个函数时，this对象总是指向Global对象（在浏览器中就是window对象）。因此，在调用完函数之后，可以通过window对象来调用sayName()方法，并且还返回了"Greg"。

最后，也可以使用call()或者apply()在某个特殊对象的作用域中调用Person()函数。这里是在对象o的作用域中调用的，因此调用后o就拥有了所有属性和sayName()方法。

# √08.31

tab菜单功能分析：

点击某个按钮：

清除三个按钮的背景色

这个按钮添加背景色

所有div都隐藏

和这个按钮对应索引的div显示

---------------------------------------------------

DOM对象有两个属性：

1、DOM对象.className：获取或设置类选择器

例：

DOM对象.calssName="active";

相当于：<p class="active"></p>

2、DOM对象.style.CSS样式（camel命名法）：获取或设置行内样式

例：

DOM对象.style.backgroundColor="#ff0";

相当于：<p style="background-color:#ff0"></p>

\*\*\*\*\*

了解：js用currentStyle和getComputedStyle获取css样式

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="UTF-8">

<title>js用currentStyle和getComputedStyle获取css样式</title>

</head>

<style type="text/css">

#div1{

width: 100px;

height: 100px;

background: red;

}

</style>

<body>

<div id="div1"></div>

</body>

<script type="text/javascript">

function getStyle(obj,attr){

if(obj.currentStyle){

return obj.currentStyle[attr];

}else{

return getComputedStyle(obj,false)[attr];

}

}

window.onload=function(){

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

alert(getStyle(oDiv,'width'));

}

</script>

</html>

----------------------------------

前端工程师：前后端交互

前端：javascript

后台：php/java/python/asp.net/其他

前后数据交换标准：

古代：xml

现代：json字符串

-----------------------------------

json字符串是前后端数据交互标准（格式）。

字符串→json字符串→ajax→php→sql语句→数据库；

数据库→php→json字符串→ajax→js将json字符串转换为js对象→遍历

json语法格式要求：

两端必须加''，然后是{}→'{}'

属性必须加""，不加就出错

属性的值如果是字符串加""，数字不用加""，值可以是数字、字符串、数组、对象，但不能是函数

属性和值之间用冒号“:”

“属性:值”之间用逗号“,”

简单json字符串举例：'{"name":"tom","age":19,"sex":"男"}'

复杂json字符串举例（开发常见）：

'{"status":"10001","msg":"success","data":[{"name":"Tom","age":19,"sex","男"},{"name":"Jay","age":33,"sex","男"}]}'

------------------------------------------------------

js无法使用字符串，需要将json字符串转换为js对象才能使用。

**json字符串转换为js对象：**

(1) eval("("+json字符串+")")； 将字符串当做js运行

优点：兼容性好

缺点：若不加控制，有安全风险

例：

var~json='{"status":"10001","msg":"success","data":[{"name":"Obama","age":22,"sex":"男"},{"name":"Trump","age":66,"sex":"女"}]}';

eval("("+json+")");

console.log(eval("("+json+")").data [0].name); //Obama

(2) JSON.parse(json字符串); //IE9+以上支持，ECMAScript5标准

例：

var~myData='{"status":"10001","msg":"success","data":[{"name":"Tom","age":22,"sex":"男"},{"name":"Jarry","age":21,"sex":"女"}]}';

//将myData转换为js对象

var obj=JSON.parse(myData); //IE9+支持

//遍历js对象

var str="";

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

str=str+"姓名："+obj.data[i].name+",年龄："+obj.data[i].age+",性别："+obj.data[i].sex+"<br>";

}

document.write(str);

---------------------------------------

1、通过函数对象的call方法，改变函数内部this指向

函数对象.call(要改变的对象,实参1,实参2……);

函数对象的call方法功能是：

(1)调用前面的函数对象

(2)改变函数内部的this指向为要改变的对象

(3)并且传递实参

例：

function test(x,y){

alert(this); //abc

alert(x+y); //8

}

test.call("abc",3,5);

2、通过函数对象的apply方法，改变函数内部this指向

函数对象.apply(要改变的对象,[实参1,实参2……]);

函数对象的apply方法功能是：

(1)调用前面的函数对象

(2)改变函数内部的this指向为要改变的对象

(3)并且传递实参

例：

function test(x,y){

alert(this); //abc

alert(x+y); //8

}

test.apply("abc",[3,5]);

补充：

对于apply、call二者而言，作用完全一样，只是接受参数的方式不太一样：call需要把参数按顺序传递进去，而apply则是把参数放在数组里。

3、触发事件的时候，处理函数中this变成触发事件的DOM对象

4、通过函数对象的bind方法，改变函数内部this指向

函数对象.bind(要改变的对象);

bind方法功能：

改变函数调用时，函数内部的this指向

说明：bind方法不会执行函数，而 apply/call方法则会立即执行函数。

---------------------------------------------------------------------------------------------

**对象的继承：**

预备知识：for-in

例1：

var obj={name:"Obama",age:22,sex:"男"};

for(var key in obj){

console.log(key,obj[key]);

}

console.log(typeof key); //string

输出：

name Obama

age 22

sex 男

string

注意：key是string类型数据

例2：

var obj1={name:"Obama",age:22,sex:"男"};

var obj2={};

for(var key in obj1){

obj2[key]=obj1[key];

}

console.log(obj1,obj2); //obj1和obj2相同

例3：

function A(){

this.name="GaoYuanYuan";

}

A.prototype.getName=function(){

alert(this.name);

}

function B(){

console.log("随便写的");

}

for(var i in A.prototype){

B.prototype[i]=A.prototype[i];

}

console.dir(A);

console.dir(B);

-------------------------------

为什么要继承?

答：子对象可以继承父对象的所有的属性和方法，实现代码的高度复用。

例1：

function A(){

this.abc="123"; //工厂模式添加属性

this.def="999";

}

A.prototype.fn=function(){ //原型模式添加方法fn

alert(this.abc); //输出：123

console.log(this); //说明这个this指向B对象

}

//A.prototype→{constructor:function(){},fn:function(){}}

A.prototype.fm=function(){ //原型模式添加方法fm

alert(this.def); //输出：999

console.log(this); //说明这个this指向B对象

}

function B(){

A.call(this); //使函数A内部的this指向B类对象，即obj

}

//B.prototype→{constructor:function(){}}

for(var i in A.prototype){

B.prototype[i]=A.prototype[i]; //继承父类A的方法

}

var obj=new B();

obj.fn();

obj.fm();

例2：ES5继承

//继承：实现代码高度复用

//父类：

function Human(name,age,sex){

this.name=name;

this.age=age;

this.sex=sex;

}

Human.prototype.eat=function(){

alert("吃出健康来！");

}

//子类：子类继承父类的属性和方法

//方法继承：原型拷贝

//属性继承：通过call方法

function Student(sno,sdept,name,age,sex){

//继承父类的属性

Human.call(this,name,age,sex);

//子类自身属性

this.sno=sno;

this.sdept=sdept;

}

//自身方法

Student.prototype.study=function(){

alert("好好学习~");

}

for(var key in Human.prototype){

Student.prototype[key]=Human.prototype[key];

}

var s1=new Student("1001","计算机系","Obama",22,"男");

例3：ES6继承（了解）

class Human{

constructor(name,age,sex){

this.name=name;

this.age=age;

this.sex=sex;

}

eat(){

alert("吃出健康来！");

}

}

//子类继承父类的属性和方法

class Student extends Human{

constructor(sno,sdept,name,age,sex){

super(name,age,sex); //继承父类的属性

//super()功能：调用父类的构造器函数，继承父类的属性

this.sno=sno;

this.sdept=sdept;

}

study(){

alert("好好学习~");

}

}

const s1=new Student("1001","计算机系","Obama",22,"男");

-----------------------------------------------------------------------------------------

鼠标划入：滑动门

点击事件：tab菜单

例：

<body>

<div id="div1">

<input type="button" value="导航1" class="active"/>

<input type="button" value="导航2"/>

<input type="button" value="导航3"/>

<div style="display: block;">内容1</div>

<div>内容2</div>

<div>内容3</div>

</div>

</body>

<script type="text/javascript">

window.onload=function(){

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

var inputs=div1.getElementsByTagName("input");

var divs=div1.getElementsByTagName("div");

//给每个按钮绑定click事件

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

inputs[i].x=i;

//即：inputs[0].x=0,inputs[1].x=1,inputs[2].x=2

inputs[i].onclick=function(){

//三个按钮的class都清除，三个div都隐藏

for(var j=0;j<inputs.length;j++){

inputs[j].className="";

divs[j].style.display="none";

}

this.className="active";

divs[this.x].style.display="block";

}

}

}

</script>

--------------------------

补充：

**javascript对象：**

ECMAScript中，JavaScript对象被描述成无序的属性集合。

属性名——变量或字符串

属性值可以包含字符串、数值、布尔值、函数、数组、对象、null等。

JavaScript对象示例：

var person={

name:"张三", //注意这里是“,”

qq:"3434344",

showName:function(){

alert(this.name);

},

showQQ:function(){

alert(this.qq);

}

}

person.fn=function(){ //给对象添加方法

alert("测试");

}

person.showName();

person.showQQ();

person.fn();

Javascript对象特点：

创建对象简单

使用方便

单体——创建多个对象不方便

**json对象：**

JSON(JavaScript Object Notation，JavaScript对象表示法) 是一种轻量级的数据交换格式，JSON格式的数据，主要是为了跨平台交流数据用的。JSON是JavaScript对象的一个子集（对象和数组混合组成的数据结构）。

属性名——字符串

属性值——属性值可以包含字符串、数值、布尔值、函数、数组、对象、null等

JSON 语法规则：

* 数据在名称/值对中
* 数据由逗号分隔
* 花括号保存对象
* 方括号保存数组

JavaScript对象和JSON的关系：

json简单说就是javascript中的对象和数组，所以这两种结构就是对象和数组两种结构，通过这两种结构可以表示各种复杂的结构。

对象：对象在js中表示为“{}”括起来的内容，数据结构为{key：value,key：value,...}的键值对的结构，在面向对象的语言中，key为对象的属性，value为对应的属性值，所以很容易理解，取值方法为“对象.key”获取属性值，这个属性值的类型可以是数字、字符串、数组、对象几种。

数组：数组在js中是中括号“[]”括起来的内容，数据结构为 ["java","javascript","vb",...]，取值方式和所有语言中一样，使用索引获取，字段值的类型可以是数字、字符串、数组、对象几种。

经过对象、数组2种结构就可以组合成复杂的Json数据结构了。

JSON特点：

JSON 是纯文本

JSON 具有“自我描述性”（人类可读）

JSON 具有层级结构（值中存在值）

JSON 可通过 JavaScript 进行解析

JSON 数据可使用 AJAX 进行传输

JSON实例：

var users={

"userName":"李三",

"pwd":"123456",

"positions":"技术部经理",

"employees": [

{ "eName":"李明","phone":"133422334"},

{ "eName":"张萍","phone":"183222334"},

{ "eName":"郑鑫","phone":"153442223"}

]

}

document.write("输出Json内容:<br>");

document.write("用户名是:"+users.userName+"<br>");

document.write("密码是:"+users.pwd+"<br>");

document.write("职位是:"+users.positions+"<br>");

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

document.write("雇员有:姓名是:"+users.employees[i].eName

+"，电话号码是:"+users.employees[i].phone+"<br>");

}

# 09.03

最早出现的编码：ASCII码(美国信息交换标准代码)，它定义了128个字符和128个扩展码。

a 97 01100001

A 65 01000001

支持中文的编码：GB2312 简体中文 2个字节表示一个汉字

支持全球各种主流语言的编码：UTF-8 3个字节代表一个字符

各种编码的前面与ASCII码相同



**JavaScript对象的分类：**

1、用户自定义对象{} 第7章

2、内置对象 第8章

3、BOM对象 第9章

4、DOM对象 第10章

5、事件对象 第11章

6、ajax对象 第14章

-----------------------------------

**JavaScript 内置对象**

* String(字符串对象)
* Array（数组）
* Math（数学）
* Date（时间）
* Number （数字）
* Function （函数/方法）
* RegExp（正则表达式）
* Error （异常）

## String字符串对象：

说明：凡是用""或''包裹的都是字符串

创建字符对象的方式：

①var 字符串对象="abc";

②var 字符串对象 =new String("abc"); //构造器函数创建

String对象的属性：

①length属性

②prototype属性(原型)

注：prototype属性

为对象增加新方法

覆盖旧方法实现新功能

----------------------------------------

扩展内置对象的功能（方法）：

1.在内置对象构造器的原型上添加新方法（开发使用）

例：

var str="我是帅哥";

String.prototype.newInterface=function(){

alert("字符串对象新加功能。");

}

str.newInterface();

2.在内置对象构造器的原型上覆盖原有方法

团队开发的时候，不能用→代码合并可能会影响其他人

例：

var str="我不是药神";

console.log(str.substr(3,2));

String.prototype.substr=function(){

return "改变了截取函数的功能。";

}

console.log(str.substr(0,3));

/\*String.prototype属性表示 String原型对象。所有 String的实例都继承自String.prototype，任何String.prototype上的改变都会影响到所有的 String实例\*/

-------------------------------------------------------------------------------------------------------

**String对象的方法：**

1、charAt(); 返回字符串中的第n个字符

2、charCodeAt(); 返回字符串中的第n个字符的Unicode编码

3、concat(); 连接一个字符串

例：

var str="abc";

console.log(str.concat("哈哈")); //abc哈哈

console.log(str+"哈哈"); //abc哈哈

4、fromCharCode(); 根据编码返回字符。这是一个静态方法。

说明：面向对象编程中，可以通过类名直接调用的方法，称为静态方法static。

使用：类名.静态方法名();

5、indexOf(); 检索指定字符串，搜索到第一个匹配的位置后，停止搜索并返回字符串下标，搜索不到返回-1。

6、lastIndexOf(); 从后向前检索字符串

7、localeCompare(); 用本地特定的顺序比较两个字符串是否相等

例：

var str1="赵"; /\*\u8d75→36213\*/

var str2="钱"; /\*\u94b1→38065\*/

var str3="孙"; /\*\u5b59→23385\*/

console.log(str1.localeCompare(str2)); //小于返回负数-1

console.log(str1.localeCompare(str3)); //大于返回正数1

8、match(); 在字符串内检索指定的值，或找到一个或多个正则表达式的匹配。

该方法类似indexOf()和lastIndexOf()，但是它返回指定的值，而不是字符串的位置。

说明：如果具有标志 g，则 match()方法将执行全局检索，找到字符串中的所有匹配子字符串。若没有找到任何匹配的子串，则返回null。

9、replace( ); 替换一个与正则表达式匹配的字符串（非变异方法）

replace("被替换的字符串","替换成的字符串")方法：替换第一个匹配的字符串或所有匹配正则表达式的字符串。

10、search(); 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串。

说明：返回第一个与指定值匹配的子串的起始位置。如果没有找到任何匹配的子串，则返回-1。

11、slice(); 该方法可提取字符串的某个部分，并以新的字符串返回被提取的部分。

slice(start, end)，返回一个新的字符串，包含从start到end（不包括该元素）的元素。

例：

var str="Hello Obama!";

console.log(str.slice(6)); //Obama!

console.log(str.slice(6,11)); //Obama

console.log(str.slice(3,7)); //lo O

console.log(str.slice(-6,-1)); //Obama

注意：不要忽略空格！

12、split(); 用于把一个字符串分割成字符串数组。

split(separator,limit)，separator为可选参数，可以是字符串或正则表达式，表示从该参数指定的地方分割；limit为可选参数，可指定返回的数组的最大长度。

补充：join()函数，用于把数组中的所有元素转换一个字符串。

13、substr(); 字符串截取函数

substr(start,length)，该方法可在字符串中抽取从开始下标开始的指定数目的字符。

注意：substr方法的起始位置可以是负数。

14、substring(); 该方法用于提取字符串中介于两个指定下标之间的字符。返回的子串包括开始处的字符，但不包括结束处的字符。

注意：与slice方法不同的是，substring方法的参数不能为负数。

15、toLowerCase( ); 将字符串转换成小写

toUpperCase( ); 将字符串转换成大写

16、valueOf(); 方法返回String对象的原始值。

说明：valueOf()方法通常由JavaScript在后台自动调用，并不显式地出现在代码中，且valueOf()方法不会改变原字符串。

例：

var p=new String("韩寒会画画后悔画韩红");

console.dir(p);

console.log(p.valueOf());

17、toString(); 返回字符串

# 09.04

1、面试题： undefined和null的区别

undefined：未定义——变量被创建，未初始化

null：对象为空——对象值是空值，不指向任何内存块

undefined==null //true

undefined===null //false

2、面试题：typeof和intanceof的区别

typeof：数据类型检测运算符（结果是字符串）

"number"、"string"、"boolean"、"undefined"、"object"、"function"、"symbol"

instanceof：判断对象的类型（结果是true /false）

判断对象是否是某个类的实例

判断对象是否是某个类创建的

判断对象的构造器

例：

var y=[1,2,3];

console.log(y instanceof Object); //true

console.log(y instanceof Array); //true

补充：ES6引入了一种新的原始数据类型Symbol，表示独一无二的值，最大的用法是用来定义对象的唯一属性名。

## Array数组对象

数组对象的创建/定义/声明：

(1)创建一个空数组：

var 数组名=new Array();

然后给数组元素赋值：

数组名[下标]=值;

说明1：数组一旦创建好之后，数组元素可以像普通变量一样使用。

说明2：数组元素可以是js中的任何类型。

(2)创建一个空数组，并且给数组对象的length属性赋值：

var 数组名=new Array(大小);

然后给数组元素赋值：

数组名[下标]=值;

(3)创建一个数组，并且给数组元素和length赋值：

var 数组名=new Array(值1,值2,值3,……);

说明：同一个函数或方法，根据参数个数不同或数据类型不同，实现不同的功能——函数重载。

(4)开发中常用方式：创建一个数组，并且给数组元素和length赋值

var 数组名=[值1,值2,值3,……];

---------------------------------------------------

Array对象的属性：

①length属性

②prototype属性(原型)

注：prototype属性

为对象增加新方法

覆盖旧方法实现新功能

**Array对象的方法：**

1、concat( ); 连接数组

2、join( ); 将数组元素按指定分隔符连接起来，返回一个字符串

3、pop( ); 删除并返回数组的最后一个元素（出栈）

4、push( ); 给数组添加元素（入栈）

5、reverse( ); 颠倒数组中元素的顺序

6、shift( ); 将头部元素移出数组

7、unshift( ); 在数组头部插入一个元素

8、slice( ); 返回数组的一部分(不包含end)

例：

var arr=["Obama","Trump","Putin","Elizebeth"];

console.log(arr.slice(2,4)); //["Putin","Elizebeth"]

9、sort( ); 对数组元素进行排序

(1)字符串排序——不带参数

(2)数字排序——带回调函数，要求能比较出来大小

例1：字符串排序

var names=["Tom","Susan","Rose","Jack","John","Petter"];

alert("排序前："+names);

alert("排序后："+names.sort()); //sort不带参数

例2：数字排序错误示范

var nums1=["100","202","10000","101","1"];

alert("排序前："+nums1);

alert("排序后："+nums1.sort());

//输出结果：["1", "100", "10000", "101", "202"]

例3：数字排序

sort方法必须加参数,参数必须是回调函数，而且回调函数必须能区别数字大小。

格式：sort(回调函数);

var nums2=[100,202,10000,101,1];

alert("排序前："+nums2);

alert("排序后："+nums2.sort(function(x,y){return x-y})); /\*匿名回调函数\*/

//输出结果：[1, 100, 101, 202, 10000]

10、splice( ); 插入、删除或替换数组的元素。

splice(开始删除位置,[删除数目,插入元素1,插入元素2…]);

splice()函数返回删除元素的数组，如果未删除任何元素，则返回空数组。该方法会改变原始数组，是变异方法。

例：

var arr=["Obama","Trump","Putin","Elizebeth"];

arr.splice(1,2,"Michael","Jackson");

console.log(arr); //["Obama", "Michael", "Jackson", "Elizebeth"]

11、toString( ); 将数组转换成一个字符串

补充：字符串反序排列的做法：

var str="abcdefghijk";

console.log(str.split("").reverse().join("")); // kjihgfedcba

**二维数组**：一维数组的元素是另一个数组

一维数组

绘图6

二维数组



对象的方法，按照是否改变原有对象元素分为两类：

1、变异方法：方法会改变对象本身

2、非变异方法：方法不会改变对象本身

数组中所有的变异方法：

push()、pop()、shift()、unshift()、splice()、sort()、reverse()

## Math对象：

Math对象的属性：

E属性：返回算术常量 e，即自然对数的底数（约等于2.718）

LN10属性：返回 10 的自然对数，即ln10（约等于2.302）

LN2属性：返回 2 的自然对数（约等于0.693）

LOG2E：返回以 2 为底的 e 的对数（约等于 1.414）。

LOG10E： 返回以 10 为底的 e 的对数（约等于0.434）。

PI属性：返回圆周率（约等于3.14159）

SQRT1\_2：返回 2 的平方根的倒数（约等于0.707）

SQRT2：返回 2 的平方根（约等于1.414）

**Math对象的方法：**

1、floor(x) 对数进行下舍入

Math.floor(3.1); //3

Math.floor(3.9); //3

Math.floor(-3.9); //-4

2、log(x) 返回数的自然对数（底为e）

3、max(x,y) 返回x和y中的最高值

4、min(x,y) 返回x和y中的最低值

5、pow(x,y) 返回x的y次幂

6、random() 返回0 ~ 1之间的随机数

7、round(x) 把数四舍五入为最接近的整数

Math.round(2.1); //2

Math.round(2.5); //3

8、sin(x) 返回数的正弦

9、sqrt(x) 返回数的平方根

10、tan(x) 返回角的正切值

11、valueOf 返回Math对象的原始值

12、abs(x) 返回数的绝对值

13、acos(x) 返回数的反余弦值

14、asin(x) 返回数的反正弦值

15、atan(x) 以介于-PI/2与PI/2弧度之间的数值来返回数值的反正切值

16、atan2(y,x) 返回从x轴到点(x,y)的角度（介于-PI/2与PI/2弧度之间）。

17、ceil(x) 对数进行上舍入

Math.ceil(3.9); //4

Math.ceil(-3.9); //-3

Math.ceil(3.1); //4

18、cos(x) 返回数的余弦

19、exp(x) 返回e的指数

# 09.05

补充：DOMContentLoaded事件（DOM对象创建完毕后触发）

<body>

<div id="div1">红红火火恍恍惚惚</div>

</body>

<script type="text/javascript">

if(window.addEventListener){

//IE9+

document.addEventListener("DOMContentLoaded",function(){

alert("DOM对象创建完毕~");

});

}else{

//IE6/7/8

document.attachEvent("onreadystatechange",function(){

if(document.readyState=='complete'){

alert("DOM对象创建完毕~");

}

});

}

window.onload=function(){

alert("window.onload");

}

</script>

-------------------------------------------------------------

**扩展内容——addEventListener、attachEvent**

预备知识：

1.使用事件监听器方法绑定事件：要求兼容

DOM对象.addEventListener("事件名",function(){});

DOM对象.attachEvent("on事件",function(){});

2.DOM对象和内置对象都是混合模式创建的对象。

属性——工厂模式

方法——原型模式

DOM对象和用户自定义对象有什么区别呢?

