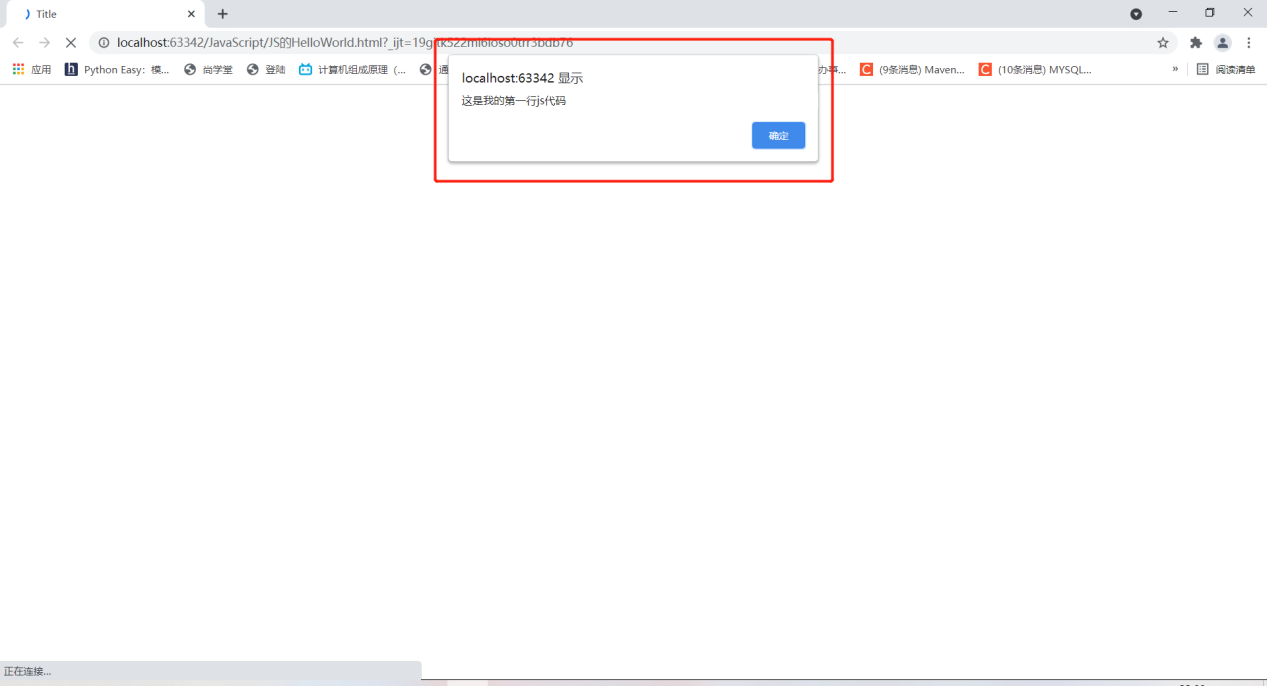
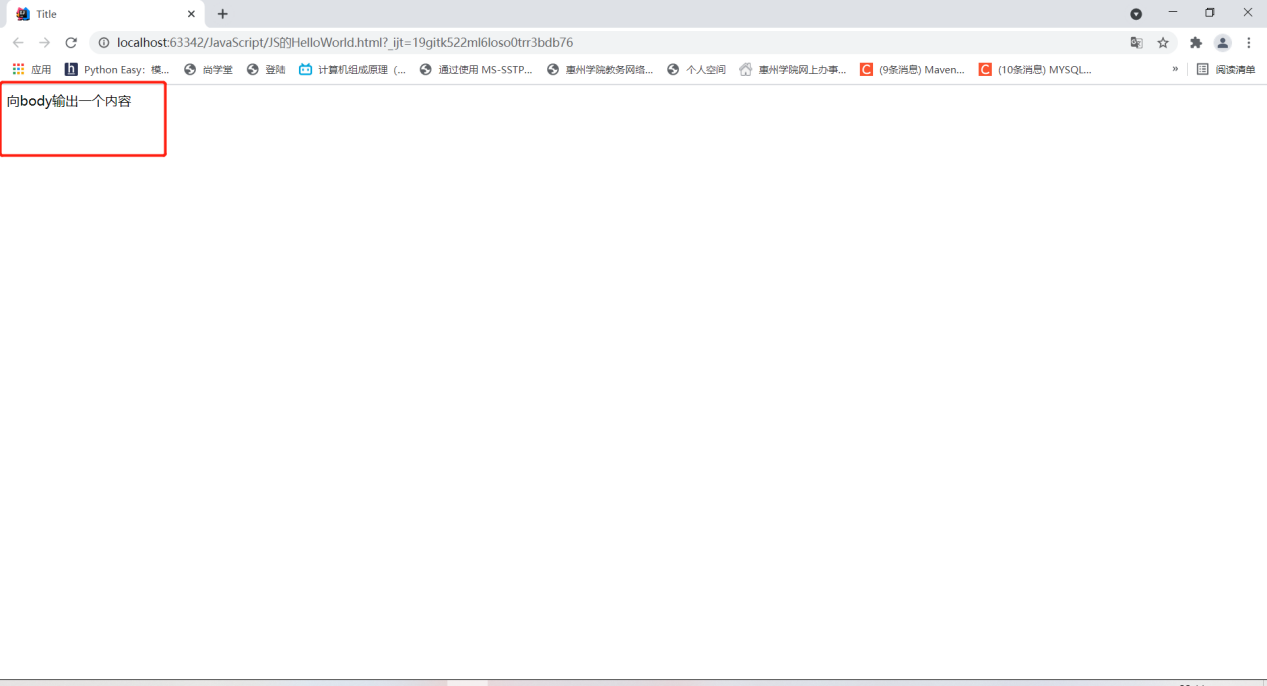
# JS的输出

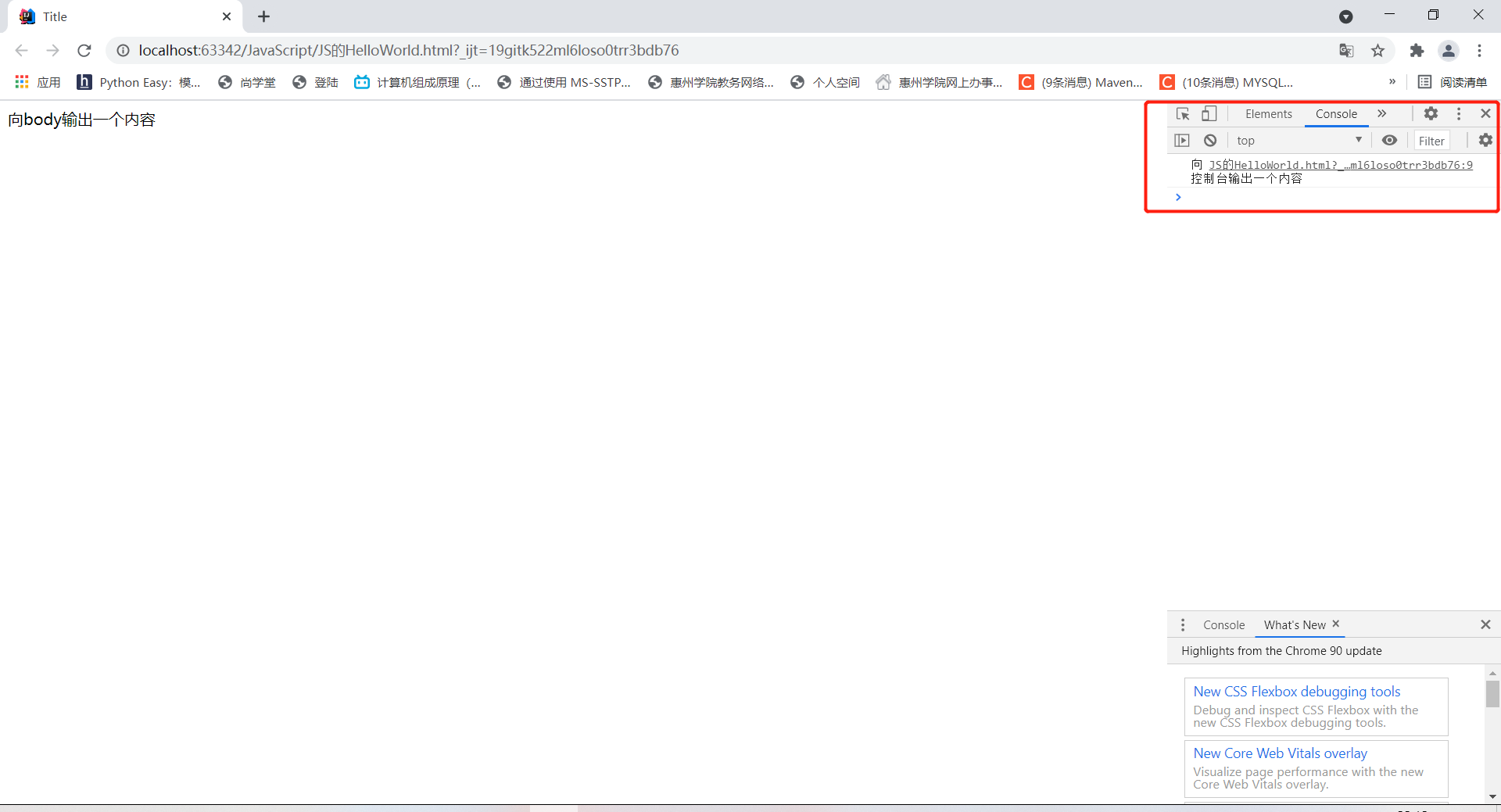
·alert("这是我的第一行js代码");//警告框



·document.write("向body输出一个内容");//输出



·console.log("向控制台输出一个内容");



<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<script type="text/javascript">

alert("这是我的第一行js代码");//警告框

document.write("向body输出一个内容");//输出

console.log("向控制台输出一个内容");

</script>

</head>

<body>

</body>

</html>

# JS编写位置

·将js代码编写到标签的onclick属性中

<button onclick="alert('你真棒')">点我一下</button>

属于结构与行为耦合，不方便维护，不推荐使用。

·将js代码写在超链接的href属性中

<a href="JavaScript:alert('让你点你就点')">你也点我一下</a>

·写在script标签中

<script type="text/javascript">

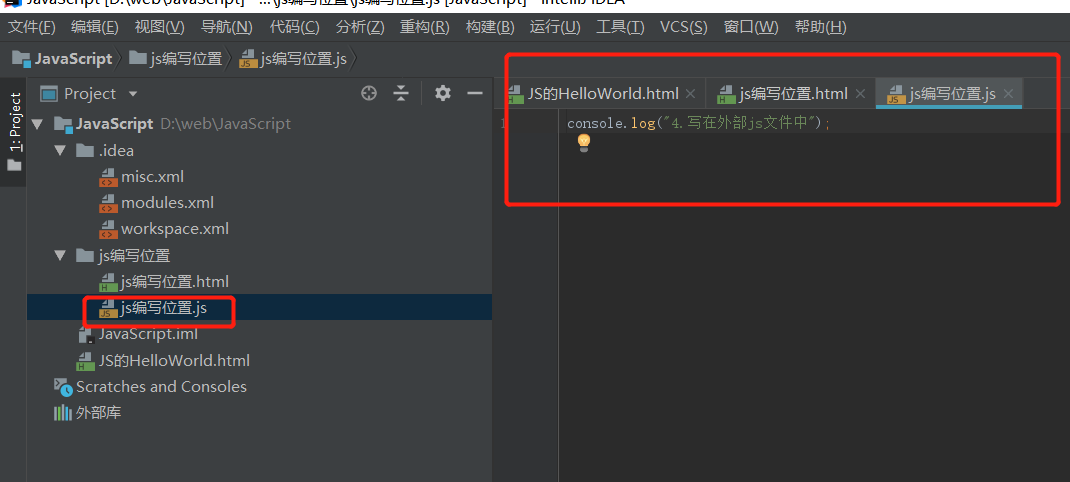
alert("也可以写在这里");

</script>

·写在外部js文件中

<!--引入外部js文件-->

<script type="text/javascript" src="js编写位置.js"></script>



写在外部文件中可以在不同的页面同时打开，也可以利用浏览器的缓存机制，推荐使用该方式。

script便签一旦引入外部文件，内部编写的代码就视为无效代码，如有需要可新建script标签。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<!--引入外部js文件-->

<script type="text/javascript" src="js编写位置.js"></script>

<!--3.写在script标签中-->

<script type="text/javascript">

alert("也可以写在这里");

</script>

</head>

<body>

<!--1.将js代码编写到标签的onclick属性中

属于结构与行为耦合，不方便维护，不推荐使用

-->

<button onclick="alert('你真棒')">点我一下</button>

<!--2.将js代码写在超链接的href属性中-->

<a href="JavaScript:alert('让你点你就点')">你也点我一下</a>

<a href="JavaScript:;">点了没反应</a>

</body>

</html>

# 基本语法

·单行注释//

//警告框

·多行注释/\* \*/

/\*

这是多行注释

\*/

·严格区分大小写

·每一条语句以 ; 结束

如果不写分号，浏览器会自动添加，但会消耗系统资源，而且有时候会加错分号，所以在开发中必须写分号。

·自动忽略多个空格和换行，可用来对代码进行格式化。

# 常量与变量

常量：不可改变的值

变量：用来保存常量，可任意改变其值

**var** a;//声明变量，未初始化的值为undefined

a = 123;//为变量赋值

**var** b = 456;//同时声明和赋值

# 标识符

·变量名、函数名、属性名

·遵守规则

- 可含有字母、数字、\_、$

- 不能以数字开头

- 不能是ES中的关键字或保留字

- 一般采用驼峰命名法

-- 首字母小写，每个单词首字母大写，其余字母小写：helloWorld。

JS底层保存标识符时实际上是采用的Unicode编码，理论上所有的utf-8中含有的内容都可以作为标识符，如中文，但是最好使用英文。

# 数据类型

JS中共有六种数据类型

·String 字符串

·Number 数值

·Boolean 布尔值

·Null 空值

·Undefined 未定义

·Object 对象

## 字符串

·使用双引号或者单引号引起来，不要混着用。

**var** str = "字符串";

·引号不能嵌套，双引号里面不能放双引号，单引号里面不能放单引号。

str = '我说："天气不错！"';

·两个字符串数值运算会返回NaN(Not a Number),表示不是数字

NaN是Number数据

## 数值

·整数和浮点数

**var** b = 456;

·Number.MAX\_VALUE返回数值最大值1.7976931348623157e+308

如果数值超过最大值，则会返回Infinity，表示正无穷

-Infinity表示负无穷

Infinity和-Infinity是Number数据

console.log(Number.MAX\_VALUE);

·Number.MIN\_VALUE返回数值最小值5e-324，表示0以上的最小值

console.log(Number.MIN\_VALUE);

·浮点数运算可能会得到不精确的结果，不要用JS进行高精准度运算。

## 布尔值

·true、false

**var** bool = **true**;

## null

·用来表示空对象，空值

bool = **null**;

## Undefined

·声明的变量未赋初值时，表示未定义

## typeof查看变量类型

typeof 变量

console.log(**typeof** str);

检查null值的变量时返回的是Object

# 转义字符\

当表示一些特殊符号时使用\进行转义。

str = "她说：\"天气不错！\"";

·\” 表示”

·\’ 表示’

·\n 换行

·\t Tab，制表符

·\\ 表示\

·\u 表示Unicode编码的字符，如”\u0054”

# 强制类型转换

·将一个数据类型强转转换为其他的数据类型

- 如将其他类型转换为String、Number、Boolean

## 其他进制的数值

·16进制以0x开头

- 输出时会自动转换为十进制数，如0xff --> 255

a = 0xff;

·8进制以0开头，如070 --> 56

a = 070;

·2进制以0b开头，但不是所有浏览器都支持，如0b10 --> 2

a = 0b10;

## 转换为String类型

以下实例a为变量

·调用被转换数据类型的toString()方法

- 该方法不会改变原变量的数据类型，只是返回String类型的数据

- 注意：null和undefined这两个值没有toString()方法，调用会报错。

a.toString();

·使用String函数，null和undefined可转换不会报错

String(a);

·任意数据类型+空串””，实际上也是调用了String()函数

a = a+””;

## 转换为Number类型

·使用Number()函数

- 字符串转数值

-- 如果是纯数值的字符串，则直接将其转换为数字

-- 如果字符串中有非数值的内容，则转换为NaN

-- 如果字符串是一个空串或者全是空格的字符，则转换为0

- 布尔转数值

-- true 转成 1

-- false 转成 0

- null转数值 转成 0

- undefined转数值 转成 NaN

Number(a);

·使用parseInt()和parseFloat()函数

- 专门转换混合字符串

- 其他的和Number()函数一样

- parseInt()字符串转整数 “123.45px6” --> 123

- parseFloat()字符串转浮点数 “123.4.5px6” --> 123.4

- “070”这种字符串，有些浏览器会当成8进制解析，有些会当成十进制解析。可传递第二参数，指定数值的进制。

parseInt(a);

parseFloat(a);

parseInt(a,10);

·任意数据类型 -0 、\*1 、+a 或 /1，实际上也是调用了Number()函数

a = a - 0;

a = +a;

## 转换为Boolean类型

·使用Boolean()函数

- 除了0、NaN、null、undefined和空字符串””，其余全是true

Boolean(a)

·任意数据类型进行两次取反!!

a = !!a;

# 运算符

·typeof 获得一个值的类型

- 他会将该值的类型以字符串形式返回

## 算数运算符

属于二元运算符

·+

·-

·\*

·/

·%

- 对非Number类型的值进行运算时，会先转为Number再运算。

- 但是当两个字符串进行加法运算时，则会拼接字符串。如”123”+”456” --> ”123456”

- 任何值和字符串做加法运算，都会先转换成字符串，再拼接字符串,可用来转换String数据类型。如123+”1” --> “1231”

- 任何值和NaN运算都得NaN

- 1 + 2 + “3”的结果为”33”

- “1” + 2 + 3的结果为”123”

- 任何值做 - 、\* 、/ 运算都会转为Number类型

## 一元运算符

只需要一个操作数

·+ 正号

·- 负号

- 对于非Number类型的值，先转换为Number再运算

- 1 + +”2” + 3 --> 6

·++ 自增

·-- 自减

- a++为自增前的值，++以后才会改变a的值。

- ++a为自增后的值，立即改变a的值。

d = 1

d++ + ++d + d = 1 + 3 + 3 = 7

## 逻辑运算符

· ！

-对于非Boolean类型的值，先转换为Boolean再运算

· &&

- 短路与,如果第一个为false，则不会检查第二个值

· ||

- 短路或,如果第一个为true，则不会检查第二个值

- 对于非Boolean类型的值，&&和||运算会先转换为Boolean再运算，然后再返回原值。

- &&如果两个都为true,则返回后边的。如5 && 6 --> 6

- &&如果两个都为false,则返回前边的。如NaN && 0 --> NaN

- ||如果两个都为false,则返回后边的。如NaN || 0 --> 0

- ||如果两个都为true,则返回前边的。如5 || 6 --> 5

## 赋值运算符

·=

·+=

·-=

·\*=

·/=

·%=

## 关系运算符

比较两个值间的大小关系,返回true/false

·>

·>=

·<

·<=

- 非数值进行比较时，会将其转换成数值，再进行比较

- 如果符号两侧都是字符串，不会将其转换成数值进行比较，而会分别比较各字符的Unicode编码，如果两位一样，则比较下一位。借用它进行英文排序，比较中文没有意义。”11” < “5” //true

- 在比较两字符串数字时，一定要转型。”11” < +”5” //false

- 任何值和NaN比较都是false

·==

- 只判断值是否相同，如果数据类型不同，则会转换成相同的进行比较

”1” == 1 //true

- undefined 衍生自 null,两值判断返回true。Undefined == null //true

- NaN不和任何值相等，包括它本身。可通过isNaN()函数判断b是否为NaN

·===

- 判断值和数据类型是否相同，类型不同直接返回false

·!=

- 只判断值是否相同，如果数据类型不同，则会转换成相同的进行比较

”1” != 1 //false

·!==

- 判断值和数据类型是否不同，类型不同直接返回true

## 条件运算符（三元运算符）

·条件表达式?语句1:语句2;

