第4章 、JavaScript

4.1、JavaScript 的历史

4.1.1、JavaScript 的历史

1992年底，美国国家超级电脑应用中心（NCSA）开始开发一个独立的浏览器，叫做Mosaic。这是人类历史上第一个浏览器，从此网页可以在图形界面的窗口浏览。但是该浏览器还没有面向大众的普通用户。

1994年10月，NCSA的一个主要程序员Jim Clark，成立了一家Netscape通信公司。这家公司的方向，就是在Mosaic的基础上，开发面向普通用户的新一代的浏览器Netscape Navigator。

1994年12月，Navigator发布了1.0版，市场份额一举超过90%。

Netscape 公司很快发现，Navigator浏览器需要一种可以嵌入网页的脚本语言，用来控制浏览器行为，因为当时，网速很慢而且上网费很贵，有些操作不宜在服务器端完成。比如，如果用户忘记填写“用户名”，就点了“发送”按钮，到服务器再发现这一点就有点太晚了，最好能在用户发出数据之前，就告诉用户“请填写用户名”。这就需要在网页中嵌入小程序，让浏览器检查每一栏是否都填写了。

管理层对这种浏览器脚本语言的设想是：功能不需要太强，语法较为简单，容易学习。

1995年5月，Netscape 公司只用了10天，就设计完成了这种语言的第一版。

Netscape 公司的这种浏览器脚本语言，最初名字叫做 Mocha，1995年9月改为LiveScript。12月，Netscape公司与Sun公司（Java语言的发明者和所有者）达成协议，后者允许将这种语言叫做JavaScript。这样一来，Netscape公司可以借助Java语言的声势，而Sun公司则将自己的影响力扩展到了浏览器，并不是因为JavaScript本身与Java语言有多么深的关系才叫做JavaScript。

1996年3月，Navigator 2.0 浏览器正式内置了 JavaScript 脚本语言，也就是可以直接在浏览器中运行 JavaScript 脚本语言。

4.2、JS的引入方式

```
1 直接编写
    <script>
        console.log('hello bobo')
    </script>
2 导入文件
    <script src="hello.js"></script>
```

- hello.js

```
function add(x, y) {
    return x + y
}
```

4.3、JS基本语法

js是一门弱类型的编程语言,属于基于对象和基于原型的脚本语言.

- 变量

```
格式:
      // 方式1 先声明再赋值
      var 变量名;   // 声明的变量如果没有进行赋值,或者没有被定义的变量,值默认是undefined
      变量名 = 变量值;

      // 方式2 声明并赋值
      var 变量名 = 变量值;

    // 方式3 一行可以声明多个变量.并且可以是不同类型
    var name="bobo", age=20, job="singer";
```

> 1、声明变量时 可以不用var. 如果不用var 那么它是全局变量
>
> 2、变量命名,首字符只能是字母,下划线，余下的字符可以是下划线或任何字母或数字字符且区分大小写

- 注释

```
// 单行注释

 /*
   多行注释
 */
```

- 语句分隔符

```
var a = 1;   // 分号和换行符作为语句分隔符号
var b = 2;
console.log(a,b);
```

4.4、JS 基本数据类型

4.4.1、数字类型

JavaScript 没有整型和浮点型，只有一种数字类型，即number类型。

```
var x = 10;
var y = 3.14;
console.log(x,typeof x);  // 10 "number"
console.log(y,typeof y);  // 3.14 "number"
```

4.4.2、字符串

字符串创建(两种方式)

- 变量 = “字符串”
- 字串对象名称 = new String (字符串)

```
var str1="hello world"; //推荐
var str1= new String("hello word");
```

```

// 字符串对象的操作
var str = "hello"; // 这就是字符串对象
console.log(str);

// 字符串对象内置属性
// length 计算字符串的长度
console.log( str.length );

// 字符串对象内置方法
// toUpperCase();  字母大写转换
// toLowerCase();  字母小写转换

console.log( str.toUpperCase() );
console.log( str.toLowerCase() );

// indexOf 获取指定字符在字符串中第一次出现的索引位置
// 字符串也有下标,也可以使用中括号来提取字符串的指定字符
console.log(str[1]); // e
console.log( str.indexOf("e") ); // 1

// match  正则匹配
// js中也存在正则,正则的使用符号和python里面是一样的
//语法：/正则表达式主体/修饰符(可选)
//修饰符：
    //i:执行对大小写不敏感的匹配。
    //g:执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。

var str = "我的电话是: 13312345678,你的电话: 13512345678";
var ret = str.match(/\d{11}/g); // 匹配,提取数据
console.log(ret);

// replace  正则替换
var str = "我的电话是: 13512345678";
var ret = str.replace(/(\d{3})\d{4}(\d{4})/,"$1****$2"); //正则的捕获模式  $1$2表示的正则中第一个和第二个小括号捕获的内容
console.log(ret);
// search  正则查找,如果查找不到,则返回-1
var str = "hello";
var ret = str.search(/l/);
console.log(ret);

// 切片,当前方法支持使用负数代表倒数下标
// slice(开始下标)    从开始位置切到最后
```

```
// slice(开始下标,结束下标)  从开始下标切到指定位置之前
var str = "helloworld";
var ret = str.slice(3,6); // 开区间,不包含结束下标的内容
console.log(ret); // low
var ret = str.slice(5);
console.log(ret); // world
var ret = str.slice(2,-1);
console.log(ret); // lloworl


// split   正则分割,经常用于把字符串转换成数组
var str = "广东-深圳-南山";
var ret = str.split("-");
console.log( ret );

// substr  截取
var str = "hello world";
var ret = str.substr(0,3);
console.log(ret); // hel

// trim    移除字符串首尾空白
var password = "    ge llo   ";
var ret = password.trim();
console.log(password.length); // 13
console.log(ret.length);  // 6
```

