day21-JavaScript

今日内容

- JS基本语法---->重点
- JS的BOM对象--->window,location
- JS的DOM对象----->重点
- JS的案例------>必须练习
    - 表单校验
    - 图片轮播
    - 二级联动

第一章-JS基础

1.1 JS简介

什么是JS

- JavaScript 是一种客户端脚本语言。运行在客户端浏览器中，每一个浏览器都具备解析 JavaScript 的引擎。
- 脚本语言：不需要编译(直译)，就可以被浏览器直接解析执行了。
- JS语言和Java语言对比：

对比	Java	JS
运行环境	JVM虚拟机	JS引擎，是浏览器的一部分
是否跨平台运行	跨平台	跨平台
语言类型	强类型语言	弱类型，动态类型语言
是否需要编译	需要编译，是编译型语言	不需要编译，是解释型语言
是否区分大小写	区分大小写	区分大小写

JS的作用

具体来说，有两部分作用：

- JS代码可以操作浏览器：进行网址跳转、历史记录切换、浏览器弹窗等等
- JS代码可以操作网页标签：操作HTML的标签、标签的属性、样式等等

  注意：JS的是在浏览器内存中运行时操作，并不会修改网页源码，所以刷新页面后网页会还原

JS的组成

- ECMAScript(核心)：是JS的基本语法规范
- BOM：Browser Object Model，浏览器对象模型，提供了与浏览器进行交互的方法
- DOM：Document Object Model，文档对象模型，提供了操作网页的方法

1.2 JS引入

内部JS

语法

- 在html里增加`<script>`标签，把js代码写在标签体里

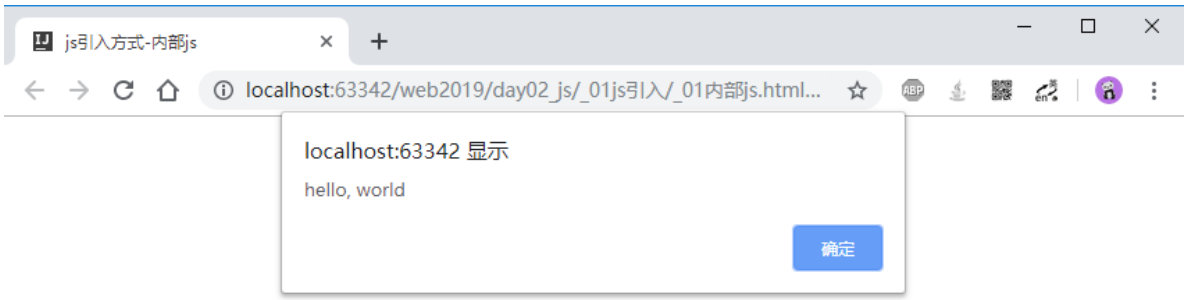
```
<script>
    //在这里写js代码
</script>
```

示例

- 创建html页面，编写js代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>js引入方式-内部js</title>
    <script>
        //操作浏览器弹窗
        alert("hello, world");
    </script>
</head>
<body>

</body>
</html>
```

- 打开页面，浏览器会弹窗



外部JS

语法

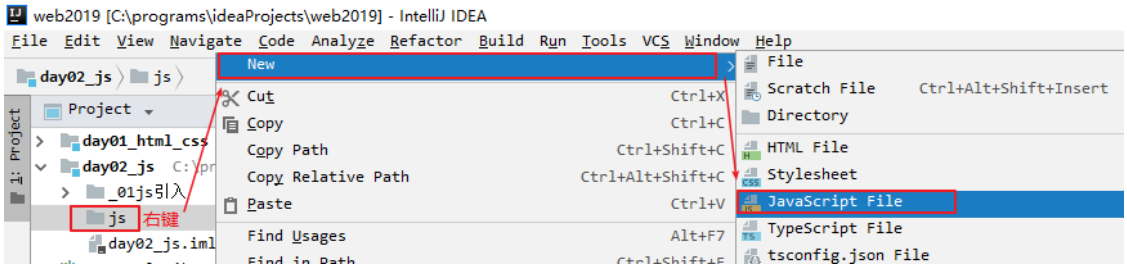
- 把js代码写在单独的js文件中，js文件后缀名是`.js`
- 在HTML里使用`<script>`标签的src属性引入外部js文件