- 条件句为非布尔值，会将其转换成布尔值，再进行判断

## ,运算符

,可分割多个语句，一般可以在声明多个变量时使用

var a , b , c;

## 运算符优先级

·. 、[] 、new

·()

·++ 、 --

·! 、~ 、 + (单目) 、- (单目)、typeof 、void、delete

·%、\*、/

·+ (双目)、- (双目)

·<< 、>> 、>>>

·< 、<= 、> 、>=

·== 、!= 、!== 、===

·&

·^

·|

·&&

·||

·?:

·= 、+= 、-= 、\*= 、/= 、%= 、<<= 、>>= 、>>>= 、&= 、^= 、|=

·,

表并不需要背，如果优先级不清楚，可以用()改变优先级。

# 代码块

使用{}为语句分组

·同一个{}中的语句我们称为是一组语句（代码块），要么都执行，要么都不执行。

{

alert("这是我的第一行js代码");//警告框

document.write("向body输出一个内容");//输出

console.log("向控制台输出一个内容");

}

·代码块内部的内容在外部是完全可见的

# 流程控制语句

通过流程控制语句可以控制程序执行流程，使程序根据一定条件执行。

·**条件判断语句 if语句**

- if(){}

- if(){}else{}

- if(){}else if(){}else{}

·**条件分支语句 switch语句**

a = 1;

**switch** (a) {

**case** 1:

console.log("1");

**break**;

**case** 2:

console.log("2");

**break**;

**case** 3:

console.log("3");

**break**;

**default**:

console.log("其他");

**break**;

}

·**循环语句**

- while循环

**while** (**true**){

}

}

-do{}while()

**do** {

}**while** (a < 10);

- for循环

for循环的三个部分都可以省略

如果3个部分都不写，则是死循环

**for** (**var** i=0;i<10;i++){

- 嵌套for循环

**for** (**var** i=0;i<10;i++){

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

}

}

# break和continue

break关键词可以用来跳出switch或循环语句

不能在if语句中使用break和continue

·break

立即终止离它最近的循环语句

可以为循环语句创建label，来标识当前循环

- label:循环语句

- 使用break时，可在其后加上label，这样将会结束指定的循环，而不是最近的。

outer:

**for** (**var** i=0;i<10;i++){

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

**break** outer;

}

}

·continue

跳出当次离它最近的循环，继续下一次循环。

# 对象的简介

对象属于一种复合的数据类型，在对象中可以保存多个不同数据类型的属性。

对象的分类：

·内建对象

- 由ES标准中定义的对象，在任何的ES的实现中都可以使用，如：Math、String、Number等。

·宿主对象

- 由JS运行环境提供的对象，目前来讲主要指由浏览器提供的对象。比如BOM、DOM。

·自建对象

- 由开发人员自己创建的对象。

# 对象的基本操作

·创建对象

- 使用new关键字调用的函数，是构造函数constructor

**var** obj = **new** Object();

- 使用对象字面量

**var** obj = {};

obj.name = "孙大圣";

使用对象字面量，可以在创建对象时，直接指定对象中的属性。

**var** obj = {

name:"猪八戒",

age:28,

gender:"男",

gege:{name:"孙悟空"}

};

对象字面量的属性名可以加引号，也可以不加，建议不加。如果要使用一些特殊名字，则必须加引号。

·向对象添加属性

在对象中保存的值称为属性

obj.name = "孙悟空";//向obj中添加一个name属性

·读取对象中的属性

console.log(obj.name);

- 如果读取对象中没有属性，不会报错，而是会返回undefined

·修改对象的属性值

obj.name = "孙猴子";

·删除对象的属性

**delete** obj.name;

# 属性名和属性值

·属性名

- 对象的属性名不强制要求遵守标识符的命名规范，但是最好遵守。

- 如果要使用特殊的属性名，不能采用.的方式来操作，需要使用

obj["123"] = 789;

console.log(obj["123"]);

- 使用[]这种形式操作属性更加灵活，在[]中可以直接传递一个变量，这样变量值是多少就会读取那个属性。

obj["123"] = 789;

**var** n = "123"

console.log(obj["n"]);//输出789

·属性值

- 可以是任意的数据类型，甚至也可以是一个对象（包括函数）。

**var** obj = **new** Object();

obj.name = "孙悟空";//向obj中添加一个name属性

**var** obj2 = **new** Object();

obj2.name = "猪八戒";

obj2.gege = obj;

·in 运算符

- 检查一个对象是否含有指定的属性

console.log("age" **in** obj);

# 基本数据类型和引用数据类型

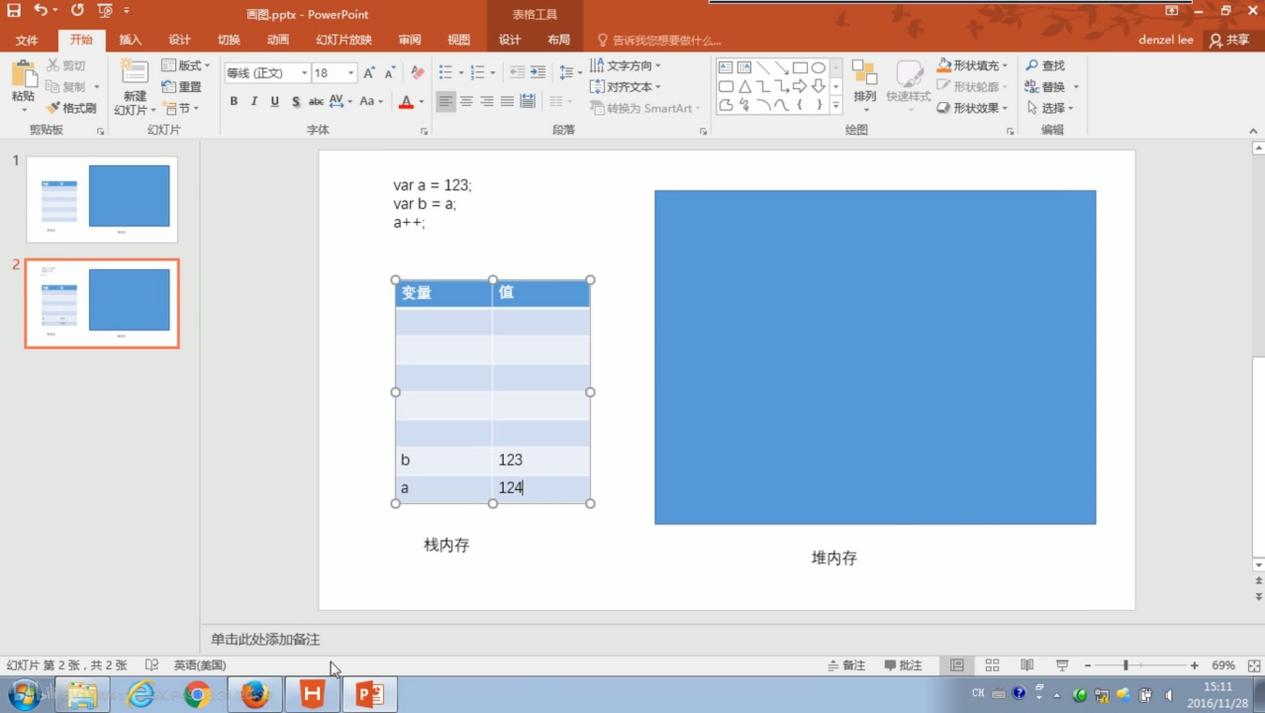
·Object为引用数据类型，其余为基本数据类型。

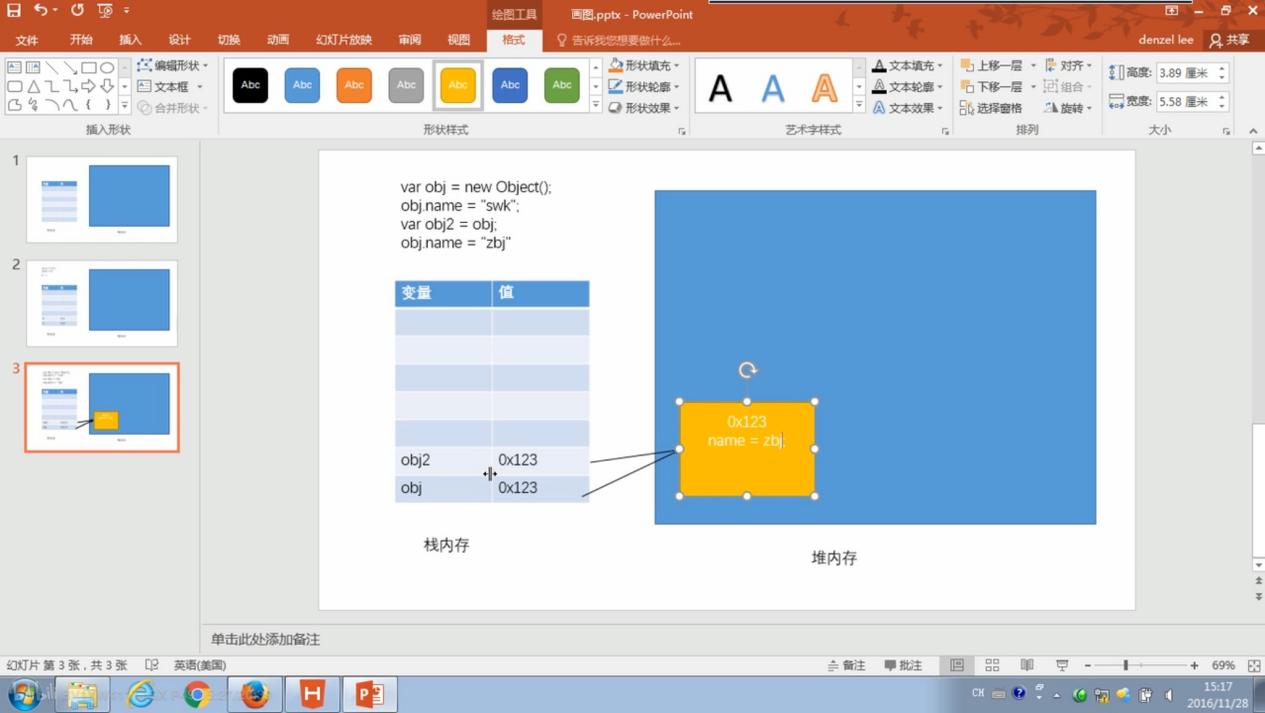
·变量保存在栈内存，基本数据类型直接在栈内存中存储，值与值之间是独立存在，修改一个变量不会影响其他变量。

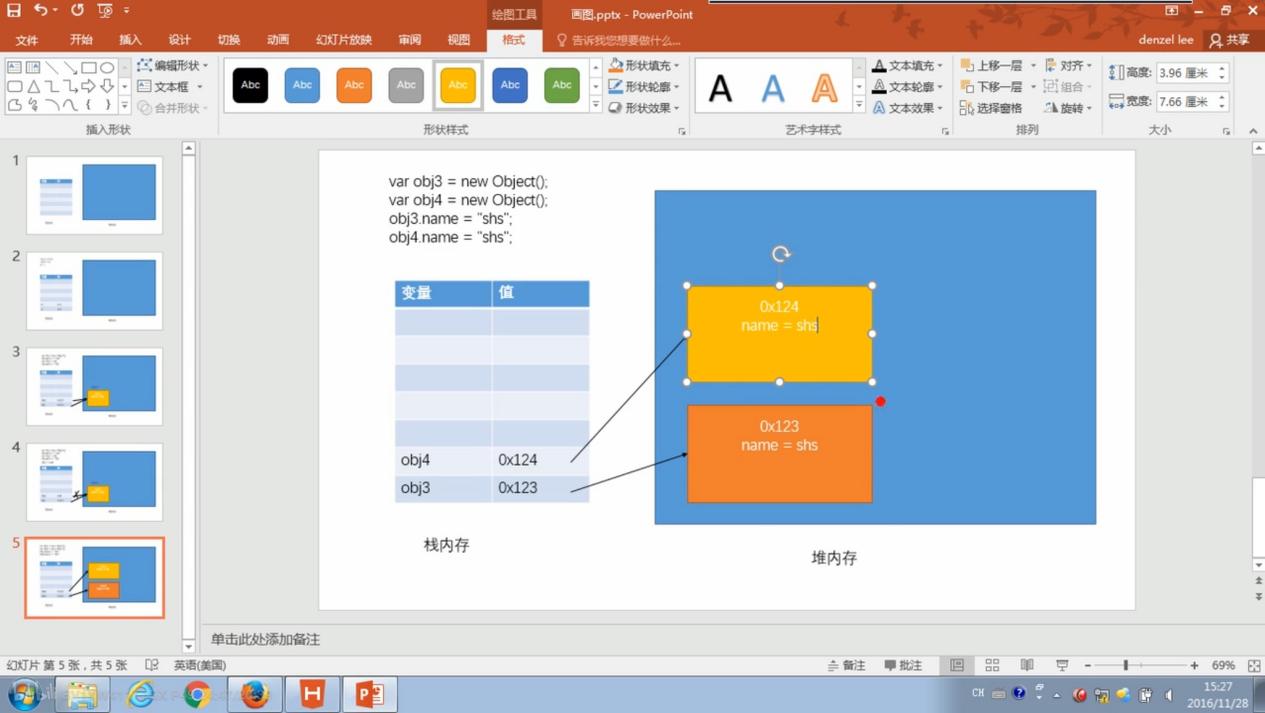
·对象是保存在堆内存中的，每创建一个新的对象，就会在堆内存中开辟新的空间，而变量保存的是对象的内存地址（对象的引用）。如果两个变量保存的是同一个对象的引用，当一个对象通过一个变量修改属性时，另一个变量也会受到影响。

·当比较两个基本数据类型时，就是比较值

·当比较两个引用数据类型时，它比较的是对象的内存地址，就算两个对象的属性值一样，但是地址不同，它也会返回false。







# 函数

·函数也是一个对象

·函数中可以封装一些功能（代码），在需要时可以调用这些功能（代码）。

·使用typeof检查一个函数对象时，会返回function

## 创建函数

·创建一个函数对象

- 函数对象具有普通对象的所有功能

**var** fun = **new** Function("console.log('这是第一个函数');");

fun();//调用函数

·使用函数声明来创建一个函数

- 调用函数时解析器不会检查实参的类型，要注意是否有可能接收到非法参数，如果有可能，则需要检查实参类型。

- 调用函数时，解析器也不会检查实参数量，多余实参不会被赋值。如果实参的数量少于形参的数量，则没有对应实参的形参将时undefined。

- 函数的实参可以是任意的数据类型。

**function** add(a,b) {

console.log(a+b);

}

add(1,2);

·使用函数表达式创建一个函数

**var** fun3 = **function**() {

console.log("我是匿名函数中封装的代码");

};

fun3();

## 函数返回值

·使用return设置返回值

**function** sum(a,b,c) {

**return** a+b+c;

}

console.log(3+sum(1,2,3));

·return后的语句都不会执行

·return;相当于return undefined;,不写return也是返回undefined。

·return可以返回任意类型的值，也可以是一个对象和函数。

·使用retuen可以跳出整个函数

## 实参

·函数的实参可以是任意的数据类型

- 实参可以是一个对象。

当我们的参数过多时，可以将参数封装到一个对象中，然后通过对象传递。

**var** role = {

name:"孙悟空",

age:18,

gender:"男",

address:"花果山"

};

**function** resume(role){

console.log(role.name+","+role.age+"岁"+","+role.gender+","+role.address);

}

resume(role);

- 实参也可以是一个函数

**var** role = {

name:"孙悟空",

age:18,

gender:"男",

address:"花果山"

};

**function** resume(role){

console.log(role.name+","+role.age+"岁"+","+role.gender+","+role.address);

}

**function** resume2(a) {

a(role);

}

resume2(resume);

## 立即执行函数

函数定义完立即被调用，往往只会执行一次。

(**function** () {

console.log("我是立即执行函数")

})();

(**function** (a,b) {

console.log("我是立即执行函数"+(a+b));

})(1,3);

# 方法

·对象的属性值可以是任意数据类型（包括函数）

**var** role = {

name:"孙悟空",

age:18,

gender:"男",

address:"花果山"

};

**function** resume(role){

console.log(role.name+","+role.age+"岁"+","+role.gender+","+role.address);

}

role.say = resume;

role.say(role);

**var** role = {

name:"孙悟空",

age:18,

gender:"男",

address:"花果山",

sayname:**function** () {

console.log(role.name);

}

};

·如果一个函数作为一个对象的属性保存，那么我们称这个函数时称为这个对象的方法，调用函数就说调用对象的方法。

# 枚举对象中的属性

使用for in语句

**var** role = {

name:"孙悟空",

age:18,

gender:"男",

address:"花果山",

sayname:**function** () {

console.log(role.name);

}

};

**for** (**var** n **in** role){

console.log(n+":"+role[n]);//输出属性名和属性值

}

# 作用域

作用域指一个变量作用的范围。

## 全局作用域

·直接编写在script标签中的JS代码，都在全局作用域。

·全局作用域在页面打开时创建，页面关闭时销毁。

·在全局作用域中，有全局对象window，它由浏览器创建，我们可以直接使用。

·在全局作用域中，创建的变量都会作为window对象的属性保存。创建的函数都会作为window对象的方法保存。可以不声明变量之间使用。

如a = 100相当于window.a = 100;

最好使用var声明，否则未声明直接使用可能会报错，如未声明xxx直接使用：console.log(xxx);