4.4.3、布尔值

> 1、Boolean类型仅有两个值：true和false，也代表1和0，实际运算中true=1,false=0
> 2、Boolean值主要用于JavaScript的控制语句

```
console.log(true);
console.log(false);
console.log(typeof true);
console.log(true == 1);
console.log(true + 1);
console.log(false + 1);
```

4.4.4、空值（Undefined）

- undefined类型

undefined类型只有一个值，即 undefined。

(1) 当声明的变量未初始化时，该变量的默认值是 undefined。

(2)当函数无明确返回值时，返回的也是值 undefined;

4.4.5、类型转换

js中,类型转换有2种.一种就是强制转换,一种就是自动转换.

> 因为js是一门弱类型的脚本语言,所以变量会在运算符的运行要求,有时候根据运算符的要求,进行自动转换的.

- 强制转换

```
 // 1. 转换数据为数值类型
// parseInt      把数据转换成整数
// parseFloat    把数据转换成小数
var box1 = "100";      // 转换会成功
var ret = parseInt(box1);
console.log(box1);
console.log(ret);
//
var box2 = "3.14";
console.log( parseFloat(box2) ); // 3.14
// 对于转换数值,如果转换失败的话,则结果为 NaN ,是 Not a Number ,但是NaN的类型也是number类型

// 2. 转换数据为字符串
  // 变量.toString()
  // String(数据)
var box4 = 3.14;
var ret = box4.toString();
console.log(ret);
```

- 自动转换

```
// 所谓的自动转换,其实弱类型中的变量会根据当前代码的需要,进行类型的自动隐式转化
var box1 = 1 + true;
// true 转换成数值,是1, false转换成数值,是0
console.log(box1); // 2

var box2 = 1 + "200";
console.log(box2);
// ‘1200’ 原因是,程序中+的含义有2种,第一: 两边数值相加, 第二: 两边字符串拼接.但是在js中运算符的优先级中, 字符串拼接的优先级要高于正数
// 值的加减乘除,所以解析器优先使用了+号作为了字符串的拼接符号了,因为程序就需要+号两边都是字符串才能完成运算操作,因此1变成字符串了。最终的结果就是 "1" +"200"

var box3 = 1 - "200";
console.log(box3); // -199;因为-号中表示的就是左边的数值减去右边的数值,因此程序就会要求"200"是数值,因此内部偷偷的转换了一下
```

4.5、运算符

- 运算符

```
/*
//算术运算符
   +    数值相加
   -    数值相减
   *    数值相乘
   /    数值相除
   %    数值求余
   **   数值求幂
   a++ 变量被使用后自增1
     var a = 10
```

```
      print(a++) 输出结果为10
      print(a) 就是11
   ++a 变量被使用前自增1
      var b = 10
      print(++b) 输出的就是11
   b-- 变量被使用后自减1
   --b 变量被使用前自减1


<script>
    var num = 10;
    //后++：先使用后加1,先使用就是说先把num的值赋值给count，然后在给num进行自增1
    var count = num ++  //var count = num; num += 1;
    console.log(count,num) //count=10 num=11
</script>

<script>
    var num = 10;
    //前++：先加1，后使用
    var count = ++num  //num += 1; var count = num;
    console.log(count,num) //count=11 num=11
</script>


//赋值运算符
   =
   +=
   -=
   *=
   /=
   %=
   **=

//比较运算符,比较的结果要么是true, 要么是false
    >   大于
    <   小于
    >=  大于或者等于
    <=  小于或者等于
    !=  不等于[计算数值]
    ==  等于[计算]

  !== 不全等[不仅判断数值,还会判断类型是否一致]
    === 全等[不仅判断数值,还会判断类型是否一致]

//逻辑运算符
  &&   并且  and    两边的运算结果为true,最终结果才是true
  ||   或者  or     两边的运算结果为false,最终结果才是false
  !    非    not    运算符的结果如果是true,则最终结果是false ,反之亦然.


//条件运算符[三目运算符]
     条件?true:false
     例如:
          var age = 12;
        var ret = age>=18?"成年":"未成年";
        console.log(ret);
```

```
 */
```