```
<script src="js文件的路径"></script>
```

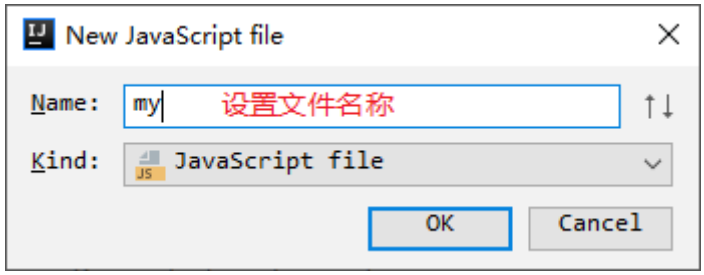
示例

- 创建一个`my.js`文件，编写js代码

- 第1步：创建js文件



- 第2步：设置js文件名称



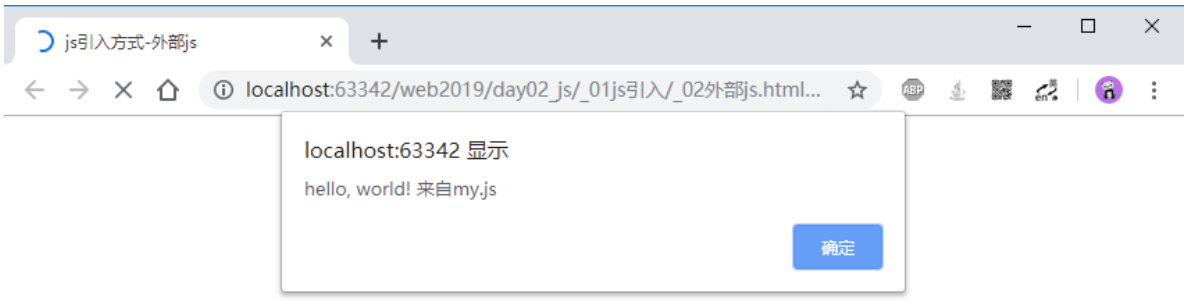
- 第3步：编写js代码

```
alert("hello, world! 来自my.js");
```

- 创建一个html，引入`my.js`文件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>js引入方式-外部js</title>
    <!--引入外部的my.js文件-->
    <script src="../js/my.js"></script>
</head>
<body>

</body>
</html>
```

- 打开页面，浏览器会弹窗



注意事项

- 一个`script`标签，不能既引入外部js文件，又在标签体内写js代码。
  - 错误演示

```
<script src="../js/my.js">
    alert("hello");
</script>
```

- 正确演示

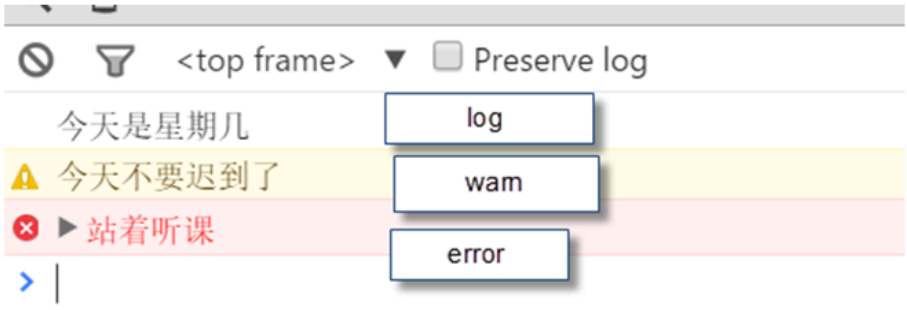
```
<script src="../js/my.js"></script>
<script>
    alert("hello");
</script>
```

1.3 JS小功能和JS调试

小功能

- alert(): 弹出警示框
- console.log(): 向控制台打印日志
- document.writte(); 文档打印. 向页面输出内容.