修改自定义对象的属性，HTML不会变；

但如果修改了DOM对象的属性，那么HTML就会变

JavaScript解释器和HTML解释器解释的是不同的代码，JavaScrip要想操作HTML，只能通过DOM对象来操作。

每个标记都会对应一个已经创建好的DOM对象。

如何从当前文档获取DOM对象？

1.通过id值获取DOM对象

var 对象名=document.getElementById("id名");

2.通过标签名获取DOM对象组成的数组

var 数组名=document.getElementsByTagName("标记名");

DOM对象的属性：

1.innerHTML代表标记中的内容

2.className代表类选择器

\*\*\*\*\*

JavaScript事件绑定方式：

1.行内绑定：angular、react、vue使用

<标签名 on事件名="处理函数1();处理函数2();……"></标签名>

缺点：js代码和html代码无法分离；维护困难。

2.DOM对象.on事件名=function(){

//事件处理语句

}

缺点：同一个DOM对象，同一个事件绑定多次，后面的会覆盖前面的，不能模拟同一个事件响应多个函数的情况。

3.事件监听器方法绑定事件

概述：javascript兼容性主要是由于IE6/7/8私有写法，不符合W3C规范造成的。

Chrome、Firefox等标准浏览器：

window.addEventListener("事件名",function(){},false/true);

IE6/7/8浏览器：

window.attachEvent("on"+"事件名",function(){});

IE9/10浏览器：两种写法都支持

①事件监听器绑定之未封装函数：

if(window.addEventListener){

//IE9+

window.addEventListener("load",function(){

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

div1.addEventListener("click",function(){

alert("触发了click111事件！");

});

div1.addEventListener("click",function(){

alert("触发了click222事件！");

});

});

}else{

//IE6/7/8

window.attachEvent("onload",function(){

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

div1.attachEvent("onclick",function(){

alert("触发了click222事件!");

});

div1.attachEvent("onclick",function(){

alert("触发了click111事件!");

});

})

}

②事件监听器绑定之addEvent封装函数：

function addEvent(elem,eventType,func){

//elem DOM对象

//eventType 事件类型

//func 事件处理函数

if(elem!=null && typeof elem=="object"){

if(window.addEventListener){ //IE9+

elem.addEventListener(eventType,func);

}else{ //IE6/7/8

elem.attachEvent("on"+eventType,func);

}

}else{

alert("对象为空或不是对象！");

}

}

补充：如果定义的变量准备在将来用于保存对象，那么最好将该变量初始化为 null 而不是其他值。（from：JavaScript高级程序设计第3版p26）

③通过封装好的函数进行绑定：

<body>

<div id="div1">

韩寒会画画后悔画韩红

</div>

</body>

<script type="text/javascript" src="addEvent.js">

</script>

<!--注意这里是两个不同的script标签-->

<script type="text/javascript">

//相当于window.onload

addEvent(window,"load",function(){

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

addEvent(div1,"click",function(){

alert("你触发了click事件哦~");

});

});

</script>

作业：

1、用事件监听器方法，实现选项卡菜单

2、用事件监听器方法，实现自动时间

3、用事件监听器方法，实现验证码

例：（addEvent.js略）

function getObject(){

var changeYzm=document.getElementById("changeYzm");

return changeYzm;

}

function showYzm(){

var yzm=document.getElementById("yzm");

var str="ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

var strYzm='';

for(var i=1;i<=4;i++){

var position=Math.floor(Math.random()\*999)%36;

strYzm+=str.substring(position,position+1);

}

yzm.innerHTML=strYzm;

addEvent(getObject(),"click",showYzm);

}

addEvent(window,"load",showYzm);

例PLUS：

addEvent(window,"load",function showYzm(){

var yzm=document.getElementById("yzm");

var changeYzm=document.getElementById("changeYzm");

var str="ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

var strYzm='';

for(var i=1;i<=4;i++){

var position=Math.floor(Math.random()\*999)%36;

strYzm+=str.substring(position,position+1);

}

yzm.innerHTML=strYzm;

addEvent(changeYzm,"click",showYzm);

});

-------------------------------------------------------------------------

遍历数组：把数据元素一个一个拿出来进行处理或显示。

1、数组元素属性.length

for(var i=0;i<数组名.length;i++){

语句;

}

2、for-in

for(var 变量 in 数组){

语句;

}

例1：

//for-in 遍历对象

var obj={name:"Tom",age:19,sex:"男"};

for(var key in obj){

console.log(key,obj[key]);

}

例2：

//for-in 遍历数组

var arr=["css","javascript","angular","react","vue"];

for(var i in arr){

console.log(i,arr[i]);

}

## Number对象：

Number对象是原始数值的包装对象。

创建一个数字对象：

(1)var num=new Number(2);

(2)var num=2;

Number对象属性：

MAX\_VALUE 可表示的最大的数。

MIN\_VALUE 可表示的最小的数。

NaN 非数字值。

NEGATIVE\_INFINITY 负无穷大，溢出时返回该值。

POSITIVE\_INFINITY 正无穷大，溢出时返回该值。

prototype 向对象添加属性和方法。

**Number对象的方法：**

1、toString 把数字转换为字符串，使用指定的基数。

2、toFixed 把数字转换为字符串，结果的小数点后有指定位数的数字。

面试题：

var x=9;

x.toFixed(0); //“9”

9.toFixed(0); //报错，因为9是常量

9.0.toFixed(2); // "9.00"（解释器的原因，之后探讨）

## Date对象（日期对象）：

Date对象：存储和操作指定日期和时间的对象。

说明：日期对象中的时间，不会自动改变。

创建日期对象：

var 日期对象=new Date(); //构造函数创建

Date对象的属性：

prototype属性

**Date对象的方法：**

1、getDate() 从Date对象返回一个月中的某一天(1--31)。

2、getDay() 从Date对象返回一周中的某一天(0--6)。

3、getMonth() 从Date对象返回月份(0--11)。

4、getFullYear() 从Date对象以四位数字返回年份。

5、getYear() （已废弃）使用getFullYear()方法代替。

6、getHours() 返回Date对象的小时(0--23)。

7、getMinutes() 返回Date对象的分钟(0--59)。

8、getSeconds() 返回Date对象的秒数(0--9)。

9、getMilliseconds() 返回Date对象的毫秒(0--999)。

**10、getTime() 返回1970年1月1日至今的毫秒数。**

11、parse() 返回1970年1月1日午夜到指定日期（字符串）的毫秒数。

12、setDate() 设置Date对象中月的某一天(1--31)。

13、setMonth() 设置Date对象中月份(0--11)。

14、setFullYear() 设置Date对象中的年份（四位数字）。

15、setYear() 请使用setFullYear()方法代替。

16、setHours() 设置Date对象中的小时(0--23)。

17、setMinutes() 设置Date对象中的分钟(0--59)。

18、setSeconds() 设置Date对象中的秒钟(0--59)。

19、setMilliseconds() 设置Date对象中的毫秒(0--999)。

**20、setTime() 以毫秒设置Date对象。**

21、toString() 把Date对象转换为字符串。

22、toTimeString() 把Date对象的时间部分转换为字符串。

23、toDateString() 把Date对象的日期部分转换为字符串。

**24、toGMTString() 请使用toUTCString()方法代替。**

**25、toUTCString() 根据世界时，把Date对象转换为字符串。**

**26、toLocaleString() 根据本地时间格式，把Date对象转换为字符串。**

27、toLocaleTimeString() 根据本地时间格式，把Date对象的时间部分转换为字符串。

28、toLocaleDateString() 根据本地时间格式，把Date对象的日期部分转换为字符串。

29、UTC() 根据世界时返回1970年1月1日到指定日期的毫秒数。

例1：自定义的日期时间

window.onload=function show(){

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

var myDate=new Date();

var year=myDate.getFullYear();

var month=myDate.getMonth()+1;

var date=myDate.getDate();

var hours=myDate.getMinutes();

hours=hours<=9?"0"+hours:hours; //防抖处理

var minutes=myDate.getMinutes();

minutes=minutes<=9?"0"+minutes:minutes;

var seconds=myDate.getSeconds();

seconds=seconds<=9?"0"+seconds:seconds;

div1.innerHTML=year+"年"+month+"月"+date+"日"+hours+":"+minutes+":"+seconds;

setInterval(show,1000);

}

例2：2020年元旦倒计数

function fangDou(time){ //防抖处理

return time<=9?"0"+time:time;

}

function getDate(){

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

var myDate=new Date();

var nowTime=myDate.getTime(); //现在距离1970年的毫秒数

var time2020=new Date("2020/1/1 00:00:00");

var aheadTime=time2020.getTime(); //2020年距离1970年的毫秒数

var seconds=Math.floor((aheadTime-nowTime)/1000); //2020年距离现在的秒数

var days=Math.floor(seconds/86400); //一天有24\*60\*60=86400秒

seconds=seconds%86400; //剩下的秒数

var hours=Math.floor(seconds/3600);

seconds=seconds%3600; //剩下的秒数

var minutes=Math.floor(seconds/60);

seconds=seconds%60; //剩下的秒数

var str=fangDou(days)+"天"+fangDou(hours)+"小时"+fangDou(minutes)+"分钟"+fangDou(seconds)+"秒";

div1.innerHTML=str;

}

setInterval(getDate,1000);

*补充1：*

在JavaScritp中使用计时事件是很容易的，两个关键方法是：

1、setInterval()：间隔指定的毫秒数不停地执行指定的代码。

setInterval(函数,毫秒); //一般使用匿名形式，1000毫秒是一秒。

2、setTimeout()：在指定的毫秒数后执行指定代码。

注意：setInterval()和setTimeout()是HTML DOM Window对象的两个方法。

*补充2：*

1、实验场景：cookie，10天免登录（将cookie存储10天），需要获取当前时间和10天以后的时间。

①创建日期对象，日期对象存储当前时间

②根据日期对象获取当前时间的时间戳

③根据当前时间戳计算10天以后的时间戳

④用10天以后的时间戳，设置日期对象

⑤得到10天以后的日期对象，即10天以后的时间

代码：

var myDate=new Date();

console.log(myDate.toLocaleString());

//获得当前时间的时间戳

var ms=myDate.getTime();

console.log(ms);

//获得10天以后的时间戳

var ms10=ms+10\*24\*3600\*1000;

console.log(ms10);

//用10天以后的时间戳,设置日期对象

myDate.setTime(ms10);

console.log(myDate.toLocaleString());

2、cookie存储时，需要指定过期时间，要求时间格式是GMT时间格式。

toGMTString(); 已废弃。请使用toUTCString()方法代替。

toUTCString(); 根据世界时，把Date对象转换为字符串。

补充：协调世界时(UTC)，英文：Coordinated Universal Time ，别称：世界统一时间，协调世界时，又称世界统一时间，世界标准时间，简称UTC。GMT（Greenwish Mean Time 格林威治平时），这是UTC的民间名称。GMT=UTC。

3、对象和字面量

字面量：3 "agc" true

补充：字面量表示如何表达这个值，一般除去表达式，给变量赋值时，等号右边都可以认为是字面量。

字面量分为字符串字面量(string literal)、数组字面量(array literal)和对象字面量(object literal)，另外还有函数字面量(function literal)。

示例：

var test="hello world!";

"hello world!"就是字符串字面量，test是变量名。

## Function对象：

JavaScript的函数属于Function对象。

创建函数的方式：

1、function 函数名(参数,参数,…){

语句;

}

2、通过内置的JavaScript函数构造器（Function()）定义函数：

var 函数名=new Function([参数1,[参数2,[参数n]]],[函数体]);

说明：方式2不参加预编译。

例：

var test = new Function("a", "b", "return a \* b");

alert(test(4, 3)); //12

**Function对象的属性：**

1、arguments属性：代表正在运行的函数，调用函数时的**实参**（以数组形式存储的，但是它不是数组）。

arguments对象的属性：

①length：**实参**个数

②callee属性：返回当前被调用的函数对象（用于匿名函数递归调用）

说明1：js中函数的实参和形参个数可以不相等。

说明2：同一个函数，参数个数不同或参数类型不同，实现不同的功能，称为函数重载。例：

function test(x,y){

switch(arguments.length){

case 0:alert("功能1");

break;

case 1:alert("功能2");

break;

case 2:alert("功能3");

break;

default:alert("功能4");

}

}

例1：

function test(){

alert("第一个实参："+arguments[0]); //1

alert("第二个实参："+arguments[1]); //8

alert("第三个实参："+arguments[2]); //5

}

test(1,8,5);

例2：

function test(){

return arguments.callee; //返回当前被调用的函数对象

}

document.write(test());

例3：

//通过arguments.callee属性实现匿名函数递归调用

(function(count){

if(count<5){

alert(count);

arguments.callee(++count);

}

})(1);

2、length属性：代表函数**形参**个数

例：

function test(x,y){

alert("形参个数为："+test.length); //函数名.length:形参个数

alert("实参个数为："+arguments.length);//arguments.length:实参个数

if(test.length!=arguments.length){

alert("参数个数不正确");

}

}

test(1,2,4);

3、caller属性：返回一个调用该函数对象的外层函数的引用

function A(){

return A.caller; //若直接调用A函数返回null

}

function B(){

return A();

}

alert(A()); //null

alert(B()); //返回B函数

4、constructor属性：返回函数构造函数Function()

说明：其他所有对象都有这个属性

例：

function test(a){

return a;

}

alert(test.constructor); //function Function(){[native code]}

5、name属性：返回函数名称

6、prototype属性

--------------------------

arguments补充：

function fn1(a, b, c) //形参——形式上，a、b、c这些名代表1、2、3

fn1(1, 2, 3); //实参——实际传递的参数

如果不写形参，1、2、3也能够传进来，都存在arguments中。

function fn1(){

//arguments→[1, 2, 3] ——实参的集合（不是数组，但是类似数组，有length，也可以用下标找到其中的数据）

}

1、当函数参数个数无法确定的时候，用arguments。

例：

alert(sum(1,2,3)); //6

function sum(){

var n=0;

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

n += arguments[i];

}

return n;

}

2、arguments用于函数重载：

说明：JS不支持函数重载，如果出现函数名相同的情况下，后者会覆盖前者。而利用arguments可以根据不同参数长度来实现让同一个函数，进行不同处理。

function test(x,y){

if(arguments.length==1){

alert("功能1");

}

if(arguments.length==2){

alert("功能2");

}

if(arguments.length==3){

alert("功能3");

}

}

3、arguments.callee的作用：

arguments 的主要用途是保存函数参数，但这个对象还有一个名叫callee的属性，返回正被执行的Function对象，也就是所指定的Function对象的正文，这有利于匿名函数的递归或者保证函数的封装性。

例1：

//经典阶乘函数

function factorial(num){

if (num <=1) {

return 1;

} else {

return num\*factorial(num-1)

}

}

document.write(factorial(5)); //120

定义阶乘函数一般都要用到递归算法；如上面的代码所示，在函数有名字，而且名字以后也不会变的情况下，这样定义没有问题。

但问题是这个函数的执行与函数名factorial紧紧耦合在了一起。为了消除这种紧密耦合的现象，可以像下面这样使用arguments.callee。

例2：

function factorial(num){

if(num<=1){

return 1;

}else{

return num\*arguments.callee(num-1);

}

}

在这个重写后的factorial()函数的函数体内，没有再引用函数名factorial。这样，无论引用函数时使用的是什么名字，都可以保证正常完成递归调用。

例3：

function factorial(num){

if(num<=1){

return 1;

}else{

return num\*arguments.callee(num-1);

}

}

var trueFactorial=factorial;

alert(trueFactorial(5)); //120

factorial=function(){

return 0;

}

alert(trueFactorial(5)); //120

在此，变量trueFactorial获得了factorial的值，实际上是在另一个位置上保存了一个函数的指针。然后，我们又将一个简单地返回0的函数赋值给factorial变量。如果像原来的factorial()那样不使用arguments.callee，调用trueFactorial()就会返回0。可是，在解除了函数体内的代码与函数名的耦合状态之后，trueFactorial()仍然能够正常地计算阶乘；至于factorial()，它现在只是一个返回0的函数。

----------------------------------------------------------------------------------------------------

1、正则表达式：字符串（文本）的匹配规则

说明：不同语言javascript php C# java等有不同的实现

常用正则表达式：点击[这里](常用正则表达式.docx)

2、正则表达式功能：匹配特殊格式字符串

举例：验证邮箱、身份证、电话号码、网址等

3、如何使用正则表达式对象？

(1)创建正则表达式对象

var 对象名=/规则/[修饰符]

修饰符i：表示搜索不区分大小写

修饰符g：表示全文搜索，如果不带g找到第一个匹配的字符串就停止搜索

(2)正则表达式对象唯一的一个方法：

正则表达式对象.test("字符串");

test()方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回true，否则返回false。

**补充：**

1.正则表达式修饰符（修饰符可以在全局搜索中不区分大小写）：

i 执行对大小写不敏感的匹配。

g 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。

m 执行多行匹配。

2.方括号（方括号用于查找某个范围内的字符）：

[abc] 查找方括号之间的任何字符。

[^abc] 查找任何不在方括号之间的字符。

[0-9] 查找任何0至9的数字。

[a-zA-Z] 查找所有字母。

3. 元字符（元字符是拥有特殊含义的字符）：

. 查找单个字符，除了换行和行结束符。

\w 查找单词字符(字母、数字、下划线)。

\W 查找非单词字符

\d 查找数字。

\s 查找空白字符。

\b 匹配单词边界。

\xxx 查找以八进制数xxx规定的字符。

\xdd 查找以十六进制数dd规定的字符。

\uxxxx 查找以十六进制数xxxx规定的Unicode字符。

x|y 匹配 x 或 y。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood"或"food"。

补充①：

对/chrome\/[\d.]+/的解释：

\代表字符转义，\d表示数字，.在中括号中就表示小数点，因此[\d.]+就表示一串数字和小数点组成的字符串。整个正则表达式代表chrome/233.233这样的字符串。

补充②：

当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）。比如，以下是等价的：

var re = new RegExp("\\w+");

var re = /\w+/;

补充③：

|  |  |
| --- | --- |
| 等价写法 | 字符类 |
| \d | [0-9] 数字字符 |
| \D | [^0-9] 非数字字符 |
| \s | [\t\n\x0B\f\r] 空白符 |
| \S | [^\t\n\x0B\f\r] 非空白符 |
| \w | [a-zA-Z\_0-9] 单词字符(字母、数字、下划线) |
| \W | [^a-zA-Z\_0-9] 非单词字符 |

4.量词：

n+ 匹配任何包含至少一个 *n* 的字符串。

例如，/a+/ 匹配 "candy" 中的 "a"，"caaaaaaandy" 中所有的 "a"。

又例如，/ba+/将匹配ba,baa,baaa。

n\* 匹配任何包含零个或多个 *n* 的字符串。

例如，/bo\*/ 匹配 "A ghost booooed" 中的 "boooo"，"A bird warbled" 中的 "b"，但是不匹配 "A goat grunted"。

又例如，/ba\*/将匹配b,ba,baa,baaa。

n? 匹配任何包含零个或一个 *n* 的字符串。

例如，/e?le?/ 匹配 "angel" 中的 "el"，"angle" 中的 "le"。

又例如，/ba？/将匹配b,ba。

^n 匹配任何开头为n的字符串。

n$ 匹配任何结尾为n的字符串。

n{X} 匹配包含X个n的序列的字符串。

例如，/a{2}/ 不匹配 "candy," 中的 "a"，但是匹配 "caandy," 中的两个 "a"，且匹配 "caaandy." 中的前两个 "a"。

n{X,} X 是一个正整数。前面的模式 n 连续出现至少 X 次时匹配。

例如，/a{2,}/ 不匹配 "candy" 中的 "a"，但是匹配 "caandy" 和 "caaaaaaandy." 中所有的 "a"。

n{X,Y} X和Y为正整数。前面的模式n连续出现至少X次，至多Y次时匹配。

例如，/a{1,3}/不匹配"cndy"，匹配"candy,"中的"a"，"caandy,"中的两个"a"，匹配"caaaaaaandy"中的前面三个"a"。注意，当匹配 "caaaaaaandy"时，即使原始字符串拥有更多的"a"，匹配项也是"aaa"。

5、支持正则表达式的String对象的方法：

search 检索与正则表达式相匹配的值

match 找到一个或多个正则表达式的匹配

replace 替换与正则表达式匹配的子串

split 把字符串分割为字符串数组

6、贪婪模式与非贪婪模式：

限定符在默认下尽可能多的匹配的即**贪婪模式**。

例如： '123456789'.match(/\d{3,5}/g); //["12345", "6789"]

结果： ["12345", "6789"]

让正则表达式尽可能少的匹配，也就是说一旦成功匹配不再继续尝试，做法很简单，在量词后加上“?”即可,即**非贪婪模式**。

例如： '123456789'.match(/\d{3,5}?/g); //["123", "456", "789"]

结果： ["123", "456", "789"]

7、有时候我们希望使用限定符的时候匹配多个字符，而不是只匹配一个。

例如，希望匹配Tom出现5次的字符串，我们如果写成 Tom{5}的话匹配的是：To出现1次，m出现5次。

怎么把Tom作为一个整体呢？使用**()**就可以，我们称为**分组。**

例如：（Tom|Jarry）{5} //匹配Tom或Jarry5次

## RegExp对象：

创建正则表达式对象的方式：

1、var reg=/表达式/对象的属性;

2、var reg=new RegExp(表达式[对象的属性]);

RegExp（Regular Expression）对象的属性：

g：global，全文搜索，默认搜索到第一个结果就停止

i：ignoreCase，忽略大小写，默认大小写敏感

m：multiplelines，多行搜索

例： var reg=/j.\*/ig;

reg.global; //true

注：global属性用于返回正则表达式是否具有"g"。如果g标志被设置，则该属性为true，否则为false。

**RegExp对象的方法：**

1、test（字符串）方法，仅仅判断字符串是否能够匹配正则表达式，匹配返回true，不匹配返回false。

例：

var reg=/[\u4E00-\u9FA5\uf900-\ufa2d]/; //匹配中文

var str=prompt("请输入文本：","韩寒");

if(reg.test(str)){

alert("输入文本包含中文");

}else{

alert("输入文本不包含中文");

}

**回顾——String对象以正则表达式为参数的函数：**

2、match(要搜索的字符串)方法，可在字符串内搜索指定子字符串，并返回到一个数组中。

match(reg)方法，匹配一个或多个正则表达式的字符串，返回到数组中。

例：

var str="张三的歌，张宇给你们";

var reg=/张./g;

alert(str.match(reg)); //张三，张宇

3、replace(要搜索的字符串,替换成的字符串) 方法，用于在字符串中用一些字符替换另一些字符。

replace(正则表达式,替换成什么字符串)方法，或替换一个与正则表达式匹配的子串。

例：

var str="张三的歌，张宇给你们";

var reg=/张./g;

var newStr=str.replace(reg,"张学友");

document.write(newStr); //张学友的歌，张学友给你们

4、search(字符串)方法用于搜索字符串中指定的子字符串，返回**第一次匹配的起始位置**，没有匹配的返回-1。

search(正则表达式)方法，或检索与正则表达式相匹配的子字符串,返回**第一次匹配的起始位置**，没有匹配的返回-1。

例：

var str="张三的歌，张宇给你们";

var reg=/你./g;

var index=str.search(reg);

alert(index); //8

5、split(字符串分隔符)方法，用于搜索指定分隔符，将字符串按照指定分隔符分割，并返回字符串数组。

split(正则表达式)方法，用于搜索匹配正则表达式，将字符串按照指定匹配正则表达式进行分割，并返回字符串数组。

例：

var str="张三的歌，张宇给你们";

var reg=/张./g;

alert(str.split(reg)); //,的歌，,给你们

## **Error对象：**

创建方式有两种：

(1)当监控代码中出现，语法错误，系统会自动创建Error对象，并赋值。

例：

//程序运行中发生错误，系统自动创建Error

try{

test();

}catch(e){

alert("e.message是："+e.message); //test is not defined

}

(2)当监控代码中出现，用户自己创建的new Error()，抛出人为异常对象。

例1：

//用户自己创建Error对象1

var err=new Error();

err.message="自定义错误";

alert(err.message);

例2：

//用户自己创建Error对象2

var err=new Error("自定义错误");

alert(err.message);

Error对象的属性：

×name 错误类型（不同浏览器有差异，IE6/7/8有的错误类型不同）

×number 错误号（仅IE支持，32位，高16位机器代码，低16位错误代码）

√message 错误信息（不同浏览器有差异）

代码监控：

try{

//要监控的代码

}catch(e){

//错误或异常处理

}finally{

//语句

}

try-catch异常处理优点：

①防止用户看到不友好的系统错误；

②由于Error在不同浏览器，报错信息有差异，可以使用try-catch进行统一报错信息；

③可以处理人为编写程序造成异常，增强程序的健壮性。

注意：

①finally 语句在try和catch之后无论有无异常都会执行。

②catch和finally语句都是可选的，但你在使用try语句时必须至少使用一个。

③当错误发生时，JavaScript会停止执行，并生成一个错误信息。使用 throw 语句来创建自定义消息(抛出异常)。

例1：

try{

//产生一个语法错误，js解释器创建第一个Error对象，并且中断程序执行，赋值给形参e

test();

console.log("韩寒");

}catch(e){

console.log(e.message); //test is not defined

console.log("出错了~"); //出错了~

}

例2：

try{

//人为抛出异常，用户自己创建的Error对象

throw new Error("韩寒会画画后悔画韩红");

console.log("ABC");

}catch(e){

alert(e.message); //韩寒会画画后悔画韩红

}

作业：

敏感词过滤

倒计时

# 09.06

JavaScript对象的分类：

1.用户自定义对象{}

2.内置对象

--String

--Array

--Math

--Date

--Function

--Number

--RegExp

--Error

3.BOM(Browser Object Mode)对象：代表整个浏览器窗口

获取或设置浏览器窗口相关信息

4.DOM(Document Object Model)对象：代表浏览器窗口中的整篇文档

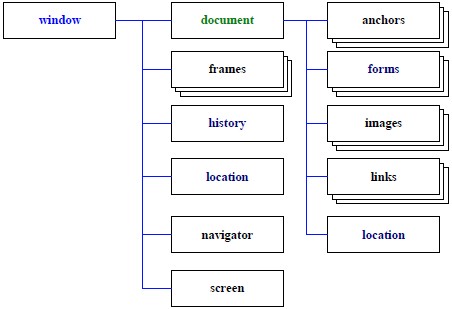
获取或设置DOM对象对应的HTML文档相关信息

5.事件对象

6.ajax对象

--------------------

**BOM、DOM、window、document对象的关系：**



## √window对象——BOM核心

window对象的属性：

1、innerHeight：获取视口高度（浏览器的视口，不包括工具栏和滚动条）

IE6/7/8不支持

兼容IE6/7/8的写法：document.documentElement.clientHeight

2、innerWidth：获取视口宽度（浏览器的视口，不包括工具栏和滚动条）→用户做移动端响应式开发

3、length：设置或返回窗口中的框架数量

说明：<iframe></iframe>：将其他网页嵌入到当前网页

--------------------------------------------------------------------------

window对象的方法：

1、alert(表达式或值);

弹出一个带确定按钮的对话框

并且程序暂停执行

纯js

2、confirm(表达式或值);

弹出一个带确定和取消按钮的对话框，用户点击确定返回true，用户点击取消返回false

能和用户交互，一般用户管理系统动态跳转

例：

//管理系统动态跳转

var flag=confirm("是否继续添加大宝贝？");

if(flag){

location.href="add.html"; //动态跳转

}else{

location.href="index.html";

}

3、prompt("提示信息",默认值);

弹出一个带确定和取消按钮的对话框，用户点击确定返回用户输入的字符串，用户点击取消返回null（开发基本不用）。

4、open("url","窗口名称","属性=值,属性=值......");