4.6、流程控制语句

编程语言的流程控制分为三种：

- 顺序结构(从上向下顺序执行)
- 分支结构
- 循环结构

之前我们学习的方式就是顺序执行，即代码的执行从上到下，一行行分别执行。

例如：

```
console.log("星期一");
console.log("星期二");
console.log("星期三");
```

4.6.1、分支结构

- if 分支语句

```
if(条件){
    // 条件为true时,执行的代码
  }

  if(条件){
    // 条件为true时,执行的代码
  }else{
    // 条件为false时,执行的代码
  }

  if(条件1){
    // 条件1为true时,执行的代码
  }else if(条件2){
    // 条件2为true时,执行的代码

  }....

  }else{
    // 上述条件都不成立的时候,执行的代码
  }
```

- switch语句

```
switch(条件){
      case 结果1:
            满足条件执行的结果是结果1时,执行这里的代码..
            break;
      case 结果2:
            满足条件执行的结果是结果2时,执行这里的代码..
            break;
      .....
      default:
            条件和上述所有结果都不相等时,则执行这里的代码
   }
```

> 1、switch比if else更为简洁
>
> 2、执行效率更高。switch...case会生成一个跳转表来指示实际的case分支的地址，而这个跳转表的索引号与switch变量的值是相等的。从而，switch...case不用像if...else那样遍历条件分支直到命中条件，而只需访问对应索引号的表项从而到达定位分支的目的。
>
> 3、到底使用哪一个选择语句，代码环境有关，如果是范围取值，则使用if else语句更为快捷；如果是确定取值，则使用switch是更优方案。

4.6.2、循环语句

- while循环

```
   while(循环的条件){
      // 循环条件为true的时候,会执行这里的代码
   }
```

循环案例：

```
var count = 0
while (count<10){
    console.log(count);
    count++;
}
```

- for循环

```
   // 循环三要素
   for(1.声明循环的开始; 2.条件; 4. 循环的计数){
      // 3. 循环条件为true的时候,会执行这里的代码
   }

   for(循环的成员下标 in 被循环的数据){
      // 当被循环的数据一直没有执行到最后下标,都会不断执行这里的代码
   }
```

循环案例：

```
// 方式1
for (var i = 0;i<10;i++){
    console.log(i)
}

// 方式2
var arr = [111,222,333]
for (var i in arr){
    console.log(i,arr[i])
}
```

- 退出循环（break和continue）

```
 for (var i = 0;i<100;i++){
           if (i===88){
               continue  // 退出当次循环
               // break  // 退出当前整个循环
           }
           console.log(i)
       }
```

4.7、数组对象

- 创建数组

```
创建方式1:
var arrname = [元素0,元素1,….];          // var arr=[1,2,3];

创建方式2:
var arrname = new Array(元素0,元素1,….); // var test=new Array(100,"a",true);
```

- 数组方法

```
var arr = ["A","B","C","D"];
// 内置属性
console.log( arr.length );
// 获取指定下标的成员
console.log( arr[3] ); // D
console.log( arr[arr.length-1] ); // 最后一个成员

// (1) pop()  出栈,删除最后一个成员作为返回值
var arr = [1,2,3,4,5];
var ret = arr.pop();
console.log(arr); // [1, 2, 3, 4]
console.log(ret); // 5


// (2) push() 入栈,给数组后面追加成员
var arr = [1,2,3,4,5];
arr.push("a");
console.log(arr); // [1, 2, 3, 4, 5, "a"]
```

```
// (3) shift是将数组的第一个元素删除
var arr = [1,2,3,4,5];
arr.shift()
console.log(arr); // [2, 3, 4, 5]

// (4) unshift是将value值插入到数组的开始
var arr = [1,2,3,4,5];
arr.unshift("bobo")
console.log(arr); // ["bobo",1,2, 3, 4, 5]


// (5) reverse() 反转排列
var arr = [1,2,3,4,5];
arr.reverse();
console.log(arr); // [5, 4, 3, 2, 1]

// (6) slice(开始下标,结束下标)  切片,开区间
var arr = [1,2,3,4,5];
console.log(arr.slice(1,3));


// (7) concat() 把2个或者多个数组合并
var arr1 = [1,2,3];
var arr2 = [4,5,7];
var ret = arr1.concat(arr2);
console.log( ret );


// (8) join()  把数组的每一个成员按照指定的符号进行拼接成字符串
var str = "广东-深圳-南山";
var arr = str.split("-");
console.log( arr ); // ["广东", "深圳", "南山"];

var arr1 = ["广东", "深圳", "南山"];
var str1 = arr1.join("-");
console.log( str1 ); // 广东-深圳-南山
```