控制台输出	
console.log()	控制台输出正常语句
console.warn()	控制台警示框
console.error()	控制台错误提示

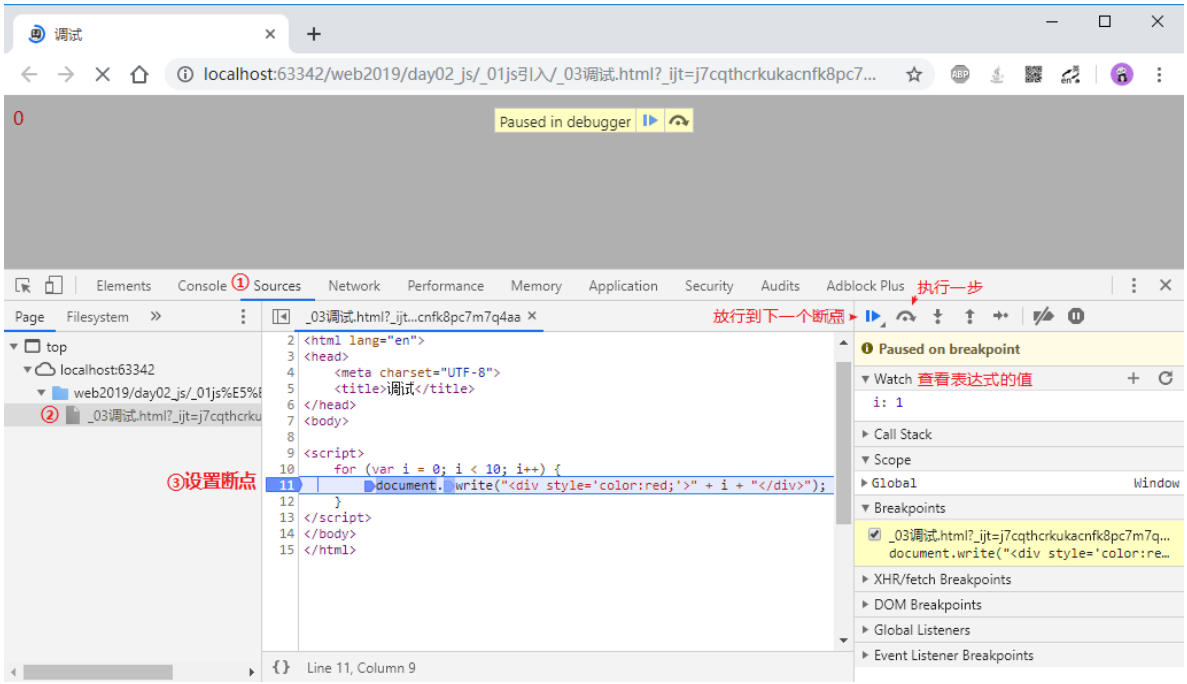



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>01_JS小功能</title>
    <script>
        // 警告框: 1.提示  2.调试(判断是否执行到了这里;判断变量是否拿到了值)
        // alert("弹出警告框...");

        // 控制台: 1.提示  2.调试(判断是否执行到了这里;判断变量是否拿到了值)
        // 向控制台打印输出
        // console.log("日志信息...");

        // 文档打印
        document.write("hello js小功能...");
        document.write("<h1>一级标题</h1>");
```

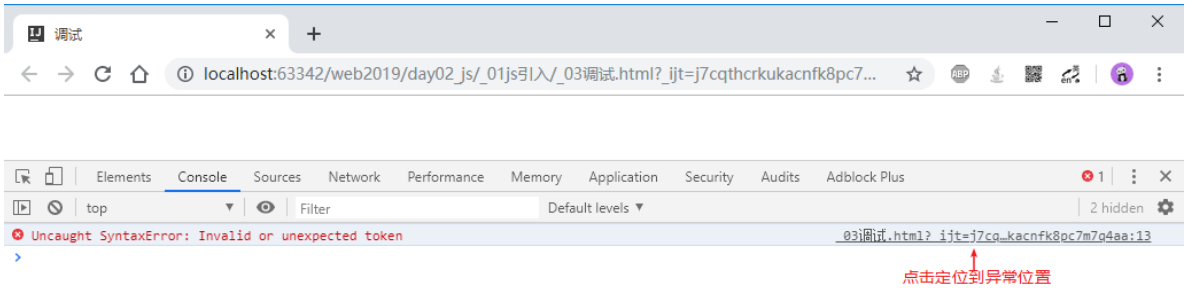
```
    </script>
</head>
<body>

</body>
</html>
```

调试

1. 按 F12 打开开发者工具
2. 找到 Source 窗口，在左边找到页面，如下图
    - 打断点之后，当代码执行到断点时，会暂停执行
    - 在窗口右侧可以查看表达式的值、单步调试、放行等等



3. 如果代码执行中出现异常信息，会在控制台 Console 窗口显示出来



4. 点击可以定位到异常位置

1.4 JS基本语法

变量

- JavaScript 是一种==弱类型语言==，**javascript的变量类型由它的值来决定。** 定义变量需要用关键字 'var'
- 格式: `var 变量名 = 值;`
- 举例:

```
var numI = 10;                    ---->相当于java中的 int i = 10;
var numD = 3.14;                  ---->相当于java中的 double d = 3.14;
var str = "java";                  ---->相当于java中的 String str = "java";
var date = new Date();            ---->相当于java中的 Date date = new Date();
...
var num1 = 10,num2 = 20,num3 = 3.14; ----> 相当于java中的  int num1 = 10,num2
=20;
注意:
numI=10;
numD = 3.14;
str = "java";
date = new Date();
```

- 注意:
  - var可以省略不写,建议保留
  - 最后一个分号可以省略,建议保留
  - 定义多个变量可以用","隔开，公用一个'var'关键字

数据类型

1.五种原始数据类型

数据类型	描述	示例
`number`	数值类型	`1`, `2`, `3`, `3.14`
`boolean`	布尔类型	`true`, `false`
`string`	字符串类型	`"hello"`, 'hello'
`object`	对象类型	`new Date()`, `null`
`undefined`	未定义类型	var a;

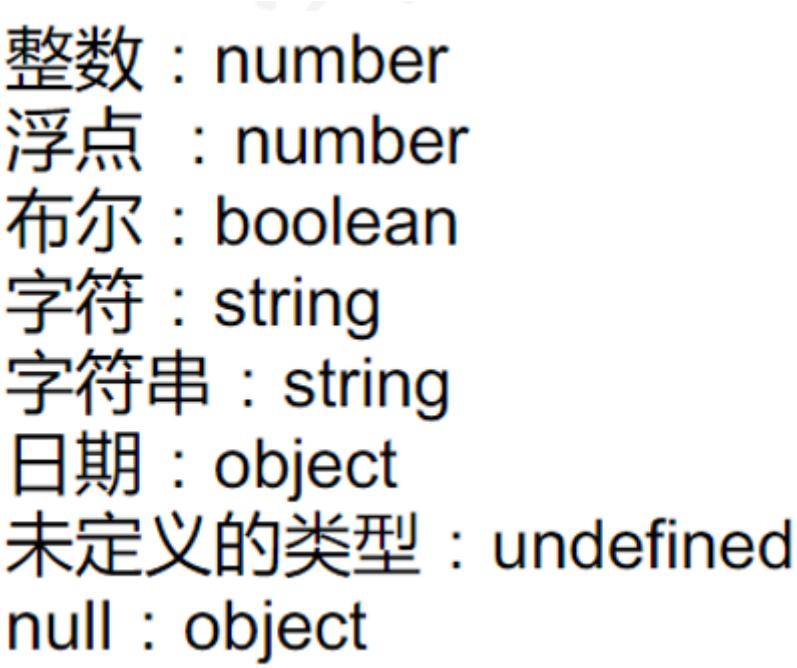
2.typeof操作符

- 作用： 用来判断变量是什么类型
- 写法：typeof(变量名) 或 typeof 变量名
- null与undefined的区别：

  null: 对象类型，已经知道了数据类型，但对象为空。 undefined：未定义的类型，并不知道是什么数据类型。

3.小练习

- 定义不同的变量,输出类型,



- 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>01_JS的变量和数据类型</title>
    <script>
        // 定义不同类型的变量,并输出类型
        // 定义变量: var 变量名 = 值;  变量的类型由值来决定
        // 变量类型: number,boolean,string,object,undefined
        // 判断变量的类型:  typeof 变量名   或者 typeof(变量名)
        // null: 对象类型，已经知道了数据类型，但对象为空。
```

```
        // undefined：未定义的类型，并不知道是什么数据类型。

        /*定义数值类型的变量*/
        var num1 = 10;
        var num2 = 3.14;

        /*定义布尔类型的变量*/
        var num3 = true;

        /*定义字符串类型的变量*/
        var num4 = "java";

        /*定义对象类型的变量*/
        var num5 = new Date();
        var num6 = null;

        /*定义未定义类型的变量*/
        var num7;

        // 输出变量类型
        console.log(typeof num1);
        console.log(typeof num2);
        console.log(typeof num3);
        console.log(typeof num4);
        console.log(typeof num5);
        console.log(typeof num6);
        console.log(typeof num7);

    </script>
</head>
<body>


</body>
</html>
```

字符串转换成数字类型

- 全局函数(方法)，就是可以在JS中任何的地方直接使用的函数，不用导入对象。不属于任何一个对象

转换函数	作用
parseInt()	将一个字符串转成整数，如果一个字符串包含非数字字符，那么 parseInt 函数会从首字母开始取数字字符，一旦发现非数字字符，马上停止获取内容。如果转换失败，则返回 NaN=Not a Number，不是一个数。
parseFloat()	将一个字符串转成小数，转换原理同上。
isNaN()	转换前判断被转换的字符串是否是一个数字，非数字返回 true　isNaN = is not a number