打开一个窗口，加载执行资源（用户体验差，开发的时候基本不用）。



5、print(); 打印当前页面内容

6、close(); 关闭当前浏览器页面

7、scrollBy(x,y); 按照指定的像素值来滚动内容

8、scrollTo(x,y); 把内容滚动到指定的坐标

scrollTo(0,0); //回到顶部

9、setTimeout(函数,毫秒); 在指定的毫秒数后调用函数或计算表达式

说明：返回一个ID（数字），可以将这个ID传递给clearTimeout()来取消执行。

例：

var x=setTimeout(function(){

alert("起床了！");

},3000);

var y=setTimeout(function(){

alert("起床了！");

},4000);

var z=setTimeout(function(){

alert("起床了！");

},5000);

console.log(x,y,z); //返回值分别为：1 2 3

10、clearTimeout(); 取消由setTimeout()方法设置的timeout

例：

var a1=setTimeout(function(){

alert("韩寒会画画后悔画韩红");

},3000);

var bt1=document.getElementById("bt1");

bt1.onclick=function(){

clearTimeout(a1);

};

11、setInterval(函数,毫秒); 每隔多少毫秒，调用一次函数

说明：返回一个ID（数字），可以将这个ID传递给clearInterval()以取消执行。

例：

var i=1;

function show(){

console.log(i++);

}

setInterval(show,1000);

补充：有JS代码如下

function test (){

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

div1.style.left=div1.offsetLeft+10+"px";

}

window.onload=function(){

setInterval(test(),100); //有误

setInterval(test,100); //正确

setInterval("test()",400); //正确

setInterval("test",400); //有误

}

这里首先涉及到括号的二义性：

①当函数名后加上括号时，通常会执行函数体本身。如果函数有返回值时，此时会得到函数的返回值；

②当函数名后不加括号时，其实质上是一个函数指针，只是用于找到函数体的位置，不会直接执行函数体；

setInterval("test()",400); //全局函数的前提下，写成字符串的时候和eval()的执行原则一样，可以这样理解，把它当成了一段字符串格式的js代码执行，还是相当于一个函数的调用。

总结：

对于setInterval()函数来说，第一个参数加()是以字符串的方式传递的，它的作用域是全局作用域；不加()是以函数引用的方式传递。

12、clearInterval(); 取消由setInterval()设置的timeout

例： //倒计时543210

window.onload=function(){

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

var i=5;

var a1=setInterval(function(){

if(i==0){

clearInterval(a1);

}

div1.innerHTML=i--;

},1000);

}

13、其他

resizeBy(); 窗口调整宽度和高度(仅IE支持)

resizeTo(); 把窗口的大小调整到指定的宽度和高度(仅IE支持)

# 09.07

浏览器内核简介：



## √Navigator对象：

navigator对象——包含有关浏览器的信息

**navigator对象的属性：**

①appCodeName：返回浏览器的代码名

说明：在所有以Netscape代码为基础的浏览器中，它的值都是 Mozilla。为了兼容起见，在Microsoft的浏览器中，它的值也是Mozilla。

浏览器代码名称：Mozilla

②appName：返回浏览器的名称

说明：chrome浏览器名称返回的是NetScape，因为根据W3C HTML 5的规范，navigator对象的appName要么返回Netscape，要么返回浏览器的全名，这是为了兼容性而考虑的！

浏览器名称：Netscape

③appVersion：返回浏览器的平台和版本信息

说明：浏览器的平台和版本信息：5.0 (Windows NT 10.0;WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

④browserLanguage：返回当前浏览器的语言

说明：当前浏览器的语言：undefined

**⑤cookieEnabled**：返回指明浏览器中是否启用cookie的布尔值

说明：浏览器中是否启用cookie 的布尔值：true

⑥platform：返回运行浏览器的操作系统平台

说明：浏览器的操作系统平台：Win32

**⑦userAgent**：返回由客户机发送服务器的user-agent头部的值

说明：由客户机发送服务器的user-agent头部的值：Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,like Gecko) Chrome/63.0.3239.132 Safari/537.36

补充：

每次向服务器发送请求，都会有headers信息。

headers信息：

请求url地址

请求方法：get/post

请求状态码

请求头request headers

navigator.userAgent

cookie

请求体(可选)：客户端浏览器向服务器发送的数据

响应头response headers：服务器发送给浏览器的信息

-----------------------------------------------------------------------------

注：以下三例代码不够完善，慎看。

例1：判断浏览器类型

function browserType(){

var explorer=window.navigator.userAgent;

//判断是否为IE浏览器

if(explorer.indexOf("MSIE")>=0||explorer.indexOf("Trident/7.0")>=0){

return "ie";

}

//判断是否为edge浏览器

else if(explorer.indexOf("Edge")>=0){

return "edge";

}

//判断是否为Firefox浏览器

else if(explorer.indexOf("Firefox")>=0){

return "Firefox";

}

//判断是否为Chrome浏览器

else if(explorer.indexOf("Chrome")>=0){

return "Chrome";

}

//判断是否为Opera浏览器

else if(explorer.indexOf("Opera")>=0){

return "Opera";

}

//判断是否为Safari浏览器

else if(explorer.indexOf("Safari")>=0){

return "Safari";

}

}

alert(browserType());

例2：判断浏览器类型及版本号

function browserType(){

var explorer=window.navigator.userAgent;

//判断是否为IE浏览器

if(explorer.indexOf("MSIE")>=0||explorer.indexOf("Trident/7.0")>=0){

return explorer.match(/MSIE[\d.]+/ig);

}

//判断是否为edge浏览器

else if(explorer.indexOf("Edge")>=0){

return explorer.match(/Edge\/[\d.]+/ig);

}

//判断是否为Firefox浏览器

else if(explorer.indexOf("Firefox")>=0){

return explorer.match(/Firefox\/[\d.]+/ig);

}

//判断是否为Chrome浏览器

else if(explorer.indexOf("Chrome")>=0){

return explorer.match(/Chrome\/[\d.]+/ig);

}

else{

return "未知浏览器类型";

}

}

alert(browserType());

例3：判断浏览器内核

function browserCore(){

var explorer=navigator.userAgent;

if(explorer.match(/webkit/i)){

return explorer.match(/webkit\/[\d.]+/ig);

}else if(explorer.match(/trident/i)){

return explorer.match(/trident\/[\d.]+/ig);

}else if(explorer.match(/gecko/i)){

return explorer.match(/gecko\/[\d.]+/ig);

}else{

return "未知内核";

}

}

alert(browserCore());

**强烈注意！**

查看Internet Explorer的userAgent的正确方法：

Internet Explorer→F12→仿真→用户代理字符串（不能用IE的模仿文档模式）

-----------------------------------------------------------------------------------------------------

navigator对象的方法：

javaEnabled(); 规定浏览器是否启用Java

说明：规定浏览器是否启用Java：false

## Screen对象：

screen对象——对象包含有关客户端显示屏幕的信息

screen对象的属性：

availHeight 返回显示屏幕的可用高度 (除Windows任务栏之外)

availWidth 返回显示屏幕的可用宽度 (除Windows任务栏之外)

height返回显示屏幕的高度

width返回显示屏幕的宽度

例：

var h1=screen.availHeight;

var w1=screen.availWidth;

var h2=screen.height;

var w2=screen.width;

console.log("当前屏幕高度(除任务栏外)："+h1);

console.log("当前屏幕宽度："+w1);

console.log("当前屏幕分辨率："+h2+"\*"+w2);

## History对象：

history对象——包含用户（在浏览器窗口中）访问过的URL

history对象的属性：

length：返回浏览器历史列表中的URL数量

history对象的方法：

1、back(); 加载history列表中的前一个URL

2、forward(); 加载history列表中的下一个URL

例：

<body>

<a href="demo02.html">测试链接</a><br />

<a href="javascript:;" id="back1">上一个URL，相当于后退</a><br />

<a href="javascript:;" id="forward1">下一个RUL，相当于前进</a>

</body>

<script type="text/javascript" src="addEvent.js">

</script>

<script>

function load(){

var back1=document.getElementById("back1");

var forward1=document.getElementById("forward1");

function forwardAction(){

history.forward();

}

function backAction(){

history.back();

}

addEvent(back1,"click",backAction);

addEvent(forward1,"click",forwardAction);

}

addEvent(window,"load",load);

</script>

addEvent.js文件：↓

function addEvent(elem,eventType,func){

if(elem&&typeof(elem)=="object"){

if(elem.addEventListener){

elem.addEventListener(eventType,func,false); //1.兼容其他浏览器；2.false:冒泡获取由里向外

}else{

elem["on"+eventType]=func; //1.兼容IE；2.相当于elem.oneventType=func……

}

}

}

3、go(); 加载history列表中的某个具体页面

例：

<body>

<a href="demo03.html">测试链接</a><br />

<a href="javascript:;" id="refresh">刷新本页面</a><br />

<a href="javascript:;" id="forward1">上一个页面</a>

</body>

<script type="text/javascript" src="01-04addEvent.js">

</script>

<script type="text/javascript">

function load(){

var refresh=document.getElementById("refresh");

var forward1=document.getElementById("forward1");

function forwardAction(){

history.go(-1); //可用console控制台测试

}

function refreshAction(){

history.go(0);

}

addEvent(refresh,"click",refreshAction);

addEvent(forward1,"click",forwardAction);

}

addEvent(window,"click",load);

</script>

## √Location对象：

location对象——对象包含有关当前URL的信息(javascript里边管理地址栏的内置对象)

location对象的属性：

1、hash 设置或返回“#”号开始的URL（锚）

例：

<style type="text/css">

tr{

vertical-align: top;

text-align: right;

}

a{

background: #6EDDC2;

cursor: pointer;

}

</style>

</head>

<body>

<table border="1" width="100%" height="500">

<tr>

<td><a name="a1">1楼</a></td>

</tr>

</table>

<table border="1" width="100%" height="500">

<tr>

<td><a name="a2">2楼</a></td>

</tr>

</table>

<table border="1" width="100%" height="500">

<tr>

<td><a name="a3">3楼</a></td>

</tr>

</table>

</body>

<script type="text/javascript" src="01-04addEvent.js">

</script>

<script type="text/javascript">

function load(){

var a=document.getElementsByTagName("a");

function hashClick(){

location.hash="#"+this.name;

}

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

addEvent(a[i],"click",hashClick);

}

}

addEvent(window,"load",load);

</script>

2、host 设置或返回主机名和当前URL的端口号

3、hostname 设置或返回当前URL的主机名

4、href 设置或返回完整的URL——通常用来动态跳转

5、pathname 设置或返回当前URL的路径部分——URI

6、port 设置或返回当前URL的端口号

7、protocol 设置或返回当前URL的协议

8、search 设置或返回从问号(?)开始的URL（查询部分）

location对象的方法：

1、assign(); 加载新的文档

例：

function load(){

var btn=document.getElementById("btn");

function btnClick(){

location.assign("http://www.mi.com/");

}

addEvent(btn,"click",btnClick);

}

addEvent(window,"load",load);

2、reload(); 重新加载当前文档(无参方法)

3、replace(); 用新的文档替换当前文档

## Frames对象：

frames对象——对象包含所有框架

## Document对象：

document对象——代表当前文档

从<!doctype html>一直到</html>

预习：第10章

1.什么是DOM对象

2.为什么要学DOM对象

3.DOM对象和其他对象有什么区别

4.怎么获得DOM对象

5.DOM对象都有哪些属性和方法

.innerHTML

.className

# 09.10

JavaScript主要功能：用来在浏览器中和用户进行交互，动态修改HTML（DOM对象）

JavaScript中对象的分类（按功能分）：

1.用户自定义对象 {属性:值,属性:值,...}

直接创建

混合模式→方法共享

2.内置对象

3.BOM对象

4.DOM对象

5.事件对象

6.ajax对象

DOM对象和用户自定义对象有什么区别？

答：修改用户自定义对象的属性，HTML不会更改；

修改DOM对象的某些属性，HTML自动修改。

总结：要想通过js动态修改HTML，必须通过修改DOM对象的属性实现

js→修改DOM对象的属性→html自动改变

html修改→DOM对象也会自动修改

为什么DOM对象可以理解为最特殊的一种对象？

答：因为修改其他对象的属性，不会引发HTML重新渲染；而修改DOM对象的属性会引发HTML重新渲染。

-------------------------------------------

页面加载流程分为两个阶段：

1.从上到下，依次创建每个标记对应的DOM对象，遇到js执行js

2.从上到下依次渲染每个标记，之后触发window.onload事件

---------------------------------------------------------------------------------

DOM对象和HTML标记之间关系：

每个DOM对象对应一个HTML标记；

修改DOM对象的属性，HTML标记自动会更改，同样，修改HTML标记，DOM对象的属性也会自动更改。

------------------------------------

HTML标记的属性分为两类：

1.内置属性：HTML自带的——DOM对象的属性

2.用户自己创建的属性

要求带“data-”

注意：HTML内置属性一定是DOM对象的属性；用户自定义属性，不是DOM对象的属性。

-------------------

如何通过JS修改HTML标记？

1.获取HTML标记对应的DOM对象

2.修改DOM对象的属性（HTML自动改变）

## √DOM对象：

DOM（文档对象模型Document Object Model）：由document根对象（代表整篇HTML文档），及其一系列子对象组成，用于动态修改HTML标记的属性和内容的一组对象。

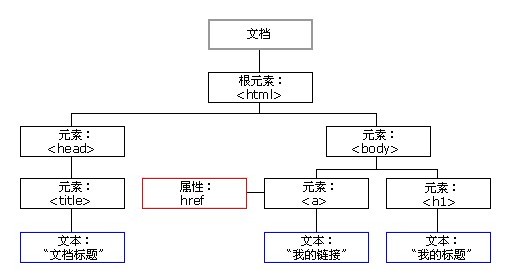
DOM特点：

通过DOM可以对整个HTML文档进行新建、添加、更新、删除等操作。

DOM模型符合WEB标准，兼容性好。

通过DOM可以操纵HTML、XHTML、XML文档。

DOM树：



HTML文档中的每一个元素都是一个节点对象，DOM中规定如下：

1.整个文档是一个文档节点

2.每个HTML标签是一个元素节点

3.包含在HTML元素中的文本是文本节点

4.每一个HTML属性是一个属性节点

5.注释属于注释节点

DOM说明：

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>DOM说明</title>

</head>

<body>

<h1>DOM Lesson one</h1>

<p>Hello world!</p>

</body>

</html>

（1）除文档节点之外的每个节点都有父节点。例如，<head>和<body>的父节点是<html>节点,文本节点”Hello world!”的父节点是<p>。大部分节点都有子节点。例如，<head>节点有一个子节点：<title>节点。<title>节点也有一个子节点：文本节点“DOM说明”

（2）当节点分享同一个父节点时，它们就是兄弟（同级节点）。例如，<h1>和<p>是同辈，因为它们的父节点均是<body>节点。

（3）节点也可以拥有后代，后代指某个节点的所有子节点，或者这些子节点的子节点，以此类推。例如，所有文本节点都是<html>点的后代，而第一个文本节点是<head>节点的后代。

（4）节点也可以拥有祖先。祖先是某个节点的父节点，或者父节点的父节点，以此类推。比如，所有的文本节点都可以把<html>节点作为先辈节点。

Document对象：

1.每个载入浏览器的HTML文档都会成为Document对象

2.Document对象使我们可以从脚本中对HTML页面中的所有元素进行访问

**获得DOM对象的方法（重点，重点，重点）：**

1、通过标记的id值，获取对应的DOM对象

var DOM对象=document.getElementById("id值");

2、通过标记的name属性的值，获取DOM对象组成的数组。即：[DOM对象,DOM对象,DOM对象……]

var 数组对象=document.getElementsByName("name属性值");

说明：一般用于表单项

3、通过标记名，获得DOM对象组成的数组。即：[DOM对象,DOM对象,DOM对象……]

var 数组对象=document.getElementsByTagName("标记名");

4、通过CSS常用选择器，获取匹配选择器的集合中第一个元素对应的DOM对象。

var DOM对象=document.querySelector("各种基本选择器");

说明：IE8+支持

5、通过CSS常用选择器，获取匹配选择器的DOM对象组成的数组。即：[DOM对象,DOM对象,DOM对象……]

var 数组=document.querySelectorAll("各种基本选择器");

说明：IE8+支持

例：

<body>

<div id="div1">韩寒会画画后悔画韩红</div>

<div id="div2" class="special">红红火火恍恍惚惚</div>

<div id="div3" class="special">明天的你，会感谢今天努力的自己</div>

</body>

<script type="text/javascript">

console.log(document.querySelector("div"));

//<div id="div1">韩寒会画画后悔画韩红</div>

console.log(document.querySelector(".special"));

//<div id="div2" class="special">红红火火恍恍惚惚</div>

console.log(document.querySelectorAll(".special"));

//NodeList(2) [div#div2.special, div#div3.special]

</script>

---------------------------------

Document对象节点信息：

（1）节点名称

元素节点的nodeName是标签的名称

属性节点的nodeName是属性的名称(获取属性节点对象名：.attributes[下标])

文本节点的nodeName永远都是#text

文档节点的nodeName永远是#document

（2）nodeValue节点值

对于文本节点，nodeValue属性包含文本

对于属性节点，nodeValue属性包含属性值

nodeValue属性对于文档节点和元素节点是不可用的（null）

（3）nodeType节点类型

|  |  |  |  |
| --- | --- | --- | --- |
| **元素类型** | **节点类型** | **元素类型** | **节点类型** |
| 元素 | 1 | 注释 | 8 |
| 属性 | 2 | 文档 | 9 |
| 文本 | 3 |  |  |

# 09.11（续DOM对象）

DOM对象的分类：

1、js解释器在页面加载的第一个阶自动创建的，并且已经在第二个阶段渲染完毕的

2、用户自己在内存中创建的DOM对象

var DOM对象=document.createElement("标记名");

说明1：内存中的DOM对象，如果不插入到现有的HTML文档中，在浏览器中看不到；创建的DOM对象，要想在HTML文档中看得到，那么必须把创建的DOM对象插入到现有的HTML文档中，引发重新渲染（重绘）。

说明2：JavaScript中最耗费性能的操作就是DOM操作，因为DOM刷新会引发重绘。

1:4000（执行一次DOM操作，相当于执行4000次普通JS语句）

所以，为了提升JS性能，尽量减少DOM刷新次数。

react和vue为了提升JS性能，采用了虚拟DOM，减少了DOM刷新。

---------------------------------------------------------------------------------------------

**Document对象操纵节点：**

**创建节点：**

1、createElement(); 创建一个元素节点（DOM对象）

例：

var div1=document.createElement("div"); //创建一个div元素对象

document.write(div1); //输出这个对象，对象在内存中并未添加到当前文档中

2、createTextNode(); 创建一个文本节点（一般不用）

例：

var t1=document.createTextNode("抽烟喝酒烫头"); //创建一个文本节点对象

document.write(t1);//输出文本节点对象，对象在内存中并未添加到当前文档

**添加节点：**

1、appendChild(node);

①添加节点到当前节点内部的后面（新创建的节点）

例：

<body>

<div id="div1">韩寒会画画后悔画韩红</div>

</body>

<script type="text/javascript">

//添加元素节点和文本节点

window.onload=function(){

var p1=document.createElement("p");

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

div1.appendChild(p1);

var t1=document.createTextNode("郭德纲");

div1.appendChild(t1);

}

</script>

②移动节点到当前节点内部的后面（已有节点）

例：

<body>

<div id="p1">郭德纲</div>

<div id="div1">于谦</div>

<div id="div2">孙越</div>

</body>

<script type="text/javascript">

window.onload=function(){

var p1=document.getElementById("p1");

var div2=document.getElementById("div2");

div2.appendChild(p1);

}

</script>

2、insertBefore(要添加或移动的节点，参考节点);

①添加节点到当前节点内部的前面（新创建的节点）

②移动节点到当前节点内部的前面（已有节点）

例：

<body>

<div id="div1">

<div id="div11">韩寒</div>

<div id="p1">韩红</div>

</div>

</body>

<script type="text/javascript">

window.onload=function(){

var p1=document.getElementById("p1");

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

var div11=document.getElementById("div11");

//将div里面的节点移动到指定节点的前面

//insertBefore(要移动的节点,参考节点);

div1.insertBefore(p1,div11);

}

</script>

**删除节点：**

1、remove(); 删除当前节点（IE不支持，可使用outerHTML代替）

2、removeChild(node); 删除节点

**复制节点：**

1、cloneNode(true); 深拷贝：除了复制节点，还复制所有子节点及文本节点

2、cloneNode(false); 浅拷贝：只复制节点

**替换节点：**

repalceChild(新节点, 被替换的节点);

例：

<body>

    <div id="div1">

        韩寒会画画后悔画韩红

        <p id="p1">红红火火恍恍惚惚</p>

    </div>

</body>

<script>

    var div1=document.querySelector("#div1");

    var p1=document.querySelector("#p1");

    var s1=document.createElement("span");

    s1.innerHTML="郭德纲";

    div1.replaceChild(s1,p1);

</script>

**判断节点是否有子节点：**

hasChildNodes(); 判断是否有子节点

**是否包含某节点：**

contains(); 判断是否包含某节点

\*\*\*\*\*

**Document对象访问相关节点：**

**更改为父节点：**

①parentNode 更改对象为当前节点的父节点

例：

<body>

<div id="div1">div1</div>

<div id="div2">div2

<span>span1</span>

<p>p1</p>

</div>

<div id="div3">div3</div>

<input type="button" id="btn" value="将div1追加到div2内部的后面" />

</body>

<script type="text/javascript">

window.onload=function(){

var p1=document.getElementsByTagName("p")[0];

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

var btn=document.getElementById("btn");

btn.onclick=function(){

//p1.parentNode:将当前对象改变为它的父对象div2

p1.parentNode.appendChild(div1);

}

}

</script>

②parentElement 更改对象为当前节点的父元素节点

**通过父节点更改为子节点：**

①firstChild 更改为当前对象的第一个子节点（前面不能有空格,否则将获得文本节点)

说明：对于firstChild来说，IE6/7/8中忽略空文本。

例：

<body>

<div id="div1">div1</div>

<div id="div2"><p>p1</p> <!--注意：前面不能有空格,否则将获得文本节点-->

<p>p2</p>

</div>

<div id="div3">div3</div>

<input type="button" id="btn" value="将div1追加到div2第一个儿子(p1)内部的后面" />

</body>

<script type="text/javascript">

window.onload=function(){

var p1=document.getElementsByTagName("p")[0];

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

var btn=document.getElementById("btn");

btn.onclick=function(){

//div2.firstChild:将div2更改为它的第一个儿子p1

div2.firstChild.appendChild(div1);

}

}

</script>

②firstElementChild IE6/7/8不支持，在其他浏览器中更改为当前对象的第一个子元素节点

③lastChild 更改为当前对象的最后一个子节点（后面不能有空格,否则将获得文本节点)

**以集合方式更改为子节点（集合方式）：**

①childNodes 更改为子节点的集合（下标从0开始）

②children 更改为子元素的集合（下标从0开始）

说明：children属性与 childNodes 属性的差别：

childNodes 属性返回所有的节点，包括文本节点、注释节点；

children 属性只返回元素节点；

例：

<body>

<div id="div1">div1</div>

<div id="div2">

<p>p1</p>

<p>p2</p>

<p>p3</p>

</div>

<div id="div3">div3</div>

<input type="button" id="btn" value="将div1追加到div2第二个儿子(p2)内部的后面" />

</body>

<script type="text/javascript">

window.onload=function(){

var p1=document.getElementsByTagName("p")[0]; //此语句没用

var div1=document.getElementById("div1"); //此语句也没用

btn.onclick=function(){

div2.children[1].appendChild(div1); //下标从0开始

}

}

</script>

**更改为兄弟节点：**

①nextSibling 更改为下一个兄弟节点

说明1：对于nextSibling来说，IE6/7/8中忽略空文本。

说明2：但是，如果当前元素是input，那么IE6/7/8的nextSibling属性不会忽略空文本，这个单独记一下。

例1：

<body>

    <div id="div1">div1

        <p id="p1">p1</p>

        <p id="p2">p2</p>

    </div>

    <div id="div2">div2</div>

</body>

<script>

    if(window.addEventListener){    //IE9+、Chrome等

        console.log(p1.nextSibling.nodeName);   //#test

        console.log(p1.nextSibling.nextSibling.nodeName);  //p

    }else{      //IE6/7/8

        console.log(p1.nextSibling.nodeName);   //p

    }

</script>

例2：当前为input控件

<body>

    <input type="text" id="t1">111

    <input type="text" id="t2">222

</body>

<script>

    if(window.addEventListener){    //IE9+、Chrome等

        console.log(t1.nextSibling.nodeName);   //#text

        console.log(t1.nextSibling.nextSibling.nodeName); //INPUT

    }else{      //IE6/7/8

        console.log(t1.nextSibling.nodeName);   //#text

        console.log(t1.nextSibling.nextSibling.nodeName); //INPUT

    }

</script>

②nextElementSibling IE6/7/8不支持，其他浏览器更改为下一个兄弟元素节点

③previousSibling 更改为上一个兄弟节点

例：

<body>

<div id="div1">div1</div>

<div id="div2"><p>p1</p><p>p2</p></div> <!--不能有空格-->

<div id="div3">div3</div>

<input type="button" id="btn" value="将div1追加到p2的上一个兄弟(p1)内部的后面" />

</body>

<script type="text/javascript">

window.onload=function(){

var p2=document.getElementsByTagName("p")[1];

btn.onclick=function(){

//p2.previousSibling:将p2改为其上一个兄弟，即p1

p2.previousSibling.appendChild(div1);