·变量的声明提前

- 使用var关键字声明的变量，会在所有的代码执行之前被声明，但是不会赋值。

- 如果声明变量时不使用var关键字，则变量不会被声明提前。

·函数声明提前

- 使用函数声明形式创建的函数function 函数(){}，他会在所有的代码执行之前被创建。所有可以在函数声明前调用函数。

- 使用函数表达式创建的函数，不会被声明提前，所以不能在声明前调用。

·全局作用域中的变量都是全局变量，可在任意部分被使用。

## 函数作用域

·调用函数时创建函数作用域，函数执行完毕后，函数作用域销毁。

·每调用一次函数就会创建一个新的函数作用域，他们之间是相互独立的。

·在函数作用域中可以访问全局变量，在全局作用域中无法访问函数作用域的变量。

·当在函数作用域中操作一个变量时，他会先在自身作用域中寻找，如果有就直接使用，如果没有，则向上一级作用域中寻找。

·在函数中访问全局变量可以使用window对象，如window.a；

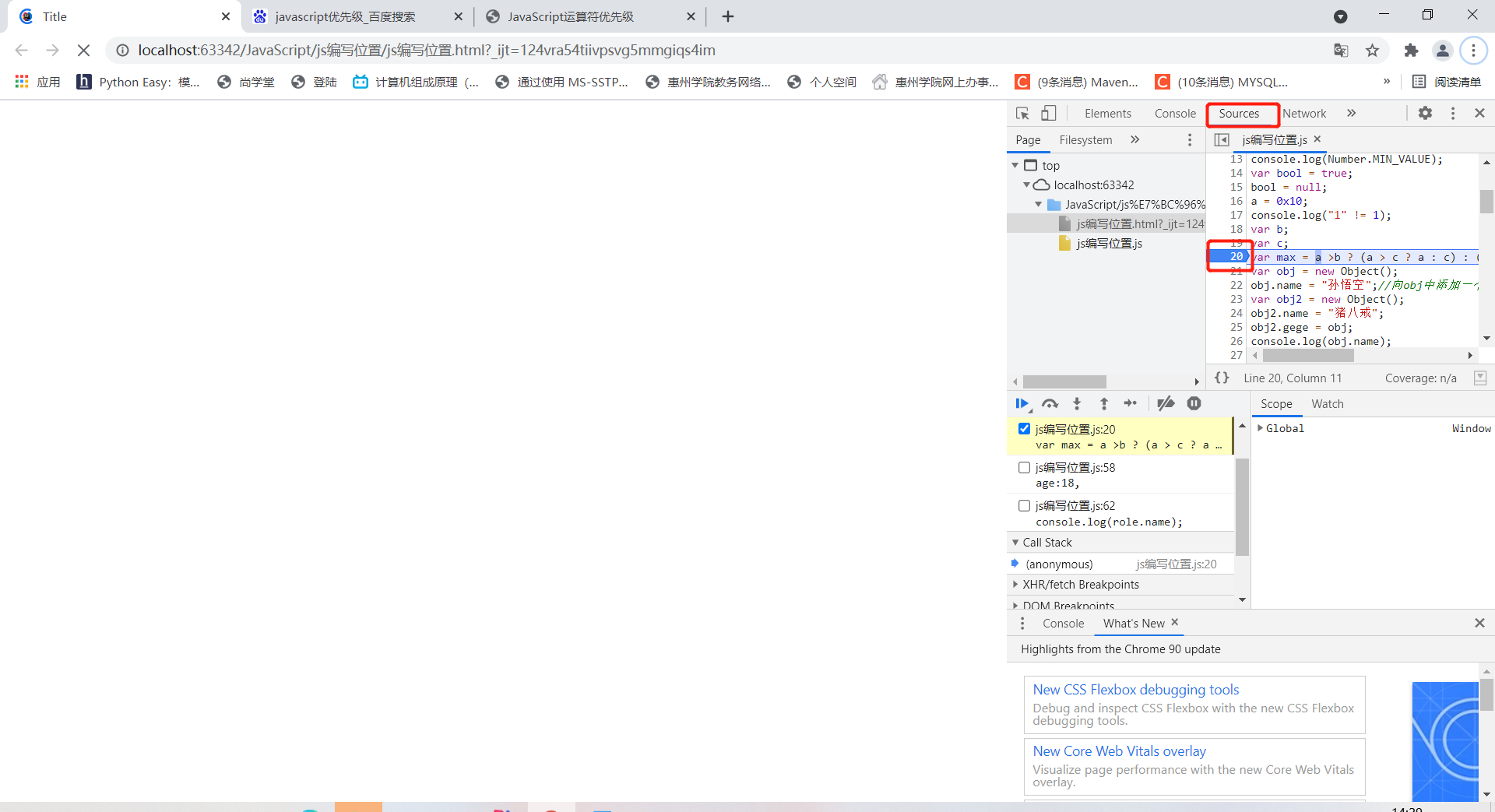
·在函数作用域中也有声明提前的特性。

·在函数中，不使用var声明的变量都会成为上一级变量，会改变上一级变量的值。

·定义形参就相当于声明了变量。

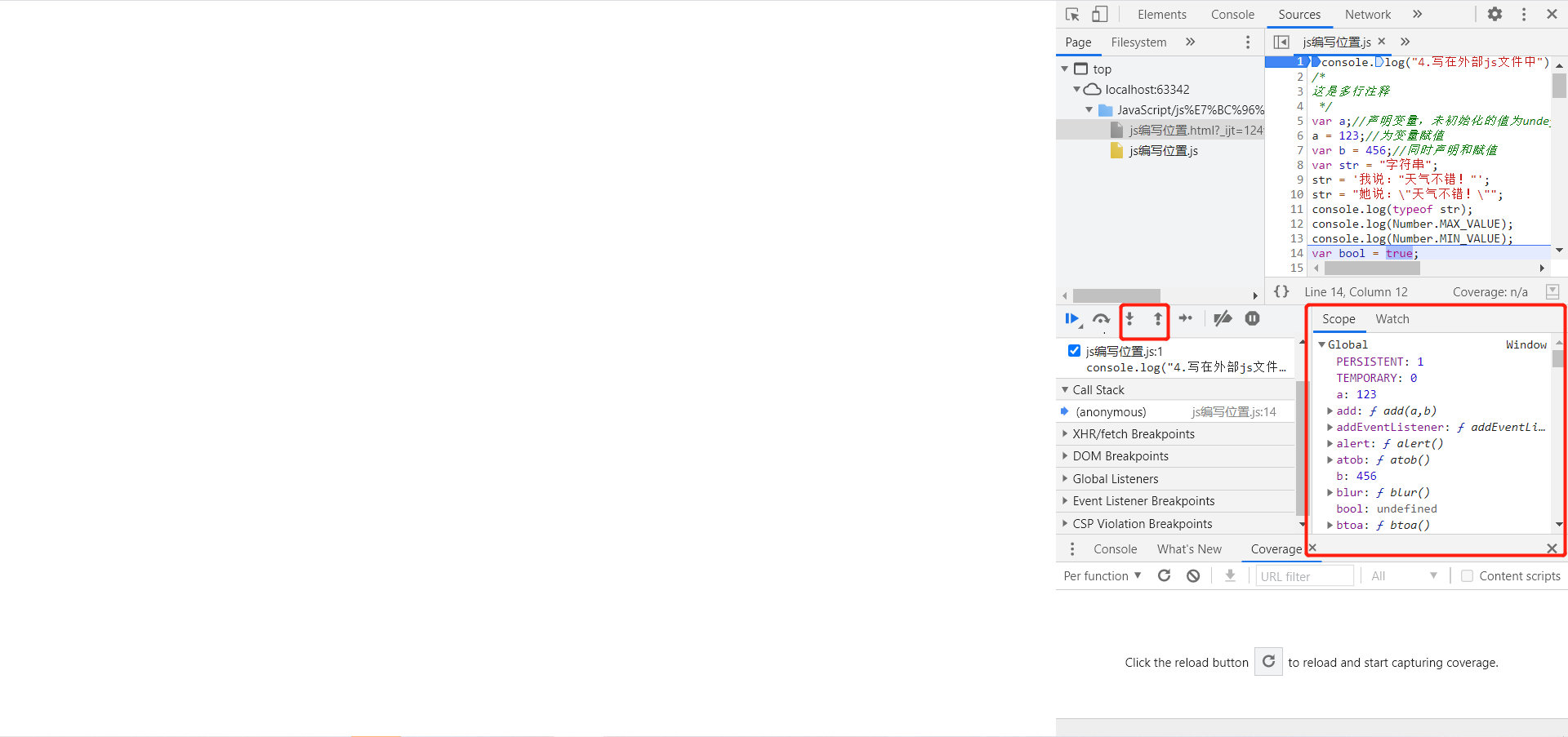
# debug

断点



然后刷新一下

上一步和下一步，查看变量值



# this

解析器在调用函数每次都会向函数内部传递进一个隐含的参数，这个隐含的参数就是this。

this指向的是一个对象，即函数执行的上下文对象。

根据函数的调用方式不同，this会指向不同的对象。

1. 以函数形式调用，this指向window。即fun();
2. 以方法形式调用，this指向该方法的对象。

# 创建对象的高级方法

·使用工厂方法创建对象

使用工厂方法创建的对象，使用的构造函数都是Object，所以创建的对象都是Object数据类型，无法区分多种不同类型的对象。

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

//创建一个新对象

**var** obj = **new** Object();

//向对象添加属性

obj.name = name;

obj.age = age;

obj.gender = gender;

obj.sayName = **function** () {

console.log(**this**.name);

};

//将新的对象返回

**return** obj;

}

**var** obj2 = createPerson("猪八戒",18,"男");

**var** obj3 = createPerson("孙悟空",19,"男");

obj2.sayName();

obj3.sayName();

·构造函数

- 构造函数习惯首字母大写

- 构造函数和普通函数的区别就是调用方式的不同，普通函数是直接调用，而构造函数需要通过new关键字调用。

- 构造函数的执行流程

-- 立刻创建新的对象

-- 将新建的对象设置为函数中的this，在构造函数中可以使用this来引用新建的对象。

-- 逐行执行函数中的代码

-- 将新建的对象作为返回值返回

- 使用一个构造函数创建的对象，称为一类对象，也将一个构造函数称为一个类。

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

**this**.name = name;

**this**.age = age;

**this**.gender = gender;

**this**.sayName = **function** () {

console.log(**this**.name);

};

}

**var** per = **new** Person("张三",18,"男");

**var** per1 = **new** Person("貂蝉",17,"女");

per.sayName();

per1.sayName();

- 将通过一个构造函数创建的对象，称为该类的实例。

- 使用instanceof可以检查一个对象是否是一个类的实例。

console.log(per **instanceof** Person);

- 所有的对象都是Object的后代，都是Object的实例。

- 缺点：每创建一个实例，就会创建一个新的方法，消耗内存。

·构造函数的优化

- 缺点：将函数定义在全局作用域中，会污染全局作用域的命名空间。定义在全局作用域中也很不安全。

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

**this**.name = name;

**this**.age = age;

**this**.gender = gender;

**this**.sayName = fun;

}

**function** fun() {

console.log(**this**.name);

};

**var** per = **new** Person("张三",18,"男");

**var** per1 = **new** Person("貂蝉",17,"女");

per.sayName();

per1.sayName();

console.log(per **instanceof** Person);

# 原型对象

原型 prototype

- 我们所创建的每一个函数，解析器都会向函数添加一个属性prototype。如果函数作为普通函数调用，prototype没有任何作用

- 当以构造函数形式调用时，它所创建的对象中都会有一个隐含的属性，指向该构造函数的原型对象，我们可以通过\_\_proto\_\_来访问该属性。

console.log(per.\_\_proto\_\_);

·原型对象就相当于一个公共的区域，所有同一个类的实例都可以访问到这个原型对象。我们可以将对象中共有的内容，统一设置到原型对象中。

//向Person类的原型中添加属性a

Person.prototype.a = 123;

console.log(per.a);//输出123

·当我们访问对象的一个属性或方法时，它会先在对象自身中寻找。如果有则直接使用，否则会在原型对象中寻找，如果找到直接使用。

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

**this**.name = name;

**this**.age = age;

**this**.gender = gender;

}

// 向原型中添加sayName方法

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

console.log("大家好我是：" + **this**.name);

};

**var** per = **new** Person("张三",18,"男");

**var** per1 = **new** Person("貂蝉",17,"女");

per.sayName();

per1.sayName();

·创建构造函数时，可以将这些对象共有的属性和方法，统一添加到构造函数的原型对象中，这样不用分别为每个对象添加，也不会影响到全局作用域，就可以使每个对象都具有这些属性和方法。

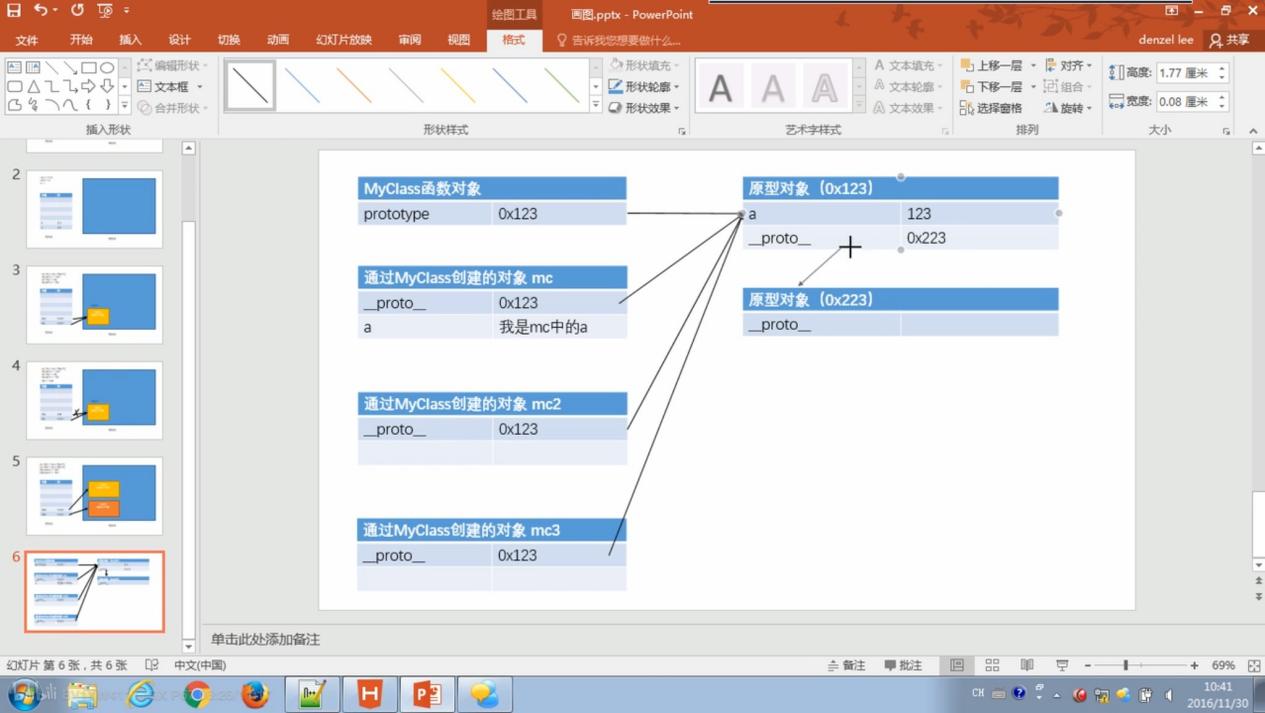
·使用in检查对象中是否含有某个属性时，如果对象中没有但是原型中有，也会返回true。

·可以使用对象的hasOwnProperty()来检查对象自身中是否含有该属性。该方法存储在对象的原型里。

console.log(per.hasOwnProperty("sayName"));

·原型对象它也是对象，它也有原型。当我们使用一个对象的属性或方法时，会先在自身中寻找，自身中没有，则在原型对象中找，原型中没有，则去原型的原型中找。

·Object对象的原型没有原型对象。



# toString()

·当我们直接在页面中打印一个对象时，实际上是输出该对象toString()方法的返回值。

·可重写toString()方法

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

**this**.name = name;

**this**.age = age;

**this**.gender = gender;

}

// 向原型中添加sayName方法

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

console.log("大家好我是：" + **this**.name);

};

**var** per = **new** Person("张三",18,"男");

**var** per2 = **new** Person("貂蝉",18,"女");

//重写toString方法

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

**return this**.name+**this**.age+**this**.gender;

};

console.log(per.toString());

console.log(per2.toString());

# 垃圾回收机制(GC)

·处理程序运行过程中产生的垃圾

·当一个对象没有任何的变量或属性对它进行引用，此时我们将永远无法操作该对象，这种对象称为垃圾，会占用大量的内存空间，导致程序运行变慢。

·JS中拥有自动的垃圾回收机制，会自动将这些垃圾对象从内存中销毁，我们不需要也不能手动进行垃圾回收操作。

·我们需要做的只是将不再使用的对象设置为null即可。

per2 = **null**;

# 数组(Array)

·数组也是一个对象，它和普通对象功能类似，也是用来存储一些值的。不同的是普通对象是使用字符串作为属性名的，而数组是使用数字来作为索引操作元素。

·数组的存储性能比普通对象好，在开发中我们经常使用数组来存储一些数据。

## 数组的使用

·创建数组

- 使用new关键字创建数组对象

**var** arr = **new** Array();//创建数组对象

**var** arr1 = **new** Array(10,20,30);//创建数组对象,有10，20，30三个元素

**var** arr2 = **new** Array(10);//创建长度为10的数组

- 使用字面量创建数组

**var** arr = [];

**var** arr1 = [1,2,3,4,5];

console.log(arr1);

·向数组中添加元素

arr[0] = 10;

·数组中的元素可以是任意数据类型,以下的per为Person对象

**var** arr2 = ["hello",1,**true**,**null**,**undefined**,per];

console.log(arr2);

console.log(arr2[5].name);

**var** arr3 = [{name:"张三"},{name:"李四"},{name:"王五"}];

console.log(arr3[1].name);

//数组存放两个函数并调用

**var** arr4 = [**function** () {

console.log("1");

},

**function** () {

console.log("2");

}];

arr4[1]();

//数组中存放数组，称为二维数组

**var** arr5 = [[1,2,3],[4,5,6],[7,8,9]];

console.log(arr5[1]);

·读取数组中的元素

console.log(arr[0]);

- 如果读取不存在的索引，它不会报错，而是返回undefined

·获取数组长度

console.log(arr.length);

- 该方法实际上是获取到数组的最大索引+1。

- 尽量不要创建非连续的数组，未赋值的中间元素都会变为空。

·修改数组长度

arr.length = 5;

- 修改后大于原长度，多出来的元素是空值。

- 修改后小于原长度，多出来的元素会被删除。

·向数组的最后一位添加元素

arr[arr.length] = 70;

## 数组的方法

·push()向数组末尾添加一个或多个元素，并返回数组的新长度

arr2.push("唐僧","蜘蛛精");

·pop()删除并返回数组的最后一个元素

arr2.pop();

·unshift()向数组开头添加一个或多个元素，并返回数组的新长度

arr2.unshift("貂蝉","吕布");

·shift()删除并返回数组的第一个元素

arr2.shift();

·slice()从数组中返回指定元素

- 该方法不会改变原数组，包含开始索引，不包含结束索引。

- 可以只传递开始索引，相当于开始后面的元素全都截取。

- 索引可以传递赋值，表示倒数第几位。

console.log(arr2.slice(1,3));//前闭后开

·splice()删除元素，并向数组添加新元素

- 删除指定元素，并返回删除的元素，会影响到原数组

- 第一个参数表示开始位置索引，第二个参数表示删除的数量

arr2.splice(1,3);

- 将新元素插入到起始位置。

arr2.splice(1,3,"替换1","替换2");

·concat()拼接多个数组，并返回结果，不会对原数组产生影响。

**var** arr6 = arr.concat(arr2,arr3,arr4,arr5);//arr也会拼接，并不会改变arr原来的值

console.log(arr6);

- 不仅可以传数组，还可以传元素

**var** arr6 = arr.concat(arr2,arr3,arr4,arr5,"牛魔王","紫霞");//并不会改变arr原来的值

console.log(arr6);

·jion()把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。

- 该方法可以将一个数组转换为字符串，默认使用,号连接，不会对原数组产生影响。

console.log(arr2.join());

- 使用指定连接符

console.log(arr2.join(":"));

·reverse()反转数组

- 该方法会直接修改原数组

arr2.reverse();

console.log(arr2);

·sort()排序数组

- 该方法会直接修改原数组，默认按照unicode编码进行排序

- 对数字进行排序时可能会得到错误结果

**var** arr7 = ["b","d","a","c","z"];

arr7.sort();

console.log(arr7);

- 自定义排序规则。在sort中添加一个回调函数，来指定排序规则。回调函数中需要定义两个形参，用来排序。

数组中a在b前面

如果返回值大于0，则a,b交换位置

如果返回值小于或等于0，位置不变

**var** arr7 = [11,4,333,24,8];

arr7.sort(**function** (a, b) {

**if** (a > b){

**return** 1;

} **else if**(a < b){

**return** -1;

}**else** {

**return** 0;

}

});

console.log(arr7);

如果需要升序则返回a - b，降序b - a。

**var** arr7 = [11,4,333,24,8];

arr7.sort(**function** (a, b) {

**return** a - b;

});

console.log(arr7);

## 数组的遍历

·使用for循环遍历数组

**for** (**var** i=0;i<arr2.length;i++){

console.log(arr2[i]);

}

·使用forEach()遍历数组，只支持IE8以上的浏览器

- 需要一个函数作为参数

- 像这种函数，由我们创建但是不由我们调用的，称为回调函数

- 数组中有几个元素就会执行几次，每次执行时，浏览器会将遍历到的元素以实参的形式传递进来，我们可以定义形参读取这些内容。

- 浏览器会在回调函数中传递三个参数

-- 第一个参数是正在遍历的元素

-- 第二个参数是正在遍历的元素的索引

-- 第三个参数是正在遍历的数组

arr2.forEach(**function** (value,index,obj) {

console.log(value + "," + index + "," + obj);