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>02_字符串转换成数字类型</title>
    <script>
        console.log(parseInt("123"));// 123
        console.log(typeof parseInt("123"));// number
        console.log("--------");
```

```
        console.log(parseFloat("3.14"));// 3.14
        console.log(typeof parseFloat("3.14"));// number
        console.log("--------");
        console.log(isNaN(123));// false 数字
        console.log(isNaN("123"));// false 数字
        console.log(isNaN("123a"));// true 非数字
        console.log(isNaN("a123"));// true 非数字
        console.log("--------");
        // 注意事项:
        console.log(parseInt("123a"));// 123
        console.log(typeof parseInt("123a"));// number

        console.log(parseInt("12a3"));// 12
        console.log(typeof parseInt("12a3"));// number

        console.log(parseInt("a123"));// NaN

    </script>
</head>
<body>

</body>
</html>
```

运算符

- 关系运算符:> >= < <=,==,!=, **===**
- 算术运算符: +,-,*,/,%
- 自增自减: ++,--
- 逻辑运算符: &&, ||, !
- 赋值运算符: =, +=,-=,*=,/=,%=
- 三元运算符:
- **注意:**
  - number类型和字符串做-,*,/,%的时候,字符串自动的进行类型转换,前提字符串里面的数值要满足**number**类型
  - /除法运算会保留小数

```
var i = 3;
var j = "6";
alert(j-i);//结果是3, "6" ==> 6
alert(j*i);//结果是18,
alert(j/i);//结果是2,
```

- 除法,保留小数

```
var i = 2;
var j = 5;
alert(j/i);
```

- `==` 比较数值, `===` 比较数值和类型

```
var i = 2;
var j = "2";
alert(i==j); // ==比较的仅仅是数值, true
alert(i===j); // ===比较的是数值和类型.false
```

流程语句

- for\while循环

```
  //99乘法表
  <script>
    for(var i = 1; i<=9 ; i++){
        for(var j =1; j <= i;j++){
            document.write(j+""+i+"="+ji);
            //空格
            document.write(" ");
        }
        //换行
    document.write("<br />");
   }
 </script>
```

- if... else

```
var a = 6;
if(a==1)
{
    alert('语文');
}
else if(a==2)
{
    alert('数学');
}
else
{
    alert('不补习');
}
```

- switch

```
<script>
    var str = "java";

    switch (str){
        case "java":
            alert("java");
            break;
        case "C++":
            alert("C++");
            break;

        case "Android":
            alert("Android");
            break;  }
</script>
```

1.5 函数

什么是函数

- 函数： 是被设计为执行特定任务的代码块， 在被调用时会执行
- 函数类似于Java里的方法，用于封装一些可重复使用的代码块

普通(有名)函数

语法

- 定义普通函数

```
function 函数名(形参列表){
    函数体(代码块)
    [return 返回值;]
}
注意:
    1.小括号中的形参是没有形参类型的,也就是不要写var(写了会报错),只需要直接写形参名即可
    2.js中的行数没有重载,如果函数一样,后面的函数会覆盖前面的函数
```

- 调用普通函数

```
var result = 函数名(实参列表);
```

示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>05_函数</title>
    <script>
        /*
            function 函数名(形参列表){
                函数体(代码块)
                [return 返回值;]
            }
            注意:
                1.小括号中的形参是没有形参类型的,也就是不要写var(写了会报错),只需要直接写形参名即可
                2.js中的行数没有重载,如果函数一样,后面的函数会覆盖前面的函数
                3.函数没有调用不会执行
           调用函数的语法: var 变量 = 函数名(实参);
        */
        function f1() {
            console.log("f1没有参数没有返回值的函数...")
        }

        f1();

        function f2(num1, num2) {
            console.log("f2有参数没有返回值的函数1,参数num1:"+num1+",参数num2:"+num2);
        }
```

```
        f2(10,"java");

        function f3(num1) {
            console.log("f3有参数有返回值的函数,参数num1:"+num1);
            return "itheima";
        }

        var result = f3(100);
        console.log("result:" + result);
        console.log("--------------------");

        function f2(num1, num2) {
            console.log("f2有参数没有返回值的函数2,参数num1:"+num1+",参数
num2:"+num2);
        }

        f2(4.23,"python");

    </script>
</head>
<body>

</body>
</html>
```