}

}

</script>

# 09.12（续DOM对象）

属性分类：

1.内置属性——HTML自带的属性

获取内置属性：

DOM对象.属性

DOM对象.getAttribute("属性名");

设置内置属性的值：

DOM对象.属性=值

DOM对象.setAttribute("属性名","值");

删除内置属性：

DOM对象.属性=""；

DOM对象.removeAttribute("属性名");

说明：HTML标记的属性也是DOM对象的属性

2.用户自定义属性

HTML5规范，用户自定义属性必须以“data-”开头

注意：用户自定义属性的获取、设置、删除只有一种方式！

获取自定义属性：

DOM对象.getAttribute("属性名");

设置自定义属性的值：

DOM对象.setAttribute("属性名",值);

删除自定义属性：

DOM对象.removeAttribute("属性名");

--------------------------------------------------------

DOM对象的CSS样式处理：

HTML属性怎么写，JS就怎样写（操作的是HTMl内置属性）

1、DOM对象.style.样式

说明：只能获取或设置行内样式

一次只能设置一种样式

CSS写法改成camel(驼峰)命名法

例：

<body>

<p>红红火火恍恍惚惚</p>

<input type="button" id="btn1" value="添加行内样式-背景粉色" />

<input type="button" id="btn2" value="添加行内样式-文字变大" />

</body>

<script type="text/javascript">

window.onload=function(){

var p1=document.getElementsByTagName("p")[0];

var btn1=document.getElementById("btn1");

var btn2=document.getElementById("btn2");

btn1.onclick=function(){

p1.style.backgroundColor="pink";

}

btn2.onclick=function(){

p1.style.fontSize="30px"; //必须加单位

}

}

</script>

2、DOM对象.style.cssText="css写法"

说明：一次可以设置/获取多个CSS行内样式，且使用CSS样式写法，符合国际命名规范。

例：

<body>

<p>韩寒会画画后悔画韩红</p>

<input type="button" id="btn1" value="一次性添加行内样式" />

</body>

<script type="text/javascript">

window.onload=function(){

var p1=document.getElementsByTagName("p")[0];

var btn1=document.getElementById("btn1");

btn1.onclick=function(){

//注意和.style属性写法不一样，完全是CSS写法

p1.style.cssText="background-color: pink;font-size: 20px;";

}

}

</script>

3、DOM对象.className="属性名"; //获取或设置类选择器

**---------------------------------------------------------------------------------**

注：JavaScript CSS Style属性对照表

| **盒子标签和属性对照** | |
| --- | --- |
| **CSS语法 (不区分大小写)** | **JavaScript语法 (区分大小写)** |
| border | border |
| border-bottom | borderBottom |
| border-bottom-color | borderBottomColor |
| border-bottom-style | borderBottomStyle |
| border-bottom-width | borderBottomWidth |
| border-color | borderColor |
| border-left | borderLeft |
| border-left-color | borderLeftColor |
| border-left-style | borderLeftStyle |
| border-left-width | borderLeftWidth |
| border-right | borderRight |
| border-right-color | borderRightColor |
| border-right-style | borderRightStyle |
| border-right-width | borderRightWidth |
| border-style | borderStyle |
| border-top | borderTop |
| border-top-color | borderTopColor |
| border-top-style | borderTopStyle |
| border-top-width | borderTopWidth |
| border-width | borderWidth |
| clear | clear |
| float | floatStyle |
| margin | margin |
| margin-bottom | marginBottom |
| margin-left | marginLeft |
| margin-right | marginRight |
| margin-top | marginTop |
| padding | padding |
| padding-bottom | paddingBottom |
| padding-left | paddingLeft |
| padding-right | paddingRight |
| padding-top | paddingTop |
| **颜色和背景标签和属性对照** | |
| **CSS语法 (不区分大小写)** | **JavaScript语法 (区分大小写)** |
| background | background |
| background-attachment | backgroundAttachment |
| background-color | backgroundColor |
| background-image | backgroundImage |
| background-position | backgroundPosition |
| background-repeat | backgroundRepeat |
| color | color |
| **样式标签和属性对照** | |
| **CSS语法 (不区分大小写)** | **JavaScript语法 (区分大小写)** |
| display | display |
| list-style-type | listStyleType |
| list-style-image | listStyleImage |
| list-style-position | listStylePosition |
| list-style | listStyle |
| white-space | whiteSpace |
| **文字样式标签和属性对照** | |
| **CSS语法 (不区分大小写)** | **JavaScript语法 (区分大小写)** |
| font | font |
| font-family | fontFamily |
| font-size | fontSize |
| font-style | fontStyle |
| font-variant | fontVariant |
| font-weight | fontWeight |
| **文本标签和属性对照** | |
| **CSS语法 (不区分大小写)** | **JavaScript语法 (区分大小写)** |
| letter-spacing | letterSpacing |
| line-break | lineBreak |
| line-height | lineHeight |
| text-align | textAlign |
| text-decoration | textDecoration |
| text-indent | textIndent |
| text-justify | textJustify |
| text-transform | textTransform |
| vertical-align | verticalAlign |

**文本节点处理：**

兼容性问题：

IE6/7/8支持innerText属性，不支持textContent

firefox不支持innerText属性，支持textContent

chrome支持innerText属性和textConent

兼容写法：对象名.textContent=对象名.innerText=“字符串”

innerText属性：用文本替换当前节点的子节点及文本节点

innerHTML属性：替换节点所有子节点及文本节点

例：

<body>

<p id="p1">p1<span>span1</span></p>

<input type="button" id="btn" value="替换p节点所有子节点及文本节点" />

</body>

<script>

window.onload=function(){

var p1=document.getElementById("p1");

var btn=document.getElementById("btn");

btn.onclick=function(){

p1.innerHTML="<i>韩寒</i>"; //*韩寒*

console.log(p1.innerHTML); //<i>韩寒</i>

console.log(p1.innerText); //韩寒

}

}

</script>

outerText属性：firefox不支持

outerHTML属性：替换当前节点

**文档碎片：**

createDocumentFragment(); 创建文档碎片

①引入：向body中添加节点—逐次添加效率低（应尽量减少DOM操作）

例：

<body>

<input type="button" name="btn1" id="btn1" value="不使用文档碎片逐次添加图片-效率低" />

</body>

<script type="text/javascript">

window.onload=function(){

var btn1=document.getElementById("btn1");

btn1.onclick=function(){

for(var i=1;i<=3;i++){

var img=document.createElement("img");

img.src="images/qsmy"+i+".jpg";

document.body.appendChild(img); //逐次添加节点效率低，应该尽量减少DOM操作

}

}

}

</script>

②解决：添加到div（或其他节点中），之后在统一添加到body中

例：

<body>

<div id="div1">div1</div>

<input type="button" id="btn1" value="不使用文档碎片添加多张图片"/>

</body>

<script type="text/javascript">

window.onload=function(){

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

var btn1=document.getElementById("btn1");

btn1.onclick=function(){

for(var i=1;i<=3;i++){

var img=document.createElement("img");

img.src="images/qsmy"+i+".jpg";

div1.appendChild(img); //内存操作，不会刷新DOM

}

document.body.appendChild(div1); //刷新DOM1次

}

}

</script>

③问题：不想增加额外的节点，直接向body(或指定节点中添加)——使用文档碎片。

例：

<body>

<input type="button" id="btn1" value="不使用文档碎片添加多张图片"/>

</body>

<script type="text/javascript">

window.onload=function(){

var frag=document.createDocumentFragment(); //创建一个文档碎片

var btn1=document.getElementById("btn1");

btn1.onclick=function(){

for(var i=1;i<=3;i++){

var img=document.createElement("img");

img.src="images/qsmy"+i+".jpg";

frag.appendChild(img);

}

document.body.appendChild(frag); //刷新DOM

}

}

</script>

说明：什么情况下，HTML标记才会刷新或重绘？

答：将创建的DOM对象（内存中），添加到现有HTML结构中（已经显示页面），才会引发重绘。

----------------------

DOM模型中分为5类节点：

1.整篇HTML文档——文档节点

2.每个标记——元素节点

3.标记的每个属性——属性节点

4.标记中的文本——文本节点

5.文档中的注释——注释节点

说明：

1.每个节点都是一个DOM对象→{}

2.文本节点不是字符串对象，而是DOM对象

-------------------------------------------------------------

tab菜单难点：如何知道点击了某个按钮，对应的div显示→共享属性

获得DOM A——操作A

获得DOM B——操作B

获得DOM C——操作C

----------------------------------

获得DOM A——操作A，切换到B操作B，切换到C操作C

<div id="div1">

<input type="button">

<div></div>

<input type="button">

<div></div>

<input type="button">

<div></div>

</div>

--------------

原生JS提供文档碎片对象，就是为了减少DOM刷新次数。

讲解顺序：

1.每次向div1中添加一个p标记，一共添加10个p标记，刷新DOM 10次。

2.先创建一个空div,循环添加p标记，一次添加一个，添加10次，这些操作全都在内存中完成。最后将新建的div添加到div1中，这样只刷新一次DOM。

3.希望有个DOM对象，能够减少DOM刷新次数，这个对象挂载到现有DOM不渲染——文档碎片对象。

文档碎片对象主要作用：减少DOM刷新次数。

-----------------------------------------------------------------

**DOM操纵表格：**

表格标记及属性回顾

<table> 定义表格

<caption> 定义表格标题

<th> 定义表格的表头

<tr> 定义表格的行

<td> 定义表格单元格

<thead> 定义表格的页眉

<tbody> 定义表格的主体

<tfoot> 定义表格的页脚

表格标题：

createCaption() 创建表格标题（对应<caption>）

deleteCaption() 删除表格标题

表格表头：

createTHead() 创建表格表头（对应标记<thead>）

deleteTHead() 删除表格表头

表格主体：

createTBody()方法必须创建一个新的tbody元素、将其插入到table元素最后一个tbody元素之后或者如果没有这样的元素则插入到table元素的结尾，并返回该元素（对应<tbody>标记）（ie6/7/8不支持）

表格行：

insertRow() 创建表格的行（对应<tr>）

deleteRow() 删除表格的行

表格单元格（列）：

insertCell() 插入单元格（对应<td>）

deleteCell() 删除单元格

例1：DOM操纵表格

window.onload=function(){

var tab=document.createElement("table"); //创建一个table节点

tab.border="1"; //HTML内置属性：DOM对象.属性=值

var the=tab.createTHead(); //在table节点中创建一个表头节点

// var tby=tab.createTBody(); //在table节点中创建一个表体节点 IE6/7/8不支持

var tbf=tab.createTFoot(); //在table节点中创建一个表底节点

var tro=tab.insertRow(); //在表格节点中创建一个行节点

for(var i=1;i<=10;i++){

var tce=tro.insertCell();//在当前行插入一个列（单元格）

tce.appendChild(document.createTextNode(i));

}

document.body.appendChild(tab); //将表格追加到body中

}

例2：传统节点方式操纵表格

window.onload=function(){

var tab=document.createElement("table"); //创建一个table节点

tab.border="1"; //HTML内置属性：DOM对象.属性=值

var tah=document.createElement("thead"); //创建一个thead节点

var tay=document.createElement("tbody"); //创建一个tbody节点

var taf=document.createElement("tfoot"); //创建一个tfoot节点

tab.appendChild(tah);

tab.appendChild(tay);

tab.appendChild(taf);

var tce=tay.insertRow(); //在当前tbody中创建一个tr节点

for(var i=1;i<=10;i++){

var td=tce.insertCell();

td.appendChild(document.createTextNode(i));

}

document.body.appendChild(tab);

}

例3：实现99乘法表

window.onload=function(){

var tab=document.createElement("table"); //创建一个table节点

tab.border="1"; //HTML内置属性：DOM对象.属性=值

var tah=document.createElement("thead"); //创建一个thead节点

var tay=document.createElement("tbody"); //创建一个tbody节点

var taf=document.createElement("tfoot"); //创建一个tfoot节点

tab.appendChild(tah);

tab.appendChild(tay);

tab.appendChild(taf);

var tce=tay.insertRow(); //在当前tbody中创建一个tr节点

//第一行第一列

tce.insertCell().appendChild(document.createTextNode("\*"));

//第一行其他9列

for(var i=1;i<10;i++){

var td=tce.insertCell();

td.appendChild(document.createTextNode(i));

}

//乘法表主体部分

for(var i=1;i<=9;i++){

//每行

var tce1=tay.insertRow();

//每行第1列

tce1.insertCell().appendChild(document.createTextNode(i));

for(j=1;j<10;j++){

//每行其他9列

tce1.insertCell().appendChild(document.createTextNode(i+"\*"+j+"="+i\*j));

}

}

document.body.appendChild(tab); //将表格追加到body中

}

**style.left和offsetLeft：**

补充：

offsetParent：是指自己祖先元素中，离自己最近的已经定位的元素，如果自己的祖先元素中，没有任何盒子进行了定位，那么offsetParent对象就是body。

offsetLeft：一个元素的offsetLeft值，就是这个元素左边框外，到自己的offsetParent对象的左边框内的距离。

DOM对象offsetLeft和style.left的区别：

1.offsetLeft是只读的

style.left是可读写的（改变位置必然使用它）

2.offsetLeft值是一个整数20

style.left是一个字符串"20px"

3.offsetLeft可以获得：

行内定位样式值

内嵌定位样式值

独立样式表样式值

style.left只能设置或获得行内定位样式。

例1：

<style type="text/css">

body{

margin: 10px;

}

#father1{

position: relative;

left:40px;

background: #ff0;

padding: 20px;

}

#son1{

background: pink;

}

#son2{

background:orange;

}

#father2{

background: #f00;

}

</style>

</head>

<body>

<div id="father1">father1

<div id="son1">儿子1</div>

<div id="son2">儿子2</div>

</div>

<div id="father2">

father2

</div>

</body>

<script type="text/javascript">

window.onload=function(){

function getId(\_id){

var oId=document.getElementById(\_id);

return oId;

}

var father1=getId("father1");

var son1=getId("son1");

var son2=getId("son2");

console.log("style.left的值是：100px；offsetLeft是一个数字：100");

//father1如果没有设置left样式或只是在CSS样式表中设置，没有用行内样式设置style.left，则style.left是空值

console.log("father1.style.left="+father1.style.left); //

//father1如果没有设置left样式则为0，如果在CSS样式或行内样式设置,则offsetLeft=margin+left实际值

console.log("father1.offsetLeft="+father1.offsetLeft); //50

}

</script>

例2：

……

<script type="text/javascript">

window.onload=function(){

function getId(\_id){

var oId=document.getElementById(\_id);

return oId;

}

var father1=getId("father1");

var son1=getId("son1");

var son2=getId("son2");

console.log("style.left的值是：100px；offsetLeft是一个数字：100");

//father1如果没有设置left样式或只是在CSS样式表中设置，没有用行内样式设置style.left，则style.left是空值

console.log("son1.style.left="+son1.style.left); //

//son1如果没有设置left样式则为0，如果在CSS样式或行内样式设置,则offsetLeft为相对于父元素的left值实际值

console.log("son1.offsetLeft="+son1.offsetLeft); //20

}

</script>

例3：

<style type="text/css">

div{

width: 50px;

height: 50px;

background: #0099FF;

position:absolute; /\*注意:这里不能少position,否则无法设置left\*/

}

</style>

</head>

<body>

<div id="div1">哈哈哈</div>

</body>

<script type="text/javascript">

window.onload=function(){

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

div1.onclick=function a1(){

setInterval(function(){

div1.style.left=div1.offsetLeft+1+"px";

},100);

};

}

</script>

# 09.14

事件：窗体、对象、鼠标、键盘的动作称为事件。

事件驱动过程：

首先，在这个对象上**绑定**这个事件；

其次，又在这个对象上，**发生**了这个事件；

最后，系统（js解释器）**自动调用**处理函数进行响应。

事件绑定方式（主要有三种）：

1、行内绑定(不建议)：不利于维护，无法实现html标记和js动作分离（angular react vue使用）

<标记名 事件名="处理函数();"></标记名>

例：

<body>

<p onclick="handlleClick1();handlleClick2()">红红火火恍恍惚惚</p>

</body>

<script type="text/javascript">

function handlleClick1(){

alert("你触发了click1事件");

}

function handlleClick2(){

alert("你触发了click2事件");

}

</script>

2、DOM对象.属性名

DOM对象.属性名=function(){

//处理函数

}

说明1：因为所有的事件都是DOM对象的属性，只不过这些事件属性的值默认是null，必须是函数。

说明2：同一个DOM对象绑定同一个事件多次，后面的会覆盖前面的。

3、事件监听器方法绑定事件：

IE9+：

DOM对象.addEventListener("事件名",函数名,捕获过程true/冒泡过程false);

IE6/7/8：

DOM对象.attachEvent("on事件名",函数名);

IE9/10：两个都支持

例1：

//null undefined "" '' NaN false 0

if(window.addEventListener){

//IE9+

alert("此浏览器支持addEventListener方法");

}else{

//IE6/7/8

alert("此浏览器支持attachEvent方法");

}

例2：

<body>

<p id="p1">哈哈哈</p>

</body>

<script type="text/javascript">

if(window.addEventListener){

//IE9+

window.addEventListener("load",function(){

//相当于window.onload=function(){}

var p1=document.getElementById("p1");

p1.addEventListener("click",function(){

alert("你触发了click11事件~");

});

p1.addEventListener("click",function(){

alert("你触发了click12事件~");

});

});

}else{

//IE6/7/8

window.attachEvent("onload",function(){

var p1=document.getElementById("p1");

p1.attachEvent("onclick",function(){

alert("你触发了click22事件~");

});

//交换了顺序，因为先绑定的后触发

p1.attachEvent("onclick",function(){

alert("你触发了click21事件~");

});

});

}

</script>

-----------------------------

事件监听器绑定事件，为什么需要封装成函数?

1.写兼容

2.代码太多

3.方便使用

例1：封装函数

//param对象:{elem:DOM对象,eventType:"click",func:函数地址}

function addEvent(param){

try{

if(param.elem!=null&&typeof param.elem=="object"){

if(window.addEventListener){

//IE9+

param.elem.addEventListener(param.eventType,param.func);

}else{

//IE6/7/8

param.elem.attachEvent("on"+param.eventType,param.func);

}

}else{

throw new Error("不是对象或对象为空！");

}

}catch(e){

alert(e.message);

}

}

例2：用封装好的函数实现事件绑定

<body>

<p id="p1">韩寒会画画后悔画韩红</p>

</body>

<script type="text/javascript" src="01-09addEvent.js"></script>

<script type="text/javascript">

addEvent({

//相当于window.onload=function(){}

elem:window,

eventType:"load",

func:function(){

var p1=document.getElementById("p1");

addEvent({

elem:p1, //注意：这里是“，”

eventType:"click",

func:function(){

alert("你触发了click事件！");

}

});

}

});

</script>

补充：焦点轮播图

<style type="text/css">

#focusing{

border: 1px solid #090;

width: 600px;

position: relative; /\*子绝父相\*/

left: 50%;

margin-left: -300px;

}

#imgs img{

vertical-align: top; /\*解决img3像素bug\*/

border: none;

}

#focus{

position: absolute;

top: 350px;

left: 230px;

text-align: center;

}

#focus span{

display: inline-block;

\_overflow:hidden; /\*仅IE6支持其他浏览器忽略\*/

width: 16px;

height: 16px;

background: url(../images/dot1.png);

margin-right: 10px;

}

</style>

</head>

<body>

<div id="focusing">

<div id="imgs">

<a href="http://www.baidu.com">

<img src="../images/1.jpg"/>

</a>

</div>

<div id="focus">

<span class="s1" style="background: url(../images/dot2.png);"></span>

<span class="s2"></span>

<span class="s3"></span>

<span class="s4"></span>

<span class="s5"></span>

</div>

</div>

</body>

<script type="text/javascript">

window.onload=function(){

var M=function(id){

return document.getElementById(id);

}

var focusing=M("focusing");

var imgs=M("imgs");

var focus=M("focus");

var s1=focus.getElementsByTagName("span");

var currentIndex; //点击焦点后的当前索引值，默认是undefined

//实现点击切换焦点及相应图片

var h1=["http://www.baidu.com","http://www.tencent.com","http://www.alibaba.com","http://www.163.com","http://www.mi.com"];

for(var i=0;i<s1.length;i++){

s1[i].m=i; //s1[0].m=0;s1[1].m=1;s1[2].m=2;s1[3].m=3;s1[4].m=4

s1[i].onclick=function(){

for(var j=0;j<s1.length;j++){ //焦点初始化

s1[j].style.background="url(../images/dot1.png)";

}

this.style.background="url(../images/dot2.png)"; //设置当前焦点

currentIndex=this.m;

if(-[1,]){ //IE6/7/8不支持firstElementChild

imgs.firstElementChild.firstElementChild.src="../images/"+(this.m+1)+".jpg";

imgs.firstElementChild.href=h1[this.m];

}else{ //对于firstChild来说，IE6/7/8中忽略空文本。

imgs.firstChild.firstChild.src="../images/"+(this.m+1)+".jpg";

imgs.firstChild.href=h1[this.m];

}

}

}

//点击焦点后，切换到当前焦点，并继续轮播

var k=1;

setInterval(function(){

for(var j=0;j<s1.length;j++){ //焦点初始化

s1[j].style.background="url(../images/dot1.png)";

}

if(currentIndex!=undefined){ //判断是否有焦点被点击

k=currentIndex;

currentIndex=undefined;

}

s1[k].style.background="url(../images/dot2.png)";

if(-[1,]){ //IE6/7/8不支持firstElementChild

imgs.firstElementChild.firstElementChild.src="../images/"+(k+1)+".jpg";

imgs.firstElementChild.href=h1[k];

}else{ //对于firstChild来说，IE6/7/8中忽略空文本。

imgs.firstChild.firstChild.src="../images/"+(k+1)+".jpg";

imgs.firstChild.href=h1[k];

}

if(k>=4){

k=-1;

}

k++;

},3000);

}

</script>

说明：if(!-[1,])

[1,] ——IE认为是2个元素，其它浏览器认为是1个元素

对它进行-取负运算 ，IE出现了NaN,其它浏览器出现了-1

NaN为false ，-1为true

!号取反 ，所以IE为true，其它浏览器为false

---------------------------------------------------------------

HTML相关事件：

onload //在页面或图像加载完成后立即发生

支持该事件的标记：<body>,<frame>,<frameset>,<iframe>,<img>,<link>, <script>

支持该事件的对象：image, layer, window

onresize //在窗口或框架被调整大小时发生

支持该事件的标记：

<a>,<address>,<b>,<big>,<blockquote>,<body>,<button>,<cite>,<code>,<dd>,<dfn>,<div>,<dl>,<dt>,<em>,<fieldset>,<form>,<frame>,<h1>-<h6>,<hr>,<i>,<img>,<input>,<kbd>,<label>,<legend>,<li>,<object>,<ol>,<p>,<pre>,<samp>,<select>,<small>,<span>,<strong>,<sub>,<sup>,<table>,<textarea>,<tt>,<ul>,<var>

支持该事件的对象：window

onscroll //在元素滚动条在滚动时触发

支持该事件的标记：<a>,<address>,<b>,<big>,<blockquote>,<body>,<button>,<cite>,<code>,<dd>,<dfn>,<div>,<dl>,<dt>,<em>,<fieldset>,<form>,<frame>,<h1>-<h6>,<hr>,<i>,<img>,<input>,<kbd>,<label>,<legend>,<li>,<object>,<ol>,<p>,<pre>,<samp>,<select>,<small>,<span>,<strong>,<sub>,<sup>,<table>,<textarea>,<tt>,<ul>,<var>

支持该事件的对象：window

兼容写法：

window.onscroll=function(){

if(window.pageYOffset){

//IE9+

console.log(window.pageYOffset);

}else{

//IE6/7/8

console.log(document.documentElement.scrollTop);

}

}

onscroll实例：

<script type="text/javascript">

var div1=document.createElement("div");

document.body.appendChild(div1);

div1.innerHTML="回顶部";

div1.onclick=function(){

window.scrollTo(0,0);

}

window.onscroll=function(){

//注意：IE6不支持position：fixed

div1.style.cssText="background:pink;height: 50px;width: 50px;cursor: pointer;position: fixed;bottom: 0;right: 0;display: none;";

var x=0;

if(window.pageYOffset){

//IE9+

x=window.pageYOffset;

}else{

//IE6/7/8

x=document.documentElement.scrollTop;

}

if(x>200){

div1.style.display="block";

}else{

div1.style.display="none";

}

}

</script>

键盘鼠标相关事件：

单击事件onclick

双击事件ondblclick

卸载事件onunload

鼠标左键按下onmousedown

鼠标左键抬起onmouseup

获取鼠标onmouseover //在鼠标指针移动到元素上时触发

失去鼠标onmouseout //在鼠标指针移出指定的对象时发生

键盘按下onkeydown

键盘抬起onkeyup

按键盘onkeypress

补充：onkeydown事件，是在按下某个键时就会触发，如果按着不放，则会一直触发；onkeypress事件，也是在按下键时触发，但不同的是，这个键必须是字符键，比如说1,2,3,a,b,c,`,在有字符输入的情况下它才会触发，当你按下非字符键如Shift,F4之类的，它是不会触发的。

表单相关事件：

改变事件onchange

选中事件onselect //在文本框中的文本被选中时发生

获得焦点事件onfocus

失去焦点onblur

重置事件onreset

提交事件onsubmit

# 09.17

学习流程：

1.事件对象的创建

复习：简短或 ||

||：有true就是true,全false才是false

方式1：事件对象创建兼容

var e=e||window.event

补充：这样理解

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

div1.onclick=function(x){ //创建事件对象并给x赋值

var y=x||window.event;

console.log(y);

}

方式2：（一般不用）

if(window.addEventListener){

//IE9+

e...