});

# 函数的方法

·call()和apply()方法

- 当对函数调用call()和apply()方法都会执行函数

**function** fun4() {

alert("我是函数");

}

fun4();

fun4.call();

fun4.apply();

- 在调用call()和apply()可以将一个对象指定为第一个参数，即this表示传入的对象。

**var** obj4 = {name:"obj4"};

**function** fun4() {

alert(**this**.name);

}

fun4.call(obj4);

fun4.apply(obj4);

- call()方法可以将实参在对象之后依次传递

- apply()方法需要将参数封装到一个数组中传递

**var** obj = {name:"李四",age:12};

**var** fn2 = **function** (a,b) {

**this**.name = "张三";

**this**.age = a+b;

};

fn2.call(obj,10,7);

console.log(obj);

fn2.apply(obj,[10,8]);

console.log(obj);

# arguments

在调用函数时，浏览器每次都会传递进两个隐含的参数

·函数的上下文对象this

·封装实参的对象arguments

- arguments是一个类数组对象，它可以通过索引来操作数据，也可以获取长度

- 在调用函数时，我们所传递的实参都会在arguments中保存

**function** fun4(a,b) {

console.log(arguments[0]);//张三

}

fun4("张三","李四");

- 我们即使不定义形参，也可以通过arguments来使用实参，只不过麻烦。

- 它有callee属性，表示正在执行的函数的对象

**function** fun4(a,b) {

console.log(arguments.callee);

}

fun4("张三","李四");

# Date对象

Date对象为时间对象

·创建一个Date对象

- 如果直接使用构造函数创建一个Date对象，则会封装为当前代码执行的系统时间。

**var** d = **new** Date();

console.log(d);//返回当前时间

- 创建一个指定的时间对象，格式:月/日/年 时:分:秒

**var** d2 = **new** Date("12/03/2021 11:10:30");

console.log(d2);

## Date对象的方法

·getDate()获取当前日期对象是几号

console.log(d2.getDate());

·getDay()获取当前日期对象是周几，0~6，0表示周日

console.log(d2.getDay());

·getMonth()获取当前日期对象的月份，0~11,0表示1月

console.log(d2.getMonth());

·getFullYear()获取当前日期对象的年份

console.log(d2.getFullYear());

·getHours()获取当前日期对象的小时数,0~23,0表示0点

console.log(d2.getHours());

·getMinutes()获取当前日期对象的分钟数

console.log(d2.getMinutes());

·getSeconds()获取当前日期对象的秒数

console.log(d2.getSeconds());

·getMilliseconds()获取当前日期对象的毫秒数

console.log(d.getMilliseconds());

·getTime()获取从格林威治标准时间的1970年1月1日至今的毫秒数(时间戳)

- 计算机底层在保存时间时使用的都是时间戳

- 计算机会把系统时间转成标准时间再计算

console.log(d.getTime());

·Date.now()获取该行代码的时间戳

- 可利用时间戳测试代码的性能

console.log(Date.now());

# Math对象

Math和其他对象不同，它不是一个构造函数，它属于工具类，不用创建对象，它里面封装了数学运算相关的属性和方法。

·Math.PI返回圆周率

·Math.abs()计算一个数的绝对值

console.log(Math.abs(-23));

·Math.ceil()对一个数进行向上取整，1.2-->2

console.log(Math.ceil(1.2));

·Math.floor()对一个数进行向下取整，1.6-->1

·Math.round()对一个数进行四舍五入取整

·Math.random()生成0~1之间的随机数

console.log(Math.round(Math.random()\*10));//生成0~10的随机数，包含0和10

console.log(Math.round(Math.random()\*8)+2);//生成2~10的随机数，包含2和10

·Math.max()获取一组数中的最大值

console.log(Math.max(10,20,30,100,50,32));

·Math.min()获取一组数中的最大值

console.log(Math.min(10,20,30,100,50,32));

·Math.pow(x,y)返回x的y次幂

console.log(Math.pow(2,10));

·Math.sqrt()对一个数进行开方(根号)运算

console.log(Math.sqrt(3));

# 包装类

JS为我们提供了三个包装类，通过这三个包装类可以将基本数据类型的数据转换为对象

·String()

- 可以将基本数据类型的字符串转换为String对象

**var** str = **new** String("hello");//创建一个String类型的对象

·Number()

- 可以将基本数据类型的字符串转换为Number对象

**var** num = **new** Number(3);//创建一个Number类型的对象

·Boolean()

- 可以将基本数据类型的字符串转换为Boolean对象

**var** bool = **new** String(**true**);//创建一个String类型的对象

但是注意，在实际开发过程中不会使用基本数据类型的对象，如果使用基本数据类型的对象，在做一些比较时可能会带来不可预期的结果。

当我们对一些基本数据类型的值去调用属性和方法时，浏览器会临时使用包装类将其转换为对象，然后再调用对象的属性和方法，调用完以后，再将其转换成基本数据类型。

# 字符串的方法

字符串中的方法一般不会影响原字符串

·创建字符串，在底层字符串是以字符数组（char类型的数组）的形式保存的

**var** str = "hello atguigu";

·length属性获取字符串长度

·trim()去除字符串中的首尾空格

·根据索引返回指定位置的字符

console.log(str.charAt(3));//返回第四个字符

console.log(str[3]);//返回第四个字符

·charCodeAt()获取指定位置字符的字符编码（Unicode编码）

console.log(str.charCodeAt(3));//返回第四个字符的Unicode编码

·String.formCharCode()可以根据字符编码获取字符

console.log(String.fromCharCode(108));//返回字符编码为108的字符

·concat()用来连接两个或多个字符串，作用和+号一样

**var** str = "hello atguigu";

**var** str2 = "你好";

console.log(str.concat(str2,"再见"));

·IndexOf()检索一个字符串是否含有指定内容，如果有则返回初始索引，如果无则返回-1

console.log(str.indexOf("ll1o"));//查看str是否含有llo字符

- 可以指定从第几个字符开始查找

console.log(str.indexOf("l",3));//从第4个字符开始查看str是否含有llo字符

·lastIndexOf()从后往前查找检索一个字符串是否含有指定内容，如果有则返回初始索引，如果无则返回-1。

console.log(str.lastIndexOf("l"));//从后往前查看str是否含有llo字符

console.log(str.lastIndexOf("l",2));//从第3个字符开始往前查看str是否含有l字符

·slice()截取字符串

console.log(str.slice(1,4));//截取2到4个字符串，包括2也包括4

console.log(str.slice(1));//截取2以后的字符串

console.log(str.slice(1,-2));//截取2到倒数第3个字符串

·substring()截取字符串

- 与slice()的区别

-- substring()不能接受负数作为参数，如果传递负数则默认改为0，而且它会自动调整参数位置，如果第二个参数小于第一个，则自动交换。

·substr()截取字符串

- 与slice()的区别

-- substr()的第二个参数是截取字符串的长度，未标准化，不建议使用。

·split()将一个字符串拆分成数组

**var** str = "abc:bcd:cde:efg";

**var** arr = str.split(":");//按:号拆分字符串并存入数组arr中

console.log(arr);

- 如果传递一个空串，则会将每个字符都拆分

·toUpperCase()将字符串转大写

·toLowerCase()将字符串转小写

# 正则表达式

正则表达式用于定义一些字符串的规则，计算机可以根据正则表达式检查一个字符串是否符合规则，获取将字符串中符合规则的内容提取出来。

·创建正则表达式对象

- 语法var 变量 = new RegExp(“正则表达式”,”匹配模式”);

**var** reg = **new** RegExp("a");//检查字符串中是否含有a，严格区分大小写

console.log(reg.test("das"));

- 在构造函数中可传入匹配模式作为第二个参数，可以是**i(忽略大小写),g(全局匹配模式)**

**var** reg = **new** RegExp("a","i");//检查字符串中是否含有a，忽略大小写

console.log(reg.test("dAs"));

·使用字面量创建正则表达式

- 语法：var 变量 = /正则表达式/匹配模式

**var** reg = /a/i;

console.log(reg.test("dAs"));

使用字面量方式更加简单，使用构造函数方式更加灵活

·test()检查一个字符串是否符合正则表达式规则

console.log(reg.test("das"));

## 正则语法

·检查字符串中是否含有a

**var** reg = /a/;

·检查字符串中是否含有a或b或c

**var** reg = /a|b|c/;

**var** reg = /[abc]/;

·检查字符串是否含有字母

- /[a-z]/任意小写字母

- /[A-Z]/任意大写字母

- /[A-z]/任意大小写字母

- /[0-9]/任意数字

**var** reg = /[a-z]/;

·检查一个字符串是否含有abc或adc或aec

**var** reg = /a[bde]c/;

·检查一个字符串是否含有 除了a和b 以外的字符

**var** reg = /[^ab]/;

## 字符串和正则表达式相关的方法

·split()方法可以传递正则表达式拆分

- 根据任意字母将字符串拆分

**var** str = "1a2b3c4d5e6f7";

**var** reg = /[a-z]/;

console.log(str.split(reg));//[1,2,3,4,5,6,7]

- 即使不指定全局匹配，也都会全局拆分

·search()搜索字符串中是否含有指定内容，如果有则返回初始索引

- 搜索字符串是否含有abc

**var** str = "hello abc hello abc";

console.log(str.search("abc"));

- 搜索一个字符串是否含有abc或adc或aec

**var** str = "hello adc hello abc";

**var** reg = /a[bef]c/;

console.log(str.search(reg));

- 只会查找一个，不能全局匹配模式

·match()根据正则表达式将符合条件的内容提取出来

- match()会将匹配到的内容封装到一个数组中返回，即使只查询到一个结果。

- 默认情况下，只会找到第一个符合要求的内容，然后停止检索。可设置正则表达式为全局匹配模式，返回所有符合条件的字符串。

- 提取字符串的所有字母

**var** str = "1a2b3C4D5e6f";

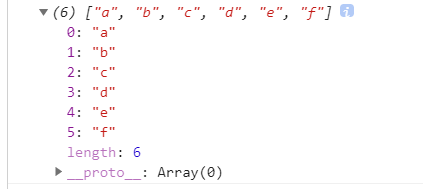
**var** reg = /[A-z]/g;

console.log(str.match(reg));

**var** str = "1a2b3C4D5e6f";

**var** reg = /[a-z]/ig;

console.log(str.match(reg));



·replace()将字符串中指定内容替换为新的内容

- 默认只会替换一个

**var** str = "1a2b3C4D5e6fa";

console.log(str.replace("a","ccc"));//将字符串里的第一个a替换成ccc

- 使用全局匹配模式将所有的a替换成ccc

**var** str = "1a2b3C4D5e6fa";

**var** reg = /a/g;

console.log(str.replace(reg,"ccc"));//将字符串里所有的a替换成ccc

- 将字符串的所有字母删去

**var** str = "1a2b3C4D5e6fa";

**var** reg = /[a-z]/gi;

console.log(str.replace(reg,""));//将字符串里所有的字母删去，123456

## 正则表达式语法

·检查字符串是否含有连续的3个a（aaa）

**var** reg = /a{3}/;

console.log(reg.test("aasaaa"));

·检查字符串是否含有连续的3个ab(ababab)

**var** reg = /(ab){3}/;

console.log(reg.test("aaaabababbbb"));

·检查字符串是否含有abc或abbc或abbbc，b出现1到3次

**var** reg = /ab{1,3}c/;

console.log(reg.test("ccabbbc"));

- {n}正好出现n次

- {n,m}出现n到m次

- {n,}出现n次以上

- /ab+c/ 相当于 /ab{1,}c/ 出现1次及以上

- /ab\*c/ 相当于 /ab{0,}c/ 出现0次及以上

- /ab?c/ 相当于 /ab{0,1}c/ 出现0或1次

·检查一个字符串是否以ac开头

**var** reg = /^ac/;

console.log(reg.test("acbbbc"));

·检查一个字符串是否以ac结尾

**var** reg = /ac$/;

console.log(reg.test("bbbcac"));

·/^a$/只能用来表示”a”

·检查一个字符串是否以ac开头或者以ac结尾

**var** reg = /^ac|ac$/;

console.log(reg.test("acbbbc"));

·检查一个字符串是否是一个合法的手机号

- 手机号规则

-- 11位

-- 以1开头 ^1

-- 第二位只能是3~9任意数字 [3-9]

-- 第三位以后是任意数字，且以这9个数字结尾 [0-9]{9}$

**var** phoneStr = "13567890123";

**var** phoneReg = /^1[3-9][0-9]{9}$/;

console.log(phoneReg.test(phoneStr));

·检查一个字符串是否含有”.”

- “.”默认表示除换行符以外的任意字符

- 使用\作为转义字符

**var** phoneReg = /\./;

console.log(phoneReg.test("da.s"));

- 注意：使用构造函数时，由于它的参数是一个字符串，而\是字符串中的转义字符，如果要使用\则需要用\\来代替。

**var** phoneReg = **new** RegExp("\\.");

console.log(phoneReg.test("das"));

·\w 任意字母、数字和\_ [A-z0-9\_]

·\W 除了字母、数字和\_ [^A-z0-9\_]

·\d 任意的数字 [0-9]

·\D 除了数字 [^0-9]

·\s 空格

·\S 除了空格

·\b 单词边界

- 检查一个字符串是否含有单词child

**var** phoneReg = /\bchild\b/;

console.log(phoneReg.test("hello children"));

·\B 除了单词边界

·去除字符串中的首尾空格

- 去除开头的空格 /^\s\*/

- 去除结尾的空格 /\s\*$/

- 去除开头和结尾的空格 /^\s\*|\s\*$/g

**var** phoneReg = /^\s\*|\s\*$/g;

console.log(phoneStr.replace(phoneReg,""));

## 邮件的正则表达式

Hello.nihao@abc.com.cn

·任意字母数字下划线开头，3个以上 ^\w{3,}

·.任意字母数字下划(可有可无)，1个以上 (\.\w+)\*

·@

·任意字母数字 [A-z0-9]+

·.任意字母结尾（2-5位）（出现1到2次） (\.[A-z]{2,5}){1,2}$

**var** emailStr = "city252@qq.com";

**var** emailReg = /^\w{3,}(\.\w+)\*@[A-z0-9]+(\.[A-z]{2,5}){1,2}$/;

console.log(emailReg.test(emailStr));

正则表达式大全：<https://www.cnblogs.com/hsinfo/p/13584432.html>

# DOM

全称Document Object Model文档对象模型

可以操作网页

·文档

- 文档表示的就是整个HTML网页文档

·对象

- 对象表示将网页中的每个部分都转换成一个对象

·模型

- 使用模型来表示对象之间的关系，这样方便我们获取对象

·浏览器为我们提供文档节点对象，这个对象是window属性，可以在页面中直接使用，文档节点代表的是整个网页

## 节点

Node——构成HTML文档最基本的单元

常用的节点分为四类

·文档节点：整个HTML文档

·元素节点：HTML文档中的HTML标签

·属性节点：元素的属性

·文本节点：HTML标签的文本内容

·通过id获取button对象

**var** btn = document.getElementById("btn");//通过id获取button对象

·修改button对象的文字

btn.innerHTML = "按钮1";//修改button内部的html代码

·innerHTML获取元素的内容

·innerText获取元素的文本内容

<body>

<button id="btn">我是一个按钮</button>

<script type="text/javascript">

**var** btn = document.getElementById("btn");//通过id获取button对象

btn.innerHTML = "按钮1";//修改button内部的html代码

</script>

</body>

## 事件

·事件是文档或浏览器窗口中发生的一些特定的交互瞬间

·JavaScript与HTML之间的交互是通过事件实现的

·如：点击某个元素、将鼠标移动至某个元素上、按下键盘上的某个健等

·onclick点击事件

<button id="btn" onclick="alert('别点我')">我是一个按钮</button>

·ondblclick双击事件

<button id="btn" ondblclick="alert('别双击我')">我是一个按钮</button>

·onmousemove鼠标移动事件

<button id="btn" onmousemove="alert('别移到我这')">我是一个按钮</button>

·为按钮的对应事件绑定处理函数的形式来响应事件，当事件被触发时，其对应的函数将会被调用。

<body>

<button id="btn">我是一个按钮</button>

<script type="text/javascript">

**var** btn = document.getElementById("btn");//通过id获取button对象

//给按钮绑定单击事件

btn.onclick = **function** () {

alert("你不要过来！！！");

};

</script>

</body>

## 文档的加载

浏览器在加载一个页面时，是按照自上而下的顺序加载的，读取到一行就运行一行，如果将script标签写到页面的上边，在代码执行时，页面还没有加载，可能会报错。

为了在页面加载完毕后再执行js代码，可以将js代码编写到页面的下部。

·onload事件会在整个页面加载完成之后才触发

- 为window绑定一个onload事件

- 可确保代码执行时所有的DOM对象已经加载完毕。

<script type="text/javascript">

window.onload = **function** () {//页面加载完成后触发

**var** btn = document.getElementById("btn");//通过id获取button对象

//给按钮绑定单击事件

btn.onclick = **function** () {

alert("你不要过来！！！");

};

};

</script>

## 获取元素对象

## 通过document调用的方法

·getElementById()通过id获取button对象

**var** btn1 = document.getElementById("btn1");//通过id获取button对象

·getElementsByTagName()通过标签名获取一组元素对象

**var** btn = document.getElementsByTagName("button");//通过标签名获取一组元素对象

·getElementsByName()通过name属性获取一组元素对象

btn = document.getElementsByName("btn");//通过name属性获取一组元素对象

·querySelector与querySelectorAll()

- 分别表示满足条件的第一个元素或者所有元素。括号内的写法与CSS选择器的写法一样

·getElementsByClassName()

- 通过类名来获取元素

·document.title和document.body

- 获取一个页面仅有的一个title和body元素

<body>

<button id="btn1" name="btn">按钮1</button>

<button id="btn2" name="btn">按钮2</button>

<script type="text/javascript">

**var** btn1 = document.getElementById("btn1");//通过id获取button对象

console.log(btn1);

**var** btn = document.getElementsByTagName("button");//通过标签名获取一组元素对象

console.log(btn);

btn = document.getElementsByName("btn");//通过name属性获取一组元素对象

console.log(btn);

</script>

</body>

·元素.属性名获取元素属性

console.log(input.value);//获取value属性

- 注意：class属性不能采用这种方式

- 读取元素class属性时需要使用 元素.className

## 通过元素节点调用的方法

·getElementsByTagName()

- 返回当前节点的指定标签名节点

**var** lis = city.getElementsByTagName("li");//查找#city下所有li节点

·childNodes

- 当前节点的所有子节点

- 它会获取包括文本节点在内的所有节点，标签间的空白也会当成文本节点。

- 但是IE8及以下浏览器不会将空白文本当成子节点。

**var** cns = city.childNodes;//返回#city的所有子节点

·children

- 获取当前元素的所有子元素

**var** cns = city.children;

·firstChild

- 当前节点的第一个子节点，可能获取空白文本

**var** fir = phone.firstChild;//返回#phone的第一个子节点

·firstElementChild

- 获取当前节点的第一个子元素

- 不兼容ie8及以下的浏览器

**var** fir = phone.firstElementChild;//返回#phone的第一个子元素

·lastChild

- 当前节点的最后一个子节点，可能获取空白文本

**var** fir = phone.lastChild;//返回#phone的最后一个子节点

·lastElementChild

- 获取当前节点的最后一个子元素

- 不兼容ie8及以下的浏览器

**var** fir = phone.lastElementChild;//返回#phone的最后一个子元素

·定义一个函数，专门用来为指定元素绑定单机响应事件

//定义一个函数，专门用来为指定元素绑定单机响应事件

**function** myClick(idbtn,fun) {//第一个是btn的id，第二个是事件回调函数，当单机元素时，该函数将会被触发

**var** btn = document.getElementById(idbtn);

btn.onclick = fun;