匿名函数

匿名函数，也叫回调函数，类似于Java里的函数式接口里的方法

```
function (形参列表){
    函数体
    [return 返回值;]
}
```

1.6 JS事件

事件介绍

- HTML 事件是发生在 HTML 元素上的“事情”， 是浏览器或用户做的某些事情
- **事件通常与函数配合使用，这样就可以通过发生的事件来驱动函数执行。**

常见事件

属性	此事件发生在何时...
onclick	当用户点击某个对象时调用的事件句柄。
ondblclick	当用户双击某个对象时调用的事件句柄。
onchange	域的内容被改变。
onblur	元素失去焦点。
onfocus	元素获得焦点。
onload	一张页面或一幅图像完成加载。
onsubmit	确认按钮被点击；表单被提交。
onkeydown	某个键盘按键被按下。
onkeypress	某个键盘按键被按住。
onkeyup	某个键盘按键被松开。
onmousedown	鼠标按钮被按下。
onmouseup	鼠标按键被松开。
onmouseout	鼠标从某元素移开。
omouseover	鼠标移到某元素之上。
onmousemove	鼠标被移动。

事件绑定

普通函数方式

说白了设置标签的属性

```
<标签 属性="调用js函数"></标签>
属性: 根事件名一致
属性值: 调用js函数代码
```

匿名函数方式

```
<script>
    标签对象.事件属性 = function(){
        //执行一段代码
    }
</script>
```

事件使用

重要的事件

- 点击事件

  需求: 每点击一次按钮 弹出hello...

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        // 需求: 每点击一次按钮 弹出hello...
        // 方式一: 普通函数绑定:为标签设置事件属性
        function f1() {
            alert("hello1...")
        }

    </script>
</head>
<body>

<input type="button" value="点击事件1" onclick="f1()">
<input id="inputId" type="button" value="点击事件2">

</body>
<script>
    // 方式二: 匿名函数绑定: 获得标签DOM对象,设置事件属性
    document.getElementById("inputId").onclick = function () {
        alert("hello2...");
    }
</script>

</html>
```

- 获得焦点(onfocus)和失去焦点(onblur)

  需求:给输入框设置获得和失去焦点

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>07_获得焦点(onfocus)和失去焦点(onblur)</title>
</head>
<body>

<input type="text" value="it" onfocus="f1()" onblur="f2(this)">

</body>
<script>
    function f1() {
        console.log("获得焦点...")
    }

    function f2(obj) {
        console.log("失去焦点...文本输入框中的内容:"+obj.value)
    }
</script>
</html>
```

- 内容改变(onchange)

  需求: 给select设置内容改变事件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>08_内容改变(onchange)</title>
</head>
<body>
<select onchange="f1(this)">
    <option value="gd">广东</option>
    <option value="sd">山东</option>
    <option value="gx">广西</option>
</select>
</body>
<script>
    function f1(obj) {
        console.log("下拉框选择的值是:"+obj.value);

    }
</script>
</html>
```

- 等xx加载完成(onload) 可以把script放在body的后面/下面, 就可以不用了

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body onload="f()">
<input id="inputId" type="text" value="hello..."/>
</body>
<script>
    function f() {
        document.getElementById("inputId").value = "你好";
    }
</script>
</html>
```

掌握的事件

- 键盘相关的, 键盘按下(onkeydown) 键盘抬起(onkeyup)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<!--键盘相关的, 键盘按下(onkeydown)  键盘抬起(onkeyup)-->
<input type="text" onkeydown="f1()" onkeyup="f2()">
</body>
<script>
    function f1() {
        console.log("键盘按下了...");
```

```
    }

    function f2() {
        console.log("键盘抬起了...");
    }
</script>
</html>
```

- 鼠标相关的, 鼠标在xx之上(onmouseover), 鼠标按下(onmousedown),鼠标离开(onmouseout)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<!--鼠标相关的, 鼠标在xx之上(onmouseover ), 鼠标按下(onmousedown),鼠标离开
(onmouseout)-->
<input type="button" value="点我吧" onmouseover="f1()" onmousedown="f2()"
onmouseout="f3()">
</body>
<script>

    function f1() {
        console.log("鼠标在按钮之上");
    }

    function f2() {
        console.log("鼠标在按钮上按下");
    }

    function f3() {
        console.log("鼠标在按钮上离开");
    }

</script>
</html>
```