}else{

//IE6/7/8

window.event...

}

2.事件对象的属性

3.事件对象的方法

## 事件对象

事件对象的创建：

1.标准方式：触发事件的时候，系统自动创建并且赋值

chrome支持e和window.event写法

firefox支持e和window.event写法

IE9/10//11 支持e和window.event写法

2.兼容IE6/7/8方式（IE6/7/8只支持window.event）：

x=e||window.event; //e是函数参数，自动生成的事件对象

补充：在W3C规范中，event对象是随事件处理函数传入的，chrome、FireFox、Opera、Safari、IE9.0及其以上版本都支持这种方式；但是对于IE8.0及其以下版本，event对象必须作为window对象的一个属性。

事件对象的属性：

1.button 鼠标左键、中键、右键

标准方式（chrome/firefox）：

0 规定鼠标左键

1 规定鼠标中键

2 规定鼠标右键

IE6/7/8拥有不同的参数：

1 规定鼠标左键

4 规定鼠标中键

2 规定鼠标右键

例1：

window.onload=function(){

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

div1.onmousedown=function(x){

//兼容IE6/7/8

if(window.event&&!x){

switch(window.event.button){

case 1:alert("你点击了鼠标左键");

break;

case 4:alert("你点击了鼠标中键");

break;

case 2:alert("你点击了鼠标右键");

break;

default:alert("你点击了鼠标的其他键");

}

}

//兼容chrome、Firefox、IE9+

if(window.event&&x){

switch(x.button){

case 0:alert("你点击了鼠标左键");

break;

case 1:alert("你点击了鼠标中键");

break;

case 2:alert("你点击了鼠标右键");

break;

default:alert("你点击了鼠标的其他键");

}

}

}

}

例2：

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

div1.onmousedown=function(x){

var y=x||window.event;

if(window.addEventListener){ //也可以用：if(x){…}else{…}

switch(y.button){

case 0:alert("左键");

break;

case 1:alert("中键");

break;

case 2:alert("右键");

break;

}

}else{

switch(y.button){

case 1:alert("左键");

break;

case 4:alert("中键");

break;

case 2:alert("右键");

break;

}

}

}

2.shiftKey 键盘shift返回true

3.ctrlKey 键盘ctrl返回ture

4.altKey 键盘alt返回true

5.target 返回触发当前事件的对象

说明：有些情况下，我们封装的addEvent函数，在IE6/7/8中，传递this的时候，this会被更改为window。

解决方法：使用e.target/e.srcElement。

例：

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

div1.onmousedown=function(x){

var y=x||window.event;

console.log(this); // <div id="div1"></div>

console.log(x.target); // <div id="div1"></div>

}

6.type 返回当前事件的名称

7.screenX 鼠标指针的屏幕水平坐标

8.screenY 鼠标指针的屏幕垂直坐标

9.clientX 鼠标指针相对于窗口的水平坐标，不包括左右侧边栏和滚动条

10.clientY 鼠标指针相对于窗口的垂直坐标

说明：clientX和clientY所有浏览器都支持

11.offsetX 相对于对象区域的x坐标（掌握）

12.offsetY 相对于对象区域的y坐标（掌握）

**事件对象的方法：**

1、对象.preventDefault(); 标准写法，**阻止默认动作**（事件）

说明：IE6/7/8兼容写法：对象.returnValue=false;

例：

<body>

<a href="http://www.bilibili.com" id="a1">去B站</a>

</body>

<script type="text/javascript">

window.onload=function(){

var a1=document.getElementById("a1");

a1.onclick=function(x){

alert("触发了链接的点击事件，除了响应点击事件外，还会自动跳转到指定链接。");

if(x){

x.preventDefault();

}

if(event){ //因为在IE6/7/8中，event对象是window对象的一个属性， “window”可以省略

event.returnValue=false;

}

}

}

</script>

2、对象.stopPropagation(); 标准写法，阻止冒泡（事件）

说明：IE6/7/8兼容写法：对象.cancelBubble=true;

例：

<body>

<div id="div1">1

<div id="div2">2</div>

</div>

</body>

<script type="text/javascript">

window.onload=function(){

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

var div2=document.getElementById("div2");

div1.style.cssText="background: #0f0;width: 200px;height: 200px;"

div2.style.cssText="background: #00f;width: 100px;height: 100px;";

document.body.onclick=function(){alert("你点击了body");};

div1.onclick=function(x){

y=x||window.event;

alert("你点击了div1");

if(y.stopPropagation){

y.stopPropagation();

}else{

y.cancelBubble=true;

}

}

div2.onclick=function(x){

y=x||window.event;

alert("你点击了div2");

if(y.stopPropagation){

y.stopPropagation();

}else{

y.cancelBubble=true;

}

}

}

</script>

**事件流分类：**

1、冒泡型事件流（由内向外）——常用

例1：冒泡之传统方式

<body>

<div id="div1">1

<div id="div2">2</div>

</div>

</body>

<script type="text/javascript">

window.onload=function(){

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

var div2=document.getElementById("div2");

div1.style.cssText="background: #0f0;width: 200px;height: 200px;"

div2.style.cssText="background: #00f;width: 100px;height: 100px;";

document.body.onclick=function(){alert("你点击了body");};

div1.onclick=function(){alert("你点击了div1");}

div2.onclick=function(){alert("你点击了div2");}

}

</script>

例2：冒泡之监听器方式

<body>

<div id="div1">1

<div id="div2">2</div>

</div>

</body>

<script type="text/javascript">

window.onload=function(){

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

var div2=document.getElementById("div2");

var body=document.body;

div1.style.cssText="background: #0f0;width: 200px;height: 200px;"

div2.style.cssText="background: #00f;width: 100px;height: 100px;";

function div1Fun(){

alert("你点击了div1");

}

function div2Fun(){

alert("你点击了div2");

}

//定义兼容的绑定函数

function bind(elem,event,func){

if(elem.addEventListener){

elem.addEventListener(event,func,false);//这里false或省略都是冒泡

}else{

elem.attachEvent("on"+event,func);

}

}

//绑定事件

bind(body,"click",function(){alert("你点击了body");});

bind(div1,"click",div1Fun);

bind(div2,"click",div2Fun);

}

</script>

2、捕获型事件流（由外向内）——很少使用

说明：IE6/7/8不支持

# 09.18

匀速动画：动画类型少、效率低、卡顿

动画中如何做非匀速运动：

非匀速

不卡顿

flash中的编程脚本语言→actionscript可以实现

脚本标准：ECMASCRIPT

javascript

actionscript

actionscript的运动类（运动框架），可以实现非匀速动画：

tween

tweenLite

tweenMax

移植成JavaScript版本：

tween.js（了解）

tweenLite.js（掌握）

--------------------------------

tween和tweenLite运动框架：能够实现平滑的缓动动画（非匀速运动）

tween没有封装，不好用；tweenLite是将tween重新封装成一个函数，并且将tween改名为easing（可以实现动画队列）。

在jQuery就不叫tween，叫easing。

运动类TweenLite概述：

TweenLite($target , $duration, $vars)参数说明：

$target：Object，执行动作的目标对象

$duration：动画持续时间（单位：秒）

$vars：属性+功能函数，数据格式是Object

基本属性如：alpha, left, top, width, height等

功能函数如：

ease→具体缓动类型

onComplete→动作运行结束时调用

onUpdate→动作正在运行时调用（动画执行过程中周期触发的函数）

delay→动作延迟指定秒数后执行

用法如下：

TweenLite(target,duration,{alpha:50,left:100,top:100,ease:Easing.Expo.easeOut,delay:2,[onUpdate:func, onComplete:func ]});

说明：

①“[]”中的内容代表可选内容

②onUpdate:函数名——动画执行过程中调用的函数

③onComplete:函数名——动画执行结束后调用的函数

④easing类型：

Linear(Linear)：无缓动效果

Quad(Quadratic)：二次方的缓动（t^2）；

Cubic(Cubic)：三次方的缓动（t^3）；

Quart(Quartic)：四次方的缓动（t^4）；

Quint(Quintic)：五次方的缓动（t^5）；

Sine(Sinusoidal)：正弦曲线的缓动（sin(t)）；

Expo(Exponential)：指数曲线的缓动（2^t）；

Circ(Circular)：圆形曲线的缓动（sqrt(1-t^2)）；

Elastic(Elastic)：指数衰减的正弦曲线缓动；

Back(Back)：超过范围的三次方缓动（(s+1)\*t^3 - s\*t^2）；

Bounce(Bounce)：指数衰减的反弹缓动。

⑤每个效果都分三个缓动方式（方法），分别是：

easeIn：从0开始加速的缓动；

easeOut：减速到0的缓动；

easeInOut：前半段从0开始加速，后半段减速到0的缓动。

其中Linear是无缓动效果，没有以上效果。

动画队列：一个动画结束，另一个动画才开始

如何实现动画队列：

TweenLite(...{onComplete:function(){

TweenLite(...{});

}});

例：

<style type="text/css">

#div1{

background: pink;

width: 50px;

height: 50px;

position: absolute; //注意别忘了position

}

</style>

</head>

<body>

<div id="div1"></div>

</body>

//别忘了外部导入js文件

<script type="text/javascript" src="01-11TweenLite.js">

</script>

<script type="text/javascript">

window.onload=function(){

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

var i=1;

div1.onclick=function(){

TweenLite(div1,3,{alpha:50,top:300,ease:Easing.Bounce.easeOut,delay:2,

onUpdate:function(){console.log("调用了"+i+++"次")},

onComplete:function(){console.log("动画结束了~");

//TweenLite() 动画队列,写在onComplete内部

}});

}

}

</script>

-------------------------------

cookie：小糕点；饼干

cookie是网站服务器向客户端浏览器存储的信息，例如IE浏览器中的cookie是网站向客户端主机C:\Users\11816\AppData\Local\Microsoft\Windows\INetCache下创建的一个文本文件，里面存储网站相关信息。用户再次访问此网站的时候服务器会读取这个cookie文件，来获取相关信息。

当我们提到cookie时，要说是哪个网站的cookie，cookie根据域名来进行区分。

IE——cookie以文件形式，存储到临时文件目录

chrome——cookie存储到浏览器中

\*\*\*\*\*

各个浏览器cookie个数限制：

①Microsoft指出InternetExplorer8增加cookie限制为每个域名50个，但IE7似乎也允许每个域名50个cookie，IE6为20个。

②Firefox每个域名cookie限制为50个。

③Opera每个域名cookie限制为30个。

④Safari/WebKit貌似没有cookie限制。但是如果cookie过多，会使header大小超过服务器的处理的限制，会导致错误发生。

注意：每个cookie文件大小不要超过4KB。

\*\*\*\*\*

判断浏览器是否支持cookie：

//IE放到远程服务器才能看到效果

if(navigator.cookieEnabled){

document.write("你的浏览器支持cookie。");

}else{

document.write("你的浏览器不支持cookie。");

}

------------------------------

网站中cookie的应用：

1.实现免登录

说明：选中10天免登录→点击登录→服务器自动将用户名和加密后的密码存储到浏览器的cookie中。下次进入mail.163.com时，浏览器在发送请求的时候，会自动将cookie放到request headers中一起发送给mail.163.com，服务器得到用户名和密码，跳转进入邮箱。

2.用户名提示

3.判断用户指定操作

4.动态语言的session是用来控制权限的，它是基于cookie实现的

---------------------------------------------------------------------------------------

cookie原理：网站服务器将cookie存储到浏览器并且指定过期时间（如果不指定过期时间，那么浏览器关闭cookie即可销毁），下次用户访问这个网站，浏览器会自动将这个网址的所有cookie自动发送给远程服务器，服务器得到后便进行处理。

cookie的特点：

1.体积小

2.可以长期存储

3.自动发送给网站服务器

说明：PHP、java、python、javascript都能设置和获得cookie。

-----------------------------------------------------------------------------------

JavaScript如何创建和获取cookie？

1.获取cookie

document.cookie //文档对象的一个属性cookie存储数据

2.设置cookie

难点：cookie存储数据的格式比较特殊

document.cookie="变量=值; expires=GMT时间格式"

说明：值需要encodeURI编码

一次只能存储一个cookie变量

cookie本身不会加密数据

---------------------------------------------

**注意1：所有cookie实例必须放到Web服务器上才能运行（HBuilder自带一个localhost：8020；wamp有localhost）。**

**注意2: cookie之间用“； ”分隔，注意不要丢了分号后的空格！**

例1：创建cookie变量（不封装成函数）

var myDate=new Date();

var t=myDate.getTime();

//获取10天以后的时间

var t10=t+10\*24\*3600\*1000;

myDate.setTime(t10);

document.cookie="x=999;expires="+myDate.toUTCString();

//注意cookie中包含中文的不同

document.cookie="y="+encodeURI("哈哈")+";expires="+myDate.toUTCString();

例2：将创建cookie变量的过程封装成函数

function setCookie(name,value,day){

var myDate=new Date();

var t=myDate.getTime();

//获取day天以后的时间

var tDay=t+day\*24\*3600\*1000;

myDate.setTime(tDay);

document.cookie=name+"="+encodeURI(value)+";expires="+myDate.toUTCString();

}

例3：获取指定cookie变量的值

function getCookie(name){

var data=document.cookie.split("; ");

//["x=999", " y=%E5%93%88%E5%93%88", " z=555"]

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

var data1=data[i].split("=");

//第一次循环得到：["x", "999"]

if(data1[0]==name){

return decodeURI(data1[1]); //解码

}

}

}

补充：获取cookie时如果得到了undefined，那可能是变量前边的空格导致的。

例4：删除指定cookie

原理：设置cookie变量值为""，设置过期时间为过期

function delCookie(name){

var myDate=new Date();

var t=myDate.getTime();

myDate.setTime(t-1000); //设置过期1秒

document.cookie=name+"=’’"+";expires="+myDate.toUTCString();

}

例5：删除所有cookie变量

function delAllCookie(){

var myDate=new Date();

myDate.setTime(-1000);

var data=document.cookie;

var dataArray=data.split("; ");

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

var varName=dataArray[i].split("=");

document.cookie=varName[0]+"=’’;expires="+myDate.toUTCString();

}

}

# 09.19

1.公司项目组构成：

组长：1人

UI组4-6人，组长1人：设计网站效果图，切图、优化

前端组4-6人，组长1人：网站重构（css、js）、前后端交互、性能优化、网站兼容性等

后台组7-10人，组长1人：接口文档书写、部分业务逻辑、数据库设计及实现

2.从技术角度分为前端和后台

3.从使用者（客户）角度：前台和后台

比如说网站共100个页面，给客户看到30个——前台页面，管理员使用70个——后台页面。

4.PHP环境搭建及工作原理

--------------------------------------

1、服务器：提供服务的软件或程序(php)

2、php环境搭建：

wamp集成环境(IDE环境)：windows+apache+php+mysql

浏览器：负责将请求发送给web服务器和接受服务器返回的html

apache：web服务器，只负责接收客户端请求，响应客户端请求，不负责任何的php脚本的解释

php解释器：将php脚本翻译成html /json，和原来的html一起，返回给apache

mysql数据库：对数据库的管理，对数据库中的表进行添删改查

3、Apache服务器的目录及配置文件：

①Apache2目录：Apache服务器文件目录

②Apache2/conf目录：Apache服务器配置文件目录

③Apache2/conf/http.conf：Apache服务器的主配置文件

网站根目录默认是www

Apache服务器端口号

网站首页配置等等

说明：wamp环境，要修改http.conf配置文件，不能直接修改文件，必须通过控制台进行修改。

④PHP配置文件：php/php.ini

如何通过PHP脚本读取,php.ini配置文件呢？

答：使用php函数→phpinfo();

4、实验：搭建PHP开发环境

Dreamweaver+wamp

通过Dreamweaver搭建PHP开发环境

原理：就是让Dreamweaver

①自动打开浏览器

②自动给当前文件夹前缀 http://localhost/myphp

例如，当前文件是test.php，地址栏就是http://localhost/test.php

拓展：

1.http://localhost找到的是apache的网站根目录www

2.所有的网站页面，不可能都放到www下，所以一般会在www下创建子目录，以存放不同的网站

www/myphp→http://localhost/myphp

------------------------------------------------------

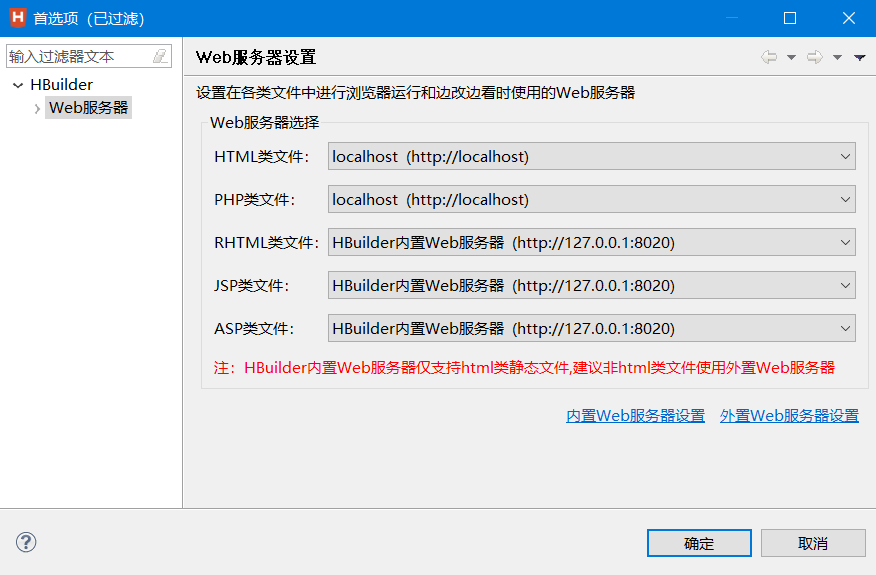
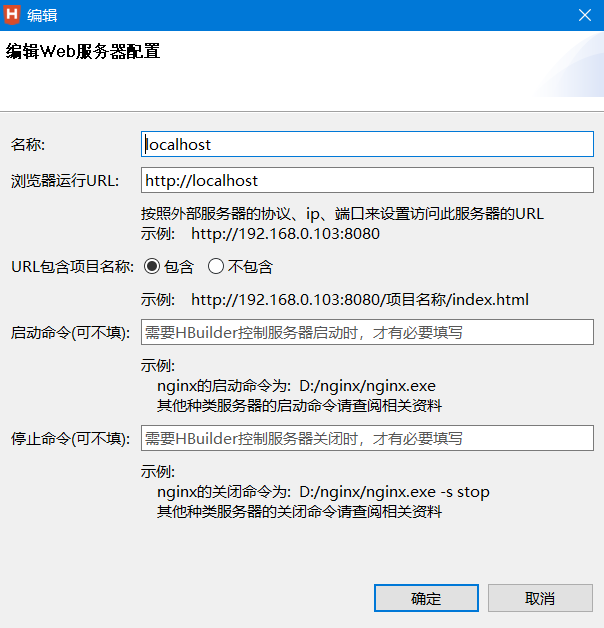
HBuilder搭建PHP开发环境：

注意：HBuilder自带一个web服务器，没有php解释器

解决方法：

①安装好wamp

②在HBuilder中，添加外部服务器，如图：



PHP基本语法：

1.变量：

①普通变量和js命名规则一样，变量以“$”开头

②预定义变量（系统变量）：如$\_GET、$\_POST、$\_SESSION等

例：

<?php

header("content-type:text/html;charset=utf-8"); //设置文档的字符集，否则中文乱码

$name="Tom";

echo $name;

$a1="韩寒会画画后悔画韩红";

echo $a1;

?>

补充1：echo 表达式或值;

//将表达式或值输出到输出缓冲中，相当于document.write()。

补充2：在“<？php”之前不能有任何输出，注释也不行。因为如果在header之前有任何输出，那么这时请求头已经发送了（使用默认请求头），php默认会先输出一些header信息比如cookie之类的，然后输出页面内容，这个时候再调用header，就违反http协议了。

而且，header()必须在任何实际输出之前调用，不管是普通的HTML标签，还是文件或PHP输出的空行、空格。

补充3：把<meta charset="utf-8">写在最前边也行，但这不是php的写法。

2.常用数据类型：

(1)number

(2)string

"" 自动识别里面的变量

'' 纯字符串

(3)boolean：true false

(4)对象类型

①数组类型：

索引数组：下标都是从0开始的数字

var 数组名=array(元素,元素,元素);

关联数组：下标是用户指定的字符串

var 数组名=array("键名"=>键值,"键名"=>键值,"键名"=>键值);

例1：

<?php

//创建一个索引数组

$arr=array(12,15,99);

echo $arr[0].'<br>'; //“.”代表字符串连接符

echo $arr[1].'<br>';

echo $arr[2].'<br>';

print\_r($arr); //打印数组结构，输出：Array ( [0] => 12 [1] => 15 [2] => 99 )

foreach($arr as $value){ //遍历数组

echo $value."<br>";

}

?>

例2：

<?php

header("content-type:text/html;charset=utf-8"); //不要丢分号

//创建一个关联数组：存储数据库的表里面的数据

$arr=array("hebei"=>"石家庄","henan"=>"郑州","shandong"=>"青岛");

echo $arr["hebei"];

print\_r($arr);

?>

②资源类型

读取一个文件

连接数据库资源

3.基本结构：和js一样

# 09.20

**重点：PHP处理表单（同步提交，浏览器直接和后台进行交互，不通过中间件）**

回顾：

<form name="表单名称" action="数据提交的PHP页面地址" method="get/post">

</form>

预备知识：什么是数据？

答：表单中用户填写的或者选择value值对应的字符串/用户自己编写的数据。

**数据编码格式：**

(1)application/x-www-form-urlencoded（表单传输数据的默认编码）

说明：这种编码方式，只能传输字符串数据，无法传输文件。

格式：

变量名=值&变量名=值&变量名=值……

//变量名:表单项的name值，值:表单项的value值

(2)application/json

格式：

'{"属性":值,"属性":值,"data":[]}'

(3)form-data（二进制数据格式）

说明：这种格式可以上传文件和字符串

-----------------------------------------------------

复习：URL统一资源定位符

格式：协议://ip或域名:端口号/uri?变量=值&变量=值

表单使用默认数据编码x-www-form-urlencoded编码：

例：

http://localhost/myphp0816/chuli.php?t1=Trump&s1=tijiao

http://localhost/myphp0816/chuli.php?t1=Obama&pas=123&s1=tijiao

http://localhost/myphp0816/chuli.php?t1=Michael&pas=555&sex=0&s1=tijiao

http://localhost/myphp0816/chuli.php?t1=Jackson&pas=555&sex=0&city=bj&s1=tijiao

http://localhost/myphp0816/chuli.php?t1=Vae&pas=666&sex=0&city=sh&t2=%E9%9F%B3%E4%B9%90%E4%BA%BA&s1=tijiao

http://localhost/myphp0816/chuli.php?t1=Elisabeth&pas=111&sex=1&city=bj&t2=hhhhh&kemu%5B%5D=yw&kemu%5B%5D=yy&s1=tijiao

注：encodeURI("[")："%5B"；encodeURI("]")："%5D"

http://localhost/myphp0816/chuli.php?t1=Jay&pas=666&sex=0&city=sh&t2=%E6%AD%8C%E6%89%8B&kemu%5B%5D=yw&kemu%5B%5D=yy&citys%5B%5D=bj&citys%5B%5D=sh&s1=tijiao

对应的HTML代码：

<form name="f1" action="chuli.php" method="get" enctype="application/x-www-form-urlencoded">

<!--以“变量=值”形式提交数据-->

输入文本框：

<input type="text" name="t1" /><br />

密码框：

<input type="password" name="pas" /><br />

单选按钮：

<input type="radio" name="sex" value="0" />男

<input type="radio" name="sex" value="1" />女<br />

单选下拉菜单：

<select name="city">

<option value="hz">杭州</option>

<option value="bj">北京</option>

<option value="sh">上海</option>

</select><br />

文本域：

<textarea name="t2" rows="5" cols="30"></textarea><br />

<!--以数组形式提交数据-->

复选框：

<input type="checkbox" name="kemu[]" value="yw" />语文

<input type="checkbox" name="kemu[]" value="sx" />数学

<input type="checkbox" name="kemu[]" value="yy" />英语<br />

多选列表：

<select name="citys[]" multiple size="3"> <!--size：可见行数-->

<option value="hz">杭州</option>

<option value="bj">北京</option>

<option value="sh">上海</option>

</select><br />

<input type="submit" name="s1" value="tijiao" />

</form>

**php如何接收客户端传递的数据？**

get→$\_GET["变量名"]

post→$\_POST["变量名"]

文件→$\_FILES

例：

<?php

header("content-type:text/html;charset=utf-8");

//接受以“变量=值”提交的数据

$t1=$\_GET["t1"];

echo '输入文本框的值是：'.$t1."<br>";

$pas=$\_GET["pas"];

echo '输入密码框的值是：'.$pas."<br>";

$sex=$\_GET["sex"];

echo '输入单选按钮的值是：'.$sex."<br>";

$city=$\_GET["city"];

echo '输入单选下拉菜单的值是：'.$city."<br>";

$t2=$\_GET["t2"];

echo '文本域的值是：'.$t2."<br>";

//接受以数组形式提交的数据

$kemu=$\_GET["kemu"];

echo '复选框的值是：';

foreach($kemu as $value){

echo $value."<br>";