- 遍历数组

```
var arr = [12,23,34]
for (var i in arr){
      console.log(i,arr[i])
}
```

4.8、Object对象

8.1、object对象的基本操作

Object 的实例不具备多少功能，但对于在应用程序中存储和传输数据而言，它们确实是非常理想的选择。

创建 Object 实例的方式有两种。

```
var person = new Object();
person.name = "alvin";
person.age = 18;
```

另一种方式是使用对象字面量表示法。对象字面量是对象定义的一种简写形式，目的在于简化创建包含大量属性的对象的过程。下面这个例子就使用了对象字面量语法定义了与前面那个例子中相同的person 对象：

```
var person = {
                name : "alvin",
                age : 18,
                            say: function(){
                      alert(123);
                  }
              };
```

- object可以通过. 和 []来访问。

```
console.log(person["age"]);
console.log(person.age)
```

- object可以通过for循环遍历

```
 for (var attr in person){
           console.log(attr,person[attr]);
      }
```

8.2、json序列化和反序列化

JSON：JavaScript 对象表示法，是一种轻量级的数据交换格式。易于人阅读和编写。

```
// json是一种数据格式, 语法一般是{}或者[]包含起来
// 内部成员以英文逗号隔开,最后一个成员不能使用逗号!
// 可以是键值对,也可以是列表成员
// json中的成员如果是键值对,则键名必须是字符串.而json中的字符串必须使用双引号圈起来
// json数据也可以保存到文件中,一般以".json"结尾.


{
   "name": "xiaoming",
     "age":12
}

[1,2,3,4]

{
   "name": "xiaoming",
     "age":22,
   "sex": true,
   "son": {
```

```
        "name":"xiaohuihui",
        "age": 2
    },
    "lve": ["篮球","唱","跳"]
}
```

js中也支持序列化和反序列化的方法：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<script>
    // js对象,因为这种声明的对象格式很像json,所以也叫json对象
    var data = {
        name: "xiaoming",
        age: 22,
        say: function(){
            alert(123);
        }
    };

    // 把对象转换成json字符串
    var ret = JSON.stringify(data);
    console.log(ret ); // {"name":"xiaoming","age":22}

    // 把json字符串转换成json对象
    var str = `{"name":"xiaoming","age":22}`;
    var ret = JSON.parse(str);
    console.log(ret);
</script>
</body>
</html>
```

4.9、Date对象

- 创建Date对象

```
//方法1：不指定参数
var nowd1=new Date(); //获取当前时间
console.log(nowd1.toLocaleString( ));
//方法2：参数为日期字符串
var d2=new Date("2004/3/20 11:12");
console.log(d2.toLocaleString( ));
var d3=new Date("04/03/20 11:12");
console.log(d3.toLocaleString( ));

//时间戳
var now=Date.parse(new Date()); //获取当前时间的时间戳
console.log(now); //js的时间戳是一个13位的整数
//python的时间戳是一个10位的整数+6位小数
```

```
//如何将python的时间戳转换成js的时间戳：int((python时间戳)*1000)
```

- 获取时间信息

```
获取日期和时间
getDate()                   获取日
getDay ()                   获取星期
getMonth ()                 获取月(0-11)
getFullYear ()              获取完整年份
getHours ()                 获取小时
getMinutes ()               获取分钟
getSeconds ()               获取秒
getMilliseconds ()          获取毫秒
```

4.10、Math对象

```
// Math对象的内置方法
// abs(x)  返回数值的绝对值
var num = -10;
console.log( Math.abs(num) ); // 10

// ceil(x)  向上取整
var num = 10.3;
console.log( Math.ceil(num) ); // 11

// floor(x) 向下取整
var num = 10.3;
console.log( Math.floor(num) ); // 10

// max(x,y,z,...,n)
console.log( Math.max(3,56,3) ); // 56
// min(x,y,z,...,n)


// random()  生成0-1随机数
console.log( Math.random() );

// 生成0-10之间的数值
console.log( Math.random() * 10 );

// round(x) 四舍五入
// 生成0-10之间的整数
console.log( Math.round( Math.random() * 10 ) );
```

4.11、Function 对象

函数在程序中代表的就是一段具有功能性的代码，可以让我们的程序编程更加具有结构性和提升程序的复用性,也能让代码变得更加灵活强大

4.11.1、声明函数

```
/*
// 函数的定义方式1
function 函数名 (参数){
    函数体;
    return 返回值;
}
功能说明：
    可以使用变量、常量或表达式作为函数调用的参数
    函数由关键字function定义
    函数名的定义规则与标识符一致，大小写是敏感的
    返回值必须使用return