}

·parentNode

- 获取当前节点的父节点

**var** pn = bj.parentNode;//返回#bj的父节点

·previousSibling

- 获取当前节点的前一个兄弟节点，可能获取空白文本

**var** ps = and.previousSibling;

·previousElementSibling

- 获取当前节点的前一个兄弟元素

**var** ps = and.previousElementSibling;

·nextSibling

- 获取当前节点的下一个兄弟节点，可能获取空白文本

**var** ps = and.nextSibling;

·nextElementSibling

- 获取当前节点的下一个兄弟元素

**var** ps = and.nextElementSibling;

·value

- 获取元素的value值

- 文本框的valude值就是文本框填写的内容。

um.value;

um.value = "李白";//设置#username的value属性值

<body>

<div id="total">

<div class="inner">

<p>

你喜欢哪个城市？

</p>

<ul id="city">

<li id="bj">北京</li>

<li id="sh">上海</li>

<li id="gz">广州</li>

<li id="sz">深圳</li>

</ul>

<br/>

<br/>

<p>

你喜欢哪个游戏？

</p>

<ul id="game">

<li id="wz">王者荣耀</li>

<li id="cj">和平精英</li>

<li id="ys">原神</li>

<li id="me">摩尔庄园</li>

</ul>

<br/>

<br/>

<p>

你手机操作系统是？

</p>

<ul id="phone">

<li id="ios">苹果</li>

<li id="android">安卓</li>

<li id="hm">鸿蒙</li>

</ul>

</div>

<div class="inner">

gender:

<input type="radio" name="gender" value="male"/>

Male

<input type="radio" name="gender" value="famale"/>

Famale

<br>

<br>

name:

<input type="text" name="name" id="username" value="abcde"/>

</div>

</div>

<div id="btnList">

<div><button id="btn01">查找#bj节点</button> </div>

<div><button id="btn02">查找所有li节点</button> </div>

<div><button id="btn03">查找name=gender的所有节点</button> </div>

<div><button id="btn04">查找#city下所有li节点</button> </div>

<div><button id="btn05">返回#city的所有子元素</button> </div>

<div><button id="btn06">返回#phone的第一个子元素</button> </div>

<div><button id="btn07">返回#bj的父节点</button> </div>

<div><button id="btn08">返回#android的前一个兄弟节点</button> </div>

<div><button id="btn09">读取#username的value属性</button> </div>

<div><button id="btn10">设置#username的value属性值</button> </div>

<div><button id="btn11">返回#bj的文本值</button> </div>

</div>

<div id="result">

</div>

<script type="text/javascript">

**var** result = document.getElementById("result");

**var** btn01 = document.getElementById("btn01");

btn01.onclick = **function** () {

//查找#bj节点

**var** bj = document.getElementById("bj");

console.log(bj);

result.innerHTML = bj.innerHTML;

};

**var** btn02 = document.getElementById("btn02");

btn02.onclick = **function** (){

//查找所有li节点

**var** lis = document.getElementsByTagName("li");

console.log(lis);

// alert(lis)

**var** str = "";

//遍历lis

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

str += lis[i].innerHTML+"<br>";//获取元素节点内容

}

result.innerHTML = str;

};

**var** btn03 = document.getElementById("btn03");

btn03.onclick = **function** (){

//查找name=gender的所有节点

**var** inputs = document.getElementsByName("gender");

**var** str = "";

//遍历inputs

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

str += inputs[i].value + "<br>";//获取value属性

}

result.innerHTML = str;

};

**var** btn04 = document.getElementById("btn04");

btn04.onclick = **function** (){

//获取id为city的元素

**var** city = document.getElementById("city");

//查找#city下所有li节点

**var** lis = city.getElementsByTagName("li");

**var** str = "";

//遍历lis

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

str += lis[i].innerHTML+"<br>";//获取元素节点内容

}

result.innerHTML = str;

};

**var** btn05 = document.getElementById("btn05");

btn05.onclick = **function** (){

//获取id为city的元素

**var** city = document.getElementById("city");

//返回#city的所有子节点

**var** cns = city.children;

**var** str = "";

//遍历lis

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

str += cns[i].innerHTML+"<br>";//获取元素节点内容

}

result.innerHTML = str;

};

**var** btn06 = document.getElementById("btn06");

btn06.onclick = **function** (){

//获取id为phone的元素

**var** phone = document.getElementById("phone");

**var** fir = phone.lastChild;//返回#phone的第一个子元素

result.innerHTML = fir.innerHTML;

};

//定义一个函数，专门用来为指定元素绑定单机响应事件

**function** myClick(idbtn,fun) {//第一个是btn的id，第二个是事件回调函数，当单机元素时，该函数将会被触发

**var** btn = document.getElementById(idbtn);

btn.onclick = fun;

}

myClick("btn07",**function** () {

//获取id为bj的元素

**var** bj = document.getElementById("bj");

**var** pn = bj.parentNode;//返回#bj的父节点

result.innerHTML = pn.innerHTML;

});

myClick("btn08",**function** () {

//获取id为android的元素

**var** and = document.getElementById("android");

//返回#android的前一个兄弟节点

**var** ps = and.previousElementSibling;

result.innerHTML = ps.innerHTML;

});

myClick("btn09",**function** () {

//获取id为username的元素

**var** um= document.getElementById("username");

//读取#username的value属性

result.innerHTML = um.value;

});

myClick("btn10",**function** () {

//获取id为username的元素

**var** um= document.getElementById("username");

//设置#username的value属性值

um.value = "李白";

});

myClick("btn11",**function** () {

//获取id为bj的元素

**var** bj = document.getElementById("bj");

//返回#bj的文本值

result.innerHTML = bj.innerHTML;

});

</script>

</body>

## 图片切换

<body>

<div id="outer">

<p id="info"></p>

<img src="https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="400px"/>

<button id="prev">上一张</button>

<button id="next">下一张</button>

</div>

<script type="text/javascript">

//获取两个按钮

**var** prev = document.getElementById("prev");

**var** next = document.getElementById("next");

//图片路径的数组

**var** imgs = ["https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg",

"https://img0.baidu.com/it/u=3243110409,1678992802&fm=26&fmt=auto&gp=0.jpg",

"https://img2.baidu.com/it/u=534486318,1094186657&fm=26&fmt=auto&gp=0.jpg",

"https://img0.baidu.com/it/u=1310634044,1597068079&fm=26&fmt=auto&gp=0.jpg",

"https://img1.baidu.com/it/u=2239891783,1520147040&fm=26&fmt=auto&gp=0.jpg"

];

//保存当前图片的索引

**var** index = 0;

//设置提示文字

//获取id为info的p元素

**var** info = document.getElementById("info");

//设置提示文字

info.innerHTML = "一共" + imgs.length + "张图,当前第" + (index+1) +"张";

//分别为两个按钮绑定单机响应事件

prev.onclick = **function** () {//上一张

//当前索引-1

index--;

//判断index是否小于0

**if** (index<0){

index = imgs.length-1;

}

//获取img标签

**var** img = document.getElementsByTagName("img")[0];

info.innerHTML = "一共" + imgs.length + "张图,当前第" + (index+1) +"张";

//修改src属性

img.src = imgs[index];

};

next.onclick = **function** () {//下一张

//当前索引+1

index++;

//判断index是否大于数组长度

**if** (index>=imgs.length){

index = 0;

}

//获取img标签

**var** img = document.getElementsByTagName("img")[0];

info.innerHTML = "一共" + imgs.length + "张图,当前第" + (index+1) +"张";

//修改src属性

img.src = imgs[index];

};

</script>

</body>

·body

- 获取body元素

**var** body = document.body;

·documentElement

- 获取HTML根标签

**var** html = document.documentElement;

·all

- 获取页面中的所有元素

- 等同于getElementsByTagName("\*")

**var** all = document.all;

·getElementsByClassName()

- 根据class获取一组元素节点

- ie8及以下的浏览器不支持

**var** box1 = document.getElementsByClassName("box1");

·querySelector()

- 根据CSS选择器来查询一个元素节点对象

- ie8以上支持

-使用该方法只会返回唯一的一个元素，如果满足条件的元素有多个，那么它只会返回第一个。

**var** div = document.querySelector(".box1 div");//获取class属性为box1下的第一个div元素

**var** div = document.querySelector("#box1");//获取id属性为box1的第一个元素

·querySelectorAll()

- 根据CSS选择器来查询一组元素节点对象

**var** div = document.querySelectorAll(".box1 div");//获取class属性为box1下的所有div元素

## DOM增删改

·createElement()

- 创建一个元素节点对象

- 它需要一个标签名作为参数，将会根据该标签名创建一个元素节点对象，并将创建好的对象返回。

**var** li = document.createElement("li");//创建li元素节点

·createTextNode()

- 创建一个文本节点对象

- 它需要一个文本内容作为参数，将会根据该内容创建一个文本节点对象，并将创建好的对象返回。

**var** hy = document.createTextNode("河源");//创建河源文本节点

·appendChild()

- 向一个父节点中添加一个新的子节点

- 用法：父节点.appendChild(子节点)

li.appendChild(hy);//将河源文本节点添加到li节点中

·insertBefore()

- 在指定的子节点前插入新的子节点

- 语法：父节点.insertBefore(新节点,旧节点)

city.insertBefore(li,bj);//将li节点插入到bj节点前面

·replaceChild()

- 使用指定的子节点替换已有的子节点

- 语法：父节点.replaceChild(新节点,旧节点)

city.replaceChild(li,bj);//使用"河源"节点替换#bj节点

·removeChild()

- 删除指定子节点

- 语法：父节点.removeChild(子节点)

city.removeChild(bj);//删除#bj节点

bj.parentNode.removeChild(bj);//获取bj的父节点再删除#bj节点

·使用innerHTML也可以完成DOM的增删改相关操作，一般会两种方式结合使用。

//获取id为city的节点

**var** city = document.getElementById("city");

// city.innerHTML += "<li id=\"hy\">河源</li>";

**var** li = document.createElement("li");//创建li元素节点

li.innerHTML = "河源";

city.appendChild(li);

<body>

<div id="total">

<div class="inner">

<p>

你喜欢哪个城市？

</p>

<ul id="city">

<li id="bj">北京</li>

<li id="sh">上海</li>

<li id="gz">广州</li>

<li id="sz">深圳</li>

</ul>

</div>

</div>

<div id="btnList">

<div><button id="btn01">创建一个"河源"节点，添加到#city下</button> </div>

<div><button id="btn02">将"河源"节点插入到#bj前面</button> </div>

<div><button id="btn03">使用"河源"节点替换#bj节点</button> </div>

<div><button id="btn04">删除#bj节点</button> </div>

<div><button id="btn05">读取#city内的HTML代码</button> </div>

<div><button id="btn06">设置#bj内的HTML代码</button> </div>

<div><button id="btn07">向#city添加河源</button> </div>

</div>

<script type="text/javascript">

//定义一个函数，专门用来为指定元素绑定单机响应事件

**function** myClick(idbtn,fun) {//第一个是btn的id，第二个是事件回调函数，当单机元素时，该函数将会被触发

**var** btn = document.getElementById(idbtn);

btn.onclick = fun;

}

myClick("btn01",**function** () {

//创建一个"河源"节点，添加到#city下

**var** li = document.createElement("li");//创建li元素节点

**var** hy = document.createTextNode("河源");//创建河源文本节点

li.appendChild(hy);//将河源文本节点添加到li节点中

//获取id为city的节点

**var** city = document.getElementById("city");

city.appendChild(li);

});

myClick("btn02",**function** () {

//将"河源"节点插入到#bj前面

**var** li = document.createElement("li");//创建li元素节点

**var** hy = document.createTextNode("河源");//创建河源文本节点

li.appendChild(hy);//将河源文本节点添加到li节点中

//获取id为bj的节点

**var** bj = document.getElementById("bj");

//获取id为city的节点

**var** city = document.getElementById("city");

city.insertBefore(li,bj);//将li节点插入到bj节点前面

});

myClick("btn03",**function** () {

//使用"河源"节点替换#bj节点

**var** li = document.createElement("li");//创建li元素节点

**var** hy = document.createTextNode("河源");//创建河源文本节点

li.appendChild(hy);//将河源文本节点添加到li节点中

//获取id为bj的节点

**var** bj = document.getElementById("bj");

//获取id为city的节点

**var** city = document.getElementById("city");

city.replaceChild(li,bj);//使用"河源"节点替换#bj节点

});

myClick("btn04",**function** () {

//删除#bj节点

//获取id为bj的节点

**var** bj = document.getElementById("bj");

bj.parentNode.removeChild(bj);//获取bj的父节点再删除#bj节点

});

myClick("btn05",**function** () {

//读取#city内的HTML代码

//获取id为city的节点

**var** city = document.getElementById("city");

alert(city.innerHTML);

});

myClick("btn06",**function** () {

//设置#bj内的HTML代码

//获取id为bj的节点

**var** bj = document.getElementById("bj");

bj.innerHTML = "河源";

});

myClick("btn07",**function** () {

//向#city添加河源

//获取id为city的节点

**var** city = document.getElementById("city");

// city.innerHTML += "<li id=\"hy\">河源</li>";

**var** li = document.createElement("li");//创建li元素节点

li.innerHTML = "河源";

city.appendChild(li);

});

</script>

</body>

# 死链接

·href="#"与href="javascript:;"的区别

#包含了一个位置信息，默认的锚是#top也就是网页的上端。

而javascript:;, 仅仅表示一个死链接。

在页面很长的时候会使用#来定位页面的具体位置，格式为：# + id。

如果你要定义一个死链接请使用javascript:;。

# 响应事件的this

响应事件的this表示点击的元素对象

# confirm()对话框

·用于弹出一个带有确认和取消按钮的提示框

- 点击确认则返回true

- 点击取消则返回false

**var** flag = confirm("确认删除吗？");

# 操作内联样式

·修改元素的样式

- 语法：元素.style.样式名 = 样式值；

box1.style.width = "300px";//修改box1的宽度

- 如果CSS的样式名中含有 - ，需要将样式名修改为驼峰命名法

box1.style.backgroundColor = "yellow";//修改背景颜色

- 通过style属性设置的样式都是内联样式，而内联样式具有较高的优先级。但是如果在样式中写了!important，则此时样式会有最高的优先级。此时会导致JS修改样式失效，所以尽量不用用!important

background-color: pink **!important**;

·读取元素的内联样式

- 语法：元素.style.样式名;

- 只能读取内联样式

alert(box1.style.width);//获取box1的内联样式宽度

·读取元素的当前样式

- 元素.currentStyle.样式名

- 只有ie浏览器支持

- 在其他浏览器中可使用getComputedStyle()方法获取元素样式，第一个参数为要获取样式的元素，第二个为伪元素，一般传null。该方法不支持ie8

**var** obj = getComputedStyle(box1,**null**);//获取box1的样式对象

alert(obj.width);//获取样式宽度

alert(getComputedStyle(box1,**null**).width);//获取样式宽度

·定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

//定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

**function** getStyle(obj, name) {

**if** (window.getComputedStyle){//如果有该方法

**return** getComputedStyle(obj,**null**)[name];//正常浏览器具有getComputedStyle方法

}**else** {

**return** obj.currentStyle[name];//ie具有该方法

}

}

**var** btn03 = document.getElementById("btn03");

btn03.onclick = **function** (){

alert(getStyle(box1,"width"));//获取样式宽度

};

·clientWidth

- 获取元素可见宽度，包括内容区和内边距

- 这些属性都是不带px的，是一个数字，可以之间计算

alert(box1.clientWidth);//获取box1可见宽度

·clientHeight

- 获取元素可见高度，包括内容区和内边距

- 这些属性都是不带px的，是一个数字，可以之间计算

alert(box1.clientHeight);//获取box1可见高度

·offsetWidth

- 获取元素的整个宽度，包括内容区、内边距、不可见的边框

alert(box1.offsetWidth);//获取box1的宽度

·offsetHeight

- 获取元素的整个高度，包括内容区、内边距、不可见的边框

alert(box1.offsetHeight);//获取box1的宽度

·offsetParent

- 可以用来获取当前元素的定位元素

- 会获取到当前元素最近的开启了定位的祖先元素

console.log(box1.offsetParent);//获取box1的定位父元素

·offsetLeft

- 当前元素相对于其定位父元素的水平偏移量

·offsetTop

- 当前元素相对于其定位父元素的垂直偏移量

·scrollwidth

- 获取元素整个滚动区的宽度

·scrollHeight

- 获取元素整个滚动区的高度

·scrollLeft

- 获取水平滚动条滚动的距离

·scrollTop

- 获取垂直滚动条滚动的距离

·当scrollHeight - scrollTop == clientHeight时，说明滚动条滚动到底了。可用于确保用户阅读协议，当滚动条到最底时，说明阅读协议。

·当scrollWidth - scrollLeft == clientWidth时，说明滚动条滚动到底了。

·onscroll响应事件

- 会在滚动条滚动时触发

·disabled属性可以设置属性是否被禁用

<body>

<button id="btn01">修改样式</button>

<button id="btn02">读取内联样式</button>

<button id="btn03">读取样式</button>

<br/>

<br/>

<div id="box1"></div>

<script type="text/javascript">

//点击按钮以后修改box1的大小

**var** box1 = document.getElementById("box1");//获取box1

**var** btn01 = document.getElementById("btn01");

btn01.onclick = **function** () {

box1.style.width = "300px";//修改box1的宽度

box1.style.height = "300px";//修改box1的宽度

box1.style.backgroundColor = "yellow";//修改背景颜色

};

**var** btn02 = document.getElementById("btn02");

btn02.onclick = **function** (){

alert(box1.style.width);//获取box1的内联样式宽度

};

//定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

**function** getStyle(obj, name) {

**if** (window.getComputedStyle){//如果有该方法

**return** getComputedStyle(obj,**null**)[name];//正常浏览器具有getComputedStyle方法

}**else** {

**return** obj.currentStyle[name];//ie具有该方法

}

}

**var** btn03 = document.getElementById("btn03");

btn03.onclick = **function** (){

alert(getStyle(box1,"width"));//获取样式宽度

};

</script>

</body>

# 事件对象

当事件响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数。在事件对象中封装了当前事件相关的信息，如鼠标坐标，哪个键盘被按下，鼠标滚轮滚动方向等。

Ie8及一下浏览器中，响应事件被触发时，浏览器不会传递事件对象，事件对象作为window对象的属性保存。window.event.clientX;

·onmousemove响应事件

- 将会在鼠标在元素中移动时被触发

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>事件对象</title>

<style type="text/css">

#areaDiv{

width: 200px;

height: 200px;

background-color: pink;

}

</style>

</head>

<body>

<div id="areaDiv"></div>

<div id="showMsg"></div>

<script type="text/javascript">

//当鼠标在areaDiv移动时，showMsg来显示鼠标的坐标

//获取两个div

**var** areaDiv = document.getElementById("areaDiv");

**var** showMsg = document.getElementById("showMsg");

areaDiv.onmousemove = **function** (event) {

//在showMsg显示鼠标坐标

// if(!event){//解决浏览器兼容问题

// event = window.event;

// }

event = event || window.event;//解决浏览器兼容问题

**var** x = event.clientX;//获取鼠标指针的水平坐标

**var** y = event.clientY;//获取鼠标指针的垂直坐标

showMsg.innerHTML = "x = " + x +" , y = " + y;

}

</script>

</body>

</html>

## div随鼠标移动

·clientX获取鼠标当前可见窗口的坐标，而div偏移量是相对于整个页面的

·pageX获取鼠标相对于当前页面的坐标，在ie8及以下浏览器中不支持

·chrme认为浏览器的滚动条是body的，可以通过body.scrollTop来获取滚动条滚动的距离。

·火狐等浏览器认为浏览器的滚动条是html的，可以通过documentElement.scrollTop来获取滚动条滚动的距离。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>div随鼠标移动</title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: red;