}

$citys=$\_GET["citys"];

echo '多选列表的值是：';

foreach($citys as $value){

echo $value."<br>";

}

?>

对应的HTML代码：

<form action="09-20chuli.php" name="f1" method="get">

<!--以“变量=值”形式提交的数据-->

输入文本框：

<input type="text" name="t1" /><br />

密码框：

<input type="password" name="pas" /><br />

单选按钮：

<input type="radio" name="sex" value="0" />男

<input type="radio" name="sex" value="1" />女<br />

单选下拉菜单：

<select name="city">

<option value="bj">北京</option>

<option value="hz">杭州</option>

<option value="sh">上海</option>

</select><br />

文本域：

<textarea name="t2" rows="5" cols="30"></textarea><br />

<!--以数组形式提交的数据-->

复选框：

<input type="checkbox" name="kemu[]" value="yw" />语文

<input type="checkbox" name="kemu[]" value="sx" />数学

<input type="checkbox" name="kemu[]" value="yy" />英语<br />

多选列表：

<select name="citys[]" multiple size="3">

<option value="bj">北京</option>

<option value="hz">杭州</option>

<option value="sh">上海</option>

</select><br />

<input type="submit" value="提交" />

</form>

说明：PHP遍历数组的方法之一

$arr=array("tom","obama","trump");

foreach($arr as $value){

echo $value."<br>";

}

-------------------------------------------------

综合实验：实现PHP版本的登录验证

逻辑：如果用户名和密码都正确，跳转到管理页面；否则报用户名或密码错误。

功能分析：

①登录验证基本功能

②带session，进行权限控制

③带cookie进行，免登录

④带数据库

⑤带验证码

预备知识：动态跳转

js：location.href="url";

php：header("location:url");

实例：

login.html：

<form action="09-20check.php" method="post">

用户名：

<input type="text" name="userName" /><br />

密码：

<input type="password" name="pas" /><br />

<input type="submit" value="登录"/>

</form>

09-20check.php：

<?php

header("content-type:text/html;charset=utf-8");

if($\_SERVER["REQUEST\_METHOD"]=="POST"){ //判断请求提交数据的方式

$userName=$\_POST["userName"];

$pas=$\_POST["pas"];

//访问数据库（还没学）

if($userName=="Obama" and $pas=="123456"){

//php动态跳转

header("location:09-20index.html");

}else{

echo '<script>alert("用户名或密码错误！");history.go(-1);</script>';

}

}else{

header("location:09-20error.php");

}

?>

补充1：

history.back(-1)和history.go(-1)都是返回之前的页面，它们的区别：

history.back(-1); 直接返回当前页的上一页，数据全部刷新，是个新页面

history.go(-1); 也是返回当前页的上一页，不过表单里的数据全部还在

补充2：PHP常见错误反馈及解决方案：

1、PHP错误反馈：Parse error: syntax error, unexpected T\_FUNCTION

错误原因：PHP版本过低

解决方案：将服务器PHP版本升级到5.3或以上即可解决

2、PHP错误反馈：Parse error: syntax error, unexpected T\_VARIABLE

错误原因：字符串拼写错误或句末缺少分号

解决方案：检查字符串拼写和句末结束符

3、PHP错误反馈：Parse error: syntax error, unexpected T\_STRING, expecting T\_FUNCTION

错误原因：关键词function后缺少空格

解决方案：在错误行中对应添加空格

4、PHP错误反馈：Warning: Unknown: failed to open stream: No such file or directory in Unknow

错误原因：存放获取php文件的目录不能有中文

解决方案：将存放获取php文件的目录的中文替换掉

# 09.21

**PHP第13章文件系统内容（部分）：**

文件上传：

文件上传可以通过HTTP协议来实现。要使用文件上传功能，首先要在php.ini配置文件中对上传做一些设置，然后了解预定义变量$\_FILES的值对上传文件做一些限制和判断，最后使用move\_uploaded\_file()函数实现上传。

**①配置php.ini文件**

要想顺利的实现上传功能，首先要在php.ini中开启文件上传，并对其中一些参数做出合理的设置。找到File\_Uploads项，可以看到5个属性值，含义如下：

1.file\_uploads：如果值是on，说明服务器支持文件上传；如果为off，则不支持。

2.upload\_tmp\_dir：上传文件临时目录。在文件被上传成功之前，文件首先被存放到服务器端的临时目录中。如果想要指定位置，可以在这里设置。否则使用系统默认目录即可。

3.upload\_max\_filesize：服务器允许上传的文件的最大值，以MB为单位。系统默认为2MB，用户可以自行设置。

4.max\_execution\_time：PHP中一个指令所能执行的最大时间，单位是秒。

5.memory\_limit：PHP中一个指令所分配的内存空间，单位是MB。

**②预定义变量$\_FILES**

$\_FILES变量存储的是上传文件的相关信息，这些信息对于上传功能有很大的作用。该变量是一个二维数组，保存相关信息。

|  |  |
| --- | --- |
| 元素名 | 说明 |
| $\_FILES[filename][name] | 存储上传文件的文件名如：test.txt、a.jpg等 |
| $\_FILES[filename][size] | 存储了文件大小。单位为字节 |
| $\_FILES[filename][tmp\_name] | 文件上传时，首先在临时目录中被保存成为一个临时文件，该变量为临时文件名 |
| $\_FILES[filename][type] | 上传文件的类型 |
| $\_FILES[filename][error] | 存储了上传文件的结果。如果返回0，说明文件上传成功。 |

**③文件上传函数**

PHP使用move\_uploaded()函数上传文件：

move\_uploaded\_file(上传文件的临时文件名，上传成功后保存的新路径和文件名)； //成功返回true，失败返回false。

---------------------------------------------------------------------

浏览器向服务器上传文件的做法：

1、<input type="file" name="ufile">

注意1：必须使用post。

注意2：需要将表单数据编码从默认的url-encoded编码，改成form-data二进制编码。

2、php服务器上获取上传文件的信息

(1)$\_FILES['ufile']['name']; 获取上传文件名

(2)$\_FILES['ufile']['tmp\_name']; 获取上传文件的临时文件名

(3)$\_FILES['ufile']['size']; 获取上传文件大小(单位：字节)

补充：第一个参数是表单的input name。

3、php如何实现上传？

move\_uploaded\_file(临时文件名，上传后的文件名);

例：

**09-21chuili.php：**

<?php

header("content-type:text/html;charset=utf-8");

$sr=$\_POST["sr"];

echo "输入文本框的值是：".$sr."<br>";

echo "上传的文件名是：".$\_FILES[ufile][name]."<br>";

echo "上传文件的临时文件名是：".$\_FILES[ufile][tmp\_name]."<br>";

echo "上传文件的大小是：".$\_FILES["ufile"]["size"]."(单位：字节)<br>"; //注意：语法松散，带不带""皆可

echo "上传文件的扩展名是：".strstr($\_FILES["ufile"]["name"],".")."<br>";

$filename=$\_FILES[ufile][name]; //上传文件名

$filenamekz=strstr($filename,"."); //文件扩展名

$filenamels=$\_FILES[ufile][tmp\_name]; //临时文件名

$filesize=$\_FILES[ufile][size]; //文件大小（单位：字节）

//解决上传文件重名的问题

$filenameuploaded=substr($filename,0,strlen($filename)-4).rand(1,999).time().$filenamekz;

//解决上传文件大小和格式的问题

if($filesize<=512000){

if(strstr(".jpg.png.gif",$filenamekz)){

move\_uploaded\_file($filenamels,'09-21upload/'.$filenameuploaded); //上传文件并指定路径

echo "上传成功！";

echo "<img src=09-21upload/".$filenameuploaded." width='200'>"; //显示上传的图片

//注1：width属性前边有一个空格↑

//注2：上传的图片名中若含中文，会因乱码导致无法显示

}else{

echo "文件名必须是.jpg.png.gif图片格式！";

}

}else{

echo "文件大小不能超过500KB！";

}

?>

**09-21form.html：**

<form action="09-21chuli.php" method="post" enctype="multipart/form-data">

输入文本框：

<input type="text" name="sr" /><br />

添加文件：

<input type="file" name="ufile" /><br />

<input type="submit" name="submit" value="上传" />

</form>

**说明1：**

strstr(字符串,子串);

php字符串截取函数：搜索子串在字符串中是否存在，如果是，返回该字符串及剩余部分，否则返回false。

**说明2：**

substr(string,start,[length]);

php字符串截取函数：该方法可在指定字符串中抽取从开始下标开始的指定数目的字符。

注意：此函数在php中与JavaScript中的使用有区别。

**说明3：**

strlen()函数返回字符串的长度。

**说明4：**

<?php

echo rand(1,100); //php函数生成随机整数

echo "<br>";

echo time(); //php时间戳，单位：秒

?>

-------------------------------------------------------------

**PHP第11章Cookie与Session内容（部分）：**

1.Session是一个会话对象，用来存取会话中的相关信息。用户访问网站过程中每个用户的私有数据存储在Session对象中。

Session生命周期：一个用户的访问过程叫做一个会话。从用户进入网站开始，直到用户关闭所有浏览器为止即离开当前网站。

特点：每个用户私有——session\_id

跨页面传值

2.开启Session会话，默认情况下PHP禁用Session（出于安全考虑），使用的时候必须开启Session。

3.关于PHP的Session变量说明：默认PHP是禁用Session的。

开启Session的方法：

(1)在页面上使用session\_start()函数；

但是页面关闭之后下次如果启用session还得执行session\_start()。

(2)永久开启session：修改php的配置文件php.ini；

设置session.auto\_start = 1（默认是session.auto\_start = 0）后，重启Apache服务器。

4.获取session\_id的方法

session\_id();

说明：在用户访问网站过程中，session\_id用来表示用户的身份，当用户关闭浏览器即离开网站就会销毁Session对象，同时session\_id也被销毁，下次访问时，会重新分配新的session\_id来标识用户。

5.创建session变量

$\_SESSION['变量名']=值;

6.使用session变量

$\_SESSION['变量名'];

7.注销session的方法

session\_destroy();

例：

**09-21a.php：**

<?php

session\_start(); //开启session

echo session\_id(); //获取session\_id

$\_SESSION["x"]="红红火火恍恍惚惚"; //创建session变量

?>

**09-21b.php：**

<?php

header("content-type:text/html;charset=utf-8");

session\_start();

echo $\_SESSION["x"]; //红红火火恍恍惚惚

?>

注意：需先打开第一个页面，再打开第二个页面。

8. 补充

Cookie是存储在客户端的文件，而Session是存储在服务器端的机制，但客户端需要存储session\_id，存储需要启用Cookie机制（session\_id存储在内存中），如果客户端禁用session\_id，客户端将无法回传session\_id，那么session将失效。

-----------------------------------------------------------------------------------------------------

Cookie是网站向客户端浏览器存储的变量值及相关信息，方便网站实现一些特殊功能。

特点：体积小

自动发送给服务器

可以长期存储

cookie变量整个网站有效

php中的cookie生命周期分为两种：

①如果指定过期时间，那么过期时间后失效

②如果没有指定过期时间，那么浏览器关闭即刻销毁（同session）

1.创建cookie

setcookie("变量名","值",过期时间);

注意：cookie是HTTP头部的组成部分，而头部必须在页面发送其他内容之前，就先发送出去。若在setcookie()函数之前输出一个HTML标记或echo语句，甚至一个空行都会导致程序错误。

2.使用cookie变量

$\_COOKIE['变量名'];

3.删除cookie

setcookie("变量名","值",过期时间-10000);

例：

**09-21c.php：**

<?php

setcookie("y","200",time()+1\*24\*3600); //创建cookie并设置过期时间

?>

**09-21d.php：**

<?php

echo $\_COOKIE["y"];

?>

补充：setcookie()函数的参数说明

|  |  |  |
| --- | --- | --- |
| name | Cookie的变量名 | 可以通过$\_COOKIE["cookiename"]调用变量名为cookiename的Cookie |
| value | Cookie变量的值，该值保存在客户端，不能用来保存敏感数据 | 可以通过S\_COOKIE["values"]获取名称为values的值 |
| expire | Cookie的过期时间 | 默认浏览器关闭销毁，可以手动设置 |
| path | Cookie在服务器端保存的有效路径 | 默认当前文件夹 |

session和cookie的区别：

1、cookie存储在浏览器中

session存储到服务器上

2、session浏览器关闭即可销毁

cookie可以设置过期时间

session和cookie的共同点：

session变量和cookie变量都可以跨页面

**实例：登录验证PLUS**

**09-21login.php：**

<?php

session\_start();

?>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>欢迎登录</title>

</head>

<body>

<?php

//读取cookie，若不为空，则验证用户名和密码

$userName=$\_COOKIE["userName"];

$pas=$\_COOKIE["pas"];

if(!empty($userName) and !empty($pas)){

if($userName=="Obama" and $pas=="123456"){

$\_SESSION["login"]="success";

$\_SESSION["userName"]=$userName;

echo "<script>location.href='09-21index.php';</script>";

}

}

?>

<form action="09-21check.php" name="login" method="post">

用户名：

<input type="text" name="userName" /><br />

密码：

<input type="password" name="pas" /><br />

<input type="checkbox" name="info" value="yes" />是否保存用户信息10天<br />

<input type="submit" name="submit" value="登录" />

</form>

</body>

</html>

**09-21check.php：**

<?php

header("content-type:text/html;charset=utf-8");

session\_start(); //开启session

if($\_SERVER['REQUEST\_METHOD']=="POST"){ //判断请求提交数据的方式

$userName=$\_POST["userName"];

$pas=$\_POST["pas"];

$info=$\_POST["info"];

if($userName=="Obama" and $pas=="123456"){

//存储登录成功的标识

$\_SESSION["login"]="success";

//存储用户名

$\_SESSION["userName"]=$userName;

if($info=="yes"){ //注意：不勾选，$info就不会传值过来

//存储cookie到浏览器

setcookie("userName",$userName,time()+10\*24\*3600);

setcookie("pas",$pas,time()+10\*24\*3600);

}

header("location:09-21index.php");

}else{

echo "<script>alert('用户名或密码错误！');history.go(-1);</script>";

}

}else{

header("location:09-21error.php");

}

?>

**09-21index.php：**

<?php

session\_start(); //开启session

?>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>欢迎进入管理系统</title>

<style type="text/css">

span,div{

font: 20px/1.5em "微软雅黑";

color: #f40;

}

span{

color: lightseagreen;

}

</style>

</head>

<body>

<?php

if($\_SESSION['login']=="success"){

?>

<div>

欢迎<span>

<?php echo $\_SESSION["userName"]; ?>

</span>进入管理系统~

</div>

<a href="09-21loginout.php">注销</a>|

<a href="09-21safeout.php">安全退出</a>

<?php

}else{

echo "<script>location.href='09-21error.php';</script>";

}

?>

</body>

</html>

**09-21error.php：**

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8" />

<title>错误页面</title>

</head>

<body>

不能绕过登录哦~<br />

系统将在3秒后，自动跳转到登录页……

</body>

<script type="text/javascript">

setTimeout(function(){

location.href="09-21login.php";

},3000);

</script>

</html>

**09-21loginout.php：**

<?php

session\_start();

session\_destroy(); //销毁session

header("location:09-21login.php");

?>

**09-21safeout.php：**

<?php

session\_start();

session\_destroy(); //销毁session

setcookie("userName","",time()-999); //删除cookie

setcookie("pas","",time()-999); //删除cookie

header("location:09-21login.php");

?>

注意：

本例的“注销”和“安全退出”存在功能冲突，若不选择“10天免登录”，即可使用“注销”销毁session。

强烈注意！

HBuilder可以在php文件中写html，在html中写php会出现未知错误！

# 09.24

**预备知识1：http**

http协议：无状态协议，客户端和服务器不会一直保持状态(连接)

一个完整的http请求过程：

第一步：建立与服务器的TCP连接

第二步：客户端浏览器向WEB服务器发送请求头信息

第三步：客户端浏览器向WEB服务器发送请求信息（可选）

第四步：WEB服务器准备应答

第五步：WEB服务器发送应答头信息

第六步：WEB服务器向浏览器发送数据

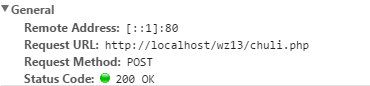
第七步：WEB服务器断开TCP连接

**预备知识2：http请求构成**

http请求的构成：

1.http请求的方法：get/post

2.http请求的URL地址



3.http请求头，包含客户端浏览器环境信息等

4.http请求正文（请求体），请求正文一般是发送给服务器的数据（表单信息等）



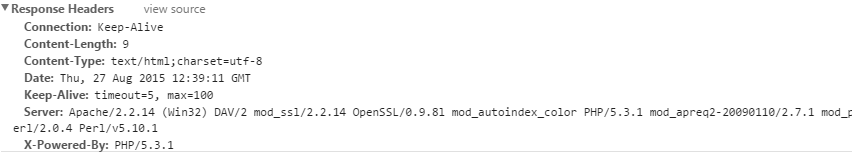
**预备知识3：http响应构成**

http响应的构成：

1.http状态码（文字和数字组成的状态码），用来显示请求成功还是失败



2.响应头：类似于请求头，包含服务器相关环境信息。



3.响应体



Ajax概述：

AJAX：(Asynchronous Javascript And XML)，异步JavaScript和XML。

注：Asynchronous读音：美/[ e'sɪŋkrənəs]

补充：

Ajax请求后台返回数据时，都是简单的字符串或数字，如果后台有大量数据要返回，我们应该怎么办？

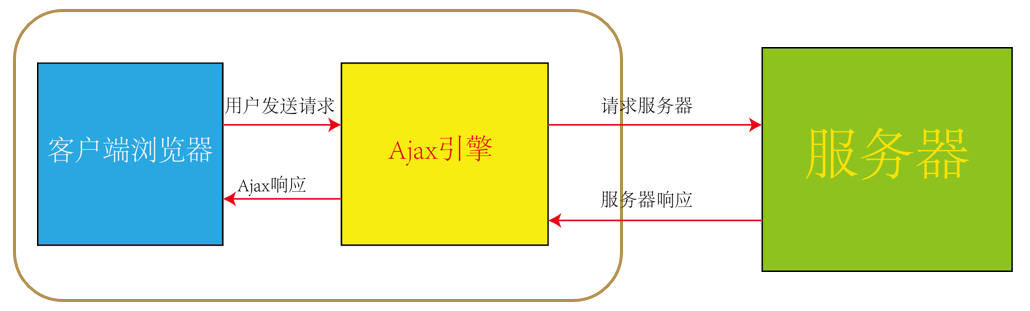
目前有两种技术方案：1：将数据打包成 XML 的格式进行传输2：将数据打包成JSON的格式进行传输。

Ajax从服务器获取的数据都是字符串，但是通过不同的解析，可以解析为XML或者JSON。一般来说，使用XML格式的数据比较通用，但是服务器和客户端解析起来都比较复杂一些；而使用JSON语句的话，服务端需要多做一些工作，但到了客户端，通过使用eval()函数来进行解析，就会获得js对象，使用起来很方便。在使用过程中，究竟使用哪种方法，这个要根据项目的实际需要，建议：在能使用JSON的情况下，尽量使用JSON。

AJAX技术要求：

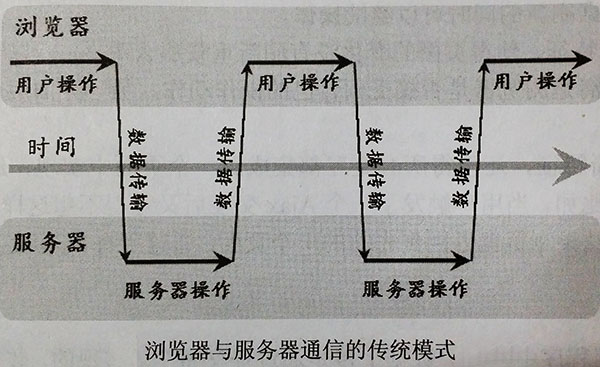
javascript+css+dom+XMLHttpRequest

Ajax工作原理：

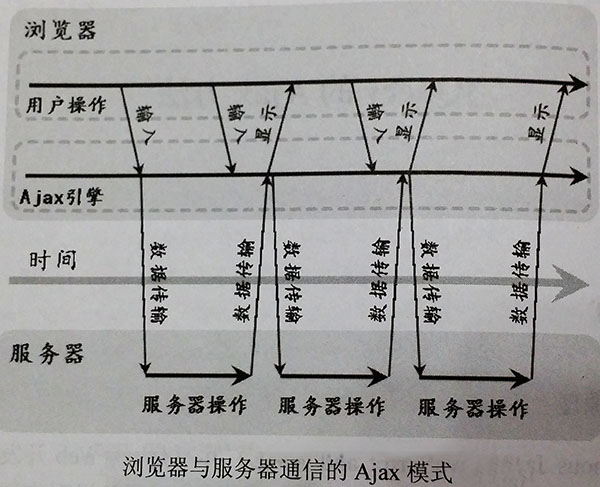


Ajax优势：

传统浏览器与服务器通信



Ajax浏览器与服务器通信



1.ajax：也就是XMLHttpRequest对象，它是JavaScript内置对象。

2.ajax功能：前后端交互

js→ajax→php→mysql→php→ajax→js

3.ajax学习的难点：

单词较多

单词较长

属性和方法配合使用

4.单词

XMLHttpRequest ajax对象的构造器函数

readyState 就绪状态(值为：0 1 2 3 4)

onreadystatechange ajax事件(ajax请求过程中自动触发)

open 打开

send 发送

responseText 响应字符串

responseXML 响应xml

ActiveXObject ajax对象的构造器函数(IE6)

-----------------------------------------------------------

ajax优点：

1.局部刷新服务器端的数据，而不是刷新整个页面，用户体验好

2.降低服务器带宽，减少服务器的负荷（瀑布流）

3.实现前后端分离，缩短项目周期，成本降低

-------------------------------------------------------------

ajax在网站中的应用（一般为局部刷新服务器端数据的情况）：

1.检测用户名是否被占用

2.登录验证

3.天猫/京东瀑布流

4.站内搜索引擎的关键字提示

5.页面初始化

--------------------

前端和后台同时进行开发，如果后台开发完了接口，我们必须得测试一下接口是否正确。如果接口正确的话，那么肯定是前端写错了，提前发现后台的错误，及时沟通，防止项目提价。

postman：后台接口测试工具

模拟前端向后台发送请求，可以测试后台接口

对json进行格式校验

# 09.25

## Ajax对象：

1.如何创建ajax对象

if(window.XMLHttpRequest){

//IE7+、Firefox、Chrome支持，IE6不支持

var xmlHttp=new XMLHttpRequest();

}else{

//IE6支持

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

console.log(xmlHttp);

**2.Ajax对象的方法**

①open方法：用于设置进行异步请求的目标URL，请求方法及其他参数信息

语法：

open("method","url",[asyncFlag]);

参数说明：

method——get/post

url——请求指定地址，并且可以传递查询字符串

asyncFlag——可选参数，**用于指定请求方式，默认为true异步**

说明：当该boolean值为true时，服务器请求是异步进行的，也就是脚本执行send()方法后不等待服务器的执行结果，而是继续执行脚本代码;

当该boolean值为false时，服务器请求是同步进行的，也就是脚本执行send()方法后等待服务器的执行结果的返回，若在等待过程中超时，则不再等待，继续执行后面的脚本代码！

实例：

open("post","chuli.php",true);

②setRequestHeader方法：为请求的HTTP头设置值（数据编码格式）。

语法：

setRequestHeader("label","value");

实例：

setRequestHeader("Content-type","application/x-www-form-urlencoded");

**注意：setRequestHeader()方法必须在调用open()方法之后才能调用**

③send方法：用于向服务器发送请求，如果请求声明为异步，则该方法将立即返回，否则将等到接收到响应为止。

语法：

send([数据])；

--------------------------

get和post的区别：

1.GET：和文档头部信息一起发送。一般用来获取服务器端数据

格式：url?变量=值&变量=值

特点：默认方法；速度快；大小受限，不能传送较大的数据量，更不能传输文件

2.POST：和文档正文一起传送。一般用于传送表单或文件

特点：大小不受限；可以传递文件

## Ajax-php数据编码：

|  |  |  |
| --- | --- | --- |
| 数据 | Ajax | php |
| 变量=值&变量=值 | application/x-www-form-urlencoded | $\_GET[‘变量名’] $\_POST[‘变量名’] |
| var formData=new FormData();  formData.append(“变量”, “值”);  formData.append(“变量”, “值”); | 不能设置setRequestHeader()，设置了会出问题 | $\_GET[‘变量名’] $\_POST[‘变量名’]  $\_FILES[ ][ ] |
| '{"变量":"值","变量":"值"}' | application/json | json\_decode(file\_get\_contents('php://input'),true); |

**3.Ajax对象事件onreadystatechange**

当请求被发送到服务器时，我们需要执行一些基于响应的任务。每当readyState属性改变时，就会触发onreadystatechange事件。

**readyState属性：**

存有XMLHttpRequest的状态信息，监控Ajax整个请求过程。

readyState存储XMLHttpRequest的状态，从0到4发生变化（该属性值由Ajax自动修改）。

0：请求未初始化（还没有调用 open()）

1：服务器连接已建立，但是还没有发送（已经调用open()，还没有调用 send()）loading……

2：请求已接收，正在处理中（已经调用send(), 接收到响应头信息） loaded……

3：请求处理中，通常响应中已有部分数据可用了，但是服务器还没有完成响应的生成。

4：请求已完成，且响应已就绪，可以获取并使用服务器的响应了。

readyState属性详解如下：

readyState存储XMLHttpRequest的状态，从0到4发生变化（将Ajax访问过程比作是运送货物去处理，并且将成品运回）。

0：XMLHttpRequest对象还没有完成初始化。(准备司机、车、货物)