//  函数的定义方式2

用 Function 类直接创建函数的语法如下：
var 函数名 = new Function("参数1","参数n","function_body");

虽然由于字符串的关系，第二种形式写起来有些困难，但有助于理解函数只不过是一种引用类型*/
```

4.11.2、函数调用

```
    //f(); --->OK
    function f(){
        console.log("hello")

    }
    f() //----->OK
```

> 不同于python，js代码在运行时，会分为两大部分———检查装载 和 执行阶段。
>
> - 检查装载阶段：会先检测代码的语法错误，进行变量、函数的声明
> - 执行阶段：变量的赋值、函数的调用等，都属于执行阶段。

4.11.3、函数参数

（1）参数基本使用

```
// 位置参数
function add(a,b){

    console.log(a);
    console.log(b);
}
add(1,2)
add(1,2,3)
add(1)


// 默认参数
```

```
function stu_info(name,gender="male"){
    console.log("姓名："+name+" 性别："+gender)
}

stu_info("yuan")
```

4.11.4、函数返回值

在函数体内，使用 return 语句可以设置函数的返回值。一旦执行 return 语句，将停止函数的运行，并运算和返回 return 后面的表达式的值。如果函数不包含 return 语句，则执行完函数体内每条语句后，返回 undefined 值。

```
function add(x,y) {
          return x,y
      }

var ret = add(2,5);
console.log(ret)  //返回5
```

> 1、在函数体内可以包含多条 return 语句，但是仅能执行一条 return 语句
>
> 2、函数的参数没有限制，但是返回值只能是一个；如果要输出多个值，可以通过数组或对象进行设计。

4.11.5、函数作用域

作用域是JavaScript最重要的概念之一。

JavaScript中，变量的作用域有全局作用域和局部作用域两种。

```
// 局部变量,是在函数内部声明,它的生命周期在当前函数被调用的时候， 当函数调用完毕以后,则内存中自动销毁当前变量
// 全局变量,是在函数外部声明,它的生命周期在当前文件中被声明以后就保存在内存中,直到当前文件执行完毕以后,才会被内存销毁掉
```

首先熟悉下var

```
    var name = "bobo"; // 声明一个全局变量 name并赋值”bobo“
    name = "张三";  // 对已经存在的变量name重新赋值 ”张三“
    console.log(name);

    var  gender = "male"
    var  gender = "female" // 原内存释放与新内存开辟，指针指向新开辟的内存
    console.log(gender)
```

作用域案例：

```
 var num = 10; // 在函数外部声明的变量, 全局变量
function func(){
  //千万不要再函数内部存在和全局变量同名的变量
  num = 20; // 函数内部直接使用变量,则默认调用了全局的变量,
}
func();
console.log("全局num：",num);
```

4.11.6、匿名函数

匿名函数，即没有变量名的函数。在实际开发中使用的频率非常高！也是学好JS的重点。

```
      // 匿名函数赋值变量
       var foo = function () {
           console.log("这是一个匿名函数！")
       };
            foo() //调用匿名函数


      // 匿名函数的自执行
      (function (x,y) {
           console.log(x+y);
       })(2,3)


     // 匿名函数作为一个高阶函数使用
     function bar() {

        return function () {
            console.log("inner函数！")
        }
    }
    bar()()
```

4.11.7、箭头函数

- 箭头函数定义
  - 箭头函数提供了一种更加简洁的函数书写方式。
  - 基本语法是：
    - 参数 => 函数体
    - (参数) => {函数体}
- 基本语法实例：
  - ```
    //普通函数
    var f = function(a){
        return a;
    }
    f(1);  //1
    ```
    ```

```
//箭头函数
var f = a => a
f(10); //10

//当箭头函数没有参数或者有多个参数，要用 () 括起来。
var f = (a,b) => a+b;
f(6,2);  //8

//当箭头函数函数体有多行语句，用 {} 包裹起来，表示代码块，当只有一行语句，并且需要返回结果时，可以省略 {} ，结果会自动返回。
var f = (a,b) => {
  var result = a+b;
  return result;
}
f(6,2);  // 8
```

4.12、BOM对象（了解）

BOM:Broswer object model,即浏览器提供我们开发者在javascript用于操作浏览器的对象。

4.12.1、window对象

- 窗口方法

```
// BOM  Browser object model 浏览器对象模型

// js中最大的一个对象.整个浏览器窗口出现的所有东西都是window对象的内容.
console.log( window );

// alert()  弹出一个警告框
window.alert("hello");

//confirm  弹出一个确认框,点击确认,返回true, 点击取消,返回false
var ret = confirm("您确认要删除当前文件么?");
console.log( ret  );

// 弹出一个消息输入框,当点击确认以后,则返回可以接收到用户在输入框填写的内容.如果点击取消,则返回null
var ret = prompt("请输入一个内容","默认值");
console.log( raet );

// close() 关闭当前浏览器窗口
window.close();

//打开一个新的浏览器窗口
window.open("http://www.baidu.com","_blank","width=800px,height=500px,left=200px,top=200px");
```