/\*开启box1的绝对定位，使left生效\*/

position: absolute;

}

</style>

</head>

<body style="height: 1000px;width: 2000px">

<div id="box1"></div>

<script type="text/javascript">

**var** box1 = document.getElementById("box1");

//给整个页面绑定鼠标响应事件

document.onmousemove = **function** (event) {

//获取滚动条滚动的距离

//chrme认为浏览器的滚动条是body的，可以通过body.scrollTop来获取

// var st = document.body.scrollTop;

//火狐等浏览器认为浏览器的滚动条是html的

// var st = document.documentElement.scrollTop;

**var** st = document.body.scrollTop || document.documentElement.scrollTop;//兼容浏览器

**var** sl = document.body.scrollLeft || document.documentElement.scrollLeft//兼容浏览器

console.log(st);

//clientX获取鼠标当前可见窗口的坐标，而div偏移量是相对于整个页面的

// var left = event.clientX;

// var top = event.clientY;

//pageX获取鼠标相对于当前页面的坐标，这两个属性在ie8及以下浏览器中不支持

**var** left = event.clientX;

**var** top = event.clientY;

//设置div的偏移量

box1.style.left = left + sl + "px";//加上滚动条距离

box1.style.top = top + st + "px";//加上滚动条距离

}

</script>

</body>

</html>

## 事件的冒泡

·事件的向上传导，当后代元素上的一个事件被触发时，其祖先元素的相同事件也会被触发。

·在开发中大部分冒泡都是有用的，如果不希望发生事件冒泡，可以通过事件对象来取消冒泡。

·event.cancelBubble = true;取消事件冒泡

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>事件冒泡</title>

<style type="text/css">

#box1{

width: 200px;

height: 200px;

background-color: pink;

}

#s1{

background-color: red;

}

</style>

</head>

<body>

<div id="box1">

我是box1

<span id="s1">我是span</span>

</div>

<script type="text/javascript">

//为s1绑定单机响应事件

**var** s1 = document.getElementById("s1");

s1.onclick = **function** (event) {

event = event || window.event;//浏览器兼容

alert("我是span的单机响应函数");

event.cancelBubble = **true**;//取消事件冒泡

};

//为box1绑定单机响应事件

**var** box1 = document.getElementById("box1");

box1.onclick = **function** (event) {

event = event || window.event;//浏览器兼容

alert("我是div的单机响应函数");

event.cancelBubble = **true**;//取消事件冒泡

};

//为body绑定单机响应函数

document.body.onclick = **function** () {

alert("我是body的单机响应函数")

}

</script>

</body>

</html>

## 事件的委派

·alert(event.target);返回触发事件的元素

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>事件的委派</title>

</head>

<body>

<button id="btn01">添加超链接</button>

<ul id="ul">

<li><a href="javascript:;" class="link">超链接一</a></li>

<li><a href="javascript:;" class="link">超链接二</a></li>

<li><a href="javascript:;" class="link">超链接三</a></li>

</ul>

<script type="text/javascript">

**var** ul = document.getElementById("ul");

//点击按钮以后添加超链接

**var** btn01 = document.getElementById("btn01");

btn01.onclick = **function** () {

//创建一个li

**var** li = document.createElement("li");

li.innerHTML = "<a href=\"javascript:;\" class=\"link\">新超链接</a>";

//将li添加到ul中

ul.appendChild(li);

};

**var** allA = document.getElementsByTagName("a");//获取所有a

//为每个超链接都绑定一个单机响应函数,这种操作比较麻烦

// 而且这些操作只能为已有的超链接设置事件，而新添加的超链接必须重新绑定

//遍历

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

// allA[i].onclick = function () {

// alert(this.innerHTML);

//

// }

// }

//只绑定一次事件，即可应用到多个元素上，即使元素是后添加的

//可以尝试将其绑定给元素的祖先元素

//为ul绑定一个单机响应函数

ul.onclick = **function** (event) {

//如果触发事件是期望的元素，则执行，否则不执行

// alert(event.target.innerHTML);//返回触发事件的元素

**if** (event.target.className == "link"){

alert(event.target.innerHTML);

}

}

</script>

</body>

</html>

## 事件的绑定

·addEventListener()

- 绑定响应事件函数的第二个方法

- 可以为该元素添加响应事件

- 参数：

-- 事件的字符串，不要on，如：“click”

-- 回调函数，当事件触发是该函数会被调用

-- 是否在捕获阶段触发事件，需要一个布尔值，一般传false

- 可以同时为一个元素的相同事件同时绑定多个响应事件，当事件被触发时，响应函数将会按照函数的绑定顺序执行。

btn01.addEventListener("click",**function** () {

alert("1");

},**false**);

btn01.addEventListener("click",**function** () {

alert("2");

},**false**);

- 以上方法不支持ie8及以下浏览器

- 在ie8中可使用attachEcent()来绑定多个事件，不支持其他浏览器

- 参数

-- 事件的字符串，如：“onclick”

-- 回调函数，当事件触发是该函数会被调用

- 当事件被触发时，后绑定的函数会先执行

btn01.attachEvent("onclick",**function** () {

alert(1);

});

btn01.attachEvent("onclick",**function** () {

alert(2);

});

btn01.attachEvent("onclick",**function** () {

alert(3);

});

·定义一个函数，为指定元素绑定响应函数,兼容所有的浏览器

addEventListener()中的this是绑定事件的对象

attachEvent()中的this是window

需要使用 函数.call() 的方法统一两个方法中的this

//定义一个函数，为指定元素绑定响应函数,兼容所有的浏览器，并统一this

//参数：1.要绑定事件的对象 2.事件的字符串 3.回调函数

**function** bind(obj,eventStr,callback) {

**if** (obj.addEventListener) {//如果有该方法

//大部分浏览器兼容的方式

obj.addEventListener(eventStr,callback,**false**);

}**else** {

//统一this,this是谁由调用方式决定的

//ie8及以下，需要+on

obj.attachEvent("on"+eventStr,**function** () {

//在匿名函数中调用回调函数

callback.call(obj);

});

}

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>事件的绑定</title>

</head>

<body>

<button id="btn01">点我一下</button>

<script type="text/javascript">

**var** btn01 = document.getElementById("btn01");

// btn01.onclick = function () {

// alert(1);

// };

//以下方法ie8不支持

// btn01.addEventListener("click",function () {

// alert("1");

// },false);

// btn01.addEventListener("click",function () {

// alert("2");

// },false);

// //以下方法ie8支持，其他不支持

// btn01.attachEvent("onclick",function () {

// alert(1);

// });

// btn01.attachEvent("onclick",function () {

// alert(2);

// });

//定义一个函数，为指定元素绑定响应函数,兼容所有的浏览器，并统一this

//参数：1.要绑定事件的对象 2.事件的字符串 3.回调函数

**function** bind(obj,eventStr,callback) {

**if** (obj.addEventListener) {//如果有该方法

//大部分浏览器兼容的方式

obj.addEventListener(eventStr,callback,**false**);

}**else** {

//统一this,this是谁由调用方式决定的

//ie8及以下，需要+on

obj.attachEvent("on"+eventStr,**function** () {

//在匿名函数中调用回调函数

callback.call(obj);

});

}

}

//调用事件响应函数

bind(btn01,"click",**function** () {

console.log(**this**);

});

</script>

</body>

</html>

## 事件的传播

·微软公司认为事件应该是由内向外传播，当事件被触发时，应该先触发当前元素的事件，然后再向当前元素的祖先元素上传播，也就是事件应该在冒泡阶段执行。

·网景公司认为事件应该是由外向内传播，当事件被触发时，应该先触发当前元素的祖先元素事件，然后再向内传播给后代元素。

·W3C综合了两个公司的方案，将事件传播分成3个阶段

- 捕获阶段

-- 在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件。

- 目标阶段

-- 事件捕获到目标元素，捕获结束时开始在目标元素上触发事件

- 冒泡阶段

-- 事件从目标元素向他的祖先元素传递，依次触发祖先元素上的事件

- 如果希望在捕获阶段就触发事件，可以将addEventListener()的第三个参数设置为true，一般情况不会用到。Ie8及以下的浏览器中没有捕获阶段。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>事件的传播</title>

<style type="text/css">

#box1{

width: 300px;

height: 300px;

background-color: pink;

}

#box2{

width: 200px;

height: 200px;

background-color: cadetblue;

}

#box3{

width: 100px;

height: 100px;

background-color: khaki;

}

</style>

</head>

<body>

<div id="box1">

<div id="box2">

<div id="box3"></div>

</div>

</div>

<script type="text/javascript">

//找到div元素

**var** box1 = document.getElementById("box1");

**var** box2 = document.getElementById("box2");

**var** box3 = document.getElementById("box3");

//定义一个函数，为指定元素绑定响应函数,兼容所有的浏览器，并统一this

//参数：1.要绑定事件的对象 2.事件的字符串 3.回调函数

**function** bind(obj,eventStr,callback) {

**if** (obj.addEventListener) {//如果有该方法

//大部分浏览器兼容的方式

obj.addEventListener(eventStr,callback,**false**);

}**else** {

//统一this,this是谁由调用方式决定的

//ie8及以下，需要+on

obj.attachEvent("on"+eventStr,**function** () {

//在匿名函数中调用回调函数

callback.call(obj);

});

}

}

bind(box1,"click",**function** () {

alert(**this**+"box1");

});

bind(box2,"click",**function** () {

alert(**this**+"box2");

});

bind(box3,"click",**function** () {

alert(**this**+"box3");

});

</script>

</body>

</html>

## 事件练习之拖拽

·onmousedown响应事件，鼠标被按下

·onmousemove响应事件，鼠标移动

·onmouseup响应事件，鼠标松开

·div的偏移量 = 鼠标.clentX - 元素.offsetLeft

·div的偏移量 = 鼠标.clentY - 元素.offsetTop

·当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容，此时会导致拖拽功能异常。可以通过return false来取消默认行为

·setCapture()设置元素捕获所以鼠标按下的事件，只有ie支持。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>拖拽</title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: cadetblue;

position: absolute;

}

#box2{

width: 100px;

height: 100px;

background-color: pink;

position: absolute;

left: 200px;

}

</style>

</head>

<body>

<div id = "box1"></div>

<div id = "box2"></div>

<script type="text/javascript">

*/\*\**

*\* 1.当鼠标在被拖拽元素上按下时，开始拖拽*

*\* 2.当鼠标移动时被拖拽元素跟随鼠标移动*

*\* 3.当鼠标松开时，被拖拽元素固定在当前位置*

*\*/*

//为box1绑定一个鼠标按下事件,当鼠标在被拖拽元素上按下时，开始拖拽

// box1.onmousedown = function (event) {

//

// // if (box1.setCapture){//ie浏览器才有该方法

// // box.setCapture();

// // }

// box1.setCapture && box.setCapture();//如果有box1.setCapture才执行box.setCapture()，跟上面的if一样

//

// event = event || window.event;

// //div的偏移量 = 鼠标.clentX - 元素.offsetLeft

// //div的偏移量 = 鼠标.clentY - 元素.offsetTop

// var ol = event.clientX - box1.offsetLeft;

// var ot = event.clientY - box1.offsetTop;

//

//

// //当鼠标移动时被拖拽元素跟随鼠标移动,为document绑定一个onmousemove事件

// document.onmousemove = function (event) {

// event = event || window.event;//浏览器兼容

// //获取元素左上角的坐标

// var left = event.clientX - ol;

// var top = event.clientY - ot;

// //修改box1的位置

// box1.style.left = left + "px";

// box1.style.top = top + "px";

// };

//

// //当鼠标松开时，被拖拽元素固定在当前位置

// // 不能给box1绑定，在兄弟元素上松开时还是会动，应该给document绑定

// document.onmouseup = function () {

// //取消document的onmousemove事件

// document.onmousemove = null;

// //取消document的onmouseup事件

// document.onmouseup = null;

// //当鼠标松开时，取消对该事件的捕获

// box1.releaseCapture && box1.releaseCapture();

// };

// //当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容

// // 此时会导致拖拽功能异常。可以通过return false来取消默认行为

// return false;

// };

//提取一个专门用来设置拖拽的函数,第一个参数是要拖拽的元素

**function** drag(obj) {

//为box1绑定一个鼠标按下事件,当鼠标在被拖拽元素上按下时，开始拖拽

obj.onmousedown = **function** (event) {

// if (box1.setCapture){//ie浏览器才有该方法

// box.setCapture();

// }

obj.setCapture && obj.setCapture();//如果有box1.setCapture才执行box.setCapture()，跟上面的if一样

event = event || window.event;

//div的偏移量 = 鼠标.clentX - 元素.offsetLeft

//div的偏移量 = 鼠标.clentY - 元素.offsetTop

**var** ol = event.clientX - obj.offsetLeft;

**var** ot = event.clientY - obj.offsetTop;

//当鼠标移动时被拖拽元素跟随鼠标移动,为document绑定一个onmousemove事件

document.onmousemove = **function** (event) {

event = event || window.event;//浏览器兼容

//获取元素左上角的坐标

**var** left = event.clientX - ol;

**var** top = event.clientY - ot;

//修改box1的位置

obj.style.left = left + "px";

obj.style.top = top + "px";

};

//当鼠标松开时，被拖拽元素固定在当前位置

// 不能给box1绑定，在兄弟元素上松开时还是会动，应该给document绑定

document.onmouseup = **function** () {

//取消document的onmousemove事件

document.onmousemove = **null**;

//取消document的onmouseup事件

document.onmouseup = **null**;

//当鼠标松开时，取消对该事件的捕获

obj.releaseCapture && obj.releaseCapture();

};

//当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容

// 此时会导致拖拽功能异常。可以通过return false来取消默认行为

**return false**;

};

};

//获取box1

**var** box1 = document.getElementById("box1");

//开启box1的拖拽功能

drag(box1);

//获取box2

**var** box2 = document.getElementById("box2");

//开启box2的拖拽功能

drag(box2);

</script>

</body>

</html>

## 滚轮事件但是火狐不支持

·onmousewheel绑定鼠标滚轮滚动事件，在火狐中需要使用DOMMouseScroll来绑定滚动事件，该事件需要通过addEvenLisener()函数来绑定。

·event.whellDalta可获取鼠标滚轮滚动的方向，向上滚是正数，向下滚是负数。在火狐中需要使用event.datail来获取滚动方向，向上滚负数，向下滚是正数。

·当滚轮滚动时，如果浏览器有滚动条。滚动条会随之滚动，这是浏览器的默认行为，如果要取消则return false,但是addEventListener()方法绑定响应函数不支持return false;取消默认。应使用event.preventDefault();但是该方法ie8会报错。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>滚轮事件</title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: pink;

}

</style>

</head>

<body style="height: 2000px;">

<div id="box1"></div>

<script type="text/javascript">

//box1随鼠标滚动变长变短

//获取box1的div

**var** box1 = document.getElementById("box1");

//为box1绑定鼠标滚轮滚动事件,会在滚轮滚动时触发，但是火狐不支持

box1.onmousewheel = **function** (event) {

event = event || window.event;

//判断鼠标滚轮滚动的方向，event.whellDalta可获取

**if** (event.wheelDelta > 0 || event.detail < 0) {//谷歌和火狐向上滚

//box1变短

**if** (box1.clientHeight > 10) {//如果高度大于10则变短，防止变没了

box1.style.height = box1.clientHeight - 10 + "px";

}

}**else** {//向下滚

//box1变长

box1.style.height = box1.clientHeight + 10 + "px";

}

//当滚轮滚动时，如果浏览器有滚动条。滚动条会随之滚动

//但是addEventListener()方法绑定响应函数不支持return false;取消默认。

// event.preventDefault();//addEventListener()取消浏览器默认行为，但是ie8会报错。

event.preventDefault && event.preventDefault();//如果有则使用，防止ie8报错。

**return false**;//取消浏览器默认行为

};

//在火狐中需要使用DOMMouseScroll来绑定滚动事件，该事件需要通过addEvenLisener()函数来绑定

//定义一个函数，为指定元素绑定响应函数,兼容所有的浏览器，并统一this

//参数：1.要绑定事件的对象 2.事件的字符串 3.回调函数

**function** bind(obj,eventStr,callback) {

**if** (obj.addEventListener) {//如果有该方法

//大部分浏览器兼容的方式

obj.addEventListener(eventStr,callback,**false**);

}**else** {

//统一this,this是谁由调用方式决定的

//ie8及以下，需要+on

obj.attachEvent("on"+eventStr,**function** () {

//在匿名函数中调用回调函数

callback.call(obj);

});

}

}

bind(box1,"DOMMouseScroll",box1.onmousewheel);//兼容火狐浏览器

</script>

</body>

</html>

## 键盘事件

·onkeydown某个键盘按键被按下，按住不松手会一直触发，连续触发时第一次和第二次直接会有小卡顿，防误触导致连续触发。

·onkeyup某个键盘按键被松开

·键盘事件一般都会绑定给一些可以获取到焦点的对象或者是document

·keyCode获取按键编码

·altKey、ctrlKey、shiftKey用来判断alt、ctrl、shift是否被按下，如果按下则返回true;

·在文本框输入内容，属于onkeydown的默认行为，如果在onkeydown中取消了默认行为，则输入的内容不会出现在文本框中，return false;

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>键盘事件</title>

</head>

<body>

<input type="text">

<script type="text/javascript">

//按键按下

document.onkeydown = **function** (event) {

event = event || window.event;//兼容浏览器

//keyCode获取按键编码

// console.log(event.keyCode);

//判断s和ctrl是否同时被按下

**if** (event.keyCode === 83 && event.ctrlKey) {

console.log("ctrl+s被按下了");

}

};

//按键松开

document.onkeyup = **function** (event) {

};

//获取input

**var** input = document.getElementsByTagName("input")[0];

//键盘在input获取焦点被按下时

input.onkeydown = **function** (event) {

event = event || window.event;

//数字48-57

//使文本框不能输入数字

**if** (event.keyCode >=48 && event.keyCode <=57){

**return false**;

}

//在文本框输入内容，属于onkeydown的默认行为

//如果在onkeydown中取消了默认行为，则输入的内容不会出现在文本框中

// return false;

}

</script>

</body>

</html>

## 键盘移动div

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>键盘移动div</title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: cadetblue;

position: absolute;

}

</style>

</head>

<body>

<div id="box1"></div>

<script type="text/javascript">

//为document绑定一个键盘按下事件

document.onkeydown = **function** (event) {

event = event || window.event;

**var** speed = 10;//移动速度

//当用户按了ctrl以后，速度加快

**if** (event.ctrlKey) {

speed = 30;

}

**switch** (event.keyCode) {

**case** 37:

//向左，left值减小

box1.style.left = box1.offsetLeft - speed + "px";

**break**;

**case** 38:

//向上

box1.style.top = box1.offsetTop - speed + "px";

**break**;

**case** 39:

//向右

box1.style.left = box1.offsetLeft + speed + "px";

**break**;

**case** 40:

//向下

box1.style.top = box1.offsetTop + speed + "px";

**break**;

}

};

</script>

</body>

</html>

# BOM

浏览器对象模型，可以通过JS操作浏览器

## BOM对象

·window

- 代表的是整个浏览器的窗口，同时window也是网页中的全局对象

·Navigator

- 代表的当前浏览器的信息，通过该对象可以识别不同的浏览器

·Location

- 代表当前浏览器的地址栏信息，通过该对象可以获取地址栏信息，或者操作浏览器跳转页面

·History

- 代表浏览器的历史记录，通过该对象操作浏览器的历史记录，由于隐私原因，该对象不能获取到具体的历史记录，只能操作浏览器向前或向后翻页，而且只在当次访问时有效。

·Screen

- 代表用户的屏幕信息。通过该对象可获取用户的显示器的相关信息

这些BOM对象在浏览器中都是作为window对象的属性保存的，可以通过window对象来使用，也可以直接使用。

### Navigator

代表的当前浏览器的信息，通过该对象可以识别不同的浏览器

由于历史原因，Navigator对象中的大部分属性都已经不能帮助我们识别浏览器了。

一般我们会用userAgent（等价于浏览器）来判断浏览器信息

·userAgent

- 是字符串，包含用来描述浏览器信息的内容

- 不同的浏览器会有不同的userAgent

- 谷歌浏览器的userAgent信息为

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36

- QQ浏览器的userAgent信息为

Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.25 Safari/537.36 Core/1.70.3861.400 QQBrowser/10.7.4313.400