1：XMLHttpRequest对象开始发送请求。(需要送十车货物，当前正在送第几车)

2：XMLHttpRequest对象的请求发送完成。(十车货送完毕)

3：XMLHttpRequest对象开始读取服务器的响应。(准备把这十车加工的货拉回来，当前第几车)

4：XMLHttpRequest对象读取服务器响应结束。(十车货全部拉回完毕)

**4.Ajax对象的属性**

**①status属性（来自接口文档）：**

200："OK"

404：未找到页面

500：服务器端错误

status属性详解如下：

1xx：信息响应类，表示接收到请求并且继续处理。(所有拉去的货，工厂还没有加工完毕)

**2xx：处理成功响应类，表示动作被成功接收、理解和接受。(所有拉去的货工厂全部加工完毕)**

3xx：重定向响应类，为了完成指定的动作，必须接受进一步处理。(所有拉去的货，工厂设备不够，让其他工厂帮忙加工)

**4xx：客户端错误，客户请求包含语法错误或者不能正确执行。(这十车货有质量问题，工厂不能正常加工)**

5xx：服务端错误，服务器不能正确执行一个正确的请求。(工厂在加工到一半过程中断电，不能继续加工)

**②responseText：**获取服务器端字符串形式数据（包括字符串形式json数据）

**③responseXML：**获取服务器端XML形式数据

----------------------------------------------------------------

ajax请求步骤：

1.不带参数请求4步：

（1）创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

（2）设置请求方法和目标地址

xmlHttp.open("GET/POST","url");

（3）绑定onreadystatechange事件，监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4 && xmlHttp.status==200){

var data=xmlHttp.responseText; //取回来的数据

}

}

（4）发送请求

xmlHttp.send();

2.带参数请求6步：

（1）创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

（2）准备发送给服务器的数据

var data="变量名=值&变量名=值";

（3）设置请求方法和目标地址

xmlHttp.open("GET/POST","url?"+data);——GET方法

xmlHttp.open("GET/POST","url");——POST方法

（4）设置数据格式编码

xmlHttp.setRequestHeader("content-type","数据编码格式");

（5）绑定onreadystatechange事件，监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4 && xmlHttp.status==200){

var data=xmlHttp.responseText; //取回来的数据

}

}

（6）发送请求

xmlHttp.send();——GET方法

xmlHttp.send(data);——POST方法

----------------------------------------------------

综合实验：

**1、get方法不带参数请求普通字符串或html**

get.html：

<body>

<div id="div1"></div>

<input type="button" id="bt1" value="点击获取服务器端字符串" />

</body>

<script type="text/javascript">

window.onload=function(){

var bt1=document.getElementById("bt1");

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

bt1.onclick=function(){

//执行ajax四步：

//1.创建ajax对象

if(window.XMLHttpRequest){

//IE7+

var xmlHttp=new XMLHttpRequest();

}else{

//IE6

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.设置请求方法和目标地址

xmlHttp.open("GET","data.php");

//3.绑定onreadystatechange事件，监控整个ajax请求过程

xmlHttp.onreadystatechange=function(){

//数据传回来而且浏览器就绪

if(xmlHttp.readyState==4&&xmlHttp.status==200){

div1.innerHTML=xmlHttp.responseText;

}

}

//4.发送请求

xmlHttp.send();

}

}

</script>

data.php：

<?php

echo 'hello,world!';

?>

-------------------------------

预备知识：图片预加载

（1）loading图片→base64→插入到css文件中→<head><link href=""></head>

（2）js实现

loading图片在线生成：

http://loadingapng.com/

http://www.ajaxload.info/

例：

get.html：

<body>

<div id="div1"></div>

<input type="button" id="bt1" value="点击获取服务器端字符串" />

</body>

<script type="text/javascript">

window.onload=function(){

var bt1=document.getElementById("bt1");

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

//图片预加载

var img1=new Image();

img1.src="../images/loading.gif";

bt1.onclick=function(){

div1.appendChild(img1);

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest(); //IE7+

}else{

//IE6

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

xmlHttp.open("GET","data.php");

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

div1.innerHTML=xmlHttp.responseText;

}

}

xmlHttp.send();

}

}

</script>

data.php：

<?php

sleep(rand(1, 5)); //延迟执行当前脚本

echo 'hello,world!';

?>

-----------------------------

复习：

前端和后台使用的语言不同：

前端——javascript

后台——php/java/python/c#

在没有json字符串之前，前后端数据交互用xml格式数据，后来被json代替。

前端和后台都认识json字符串(简称json)，前端给后台传字符串或json字符串；后台给前端传字符串或json字符串。

------------------------------------------------

json字符串语法：

1.两端加''

2.写在单引号里面{}

3.属性必须加""

4.值是字符串必须加""，其他类型（number、boolean、数组）不加双引号

5.属性和值之间用“:”分割

6.“属性:值”之间用“,”分割

举例：

1.简单json

'{"status":"10001","msg":"success"}'

2.复杂json

'{"status":"10001","meg":"success","data":[{"id":"9001","name":"tom","age":19,"sex":"男"},{"id":"9002","name":"jarry","age":21,"sex":"男"},{"id":"9003","name":"susan","age":22,"sex":"女"}]}'

后台给前端传json字符串，json字符串存储在responseText中，没法获得或没法“.”出来，怎么办?

答：将json字符串转换为js对象。有下面两种方法：

1.兼容性好；有安全隐患

var obj=eval("("+json+")");

2.IE9+以上支持；安全

var obj=JSON.parse(json);

扩展知识：

js对象转换为json字符串：

JSON.stringify(js对象);

------------------------------------

**2、get方法不带参数请求简单json**

get.html：

<body>

<div id="div1"></div>

<input type="button" id="bt1" value="点击获取简单json" />

</body>

<script type="text/javascript">

window.onload=function(){

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

var bt1=document.getElementById("bt1");

//图片预加载

var img1=new Image();

//当浏览器给img1的src属性赋值时，它会自动从服务器下载这张图片，并加载到浏览器中

img1.src="../images/loading.gif";

bt1.onclick=function(){

div1.appendChild(img1);

//执行ajax

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.设置请求方法和目标地址

xmlHttp.open("GET","data.php");

//3.监控整个ajax请求过程，能了解ajax请求进行到哪一步了

//readyState属性监控整个过程,值是系统自动赋予的

//0 没open

//1 open了，没send

//2 send请求发送了，收到响应头ResponseHeaders

//3 服务器返回数据，但是没有完全返回

//4 数据全部返回到浏览器

//status属性表示浏览器是否就绪：200、404、500

//responseText后台输出所有的数据字符串

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

//获取服务器返回的数据responseText

var data=xmlHttp.responseText;

//json对象转换为js对象，才能使用

data=JSON.parse(data);

div1.innerHTML="姓名："+data.name+"年龄："+data.age+"性别："+data.sex;

}

}

//4.发送请求

xmlHttp.send();

}

}

</script>

data.php：

<?php

header("content-type:text/html;charset=utf-8");

echo '{"name":"Tom","age":19,"sex":"男"}';

?>

补充1：使用JSON.parse()转json字符串为json对象时，如果语法报错，可能是json字符串书写有误，可以用在线JSON校验格式化工具检查一下：<http://www.bejson.com/>。

补充2：

href用于建立与外部资源的链接，与文档渲染同时进行，不影响页面加载；

src是source的缩写，所在位置将被替换为其指向的文件内容，常见的如js和image，页面遇到此类元素会暂停渲染直到src内容加载并执行编译完成，再加载后续资源。故script标签常放在body最下方。

**3、get方法不带参数请求复杂json**

get.html：

<body>

<div id="div1"></div>

<input type="button" id="bt1" value="点击获取简单json" />

</body>

<script type="text/javascript">

window.onload=function(){

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

var bt1=document.getElementById("bt1");

//图片预加载

var img1=new Image();

img1.src="../images/loading.gif";

bt1.onclick=function(){

div1.appendChild(img1);

//执行ajax

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.设置请求方法和目标地址

xmlHttp.open("GET","data.php");

//3.监控整个ajax请求过程，能了解ajax请求进行到哪一步了

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

//获取服务器返回的数据responseText

var data=xmlHttp.responseText;

//json对象转换为js对象，才能使用

data=JSON.parse(data).student;

//遍历student数组

var str="<ul>";

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

str=str+"<li>"+"姓名："+data[i].name+"年龄："+data[i].age+"性别："+data[i].sex+"</li>";

}

div1.innerHTML=str+"</ul>";

}

}

//4.发送请求

xmlHttp.send();

}

}

</script>

data.php：

<?php /\* "<?php"前边不能有任何内容，包括空格 \*/

sleep(rand(1, 5));

header("content-type:text/html;charset=utf-8");

echo'{"status":"10001","meg":"success","student":[{"id":"9001 ","name":"Obama","age":"19","sex":"男"},{"id":"9002","name":"Trump","age":"21","sex":"男"},{"id":"9003","name":"Pete","age":"22","sex":"女"}]}';

?>

-----------------------------------------------------------------------------

在没有json字符串之前，前后端数据交互用xml格式数据，后来被json代替。

xml和json一样都是数据传输标准：

①xml数据交换标准

缺点：数据量大，占带宽；不好维护；遍历困难

②json轻量级的数据交换标准

HTML文档分类：

1.xml 可扩展标记语言 1998年

2.html4.01 1999年

3.xhtml1.0 2000年

4.html5

react的jsx语法，采用的是xml规范。

xml作为数据交换标准，一定有严格规范：

1.只能有一个根元素

注：XML文档必须有一个元素是所有其他元素的**父元素**，该元素称为**根元素**。

2.标记和属性必须小写

3.标记必须关闭

4.属性值必须加""

5.标记只能嵌套不能交叉

-----------------------------------

JavaScript的DOM兼容性比较好，支持HTML、XHTML、XML。

**responseText和resposneXML的区别：**

responseText存储服务器返回来的字符串

responseXML会将服务器返回的字符串转换为xml文档对象document

**4、get方法不带参数请求xml格式数据**

get.html：

<body>

<div id="div1"></div>

<input type="button" id="bt1" value="点击获取简单json" />

</body>

<script type="text/javascript">

window.onload=function(){

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

var bt1=document.getElementById("bt1");

//图片预加载

var img1=new Image();

img1.src="../images/loading.gif";

bt1.onclick=function(){

div1.appendChild(img1);

//执行ajax

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.设置请求方法和目标地址

xmlHttp.open("GET","data.xml");

//3.监控整个ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

//responseXML会将xml字符串转换为document对象

//document对象代表整篇xml文档

var data=xmlHttp.responseXML;

//console.log(data);

//console.log(data.getElementsByTagName("stu"));

//console.log(data.getElementsByTagName("stu")[0].children[0].firstChild); //console.dir(data.getElementsByTagName("stu")[0].children[0].firstChild.nodeValue);

data=data.getElementsByTagName("stu");

var str="<ul>";

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

str=str+"<li>姓名："+data[i].children[0].firstChild.nodeValue

+"年龄："+data[i].children[1].firstChild.nodeValue

+"性别："+data[i].children[2].firstChild.nodeValue+"</li>";

}

div1.innerHTML=str+"</ul>";

}

}

//4.发送请求

xmlHttp.send();

}

}

</script>

data.xml： //data.php也是可以的

<?xml version="1.0" encoding="utf-8"?>

<students>

<stu>

<name>Obama</name>

<age>19</age>

<sex>男</sex>

</stu>

<stu>

<name>Trump</name>

<age>21</age>

<sex>男</sex>

</stu>

<stu>

<name>Elizabeth</name>

<age>22</age>

<sex>女</sex>

</stu>

</students>

**5、get方法带参数(数据格式都是urlencoded)请求json**

实验：检测用户名是否被占用

get.html：

<body>

用户名：<input type="text" id="userName"/><br />

<span id="tips"></span>

</body>

<script type="text/javascript">

window.onload=function(){

var userName=document.getElementById("userName");

var tips=document.getElementById("tips");

userName.onblur=function(){

//ajax过程：

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.准备要给服务器发送的数据(urlencoded格式)

//变量名=值&变量名=值

var data="userName="+userName.value; //"userName=孙强"

//3.设置请求方法和目标地址

xmlHttp.open("GET","data.php?"+data);

//4.设置向服务器发送数据的编码

xmlHttp.setRequestHeader("content-type","application/x-www-form-urlencoded");

//5.监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

var data=xmlHttp.responseText;

data=JSON.parse(data);

if(data.status=="20001"){

tips.innerHTML=data.msg;

}else{

tips.innerHTML=data.msg;

}

}

}

//6.发送请求

xmlHttp.send();

}

userName.onfocus=function(){

tips.innerHTML="";

}

}

</script>

data.php：

<?php

header("content-type:text/html;charset=utf-8");

//获取客户端传递过来的用户名

//判断数据库里面有没有

//如果有，就是占用了，返回一个json

//否则就是没有占用，返回一个json

$userName=$\_GET['userName'];

//假装这是一个数据库：

$names="赵匡胤钱钟书孙强李志周杰伦吴莫愁郑和王旺";

if(strstr($names, $userName)){

echo '{"status":"20001","msg":"此用户名已被占用"}';

}else{

echo '{"status":"10001","msg":"此用户名可用"}';

}

?>

补充1：

第二步的时候如果传输多个变量的写法：

var post\_data=“userName=”+userName+“&pwd=”+pwd;

补充2：

第三步，GET方法向URL添加数据，URL的长度是受限制的（URL的最大长度是2048个字符）。详情点击[这里](get和post方法区别.docx)

# 09.26

综合实验：续09.25

**6、post方法不带参数请求json**

09-26post.html：

<body>

<input type="button" id="bt1" value="点击获取服务器端json" />

<div id="div1"></div>

</body>

<script type="text/javascript">

window.onload=function(){

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

var bt1=document.getElementById("bt1");

bt1.onclick=function(){

div1.innerHTML="";

div1.className="loading";

//执行ajax过程

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.设置请求方法和目标地址

xmlHttp.open("POST","09-26data6.php?id=");

//3.绑定ajax全局事件onreadystatechange，监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

var data=xmlHttp.responseText;

data=eval("("+data+")");

if(data.status=="1001"){

//遍历

var str="<ul>";

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

str+="<li>学号："+data.data[i].id+"姓名："+data.data[i].name+"年龄："+data.data[i].age+"性别："+data.data[i].sex;

}

div1.className="";

div1.innerHTML=str+"</ul>";

}else{

div1.className="";

div1.innerHTML=data.msg;

}

}

}

//4.发送请求

xmlHttp.send();

}

}

</script>

09-26data.php：

<?php

header("content-type:text/html;charset=utf-8");

sleep(rand(1, 3));

echo '{"status":"1001","msg":"success","data":[{"id":"10001","name":"Obama","age":19,"sex":"男"},{"id":"10002","name":"Trump","age":22,"sex":"男"},{"id":"10003","name":"Elisabeth","age":18,"sex":"女"}]}';

?>

**7、post方法带参数(数据格式是urlencoded)请求json**

实验：登录验证

post.html：

<body>

用户名：<input type="text" id="userName"/><br />

密码：<input type="password" id="pas"/><br />

<div id="tips"></div>

<input type="button" id="bt1" value="登录" />

</body>

<script type="text/javascript">

window.onload=function(){

var Q=function(\_id){

return document.getElementById(\_id);

}

var userName=Q("userName");

var bt1=Q("bt1");

var tips=Q("tips");

var pas=Q("pas");

bt1.onclick=function(){

//执行ajax过程：

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.准备要发送的数据

var data="userName="+userName.value+"&pas="+pas.value;

//3.设置请求方法和目标地址

xmlHttp.open("POST","data.php");

//4.设置发送数据的编码urlencoded

xmlHttp.setRequestHeader("content-type","application/x-www-form-urlencoded");

//5.监控整个ajax过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

var data=xmlHttp.responseText;

data=JSON.parse(data);

if(data.status=="10001"){

location.href="manage.php";

}else{

tips.innerHTML=data.msg;

}

}

}

//6.发送请求

xmlHttp.send(data);

}

userName.onfocus=function(){

tips.innerHTML="";

}

}

</script>

data.php：

<?php

header("content-type:text/html;charset=utf-8");

//获得用户名和密码

//判断用户名和密码是否正确

//如果正确，返回一个json

//如果错误，返回一个json

$userName=$\_POST['userName'];

$pas=$\_POST['pas'];

if($userName=="Obama" and $pas=="122330"){

echo '{"status":"10001","msg":"success"}';

}else{

echo '{"status":"20001","msg":"用户名或密码错误！"}';

}

?>

manage.php：略

----------------------

补充：

问题：POST请求无法被缓存，但重复发送相同的GET请求，有时数据会被浏览器强制缓存。

解决：在请求目标地址尾部加上“?id="+new Date().getTime()”。

例：xmlHttp.open("POST","09-26data6.php?id="+new Date().getTime());

--------------------------------------------------------------------------------------------

将来公司里做开发的时候，前后端同时开始，接口都遵循同一个标准。这个标准包含：

目标地址

请求参数

响应体

在实际开发中，当前端和后台开始开发前，必须将接口文档发给前端和后台，前端和后台按照接口文档开发。

application——应用程序，简称app

interface——接口

application interface——应用程序接口，简称api

比如JavaScript中的api文档（帮助文档）

---------------------------------------------------------

注册模块接口：

接口地址：chuli.php

请求方法：POST

请求参数（入参）→前端给后台的数据：

userName String 20

password String 15

x file 500KB

email String 60

响应体（出参）→后台给前端的数据：

status String 10 操作码

msg String 100 提示信息

出参示例：

'{"status":"10001","msg":"success"}';

'{"status":"30001","msg":"注册失败，请联系管理员"}';

----------------------------------------------------------------------------------

**8、post带参数（formData数据）请求json**

预备知识：

①如何获取要上传的文件?

答：通过html5提供的api获取

DOM对象.files; //以数组形式存储着所有要上传的文件

②如果发送formData二进制数据，要求必须先对发送的数据进行编码。

二进制数据属于原始数据，不能设置setRequestHeader。

//先创建一个空的formdata对象

var x=new FormData();

//再给formdata对象添加键值对

x.append("key","值");

比如：x.append("userName",userName.value);

x.append("ufile",ufile.files[0]);

PHP接收：

表单数据：post→$\_POST[]

表单文件：$\_FILES[][]

move\_uploaded\_file(临时文件名，上传后的文件名); //上传

**注册实验：**

客户端浏览器通过ajax发送formData格式数据

PHP接收：

$\_POST[]

$\_FILES[][]

reg.html：

<body>

<div id="div1">

用户名：<input type="text" id="userName"/><br />

密码：<input type="password" id="pas"/><br />

email:<input type="text" id="email"/><br />

头像：<input type="file" id="pho"/><br />

<input type="button" id="bt1" value="注册" />

</div>

</body>

<script type="text/javascript">

window.onload=function(){

var O=function(\_id){

return document.getElementById(\_id);

}

var div1=O("div1");

var bt1=O("bt1");

var pas=O("pas");

var pho=O("pho");

var email=O("email");

var userName=O("userName");

bt1.onclick=function(){

//执行ajax过程：

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.准备发送数据，请求参数form-data编码

var formdata=new FormData();

//向formdata对象中添加键值对key,value

formdata.append("userName",userName.value);

formdata.append("pas",pas.value);

formdata.append("email",email.value);

formdata.append("pho",pho.files[0]);

//3.设置请求方法和目标地址

xmlHttp.open("POST","chuli.php");

//4.由于是form-data二进制数据，不能设置数据编码

//5.监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

var data=xmlHttp.responseText;

data=JSON.parse(data);

if(data.status=="10001"){

div1.innerHTML=data.msg;

setTimeout(function(){

location.href="index.php";

},3000);

}else{

alert(data.msg);

}

}

}

//6.发送请求

xmlHttp.send(formdata);

}

}

</script>

chuli.php：

<?php

header("content-type:text/html;charset=utf-8");

$userName=$\_POST['userName'];

$pas=$\_POST['pas'];

$email=$\_POST['email'];

$fileName=$\_FILES['pho']['name'];

$fileNametmp=$\_FILES['pho']['tmp\_name'];

//注册成功：

if($userName!=''&&$pas!=''&&$email!=''&&$fileName!=''){

move\_uploaded\_file($fileNametmp, "upload/".$fileName);

echo '{"status":"10001","msg":"注册成功!","userName":"'.$userName.'"}';

//"userName":"'.$userName.'"：这个代码是前后拼接起来的，为了查看是否接收到了userName，没有其他意义

}else{

echo '{"status":"20001","msg":"信息有误~"}';

}

?>

----------------------------------------------

问题：在数据准备的时候，需要一个一个的获得表单项，然后存储成键值对，如果表单项比较多的话，非常麻烦。

解决方法：使用表单序列化

表单序列化，会自动将表单的name值当成key，将表单的value值当成value。

例：

reg.html：

<body>

<div id="div1">

<form>

用户名：<input type="text" name="userName"/><br />

密码：<input type="password" name="pas"/><br />

email:<input type="text" name="email"/><br />

头像：<input type="file" name="pho"/><br />

<input type="button" id="bt1" value="注册" />

</form>

</div>

</body>

<script type="text/javascript">

window.onload=function(){

var O=function(\_id){

return document.getElementById(\_id);

}

var div1=O("div1");

var bt1=O("bt1");

bt1.onclick=function(){

//执行ajax过程：

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.准备发送数据，请求参数form-data编码，使用表单序列化（自动向formdata对象中添加键值对）

var f1=document.getElementsByTagName("form")[0];

var formdata=new FormData(f1);

//3.设置请求方法和目标地址

xmlHttp.open("POST","chuli.php");

//4.由于是form-data二进制数据，不能设置数据编码

//5.监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

var data=xmlHttp.responseText;

data=JSON.parse(data);

if(data.status=="10001"){

div1.innerHTML=data.msg;

}else{

alert(data.msg);

}

}

}

//6.发送请求

xmlHttp.send(formdata);

}

}

</script>

chuli.php：与上例（综合实验8）相同

**9、post方法带参数（json）请求，返回json——开发常用**

实验：登录验证

登录验证接口：

请求地址：chuli.php

请求方法：POST（可选）

请求参数：

userName String 20

password String 20

响应体：

status String 服务器状态

msg String 统一的提示信息

说明：响应体示例

'{"status":"10001","msg":"success"}';

'{"status":"30001","msg":"用户名或密码错误"}';

index.hml：

<body>

用户名：<input type="text" id="userName"/><br />

密码：<input type="password" id="pas"/><br />

<div id="tips"></div>

<input type="button" id="bt1" value="登录" />

</body>

<script type="text/javascript">

window.onload=function(){

var V=function(\_id){

return document.getElementById(\_id);

}

var pas=V("pas");

var userName=V("userName");

var bt1=V("bt1");

var tips=V("tips");

bt1.onclick=function(){

//执行ajax

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.准备数据

var data={userName:userName.value,pas:pas.value};

data=JSON.stringify(data); //对象转换json字符串

//写法2：var data='{"userName":"'+userName.value+'","pas":"'+pas.value+'"}';

//3.设置请求方法和目标地址

xmlHttp.open("POST","chuli.php");

//4.设置数据编码

xmlHttp.setRequestHeader("content-type","application/json");

//5.监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

var data=xmlHttp.responseText;

data=JSON.parse(data);

if(data.status=="10001"){

location.href="manage.php";

}else{

tips.innerHTML=data.msg;

}

}

}

//6.发送请求

xmlHttp.send(data);

}

userName.onfocus=function(){

tips.innerHTML="";

}

}

</script>

chuli.php：

<?php

//客户端发送数据是json，不能用$\_POST接收

$info=json\_decode(file\_get\_contents('php://input'),true);

//获取关联数组的键值

$userName=$info['userName'];

$pas=$info['pas'];

if($userName=="Obama" and $pas=="122330"){

echo '{"status":"10001","msg":"success"}';

}else{

echo '{"status":"20001","msg":"用户名或密码错误！"}';

}

?>

补充1：file\_get\_contents('php://input')是一个js对象，json\_decode将这个对象解码为（关联）数组。

补充2：用postman检验的时候，body里面写入的json字符串不要带单引号' '。

# 09.27

ajax全局事件，就是在ajax整个请求过程中自动触发的事件。

1、onreadystatechange：当readyState属性发生变化时触发

2、onloadstart：当接收到服务器响应数据第一个字节时触发（IE10+）

3、onloadend：当通信完成或触发了onerror、onabort、onload事件后触发（IE10+）

4、onprogress（了解）：

chrome中请求完成之前只调用一次的函数(一般放到open后)，而IE10、IE11会周期性触发（一直触发）。

事件对象属性loaded代表接收到字节数；

事件对象属性total代表总字节数

5、onerror：当请求发生错误的时候触发

6、onload：接收到完整响应数据时触发

触发顺序：

onloadstart→onprogress→onerror/onabort/onload→onloadend

说明：IE8以上支持的只有onload事件。

例：

get.html：

<body>

<div id="div1"></div>

<input type="button" id="bt1" value="点击获取服务器端字符串" />

</body>

<script type="text/javascript">

window.onload=function(){

var bt1=document.getElementById("bt1");

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

var img1=new Image();

img1.src="../images/loading.gif";

bt1.onclick=function(){

div1.appendChild(img1);

//执行ajax四步：

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//全局事件onloadstart：接收到服务器返回的第一个字节时触发

xmlHttp.onloadstart=function(){

console.log("开始接受服务器端响应信息");