1.7 正则表达式

正则表达式概述

正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

用我们自己的话来说: ==**正则表达式用来校验字符串是否满足一定的规则的公式**==

正则表达式的语法

创建对象

- 对象形式: `var reg = new RegExp("正则表达式")`
- **直接量形式**: `var reg = /^正则表达式$/;`

常用函数

方法	描述	参数	返回值
`boolean test(string)`	校验字符串的格式	要校验的字符串	boolean，校验通过返回true

常见正则表达式规则

符号	作用
\d	数字
\D	非数字
\w	单词：a-zA-Z0-9_
\W	非单词
.	通配符，匹配任意字符
{n}	匹配n次
{n,}	大于或等于n次
{n,m}	在n次和m次之间
+	1~n次
*	0~n次
?	0~1次
^	匹配开头
$	匹配结尾
[a-zA-Z]	英文字母
[a-zA-Z0-9]	英文字母和数字
[*xyz*]	字符集合, 匹配所包含的任意一个字符

使用示例

需求:

1. 出现任意数字3次
2. 只能是英文字母的, 出现6~10次之间
3. 只能由英文字母和数字组成，长度为4～16个字符，并且以英文字母开头
4. 手机号码: 以1开头, 第二位是3,4,5,6,7,8,9的11位数字

步骤:

1. 创建正则表达式
2. 调用test()方法

```
```

```
<script>
    //1.出现任意数字3次
    //a. 创建正则表达式
    var reg1  = /^\d{3}$/; //出现任意数字3次
    //b. 校验字符串
    var str1 = "3451";
    var flag1 = reg1.test(str1);
    //alert("flag1="+flag1);

    //2.只能是英文字母的，出现6~10次之间
    var reg2  =/^[a-zA-Z]{6,10}$/;
    var str2 = "abcdef11g";
    //alert(reg2.test(str2));

    //3 用户名：只能由英文字母和数字组成，长度为4～16个字符，并且以英文字母开头
    var reg3  =/^[a-zA-Z][a-zA-Z0-9]{3,15}$/;
    var str3 = "zs";
   // alert(reg3.test(str3));


    //4. 手机号码：以1开头，第二位是3,4,5,6,7,8,9的11位数字
    //var reg4  =/^1[3456789]\d{9}$/i; //忽略大小写的
    var reg4  =/^1[3456789]\d{9}$/; //不忽略大小写的
    var str4 = "188245899";
    alert(reg4.test(str4));


</script>
```

1.8 内置对象之Array数组【重点】

创建数组对象

语法

- `var arr = new Array(size)`
- `var arr = new Array(element1, element2, element3, ...)`
- `var arr = [element1, element2, element3, ...];`

数组的特点

- 数组中的每个元素可以是任意类型
- 数组的长度是可变的，更加类似于Java里的集合`List`

示例

- 创建数组，并把数组输出到浏览器控制台上
    - 说明：把数据输出到控制台：`console.log(value)`

```
<script>
        /*
            创建数组对象
                - var arr = new Array(size)
```

```
                - var arr = new Array(element1, element2, element3, ...)
                - var arr = [element1, element2, element3, ...];

            数组的特点
                - 数组中的每个元素可以是任意类型
                - 数组的长度是可变的，更加类似于Java里的集合List
         */
        // 方式一：定义数组
        var arr1 = new Array(3);
        arr1[0] = "java";
        arr1[1] = "php";
        arr1[2] = "python";
        arr1[3] = "c++";
        for (var i = 0;i<arr1.length;i++){
            console.log(arr1[i]);
        }

        console.log("----------");
        // 方式二：定义数组
        var arr2 = new Array("java","php","python");
        for (var i = 0;i<arr2.length;i++){
            console.log(arr2[i]);
        }

        console.log("----------");
        // 方式三：定义数组
        var arr3 = ["java","php","python",123];
        for (var i = 0;i<arr3.length;i++){
            console.log(arr3[i]);
        }

    </script>
```

数组常见的函数

API介绍

方法	描述
concat(arr)	连接两个或更多的数组，并返回结果。
join(str)	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。
reverse()	颠倒数组中元素的顺序。

3.2.2示例

```
<script>
        /*
            concat(arr) 连接两个或更多的数组，并返回结果。
            join(str) 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。
            reverse() 颠倒数组中元素的顺序。
         */
        // 创建数组
        var arr1 = [123,234,345];
        var arr2 = ["java","php","c++"];
```

```
        // 多个数组进行拼接
        var arr = arr1.concat(arr2);
        console.log(arr.length);
        for (var i = 0; i < arr.length; i++) {
            console.log(arr[i]);
        }
        console.log("------");

        // 把数组中所有的元素拼接成一个字符串,拼接符由参数指定
        var str = arr1.join("-");
        console.log(str);
        console.log("------");

        // 数组反转
        var reverseArr = arr1.reverse();
        for (var i = 0; i < reverseArr.length; i++) {
            console.log(reverseArr[i]);// 反转之后
        }
        console.log("------");
        for (var i = 0; i < arr1.length; i++) {
            console.log(arr1[i]);// 反转之后
        }

    </script>
```