- Edg浏览器的userAgent信息为

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36 Edg/91.0.864.64

- IE11浏览器的userAgent信息为

Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; rv:11.0) like Gecko

- IE8浏览器的userAgent信息为

Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)

- IE9浏览器的userAgent信息为

Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)

- IE10浏览器的userAgent信息为

Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)

- 在IE11中已经将微软和IE相关的标识去除，所以已经不能通过userAgent识别浏览器是否是ie了

- 如果userAgent不能判断，还可以通过浏览器特有的对象判断浏览器信息

比如：ActiveXObiect是ie浏览器特有的

- 判断浏览器的内核

<body>

<script type="text/javascript">

**var** ua = navigator.userAgent;

**if** ("ActiveXObject" **in** window){//判断该对象是否存在

alert("IE浏览器内核");

}**else if** (/firefox/i.test(ua)){//判断ua字符串是否含有firefox并忽略大小写

alert("火狐浏览器内核");

}**else if** (/chrome/i.test(ua)){

alert("谷歌浏览器内核");

}

</script>

</body>

# History

可以用来操作浏览器向前或向后翻页

## 属性

·length

- 获取当前访问的链接数量

alert(history.length);

·back()

- 返回上一个页面

- 相当于浏览器的

history.back();

·forward()

- 跳转到下一个页面

- 相当于浏览器的

history.forward();

·go()

- 跳转到指定的页面

- 它需要一个整数作为参数

1表示向前跳转一个页面，相当于forward()

2表示向前跳转两个页面

-1表示向后跳转一个页面

history.go(-1);

<body>

<h1>HISTORY</h1>

<button id="btn1">获取当前访问的链接数量</button>

<button id="btn2">返回上一个页面</button>

<button id="btn3">跳转到下一个页面</button>

<button id="btn4">跳转到指定的页面</button>

<a href="https://www.baidu.com/">百度</a>

<script type="text/javascript">

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

btn1.onclick = **function** () {

alert(history.length);

};

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

btn2.onclick = **function** () {

history.back();

};

**var** btn3 = document.getElementById("btn3");

btn3.onclick = **function** () {

history.forward();

};

**var** btn4 = document.getElementById("btn4");

btn3.onclick = **function** () {

history.go(-1);

};

</script>

</body>

# Location

该对象封装了地址栏信息

·直接打印location可获取到地址栏信息（）当前页面的完整路径

alert(location);

·跳转页面

- 将location属性修改为一个完整的路径或相对路径，则我们页面会跳转到该路径。并且会生成相应的历史记录。

location = "https://www.baidu.com";

## 属性

·location.hostname 返回 web 主机的域名

·location.pathname 返回当前页面的路径和文件名

·location.port 返回 web 主机的端口 （80 或 443）

·location.protocol 返回所使用的 web 协议（http:// 或 https://）

·assign()

- 用来跳转到其他的页面，作用和直接修改location一样

location.assign("https://www.baidu.com");

·reload()

- 重新加载文档（刷新）

- 可传入参数true强制清空缓存式刷新（与ctrl+f5一样）

location.reload();

·replace()

- 使用新的页面替换当前页面，类似页面跳转，不会生成历史记录，返回不了。

<body>

<button id="btn1">百度一下</button>

<button id="btn2">刷新</button>

<script type="text/javascript">

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

btn1.onclick = **function** () {

// location = "https://www.baidu.com";

location.assign("https://www.baidu.com");

};

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

btn2.onclick = **function** () {

location.reload();

};

</script>

</body>

# 定时器

·setInterval()

- 定时调用

- 将一个函数每隔一段时间执行一次

- 参数：

-- 回调函数

-- 每次调用间隔时间，单位是毫秒

setInterval(time,1000);//每隔一秒执行一次time函数

-- 返回值：Number类型的数据，用来作为定时器的唯一标识

·clearInterval()

- 关闭定时器

- 方法中需要定时器的标识作为参数

clearInterval(timer);//当sum为15时关闭当前timer定时器

<body>

<h1 id="count"></h1>

<script type="text/javascript">

**var** count = document.getElementById("count");

**var** sum = 1;

**function** time(){

count.innerHTML = sum++;

**if** (sum == 15) {

clearInterval(timer);//当sum为15时关闭当前timer定时器

}

}

//使count中的内容自动切换

**var** timer = setInterval(time,1000);//每隔一秒执行一次time函数

</script>

</body>

## 切换图片练习

<body>

<div id="outer">

<img id="img1" src="https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="400px"/>

<button id="btn01">开始</button>

<button id="btn02">停止</button>

</div>

<script type="text/javascript">

//图片路径的数组

**var** imgs = ["https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg",

"https://img0.baidu.com/it/u=3243110409,1678992802&fm=26&fmt=auto&gp=0.jpg",

"https://img2.baidu.com/it/u=534486318,1094186657&fm=26&fmt=auto&gp=0.jpg",

"https://img0.baidu.com/it/u=1310634044,1597068079&fm=26&fmt=auto&gp=0.jpg",

"https://img1.baidu.com/it/u=2239891783,1520147040&fm=26&fmt=auto&gp=0.jpg"

];

**var** img1 = document.getElementById("img1");

//保存当前图片的索引

**var** index = 0;

**var** timer;

//为btn01绑定一个单机响应函数

**var** btn01 = document.getElementById("btn01");

btn01.onclick = **function** () {

*/\*\**

*\* 目前我们每点击一次开始就会开启一个定时器*

*\* 如果点击开始过多则会导致图片切换过快*

*\* 点停止也停不下来*

*\** ***@type*** *{HTMLElement}*

*\*/*

//在开启定时器前，将上一个定时器关闭

clearInterval(timer);//关闭当前timer定时器

timer = setInterval(imgChange,1500);//开启定时器

};

//为btn01绑定一个单机响应函数

**var** btn02 = document.getElementById("btn02");

btn02.onclick = **function** () {

clearInterval(timer);//关闭当前timer定时器

};

**function** imgChange() {

//当前索引+1

index++;

//判断index是否大于数组长度

// if (index>=imgs.length){

// index = 0;

// }

//写法2

index %= imgs.length;

//修改src属性

img1.src = imgs[index];

}

</script>

</body>

## 解决div移动卡顿问题

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>键盘移动div</title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: cadetblue;

position: absolute;

}

</style>

</head>

<body>

<div id="box1"></div>

<script type="text/javascript">

//开启一个定时器，来控制div移动

setInterval(moveDiv,30);

**var** dir = 0;//创建一个变量表示方向

**var** speed;

**function** moveDiv(){

**switch** (dir) {

**case** 37:

//向左，left值减小

box1.style.left = box1.offsetLeft - speed + "px";

**break**;

**case** 38:

//向上

box1.style.top = box1.offsetTop - speed + "px";

**break**;

**case** 39:

//向右

box1.style.left = box1.offsetLeft + speed + "px";

**break**;

**case** 40:

//向下

box1.style.top = box1.offsetTop + speed + "px";

**break**;

}

}

//为document绑定一个键盘按下事件

document.onkeydown = **function** (event) {

event = event || window.event;

//当用户按了ctrl以后，速度加快

**if** (event.ctrlKey) {

speed = 30;

}**else** {

speed = 10;

}

dir = event.keyCode

};

//当按键松开时，div不再移动

document.onkeyup = **function** () {

event = event || window.event;

dir = 0;

}

</script>

</body>

</html>

# 延时调用

调用一个函数，隔一段时间再执行，并且只执行一次

<body>

<script type="text/javascript">

**var** num = 1;

// //开启一个定时器

// setInterval(function () {

// console.log(num++);

// },3000);

**var** timer = setTimeout(**function** () {

console.log(num++);

},3000);

clearTimeout(timer);//关闭延时调用

</script>

</body>

延时调用和定时调用实际上可以互相替代，在开发中可以根据自己的需要去选择

# 定时器的应用

## 一

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>定时器的应用</title>

<style type="text/css">

\*{

margin: 0;

padding: 0;

}

#box1{

width: 100px;

height: 100px;

background-color: cadetblue;

position: absolute;

left: 0px;

}

</style>

</head>

<body>

<button id="btn01">点击按钮以后box1向右移动</button>

<br><br/>

<div id="box1"></div>

<div style="width: 0;height: 1000px;border-left: 1px black solid;position: absolute;left: 800px;top: 0;"></div>

</body>

<script type="text/javascript">

**var** box1 = document.getElementById("box1");

**var** btn01 = document.getElementById("btn01");

//定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

**function** getStyle(obj, name) {

**if** (window.getComputedStyle){//如果有该方法

**return** getComputedStyle(obj,**null**)[name];//正常浏览器具有getComputedStyle方法

}**else** {

**return** obj.currentStyle[name];//ie具有该方法

}

}

**var** timer;

btn01.onclick = **function** () {

//关闭上一个定时器

clearInterval(timer);

//开启一个定时器执行动画效果

timer = setInterval(**function** () {

//获取box1的值

**var** oldValue = parseInt(getStyle(box1,"left"));

//在旧值的基础上增加

**var** newValue = oldValue + 10;

//判断newValue是否大于等于800

**if** (newValue > 800){

newValue = 800;

}

//将新值设置给box1

box1.style.left = newValue + "px";

//当元素移动到800px时停止

**if** (newValue === 800) {

//达到目标，关闭定时器

clearInterval(timer);

}

},30)

}

</script>

</html>

## 二

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>定时器的应用</title>

<style type="text/css">

\*{

margin: 0;

padding: 0;

}

#box1{

width: 100px;

height: 100px;

background-color: cadetblue;

position: absolute;

left: 0px;

}

</style>

</head>

<body>

<button id="btn01">点击按钮以后box1向右移动</button>

<button id="btn02">点击按钮以后box1向左移动</button>

<br><br/>

<div id="box1"></div>

<div style="width: 0;height: 1000px;border-left: 1px black solid;position: absolute;left: 800px;top: 0;"></div>

</body>

<script type="text/javascript">

**var** box1 = document.getElementById("box1");

**var** btn01 = document.getElementById("btn01");

//定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

**function** getStyle(obj, name) {

**if** (window.getComputedStyle){//如果有该方法

**return** getComputedStyle(obj,**null**)[name];//正常浏览器具有getComputedStyle方法

}**else** {

**return** obj.currentStyle[name];//ie具有该方法

}

}

**var** timer;

//向右移

btn01.onclick = **function** () {

moveDiv(box1,800,10);

};

//向左移动

**var** btn02 = document.getElementById("btn02");

btn02.onclick = **function** () {

moveDiv(box1,0,10);

};

//往某个方向移动的函数

**function** moveDiv(obj,target,speed) {//参数1是移动的元素，2是停止位置，3是移动速度

//关闭上一个定时器

clearInterval(timer);

//获取元素目前位置判断往左移还是往右移动

**var** current = parseInt(getStyle(obj,"left"));

//判断速度的正负值

//如果target小于当前位置则向左移动

//如果target大于当前位置则向右移动

**if** (target < current) {

speed = -speed;

}

//开启一个定时器执行动画效果

timer = setInterval(**function** () {

//获取box1的值

**var** oldValue = parseInt(getStyle(obj,"left"));

//在旧值的基础上增加

**var** newValue = oldValue + speed;

//向左移动时，判断newValue是否小于target

//向右移动时，判断newValue是否大于target

**if** ((speed < 0 && newValue < target) || (speed > 0 && newValue > target)){

newValue = target;

}

//将新值设置给box1

obj.style.left = newValue + "px";

//当元素移动到800px时停止

**if** (newValue === target) {

//达到目标，关闭定时器

clearInterval(timer);

}

},30)

}

</script>

</html>

## 三

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>定时器的应用</title>

<style type="text/css">

\*{

margin: 0;

padding: 0;

}

#box1{

width: 100px;

height: 100px;

background-color: cadetblue;

position: absolute;

left: 0px;

}

#box2{

width: 100px;

height: 100px;

background-color: aquamarine;

position: absolute;

left: 0px;

top: 200px;

}

</style>

</head>

<body>

<button id="btn01">点击按钮以后box1向右移动</button>

<button id="btn02">点击按钮以后box1向左移动</button>

<button id="btn03">点击按钮以后box2向右移动</button>

<button id="btn04">测试按钮</button>

<br><br/>

<div id="box1"></div>

<div id="box2"></div>

<div style="width: 0;height: 1000px;border-left: 1px black solid;position: absolute;left: 800px;top: 0;"></div>

</body>

<script type="text/javascript">

**var** box1 = document.getElementById("box1");

**var** box2 = document.getElementById("box2");

**var** btn01 = document.getElementById("btn01");

//定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

**function** getStyle(obj, name) {

**if** (window.getComputedStyle){//如果有该方法

**return** getComputedStyle(obj,**null**)[name];//正常浏览器具有getComputedStyle方法

}**else** {

**return** obj.currentStyle[name];//ie具有该方法

}

}

*/\*\**

*\* 目前我们的定时器的标识由全局变量timer保存*

*\* 所有的正在执行的定时器都在这个变量中保存*

*\*/*

// var timer;

//box1向右移

btn01.onclick = **function** () {

moveDiv(box1,800,10,"left");

};

//box1向左移动

**var** btn02 = document.getElementById("btn02");

btn02.onclick = **function** () {

moveDiv(box1,0,10,"left");

};

//box2向左移动

**var** btn03 = document.getElementById("btn03");

btn03.onclick = **function** () {

moveDiv(box2,800,50,"left");

};

**var** btn04 = document.getElementById("btn04");

btn04.onclick = **function** () {

loop(box2);

};

//循环动画

**function** loop(obj) {

moveDiv(obj,800,10,"width",**function** () {

moveDiv(obj,400,10,"height",**function** () {

moveDiv(obj,0,10,"top",**function** () {

moveDiv(obj,100,10,"width",**function** () {

moveDiv(obj,200,10,"top",**function** () {

moveDiv(obj,100,10,"height",**function** () {

loop(obj);

}) ;

});

});

});

});

});

}

//往某个方向移动的函数

**function** moveDiv(obj,target,speed,attr,callback) {//参数1是移动的元素，2是停止位置，3是移动速度，4是要执行动画的样式，如left,5是回调函数，这个函数将会在动画执行完毕以后执行

//关闭上一个定时器

clearInterval(obj.timer);

//获取元素目前位置判断往左移还是往右移动

**var** current = parseInt(getStyle(obj,attr));

//判断速度的正负值

//如果target小于当前位置则向左移动

//如果target大于当前位置则向右移动

**if** (target < current) {

speed = -speed;

}

//开启一个定时器执行动画效果

obj.timer = setInterval(**function** () {//将定时器的标识保存到移动元素的属性中

//获取box1的值

**var** oldValue = parseInt(getStyle(obj,attr));

//在旧值的基础上增加

**var** newValue = oldValue + speed;

//向左移动时，判断newValue是否小于target

//向右移动时，判断newValue是否大于target

**if** ((speed < 0 && newValue < target) || (speed > 0 && newValue > target)){

newValue = target;

}

//将新值设置给box1

obj.style[attr] = newValue + "px";

//当元素移动到800px时停止

**if** (newValue === target) {

//达到目标，关闭定时器

clearInterval(obj.timer);

//动画执行完毕，调用回调函数

callback && callback();//如果有回调函数才执行

}

},30)

}

</script>

</html>

# 轮播图

## 轮播图界面

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>轮播图</title>

<style type="text/css">

\*{

margin: 0;

padding: 0;

}

#outer{

width: 520px;

height: 500px;

/\*居中\*/

margin: 50px auto;

/\*设置背景颜色\*/

background-color: aqua;

/\*设置padding\*/

padding: 10px 0;

/\*开启相对定位\*/

position: relative;

/\*裁剪溢出的内容\*/

overflow: hidden;

}

#imgList{

/\*去除项目符号\*/

list-style: none;

/\*设置ul的宽度\*/

/\*width: 2600px;\*/

/\*开启绝对定位，最好给父元素开启相对定位\*/

position: absolute;

/\*设置偏移量\*/

left: 0px;

}

#imgList li{

/\*设置浮动使li水平布局\*/

float: left;

/\*设置左右外边距\*/

margin: 0 10px;

}

#navDiv{

/\*开启绝对定位\*/

position: absolute;

/\*设置位置\*/

bottom: 10px;

/\*设置left值，outer宽度：520，navDiv宽度25\*5=125，520-125=395，再除以2等于197.5\*/

/\*left: 197px;\*/

}

#navDiv a{

/\*浮动，自动变成块元素\*/

float: left;

/\*设置宽高\*/

width: 15px;

height: 15px;

background-color: red;

/\*设置左右外边距\*/

margin: 0 5px;

/\*设置透明度\*/

opacity: 0.5;

/\*兼容ie8透明\*/

filter: alpha(opacity = 50);

}

/\*设置鼠标移入的效果\*/

#navDiv a:hover{

background-color: black;

}

</style>

</head>

<body>

<!--创建一个外部的div，来作为大的容器-->

<div id="outer">

<!--创建一个ul，用于放置图片-->

<ul id="imgList">

<li><img src="https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img0.baidu.com/it/u=3243110409,1678992802&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img2.baidu.com/it/u=534486318,1094186657&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img0.baidu.com/it/u=1310634044,1597068079&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img2.baidu.com/it/u=2089609337,3012499084&fm=11&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

</ul>

<!--创建导航按钮-->

<div id="navDiv">

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

</div>

</div>

<script type="text/javascript">

**var** imgList = document.getElementById("imgList");

//获取页面中所有图片，img标签

**var** imgArr = document.getElementsByTagName("img");

//设置imglist的宽度

imgList.style.width = 520 \* imgArr.length + "px";

//设置导航按钮居中

//设置left值，outer宽度：520，navDiv宽度25\*5=125，520-125=395，再除以2等于197.5

**var** navDiv = document.getElementById("navDiv");

**var** outer = document.getElementById("outer");

//设置navDiv的left的值

navDiv.style.left = (outer.offsetWidth - navDiv.offsetWidth)/2+"px";

//默认显示图片的索引

**var** index = 0;

//获取所有的a

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

//设置默认选择的效果

allA[index].style.backgroundColor = "black";

</script>

</body>

</html>

## 点击按钮切换图片

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>轮播图</title>

<style type="text/css">

\*{

margin: 0;

padding: 0;

}

#outer{

width: 520px;

height: 500px;

/\*居中\*/

margin: 50px auto;

/\*设置背景颜色\*/

background-color: aqua;

/\*设置padding\*/

padding: 10px 0;

/\*开启相对定位\*/

position: relative;

/\*裁剪溢出的内容\*/

overflow: hidden;

}

#imgList{

/\*去除项目符号\*/

list-style: none;

/\*设置ul的宽度\*/

/\*width: 2600px;\*/

/\*开启绝对定位，最好给父元素开启相对定位\*/

position: absolute;

/\*设置偏移量\*/

left: 0px;

}

#imgList li{

/\*设置浮动使li水平布局\*/

float: left;

/\*设置左右外边距\*/

margin: 0 10px;

}

#navDiv{

/\*开启绝对定位\*/

position: absolute;

/\*设置位置\*/

bottom: 10px;

/\*设置left值，outer宽度：520，navDiv宽度25\*5=125，520-125=395，再除以2等于197.5\*/

/\*left: 197px;\*/

}

#navDiv a{

/\*浮动，自动变成块元素\*/

float: left;

/\*设置宽高\*/

width: 15px;

height: 15px;

background-color: red;

/\*设置左右外边距\*/

margin: 0 5px;

/\*设置透明度\*/

opacity: 0.5;

/\*兼容ie8透明\*/

filter: alpha(opacity = 50);