4.12.2、location (地址栏)对象

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<button onclick="func1()">查看Location对象</button>
<button onclick="func2()">跳转到百度</button>
<button onclick="func3()">F5刷新页面</button>
<script>
    function func1(){
        console.log( location );
    }
    // 地址栏对象控制和操作地址栏
    // 所谓的地址就是当前页面所在地址
    // 地址结构:
    // 协议://域名:端口/路径/文件名?查询字符串#锚点
    console.log( `协议=>${location.protocol}` );
    console.log( `域名=>${location.port}` );
    console.log( `域名=>${location.hostname}` );
    console.log( `域名:端口=>${location.host}` );
    console.log( `路径=>${location.pathname}` );
    console.log( `查询字符串=>${location.search}` );
    console.log( `锚点=>${location.hash}` );
    console.log(`完整的地址信息=>${location.href}`);

    function func2(){
        // location.href="http://www.baidu.com"; // 页面跳转
        location.assign("http://www.baidu.com"); // 页面跳转
    }

    function func3(){
        location.reload(); // 刷新页面
    }
</script>
</body>
</html>
```

4.12.3、history对象：

代表当前窗口的浏览历史记录，可以通过`history`操作浏览器的前进和后退。例如：

```
// 向前进一步（相当于点击浏览器的“前进”按钮）
history.forward();

// 向后退一步（相当于点击浏览器的“后退”按钮）
history.back();
```

4.12.4、navigator对象：

提供了浏览器的相关信息，例如浏览器类型和版本、操作系统、语言等。例如：

```
// 获取浏览器名称和版本信息
var browserName = navigator.appName;
var browserVersion = navigator.appVersion;
console.log(browserName + " " + browserVersion);

// 获取操作系统类型
var osName = navigator.platform;
console.log(osName);

// 获取语言设置
var language = navigator.language;
console.log(language);
```

4.13、DOM对象(了解)

DOM document Object Model 文档对象模型

```
// 整个html文档,会保存一个文档对象document
// console.log( document ); // 获取当前文档的对象
```

4.13.1、查找标签

- 直接查找标签

```
document.getElementsByTagName("标签名")
document.getElementById("id值")
document.getElementsByClassName("类名")
//返回dom对象，就是标签对象或者数组
```

- CSS选择器查找

```
document.querySelector("css选择器")  //根据css选择符来获取查找到的第一个元素，返回标签对象（dom对象）
document.querySelectorAll("css选择器"); // 根据css选择符来获取查找到的所有元素,返回数组
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>


<div id="i1">DIV1</div>
```

```
<div class="c1">DIV</div>
<div class="c1">DIV</div>
<div class="c1">DIV</div>


<div class="outer">
    <div class="c1">item</div>
</div>


<div class="c2">
    <div class="c3">
        <ul class="c4">
            <li class="c5" id="i2">111</li>
            <li>222</li>
            <li>333</li>
        </ul>
    </div>
</div>

<script>

   // 直接查找
   var ele = document.getElementById("i1");  // ele就是一个dom对象
   console.log(ele);

   var eles = document.getElementsByClassName("c1"); // eles是一个数组
[dom1,dom2,...]
   console.log(eles);

   var eles2 = document.getElementsByTagName("div"); // eles2是一个数组
[dom1,dom2,...]
   console.log(eles2);

  //定位outer下的c1对应的标签
   var outer = document.getElementsByClassName("outer")[0];
   var te = outer.getElementsByClassName("c1");
   console.log(te);

    // css选择器
        //层级定位(空格可以表示一个或多个层级)
    var dom = document.querySelector(".c2 .c3 .c5");
    console.log(":::",dom);
        //层级定位
    var doms = document.querySelectorAll("ul li");
    console.log(":::",doms);

</script>

</body>
</html>
```

4.13.2、绑定事件

- 静态绑定 ：直接把事件写在标签元素中

```
<div id="div" onclick="foo()">click</div>

<script>
    function foo(){
        console.log("foo函数");
    }
</script>
```

- 动态绑定：在js中通过代码获取元素对象,然后给这个对象进行后续绑定

```
<p id="i1">试一试!</p>

<script>

    var ele=document.getElementById("i1");

    ele.onclick=function(){
        console.log("ok");

    };