}

//全局事件onloadend：ajax请求结束后触发

xmlHttp.onloadend=function(){

console.log("ajax请求完成");

//继续执行其他相关的操作

//比如说给返回的数据，对应的DOM绑定事件

}

/\*全局事件onprogress（了解）：

(1)chrome中，数据全部返回后触发

(2)IE10/11，周期性触发，如果数据量小，只能看到全部数据

\*/

xmlHttp.onprogress=function(e){

console.log("loaded:",e.loaded);

console.log("total:",e.total);

}

//2.设置open，请求方法和目标地址

xmlHttp.open("GET","data.php");

//3.绑定事件，监控整个ajax请求过程

xmlHttp.onreadystatechange=function(){

//数据回来而且浏览器就绪

if(xmlHttp.readyState==4&&xmlHttp.status==200){

div1.innerHTML=xmlHttp.responseText;

}

}

//4.发送请求

xmlHttp.send();

}

}

</script>

data.php：

<?php

sleep(rand(1, 5)); //延迟执行当前脚本

echo 'hello,world!';

?>

-------------------------------

ajax封装为函数：

function ajax(param){

try{

if(typeof param=="object"&&param!=undefined){

//执行ajax过程，分为8种情况

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.设置请求方法和地址

//2.1 get带参数请求：url?变量=值&变量=值

//2.2其他情况

//加一个completed，ajax请求结束以后触发

xmlHttp.onloadend=function(){

if(param.completed!=undefined){

param.completed();

}

}

if(param.type=="GET"&&param.data!=undefined){

xmlHttp.open(param.type,param.url+"?"+param.data);

}else{

xmlHttp.open(param.type,param.url);

}

//3.设置请求数据的编码

//3.1 get方法带参数、post方法带参数（urlencoded）→urlencoded

//3.2 post方法带参数（form-data）→不能设置编码

//3.3 post方法带参数（json），设置编码为json

//补充：jQuery封装时，会将传入的Object对象类型的参数转换为urlencoded。

if(param.data!=undefined){

switch(param.contentType){

case "urlencoded":

xmlHttp.setRequestHeader("content-type","application/x-www-form-urlencoded");

break;

case "json":

xmlHttp.setRequestHeader("content-type","application/json");

break;

}

}

//4.监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

var data=xmlHttp.responseText;

if(param.dataType=="json"){

//dataType:返回的数据类型

//json字符串，需要转换

//html或普通字符串，不需要转换

data=JSON.parse(data);

}

param.success(data);

}

}

//5.发送请求

//5.1 post带参数请求

//5.2 其他情况

if(param.type=="POST"&&param.data!=undefined){

xmlHttp.send(param.data);

}else{

xmlHttp.send();

}

}else{

throw new Error("参数不正确！");

}

}catch(e){

alert(e.message);

}

}

说明1：param的属性

url:接口地址

type:请求方法

contentType:数据编码

data:请求数据

dataType:返回的数据类型

success:ajax成功回调函数,readyState==4&&status==200,onload之后

completed:ajax请求结束的操作,onloadend是触发的回调函数,ajax排队

error:ajax触发onerror

说明2：封装成函数的ajax如何使用

例：

ajax({

type:"POST",

url:"chuli.php",

contentType:"urlencoded",

data:"x=100&y=200";

dataType:"json";

success:function(data){

//返回的data**（json已转换为Object对象类型）**

}, **//别丢逗号“,”！**

conpleted:function(){

//ajax请求结束以后的操作

}

});

--------------------------------------------------------

团队协作，最后代码进行合并，由于大家使用的都是全局变量，所以代码会互相覆盖，造成程序无法执行。

问题：团队中代码合并，污染顶级变量

问题根源：所有文件中使用的变量都是全局变量

解决方法：

因为函数就是作用域，函数中的变量在函数外不可见，所以我们把全局变量变成局部变量即可。

如果要对外输出对象或函数，就将其挂到全局对象window上。

例：

public.js：

(function(){

var $={};

//addEvent封装函数（略）

//ajax封装函数（略）

$.addEvent=addEvent;

$.ajax=ajax;

window.$=$;

})();

publicjson.php：

<?php

header("content-type:text/html;charset=utf-8");

echo '{"name":"Pete","age":"22","sex":"男"}';

?>

publicget.html：

<body>

<div id="div1"></div>

<input type="button" id="bt1" value="点击获取json" />

</body>

<script type="text/javascript" src="public.js">

</script>

<script type="text/javascript">

$.addEvent({

elem:window,

eventType:"load",

func:function(){

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

var bt1=document.getElementById("bt1");

$.addEvent({

elem:bt1,

eventType:"click",

func:function(){

//ajax过程

$.ajax({

url:"publicjson.php",

type:"GET",

dataType:"json",

success:function(data){

div1.innerHTML="姓名："+data.name+"年龄："+data.age+"性别："+data.sex;

}, //别丢逗号！

completed:function(){

alert("ajax结束后可以进行的操作");

}

});

}

});

}

});

</script>

# 09.28

软件设计模式分为两类：

1.C/S Client/Server 客户端/服务器模式

特点：需要安装专用的客户端软件；服务器的一部分功能可以放到客户端实现，降低服务器的负担

举例：PC端：QQ 爱奇艺客户端 腾讯视频客户端 酷狗 绝地求生

移动端：各种APP 抖音 和平精英

缺点：需要专用客户端软件；经常更新

2.B/S Browser/Server 浏览器/服务器模式

所有的动态网站都是该类型

特点：不需要安装客户端；方便维护

--------------------------------------------------

天猫项目：

1.C/S模式：手机APP→移动网站打包生成的APP

2.B/S模式

PC端网站

移动端网站

-----------------------

手机APP分类：

1.原生App

应用技术：

Android java

IOS Swift

优点：性能高→手游

缺点：开发周期长；开发成本高

2.hybrid App（75%）：移动端网站+底层HTML5 API→打包生成Android和IOS安装包

优势：开发周期短；成本非常低

缺点：底层应用效率没有原生高

---------------------------------------------

公司架构：

业务部 技术部 人力资源 行政部 后勤部 财务部

部门经理/产品经理

技术主管

项目组1：组长1

UI组 组长1 4-6

前端组 组长1 4-6

后台组 组长1 7-10

测试人员

项目组2

项目组3

……

**公司业务流程：**

1.技术部经理从业务部接到项目：

软件需求说明书（[样本](item/ICSS软件需求说明书-日期.docx)）

产品原型图（[样本](item/FO原型设计图及功能流程最终版-产品原型图)）→伪网站

2.第一次评审大会：

参会人员：主管技术副总+技术经理+技术主管+其他（技术经理带以上两个文件）

确定：成本核算

项目周期

项目功能

3.第二次评审大会：

参会人员：技术经理+技术主管+UI组长+前端组长+后台组长+测试组长

确定：项目样式

项目功能

分配任务：UI设计效果图

后台设计接口文档（含mock数据）和数据库

其他事项

接下来的一周，UI去设计效果图，后台去写接口，设计数据库，前端认真阅读软件需求说明书和产品原型图或者熟悉框架API。

------------------------------------------------------------------

和前端开发人员相关场景：两次评审大会结束，UI设计完毕效果图、后台设计完毕接口及数据库，前端组长将UI效果图和后台接口发给前端人员。

前端组长分配任务：待诊模块、患者详情模块

前端开发前手里的文档：

第一阶段（一周）：

软件需求说明书（认真看）

产品原型图Axure（认认真真看）

第二阶段：

UI 效果图

接口文档

前端开发规范

文件命名规范

文件夹命名规范

CSS规范

JavaScript规范

兼容性规范

场景1.前端和后台同时进行开发，没有后台数据，前端使用mock数据进行DOM填充。

ajax.js ajax封装

公司自己封装的ajax

jQuery封装的ajax

阿里 fetch

axios

addEvent.js 事件封装（改变）

场景2.前端和后台同时进行开发，前端写mock数据，来做DOM填充；后台创建数据库及表，按照接口文档开发业务逻辑（主要给前端提供json和逻辑处理）。

场景3.前端和后台按照联调时间规定，在联调时间前，将代码提交到git/svn。

校验格式json

校验功能

场景4.项目联调：

大型/中型公司：联调服务器大家只需要使用联调服务器接口地址

①后台写好接口放到开发服务器

②接口文档发给前端

③前端在开发环境通过代理服务器访问接口进行开发

④后台写完真实接口后，同步到开发服务器

⑤前端开发完毕，同步到开发服务器

⑥前端联调，只需要在开发环境即可

中型或小型公司：将前端和后台代码，及数据库都布置到自己的电脑环境上，在自己的电脑上调试，url地址换成对应的地址

①需要后台在你的电脑上，安装后台开发环境

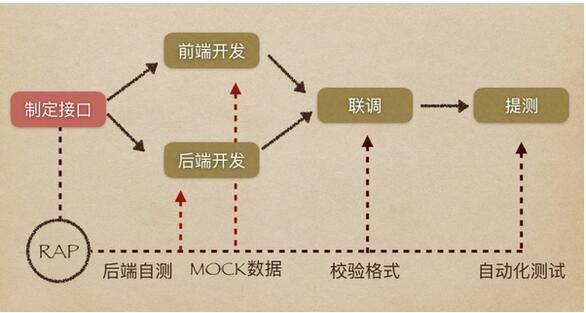
②让后台把整个项目布置到你的电脑上

③做哪个模块，用postman测试后台接口，如果有问题找后台改

④如果模块接口没有问题，改bug

联调中（格式校验，功能测试），后台接口建议使用postman测试无误后，再使用前端代码进行测试。

联调发现bug或问题，进行修改，都没有问题后，提交测试（烧机测试）。



**前后端分离开发：**

1.前端按照接口文档进行开发

问题：前端开发的时候ajax没有后台，更没有数据怎么办？

答：可以使用mock数据

mock数据有三种：

静态mock：仅支持GET，不支持模拟业务逻辑

动态mock：GET/POST，模拟业务逻辑——mock.js

大型公司提供开发测试服务器，上面有写好的所有mock数据，而且是动态的

2.后台按照接口文档进行开发

3.前端和后台所有文件代码全部完成→联调测试

-----------------------------------------------------------------

实验：静态mock数据的使用

get方法或post方法获取mock提供的json（如果是静态mock，很多服务器不支持post方法）

静态mock缺点：

必须得有服务器

有的服务器只支持GET方法

无法模拟网络延迟

无法提供动态数据

无法模拟业务逻辑

联调的时候，还得将mock接口地址换成真实接口地址，GET方法换成我们要用的方法

get1.html：

<body>

<div id="div1">

<img src="img/loading.gif"/>

</div>

</body>

<script type="text/javascript" src=" public.js">

//public.js封装了addEvent和ajax方法

</script>

<script type="text/javascript">

$.addEvent({

elem:window,

eventType:"load",

func:function(){

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

//执行ajax

$.ajax({

type:"get",

url:"mock/students.json",

dataType:"json",

success:function(data){

console.log(data);

//console.log(data.data);

var str='';

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

str+="姓名："+data.data[i].name+"年龄："+data.data[i].age+"性别："+data.data[i].sex+"<br>";

}

div1.innerHTML=str;

}

});

}

});

</script>

students.json：

{ //无法模拟网络延迟

"status":"10001",

"meg":"success",

"data":[

{"name":"Obama","age":22,"sex":"男"},

{"name":"Trump","age":28,"sex":"男"},

{"name":"Pete","age":55,"sex":"女"}

]

}

------------------------------------------------------------------

我们开发如果用的是静态mock，非常麻烦而且浪费时间。mock.js专门解决mock数据问题。

mock.js概述：

mock.js 是一款前端开发中拦截Ajax请求，再生成随机数据响应的工具。

可以用来模拟服务器响应，优点是非常简单方便，无侵入性，基本覆盖常用的接口数据类型。

mock.js特点：

不需要服务器支持（直接拦截ajax，不走网络，network监控不到）

GET/POST都支持

可以模拟网络延迟

提供动态数据

模拟业务逻辑

联调的时候，接口地址不用换，方法也不用换

注意：联调的时候，要删除mock.js和data.js

使用方法：

1.引入public.js：

<script type="text/javascript" src="js/public.js"></script>

2.引入mock.js（mock框架）

<script type="text/javascript" src="data/mock.js"></script>

3.引入data01.js（接口数据生成脚本）

<script type="text/javascript" src="data/data01.js"></script>

基本语法：数据模版格式

'name|rule': value

属性名|生成规则: 属性值

语法规范实例：

{

"user|1-3":[{ //随机生成1-3个数组元素

'name':'@cname', //中文名称

'id|+1':88, //属性值自动加1，初始值为88

'age|18-28':0, //18-28以内的随机整数，0是用来确定类型的

'birthday':'@date("yyyy-MM-dd")', //日期

'city':'@city(true)', //中国城市

'color':'@color', //16进制颜色

'isMale|1':true, //true的概率是1/2

'isFat|1-2':true, //true的概率是1/(1+2)

'fromObj1|2':obj, //从obj对象中随即获取2个属性

'fromObj2|1-3':obj, //从obj对象中随即获取1-3个属性

'brother|1':['jack','jim'], //随机选取1个元素

'sister|+1':['jack','jim','lily'], //array中顺序选取元素作为结果

'friends|2':['jack','jim'] //重复选取2次属性值生成一个新数组

}]

}

**语法规范：**

Mock.js的语法规范包括两部分：

数据模板定义规范（Data Template Definition，DTD）

数据占位符定义规范（Data Placeholder Definition，DPD）

参考网址：https://github.com/nuysoft/Mock/wiki/Syntax-Specification

实验1：get方法请求json

get.html：

<body>

<div id="div1">

<img src="images/loading.gif"/>

</div>

</body>

<script type="text/javascript" src="js/mock.js"></script>

<script type="text/javascript" src="js/public.js"></script>

<script type="text/javascript" src="data/data01.js"></script>

<script type="text/javascript">

$.addEvent({

elem:window,

eventType:"load",

func:function(){

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

$.ajax({

type:"GET",

url:"students.php",

dataType:"json",

success:function(data){

console.log(data);

var str='';

data=data.data;

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

str+="姓名："+data[i].name+"年龄："+data[i].age+"性别："+data[i].gender+"<br>";

}

div1.innerHTML=str;

}

});

}

});

</script>

data01.js：

Mock.setup({timeout:'1000-3000'}); //模拟网络延迟

Mock.mock("students.php",{

"data|1-6":[{ //随机生成1到6个数组元素

'name|1': ['Obama',"Putin","Trump"],

'age|18-30': 0, //18至30以内随机整数, 0只是用来确定类型的

'gender|1':['男',"女"]

}]

});

实验2：post方法带json请求参数，响应json

login.html：

<body>

用户名：<input type="text" id="userName"/><br />

密码：<input type="password" id="pas"/><br />

<div id="tips"></div>

<input type="button" id="bt1" value="登录" />

</body>

<script type="text/javascript" src="js/public.js"></script>

<script type="text/javascript" src="js/mock.js"></script>

<script type="text/javascript" src="data/data02.js"></script>

<script type="text/javascript">

$.addEvent({

elem:window,

eventType:"load",

func:function(){

var Q=function(\_id){

return document.getElementById(\_id);

}

var userName=Q("userName");

var pas=Q("pas");

var bt1=Q("bt1");

$.addEvent({

elem:bt1,

eventType:"click",

func:function(){

//执行ajax

$.ajax({

url:"chuli.php",

type:"POST",

data:JSON.stringify({

userName:userName.value,

pas:pas.value

}),

dataType:"json",

success:function(data){

if(data.status=="10001"){

location.href="manage.html";

}else{

tips.innerHTML=data.msg;

}

}

});

}

});

$.addEvent({

elem:userName,

eventType:"focus",

func:function(){

tips.innerHTML="";

}

});

}

});

</script>

data02.js：

Mock.mock("chuli.php","post",function(options){

console.log(options);

var json=JSON.parse(options.body);

if(json.userName=="Obama"&&json.pas=="122330"){

return Mock.mock({"status":"10001","msg":"success"});

}else{

return Mock.mock({"status":"20001","msg":"用户名或密码错误！"});

}

});

注1：options参数说明：

options:{

body:{json},

type: "POST",

url:"http://dff.com"

}

注2：设置响应时间的方法

设置4秒后再响应：Mock.setup({ timeout: 4000 });

设置1秒至4秒间响应：Mock.setup({ timeout: '1000-4000' });

实验2改：

login-gai：

……

$.ajax({

url:"chuli-plus.php",

type:"POST",

data:{

"userName":userName.value,

"pas":pas.value

},

dataType:"json",

success:function(data){

if(data.status=="10001"){

location.href="manage.html";

}else{

tips.innerHTML=data.msg;

}

}

});

……

data03.js：

Mock.mock("chuli-plus.php","post",function(options){

console.log(options);

var userName=options.body.userName;

var pas=options.body.pas;

if(userName=="Obama"&&pas=="122330"){

return Mock.mock({"status":"10001","msg":"success"});

}else{

return Mock.mock({"status":"20001","msg":"用户名或密码错误！"});

}

});

# 10.0x

诊疗管理系统-诊疗患者病历模块（有后台）：

index.html：

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>诊疗管理系统-诊疗患者病历模块</title>

<style type="text/css">

#left{

border:1px solid pink;

width: 300px;

height: 400px;

float: left;

}

#right{

border:1px solid pink;

width: 500px;

height: 400px;

float: left;

}

#left p{

cursor: pointer;

}

</style>

</head>

<body>

<div id="left">

<img src="img/qrCodeloading.gif"/>

</div>

<div id="right">

</div>

</body>

<script type="text/javascript" src="js/public.js"></script>

<script type="text/javascript">

$.addEvent({

elem:window,

eventType:"load",

func:function(){

//onload装载待诊人员列表

var left=document.getElementById("left");

var right=document.getElementById("right");

$.ajax({

type:"get",

url:"patient.php",

contentType:"urlencoded",

data:"hospitalId=10001&stffId=10002&departId=10003",

dataType:"json",

success:function(data){

data=data.data;

var str="";

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

var state="待诊";

switch(data[i].regVisitedState){

case 0:state="待诊";

break;

case 1:state="就诊中";

break;

case 2:state="已诊完";

break;

}

str+="<p data-patientId="+data[i].id+">"+data[i].patName+"("+state+")"+"</p>";

left.innerHTML=str;

}

}, //别丢逗号！！！

completed:function(){

var ps=left.getElementsByTagName("p");

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

ps[i].onclick=function(){

//请求对应id病人详细信息

$.ajax({

url:"detail.php",

type:"GET",

contentType:"urlencoded",

data:"patientId="+this.getAttribute("data-patientId"),

dataType:"json",

success:function(data){

right.innerHTML=

"<br>编号："+data.id+

"<br>姓名："+data.patName+

"<br>性别："+data.patSex+

"<br>出生日期:"+data.patBirthday+

"<br>……";

}

});

//请求对应id病人详细信息

}

}

}

});

//onload装载待诊人员列表

//onload加载正在诊疗人员

//病人详情接口

$.ajax({

url:"defaultdetail.php",

type:"GET",

contentType:"urlencoded",

data:"patientId=980001",

dataType:"json",

success:function(data){

right.innerHTML=

"<br>编号："+data.id+

"<br>姓名："+data.patName+

"<br>性别："+data.patSex+

"<br>出生日期:"+data.patBirthday+

"<br>……";

}

});

//病人详情接口

}

});

</script>

</html>

public.js：

(function(){

var $={};

//addEvent

function addEvent(param){

//param属性说明：

//elem：DOM对象

//eventType：事件类型

//func：事件处理函数

try{

if(typeof param.elem=="object"&&param.elem!=undefined&&param.elem!=null){

if(window.addEventListener){

//IE9+

param.elem.addEventListener(param.eventType,param.func);

}else{

//IE6/7/8

param.elem.attachEvent("on"+param.eventType,param.func);

}

}else{

throw new Error("不是对象或对象为空");

}

}catch(e){ //程序运行中发生错误，系统自动创建Error对象并赋值给e

alert(e.message);

}

}

//addEvent

//ajax

function ajax(param){

try{

if(typeof param=="object"&&param!==undefined){

//执行ajax过程，分为8种情况

//1.创建ajax对象

if(window.XMLHttpRequest){

var xmlHttp=new XMLHttpRequest();

}else{

var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");

}

//全局事件，ajax请求结束以后触发

xmlHttp.onloadend=function(){

if(param.completed){

param.completed();

}

}

//2.设置请求方法和地址

//2.1 get带参数请求：url?变量=值&变量=值

//2.2 其他情况

if(param.type=="GET"&&param.data!=undefined){

xmlHttp.open(param.type,param.url+"?"+param.data);

}else{

xmlHttp.open(param.type,param.url);

}

//3.设置请求数据的编码

if(param.data!=undefined){

switch(param.contentType){

case "urlencoded":

xmlHttp.setRequestHeader("content-type","application/x-www-form-urlencoded");

break;

case "json":

xmlHttp.setRequestHeader("content-type","application/json");

}

}

//4.监控ajax请求过程

xmlHttp.onreadystatechange=function(){

if(xmlHttp.readyState==4&&xmlHttp.status==200){

//xmlHttp.onload数据全部返回

var data=xmlHttp.responseText;

if(param.dataType=="json"){

data=JSON.parse(data);

}

param.success(data);

}

}

//5.发送请求

//5.1 post带参数请求

//5.2 其他情况

if(param.type=="POST"&&param.data!=undefined){

xmlHttp.send(param.data);

}else{

xmlHttp.send();

}

}else{

throw new Error("参数不正确！");

}

}catch(e){

alert(e.message);

}

}

//ajax

$.addEvent=addEvent;

$.ajax=ajax;

window.$=$;

console.log($);

})();

patient.php：

<?php

echo '{

"status":"OK",

"startTime":1434091059460,

"endTime":1434091059739,

"version":"1.0",

"msg":"操作成功",

"data":[

{"id":"980001","patCardNum":"201809019981","patId":1020,"patName":"李亮亮","feeId":"C","natId":"D","patSex":"男","patAge":19,"regDepartId":"201809019981","regDoctorId":"401","regVisitedState":1,"regFirstSign":0,"regNm":0},

{"id":"180002","patCardNum":"201709019581","patId":1220,"patName":"王亮亮","feeId":"C","natId":"D","patSex":"男","patAge":22,"regDepartId":"201809019981","regDoctorId":"401","regVisitedState":0,"regFirstSign":0,"regNm":0},

{"id":"340021","patCardNum":"201609013381","patId":1040,"patName":"赵亮亮","feeId":"C","natId":"D","patSex":"男","patAge":23,"regDepartId":"201809019981","regDoctorId":"401","regVisitedState":0,"regFirstSign":0,"regNm":0},

{"id":"528021","patCardNum":"201509012981","patId":3021,"patName":"刘亮亮","feeId":"C","natId":"D","patSex":"男","patAge":22,"regDepartId":"201809019981","regDoctorId":"401","regVisitedState":2,"regFirstSign":0,"regNm":0},

{"id":"433214","patCardNum":"201409019983","patId":2021,"patName":"张亮亮","feeId":"C","natId":"D","patSex":"男","patAge":16,"regDepartId":"201809019981","regDoctorId":"401","regVisitedState":2,"regFirstSign":0,"regNm":0}

]

}';

?>

detail.php：

<?php

$patientId=$\_GET['patientId'];

switch($patientId){

case "980001":

echo '{

"id":"980001",

"patName":"李亮亮",

"patSex":"男",

"patBirthday":"1990/1/1",

"natureId":"9001",

"patIdentityNum":"130222332233444",

"patFamAddress":"上海市黄浦区XXX",

"patPostcode":"200001",

"patContactPhone":"1372228221",

"patContacts":"李大刚",

"patPhone":"132883772",

"patWorkUnit":"阿里巴巴",

"patOperation":"前端工程师",

"patCountry":"中国",

"patNationality":"汉族",

"patMatrimony":1,

"patHisPrevious":0,

"patHisAllergic":0,

"patHisFamily":0,

"patRecordDate":"2001/1/1",

"patMemGrade":"C",

"patCardNum":"201809019981",

"feeId":"C"

}';

break;

case "180002":

echo '{

"id":"180002",

"patName":"王亮亮",

"patSex":"男",

"patBirthday":"1990/1/1",

"natureId":"9001",

"patIdentityNum":"130222332233444",

"patFamAddress":"上海市黄浦区XXX",

"patPostcode":"200001",

"patContactPhone":"1372228221",

"patContacts":"李大刚",

"patPhone":"132883772",

"patWorkUnit":"阿里巴巴",

"patOperation":"前端工程师",

"patCountry":"中国",

"patNationality":"汉族",

"patMatrimony":1,

"patHisPrevious":0,

"patHisAllergic":0,

"patHisFamily":0,

"patRecordDate":"2001/1/1",

"patMemGrade":"C",

"patCardNum":"201809019981",

"feeId":"C"

}';

break;

}

?>

defaultdetail.php：

<?php

echo '{

"id":"980001",

"patName":"李亮亮",

"patSex":"男",

"patBirthday":"1990/1/1",

"natureId":"9001",

"patIdentityNum":"130222332233444",

"patFamAddress":"上海市黄浦区XXX",

"patPostcode":"200001",

"patContactPhone":"1372228221",

"patContacts":"李大刚",

"patPhone":"132883772",

"patWorkUnit":"阿里巴巴",

"patOperation":"前端工程师",

"patCountry":"中国",

"patNationality":"汉族",

"patMatrimony":1,

"patHisPrevious":0,

"patHisAllergic":0,

"patHisFamily":0,

"patRecordDate":"2001/1/1",

"patMemGrade":"C",

"patCardNum":"201809019981",

"feeId":"C"

}';

?>