}

/\*设置鼠标移入的效果\*/

#navDiv a:hover{

background-color: black;

}

</style>

</head>

<body>

<!--创建一个外部的div，来作为大的容器-->

<div id="outer">

<!--创建一个ul，用于放置图片-->

<ul id="imgList">

<li><img src="https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img0.baidu.com/it/u=3243110409,1678992802&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img2.baidu.com/it/u=534486318,1094186657&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img0.baidu.com/it/u=1310634044,1597068079&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img2.baidu.com/it/u=2089609337,3012499084&fm=11&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

</ul>

<!--创建导航按钮-->

<div id="navDiv">

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

</div>

</div>

<script type="text/javascript">

**var** imgList = document.getElementById("imgList");

//获取页面中所有图片，img标签

**var** imgArr = document.getElementsByTagName("img");

//设置imglist的宽度

imgList.style.width = 520 \* imgArr.length + "px";

//设置导航按钮居中

//设置left值，outer宽度：520，navDiv宽度25\*5=125，520-125=395，再除以2等于197.5

**var** navDiv = document.getElementById("navDiv");

**var** outer = document.getElementById("outer");

//设置navDiv的left的值

navDiv.style.left = (outer.offsetWidth - navDiv.offsetWidth)/2+"px";

//默认显示图片的索引

**var** index = 0;

//获取所有的a

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

//设置默认选择的效果

allA[index].style.backgroundColor = "black";

//点击超链接切换到指定的图片

//为所有的超链接绑定单机响应函数

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

//为每一个超链接添加编号

allA[i].num = i;

allA[i].onclick = **function** () {

//获取点击超链接的编号,并将其设置为index

index = **this**.num;

//切换图片

// imgList.style.left = -(520\*index)+"px";

//修改正在选中的图片

setA();

//使用moveDiv函数切换图片

moveDiv(imgList,-520\*index,30,"left",**function** () {

});

};

}

//创建一个方法，用来设置选中的a

**function** setA() {

//遍历所有的a，将他们内联样式清空

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

allA[i].style.backgroundColor = "";

}

//将选中的a设置为黑色

allA[index].style.backgroundColor = "black";

}

//定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

**function** getStyle(obj, name) {

**if** (window.getComputedStyle){//如果有该方法

**return** getComputedStyle(obj,**null**)[name];//正常浏览器具有getComputedStyle方法

}**else** {

**return** obj.currentStyle[name];//ie具有该方法

}

}

//元素往某个方向移动的函数

**function** moveDiv(obj,target,speed,attr,callback) {//参数1是移动的元素，2是停止位置，3是移动速度，4是要执行动画的样式，如left,5是回调函数，这个函数将会在动画执行完毕以后执行

//关闭上一个定时器

clearInterval(obj.timer);

//获取元素目前位置判断往左移还是往右移动

**var** current = parseInt(getStyle(obj,attr));

//判断速度的正负值

//如果target小于当前位置则向左移动

//如果target大于当前位置则向右移动

**if** (target < current) {

speed = -speed;

}

//开启一个定时器执行动画效果

obj.timer = setInterval(**function** () {//将定时器的标识保存到移动元素的属性中

//获取box1的值

**var** oldValue = parseInt(getStyle(obj,attr));

//在旧值的基础上增加

**var** newValue = oldValue + speed;

//向左移动时，判断newValue是否小于target

//向右移动时，判断newValue是否大于target

**if** ((speed < 0 && newValue < target) || (speed > 0 && newValue > target)){

newValue = target;

}

//将新值设置给box1

obj.style[attr] = newValue + "px";

//当元素移动到800px时停止

**if** (newValue === target) {

//达到目标，关闭定时器

clearInterval(obj.timer);

//动画执行完毕，调用回调函数

callback && callback();//如果有回调函数才执行

}

},30)

}

</script>

</body>

</html>

## 完成轮播图

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>轮播图</title>

<style type="text/css">

\*{

margin: 0;

padding: 0;

}

#outer{

width: 520px;

height: 500px;

/\*居中\*/

margin: 50px auto;

/\*设置背景颜色\*/

background-color: aqua;

/\*设置padding\*/

padding: 10px 0;

/\*开启相对定位\*/

position: relative;

/\*裁剪溢出的内容\*/

overflow: hidden;

}

#imgList{

/\*去除项目符号\*/

list-style: none;

/\*设置ul的宽度\*/

/\*width: 2600px;\*/

/\*开启绝对定位，最好给父元素开启相对定位\*/

position: absolute;

/\*设置偏移量\*/

left: 0px;

}

#imgList li{

/\*设置浮动使li水平布局\*/

float: left;

/\*设置左右外边距\*/

margin: 0 10px;

}

#navDiv{

/\*开启绝对定位\*/

position: absolute;

/\*设置位置\*/

bottom: 10px;

/\*设置left值，outer宽度：520，navDiv宽度25\*5=125，520-125=395，再除以2等于197.5\*/

/\*left: 197px;\*/

}

#navDiv a{

/\*浮动，自动变成块元素\*/

float: left;

/\*设置宽高\*/

width: 15px;

height: 15px;

background-color: red;

/\*设置左右外边距\*/

margin: 0 5px;

/\*设置透明度\*/

opacity: 0.5;

/\*兼容ie8透明\*/

filter: alpha(opacity = 50);

}

/\*设置鼠标移入的效果\*/

#navDiv a:hover{

background-color: black;

}

</style>

</head>

<body>

<!--创建一个外部的div，来作为大的容器-->

<div id="outer">

<!--创建一个ul，用于放置图片-->

<ul id="imgList">

<li><img src="https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img0.baidu.com/it/u=3243110409,1678992802&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img2.baidu.com/it/u=534486318,1094186657&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img0.baidu.com/it/u=1310634044,1597068079&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<li><img src="https://img2.baidu.com/it/u=2089609337,3012499084&fm=11&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

<!--这张用来存下一张图-->

<li><img src="https://img2.baidu.com/it/u=3724265679,1678217865&fm=26&fmt=auto&gp=0.jpg" alt = "iu" width="500px"/></li>

</ul>

<!--创建导航按钮-->

<div id="navDiv">

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

<a href="javascript:;"></a>

</div>

</div>

<script type="text/javascript">

**var** timer;//自动切换定时器的标识

**var** imgList = document.getElementById("imgList");

//获取页面中所有图片，img标签

**var** imgArr = document.getElementsByTagName("img");

//设置imglist的宽度

imgList.style.width = 520 \* imgArr.length + "px";

//设置导航按钮居中

//设置left值，outer宽度：520，navDiv宽度25\*5=125，520-125=395，再除以2等于197.5

**var** navDiv = document.getElementById("navDiv");

**var** outer = document.getElementById("outer");

//设置navDiv的left的值

navDiv.style.left = (outer.offsetWidth - navDiv.offsetWidth)/2+"px";

//默认显示图片的索引

**var** index = 0;

//获取所有的a

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

//设置默认选择的效果

allA[index].style.backgroundColor = "black";

//点击超链接切换到指定的图片

//为所有的超链接绑定单机响应函数

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

//为每一个超链接添加编号

allA[i].num = i;

allA[i].onclick = **function** () {

//关闭自动切换的定时器

clearInterval(timer);

//获取点击超链接的编号,并将其设置为index

index = **this**.num;

//切换图片

// imgList.style.left = -(520\*index)+"px";

//修改正在选中的图片

setA();

//使用moveDiv函数切换图片

moveDiv(imgList,-520\*index,30,"left",**function** () {

//动画执行完之后开启自动切换

autoChange();

});

};

}

//开启自动切换图片

autoChange();

//创建一个函数，开启自动切换图片

**function** autoChange() {

//开启定时器定时切换图片

timer = setInterval(**function** () {

//使索引自增

index++;

//判断index的值

index %= imgArr.length;

//执行动画切换图片

moveDiv(imgList,-520\*index,30,"left",**function** () {

//修改导航点

setA();

});

},3000);

}

//创建一个方法，用来设置选中的a

**function** setA() {

//判断当前索引是否是最后一张图片

**if** (index >= imgArr.length-1) {

//则将index设置为0

index = 0;

//此时显示的是最后一张图片，而最后一张图片和第一张是一样的

//通过CSS将最后一张切换成第一张

imgList.style.left = 0;

}

//遍历所有的a，将他们内联样式清空

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

allA[i].style.backgroundColor = "";

}

//将选中的a设置为黑色

allA[index].style.backgroundColor = "black";

}

//定义一个函数，传入元素和样式名获取指定元素的当前样式,使所有浏览器都支持

**function** getStyle(obj, name) {

**if** (window.getComputedStyle){//如果有该方法

**return** getComputedStyle(obj,**null**)[name];//正常浏览器具有getComputedStyle方法

}**else** {

**return** obj.currentStyle[name];//ie具有该方法

}

}

//元素往某个方向移动的函数

**function** moveDiv(obj,target,speed,attr,callback) {//参数1是移动的元素，2是停止位置，3是移动速度，4是要执行动画的样式，如left,5是回调函数，这个函数将会在动画执行完毕以后执行

//关闭上一个定时器

clearInterval(obj.timer);

//获取元素目前位置判断往左移还是往右移动

**var** current = parseInt(getStyle(obj,attr));

//判断速度的正负值

//如果target小于当前位置则向左移动

//如果target大于当前位置则向右移动

**if** (target < current) {

speed = -speed;

}

//开启一个定时器执行动画效果

obj.timer = setInterval(**function** () {//将定时器的标识保存到移动元素的属性中

//获取box1的值

**var** oldValue = parseInt(getStyle(obj,attr));

//在旧值的基础上增加

**var** newValue = oldValue + speed;

//向左移动时，判断newValue是否小于target

//向右移动时，判断newValue是否大于target

**if** ((speed < 0 && newValue < target) || (speed > 0 && newValue > target)){

newValue = target;

}

//将新值设置给box1

obj.style[attr] = newValue + "px";

//当元素移动到800px时停止

**if** (newValue === target) {

//达到目标，关闭定时器

clearInterval(obj.timer);

//动画执行完毕，调用回调函数

callback && callback();//如果有回调函数才执行

}

},30)

}

</script>

</body>

</html>

# 类的操作

通过style属性来修改元素的样式，每修改一个样式，浏览器就需要重新渲染一次页面。

这样执行的性能是比较差的，而且这种形式当我们要修改多个样式值时，也不太方便。

box.style.width = "200px";

希望一行代码可以同时修改多个样式，修改box的class属性，可间接同时修改多个样式，浏览器只需渲染一次页面，性能较好，并且可以使表现和行为分类。

box.className = "b2";//修改box的class属性，可间接同时修改多个样式，浏览器只需渲染一次页面

在原有样式的基础上加上别的样式

box.className += " b2"//在b1的基础上加上b2的样式

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>类的操作</title>

<style type="text/css">

.b1{

width: 100px;

height: 100px;

background-color: cadetblue;

}

.b2{

width: 200px;

height: 200px;

background-color: aquamarine;

}

</style>

</head>

<body>

<button id="btn01">增加box的样式</button>

<button id="btn02">删除box的样式</button>

<button id="btn03">切换box的样式</button>

<br/>

<br/>

<div id="box" class="b1"></div>

<script type="text/javascript">

//获取box

**var** box = document.getElementById("box");

//获取btn01

**var** btn01 = document.getElementById("btn01");

btn01.onclick = **function** () {

// box.className = "b2";//修改box的class属性，可间接同时修改多个样式，浏览器只需渲染一次页面

// box.className += " b2"//在b1的基础上加上b2的样式

addClass(box,"b2");

};

//获取btn02

**var** btn02 = document.getElementById("btn02");

btn02.onclick = **function** () {

removeClass(box,"b2");

};

//获取btn03

**var** btn03 = document.getElementById("btn03");

btn03.onclick = **function** () {

toggleClass(box,"b2");

};

//定义一个函数，用来向一个元素添加指定的class属性值

//第一个参数：要添加class属性的元素

//第二个参数:要添加的class值

**function** addClass(obj,cn) {

//检查obj中是否含有cn,如果没有则添加

**if** (!hasClass(obj, cn)) {

obj.className += " " + cn;

}

}

//删除一个元素中指定的class属性

**function** removeClass(obj, cn) {

//创建一个正则表达式

**var** reg = **new** RegExp("\\b "+cn+"\\b");

//删除class

obj.className = obj.className.replace(reg , "");//用空串替换掉cn

}

//切换一个类，如果有则删除，没有则添加

**function** toggleClass(obj,cn) {

//判断obj是否含有cn

**if** (hasClass(obj, cn)) {

removeClass(obj,cn);

}**else**{

addClass(obj,cn);

}

}

//判断一个元素中是否含有指定的class属性值，如果有则返回true

//第一个参数：要添加class属性的元素

//第二个参数:要添加的class值

**function** hasClass(obj,cn) {

//判断obj中有没有cn class

//创建一个正则表达式

// var reg = /\bb2\b/;//\b是单词边界，类似空格

**var** reg = **new** RegExp("\\b"+cn+"\\b");

**return** reg.test(obj.className);

}

</script>

</body>

</html>

# 二级菜单

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>二级菜单</title>

<style type="text/css">

\* {

margin: 0;

padding: 0;

list-style-type: none;

}

a,img {

border: 0;

text-decoration: none;

}

body {

font: 12px/180% Arial, Helvetica, sans-serif, "新宋体";

}

</style>

<link rel="stylesheet" type="text/css" href="css/sdmenu.css" />

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

</head>

<body>

<div id="my\_menu" class="sdmenu">

<div>

<span class="menuSpan">在线工具</span>

<a href="#">图像优化</a>

<a href="#">收藏夹图标生成器</a>

<a href="#">邮件</a>

<a href="#">htaccess密码</a>

<a href="#">梯度图像</a>

<a href="#">按钮生成器</a>

</div>

<div class="collapsed">

<span class="menuSpan">支持我们</span>

<a href="#">推荐我们</a>

<a href="#">链接我们</a>

<a href="#">网络资源</a>

</div>

<div class="collapsed">

<span class="menuSpan">合作伙伴</span>

<a href="#">JavaScript工具包</a>

<a href="#">CSS驱动</a>

<a href="#">CodingForums</a>

<a href="#">CSS例子</a>

</div>

<div class="collapsed">

<span class="menuSpan">测试电流</span>

<a href="#">Current or not</a>

<a href="#">Current or not</a>

<a href="#">Current or not</a>

<a href="#">Current or not</a>

</div>

</div>

</body>

<script type="text/javascript">

*/\*\**

*\* 我们的每个菜单都是一个div*

*\* 当div具有collapsed这个类时，div就是折叠的状态*

*\* 当div没有这个类时，div就是展开状态*

*\*/*

//点击菜单，切换菜单的显示状态

//获取所有的class为menuSpan的元素

**var** menuSpan = document.querySelectorAll(".menuSpan");

//定义一个变量，来保存上一次点击的菜单

**var** openDiv = menuSpan[0].parentNode;

//为span绑定单机响应函数

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

menuSpan[i].onclick = **function** () {

//获取当前span的父元素

**var** parentDiv = **this**.parentNode;

//切换点击的菜单

toggleMenu(parentDiv);

//判断当前打开的openDiv和要打开的parentDiv是否相同

**if** (openDiv != parentDiv && !hasClass(openDiv,"collapsed")) {//如果上一次点击的菜单和现在点击的菜单不同，并且上一次点击的菜单没有关闭，就关闭菜单

//打开这个菜单以后应该关闭之前的菜单

// addClass(openDiv,"collapsed");

//为了更好的处理动画效果，将addClass改为toggleClass

// toggleClass(openDiv,"collapsed");

//切换上一次点击的菜单

toggleMenu(openDiv);

}

//保存上一次点击的菜单

openDiv = parentDiv;

}

}

//菜单切换的方法和动画效果

**function** toggleMenu(obj) {

//获取当前元素的高度

**var** begin = obj.offsetHeight;

//obj,切换class

toggleClass(obj,"collapsed");

//获取切换之后的高度

**var** end = obj.offsetHeight;

// console.log("begin = "+begin," end = "+end);

//动画效果就是将高度从begin向end过渡

//将元素的高度重置为begin

obj.style.height = begin +"px";

//执行动画，从begin向end过渡,修改的是内联样式

move(obj,"height",end,10,**function** () {

//当动画执行完时，删除内联样式

obj.style.height = **null**;

});

}

</script>

</html>

# JSON

·JS中的对象只有JS认识，其他语言都不认识

·JSON就是一个特殊格式的字符串，这个字符串可以被任意的语言所识别，并且可以转换为任意语言中的对象，JSON在开发中主要用来数据的交互

·JSON即JavaScript Object Notation（JS对象标记法）

·JSON的格式和JS对象的格式一样，只不过JSON字符串中属性名必须加双引号，其他的和JS语法一致

·JSON分类：

- JSON对象

- JSON数组

·JSON允许的值

- 字符串

- 数值

- 布尔值

- null

- 对象（不包含函数）

- 数组

·将JSON字符串转换为JS对象

- 在JS中，为我们提供了JSON工具类

- JSON对象可以帮我们将一个JSON转换为JS对象，也可以将一个JS对象转换为JSON

·JSON.parse()

- 将json转换为js对象

- 它需要一个JSON字符串作为参数，会将该字符串转换为JS对象并返回

// alert(JSON);

**var** json = '{"name":"孙悟空","age":18,"gender":"男"}';

//将json转换为js对象

**var** o = JSON.parse(json);

console.log(o);

- 将json转换为js数组

**var** arr = '[1,2,3,"hello",true]';

**var** o2 = JSON.parse(arr);

console.log(o2);

·JSON.stringify()

- 将JS对象转换为JSON

- 需要一个js对象作为参数，会返回一个JSON字符串

**var** obj3 = {name:"猪八戒" , age:28 , gender:"男"};

**var** str = JSON.stringify(obj3);

console.log(str);

<script type="text/javascript">

// alert(JSON);

**var** json = '{"name":"孙悟空","age":18,"gender":"男"}';

//将json转换为js对象

**var** o = JSON.parse(json);

console.log(o);

//将json转换为js数组

**var** arr = '[1,2,3,"hello",true]';

**var** o2 = JSON.parse(arr);

console.log(o2);

//将JS对象转换为JSON

**var** obj3 = {name:"猪八戒" , age:28 , gender:"男"};

**var** str = JSON.stringify(obj3);

console.log(str);

</script>

·JSON这个对象在IE7及以下的浏览器中不支持，所以在这些浏览器中调用时会报错

·eval()

- 该函数可以执行一段字符串形式的JS代码，并将执行结果返回

**var** str2 = "alert('hello');";

eval(str2);//执行以上代码

- 如果使用eval()执行的字符串中含有{},它会将{}当成是代码块

- 如果不希望将其当成代码块解析，则需要在字符串前后各加一个()

**var** str = '{"name":"孙悟空","age":18,"gender":"男"}';

**var** obj = eval("("+str+")");

console.log(obj);

- eval()这个函数的功能很强大，可以直接执行一个字符串中的js代码，但是在开发中尽量不要使用，首先它的执行性能比较差，然后它还具有安全隐患。

·如果要兼容ie7及以下浏览器的JSON操作，则可以引用外部文件json2,js来处理

<script type="text/javascript" src="js/json2.js"></script>

<!--如果要兼容ie7及以下浏览器的JSON操作，则可以引用外部文件json2,js来处理-->

<script type="text/javascript" src="js/json2.js"></script>

<script type="text/javascript">

// alert(JSON);

**var** json = '{"name":"孙悟空","age":18,"gender":"男"}';

//将json转换为js对象

**var** o = JSON.parse(json);

console.log(o);

//将json转换为js数组

**var** arr = '[1,2,3,"hello",true]';

**var** o2 = JSON.parse(arr);

console.log(o2);

//将JS对象转换为JSON

**var** obj3 = {name:"猪八戒" , age:28 , gender:"男"};

**var** str = JSON.stringify(obj3);

console.log(str);

</script>