</script>
```

> 一个元素本身可以绑定多个不同的事件, 但是如果多次绑定同一个事件,则后面的事件代码会覆盖前面的事件代码

- 在 JavaScript 中 this 不是固定不变的，它会随着执行环境的改变而改变。
  - 单独使用 this，则它指向全局(Global)对象。在浏览器中，window 就是该全局对象为
  - ```
    var x = this;
    ```
  - 在函数中(作为函数的返回值)，函数的所属者/调用者默认绑定到 this 上。
  - ```
    function myFunction() {
      return this;
    }
    ```
  - 对象方法中的this就是对象本身
  - ```
    var person = {
      firstName  : "John",
      lastName   : "Doe",
      id         : 5566,
      myFunction : function() {
        return this;
      }
    };
    ```
  - 事件中的this就是接收事件的 HTML 标签
    ```

- ```
  <button onclick="this.style.display='none'">
  点我后我就消失了
  </button>
  ```

4.14 进阶操作（重点）

- ES6新特性
  - ES6是JavaScript语言的下一代标准，已经在2015年6月正式发布了。Mozilla公司将在这个标准的基础上，推出JavaScript 2.0。
- 变量提升
  - 查看以下代码, 是否可以被运行
  - ```
    function fn(){
        console.log(name);
        var name = '大马猴';
    }
    fn()
    ```
  - 发现问题了么. 这么写代码, 在其他语言里. 绝对是不允许的. 但是在js里. 不但允许, 还能执行. 为什么呢? 因为在js执行的时候. 它会首先检测你的代码. 发现在代码中会有name使用. OK. 运行时就会变成这样的逻辑:
  - ```
    function fn(){
        var name;
        console.log(name);
        name = '大马猴';
    }
    fn();
    ```
  - 看到了么. 实际运行的时候和我们写代码的顺序可能会不一样....这种把变量提前到代码块第一部分运行的逻辑被称为**变量提升**. 这在其他语言里是绝对没有的. 并且也不是什么好事情. 正常的逻辑不应该是这样的. 那么怎么办? 在新的ES6中. 就明确了, 这样使用变量是不完善的. es6提出. 用let来声明变量. 就不会出现该问题了.
  - ```
    function fn(){
        console.log(name);  // 直接报错, let变量不可以变量提升.
        let name = '大马猴';
    }
    fn()
    ```
    - 用let声明变量是新版本javascript提倡的一种声明变量的方案
  - let还有哪些作用呢?
  - ```
    function fn(){
        var name = "周杰伦";
        var name = "王力宏";
        console.log(name);
    }
    fn()
    ```
    ```

    - 显然一个变量被声明了两次. 这样也是不合理的. var本意是声明变量. 同一个东西. 被声明两次. 所以ES6规定. let声明的变量. 在同一个作用域内. 只能声明一次.
    - ```
      function fn(){
          let name = "周杰伦";
          console.log(name);
          let name = "王力宏";
          console.log(name);
      }
      fn()
      ```
    - 注意, 报错是发生在代码检查阶段. 所以. 上述代码根本就执行不了. 在同一个作用域内. let声明的变量只能声明一次. 其他使用上和var没有差别.
- eval函数
    - eval本身在js里面正常情况下使用的并不多. 但是很多网站会利用eval的特性来完成反爬操作. 我们来看看eval是个什么鬼?
    - 从功能上讲, eval非常简单. 它可以动态的把字符串当成js代码进行运行.
    - ```
      s = "console.log('我爱你')";
      eval(s);
      ```
    - 也就是说. eval里面传递的应该是即将要执行的代码(字符串). 那么在页面中如果看到了eval加密该如何是好? 其实只要记住了一个事儿. 它里面不论多复杂. 一定是个字符串.
    - 比如:
    - ```
      eval(function(p,a,c,k,e,d){e=function(c)
      {return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?
      String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String))
      {while(c--)d[e(c)]=k[c]||e(c);k=[function(e){return d[e]}];e=function()
      {return'\\w+'};c=1};while(c--)if(k[c])p=p.replace(new
      RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}('0.1(\'我爱你
      \')',62,2,'console|log'.split('|'),0,{}))
      ```
        - 这一坨看起来, 肯定很不爽. 怎么变成我们看着很舒服的样子呢? 记住. eval()里面是字符串. 记住~!!
        - 那我想看看这个字符串长什么样? 就把eval()里面的东西拷贝出来. 执行一下. 最终一定会得到一个字符串. 要不然eval()执行不了的. 对不...于是就有了下面的操作.
        - 

        - http://tools.jb51.net/password/evalencode, 在赠送你一个在线JS处理eval的网站. 大多数的eval加密. 都可以搞定了.
      ```

    - 

- 内置对象：window
    - window对象是一个很神奇的东西. 你可以把这东西理解成javascript全局的内置对象. 如果我们默认不用任何东西访问一个标识符. 那么默认认为是在用window对象.
    - ```
      window.mm = "爱你" //定义了一个全局变量
      console.log(mm);
      ```
    - 综上，我们可以得出一个结论. 全局变量可以用window.xxx来表示.
    - ok. 接下来. 注意看了. 我要搞事情了
        - 想要在函数外部调用该函数内部定义的一个内置函数，不可使用返回值的机制，如何实现？
    - ```
      (function(){
          let chi = function(){
              console.log("我是吃")
          }
          window.w_chi = chi //全局变量chi
      })();
      //如何调用
      w_chi() //chi()

      //换一种写法. 你还认识么?
      (function(w){
          let chi = function(){
              console.log("我是吃")
          }
          w.chi = chi
      })(window);
      chi()
      ```

第5章 、jQuery

5.1、jQuery介绍

- jQuery是什么

jQuery是一个快速、简洁的JavaScript框架。jQuery设计的宗旨是“write Less，Do More”，即倡导写更少的代码，做更多的事情。它封装JavaScript常用的功能代码，提供一种简便的JavaScript设计模式，优化HTML文档操作、事件处理等功能。

jQuery兼容各种主流浏览器，如IE 6.0+、FF 1.5+、Safari 2.0+、Opera 9.0+等

- jQuery的版本

目前在市场上, 1.x , 2.x, 3.x 功能的完善在1.x, 2.x的时候是属于删除旧代码,去除对于旧的浏览器兼容代码。3.x的时候增加es的新特性以及调整核心代码的结构

- jQuery的引入

根本上jquery就是一个写好的js文件,所以想要使用jQuery的语法必须先引入到本地

```
<script src="https://cdn.bootcdn.net/ajax/libs/jquery/3.5.1/jquery.js"></script>
```

5.2、jQuery的选择器

5.2.1、直接查找

- 基本选择器

```
/*
#id          # id选择符
element      # 元素选择符
.class       # class选择符
selector1, selector2, selectorN   # 同时获取多个元素的选择符

jQ选择器：

$("#id")   == document.getElementById('id')
$(".class")
$("element")
*/
```

5.3、jQuery的事件绑定

```
/*
用法:
    直接通过事件名来进行调用
  $().事件名(匿名函数)

*/
```

案例：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

</head>
<body>
    <div id="d1">i am div tag</div>
    <input type="text" id="t">
</body>
<script src="https://cdn.bootcdn.net/ajax/libs/jquery/3.6.0/jquery.js"></script>
```

```
<script>
    $('#d1').click(function () {
        window.alert('点击了div标签！')
    })

    $('#t').blur(function () {
        window.alert('写完了吗？')
    })
</script>
</html>
```

5.4 Ajax请求(重点)

- 什么是ajax？
  - AJAX = 异步的javascript和XML（Asynchronous Javascript and XML）
  - 它不是一门编程语言，而是利用JavaScript在保证页面不被刷新、页面链接不改变的情况下与服务器交换数据并更新部分网页的技术。
  - 对于传统的网页，如果想更新内容，那么必须要刷新整个页面，但有了Ajax，便可以在页面不被全部刷新的情况下更新其内容。在这个过程中，页面实际上是在后台与服务器进行了数据交互，获得数据之后，再利用JavaScript改变页面，这样页面内容就会更新了。（微博页面加载过程的例子，加载一会下方才会出现内容，这其实就是Ajax加载的过程。）
  - 简言之，在不重载整个网页的情况下，AJAX通过后台加载数据，并在网页上进行显示。
  - 通过 jQuery AJAX 方法，您能够使用 HTTP Get 和 HTTP Post 从远程服务器上请求文本、HTML、XML 或 JSON - 同时您能够把这些外部数据直接载入网页的被选元素中。
  - ```
    //get()方式
       $("#btn").click(function(){
            var url = "yourUrl?param1=value1&param2=value2"; // 在url中添加参数
            $.ajax({
                type: "GET",
                url: url,
                dataType: "json",
                success: function(data){
                    console.log(data); // 处理回调数据
                },
                error : function(error) {
                    console.log(error); // 处理错误信息
                }
            });
        });

    //or
    $("#btn").click(function(){
            $.ajax({
                type: "GET",
                url: "yourUrl",
                data: { // 使用data选项传递参数
                    param1: "value1",
                    param2: "value2"
                },
                dataType: "json",
    ```
    ```

```
            success: function(data){
                console.log(data); // 处理回调数据
            },
            error : function(error) {
                console.log(error); // 处理错误信息
            }
        });
    });
```

- ```
  //post()方式
   $("#btn").click(function() {
          $.ajax({
              type: "POST",
              url: "yourUrl", // 请求地址
              data: { // 请求参数
                  param1: "value1",
                  param2: "value2"
              },
              dataType: "json", // 服务器返回的数据类型
              success: function(data) { // 成功回调函数
                  console.log(data); // 打印服务器返回的数据
              },
              error: function(error) { // 失败回调函数
                  console.log(error); // 打印错误信息
              }
          });
      });
  ```

- 常用的抓包工具：
  - win：fiddler
  - mac：Charles

  易语言：适合写外挂

  数据可视化：pyechars · matplotlib · Seaborn
  ```