二维数组

1. 数组里面再放数组 就是二维数组
2. 示例

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

</body>
<script>
    // 方式一:
    var citys = new Array(3);
    citys[0] = ["深圳", "广州", "东莞", "惠州"];
    citys[1] = ["武汉", "黄冈", "黄石", "鄂州", "荆州"];
    citys[2] = ["济南", "青岛", "烟台", "淄博", "聊城"];

    // 方式二:
    var citys2 = [
        ["深圳", "广州", "东莞", "惠州"],
        ["武汉", "黄冈", "黄石", "鄂州", "荆州"],
        ["济南", "青岛", "烟台", "淄博", "聊城"]
    ];

    // 遍历二维数组
    for (var i = 0; i < citys.length; i++) {
        // 获取元素
        var eArr = citys[i];
```

```
        // 遍历取出来的元素数组
        for (var j = 0; j < eArr.length; j++) {
            console.log(eArr[j]);
        }
        console.log("===============");
    }

</script>
</html>
```

1.9 内置对象之-Date日期

创建日期对象

语法

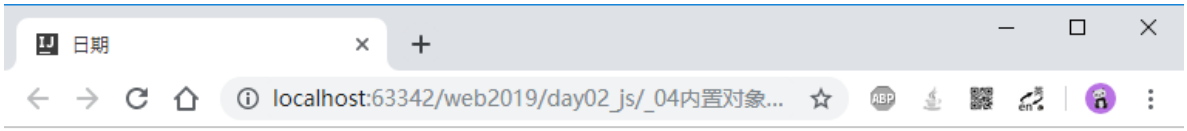
- 创建当前日期：`var date = new Date()`
- 创建指定日期：`var date = new Date(年, 月, 日)`

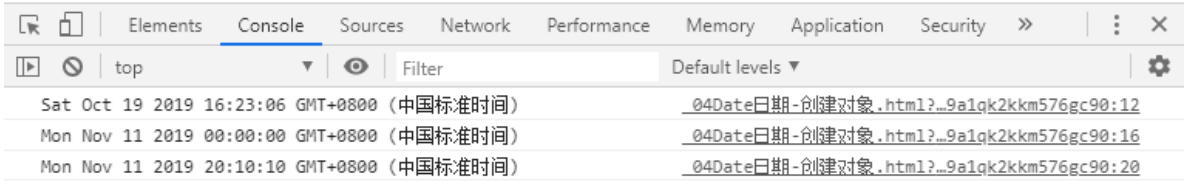
  注意：月从0开始，0表示1月
- 创建指定日期时间：`var date = new Date(年, 月, 日, 时, 分, 秒)`

  注意：月从0开始，0表示1月

示例

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

<script>
    //创建当前日期
    var date1 = new Date();
    console.log(date1);

    //创建指定日期: 2019-11-11
    var date2 = new Date(2019, 10, 11);
    console.log(date2);

    //创建指定日期时间：2019-11-11 20:10:10
    var date3 = new Date(2019, 10, 11, 20, 10, 10);
    console.log(date3);
</script>
</body>
</html>
```





日期常用方法

API介绍

方法	描述
Date()	返回当日的日期和时间。
getDate()	从 Date 对象返回一个月中的某一天 (1 ~ 31)。
getDay()	从 Date 对象返回一周中的某一天 (0 ~ 6)。
getMonth()	从 Date 对象返回月份 (0 ~ 11)。
getFullYear()	从 Date 对象以四位数字返回年份。
getHours()	返回 Date 对象的小时 (0 ~ 23)。
getMinutes()	返回 Date 对象的分钟 (0 ~ 59)。
getSeconds()	返回 Date 对象的秒数 (0 ~ 59)。
getMilliseconds()	返回 Date 对象的毫秒(0 ~ 999)。
getTime()	返回 1970 年 1 月 1 日至今的毫秒数。
parse()	返回1970年1月1日午夜到指定日期（字符串）的毫秒数。
setDate()	设置 Date 对象中月的某一天 (1 ~ 31)。
setMonth()	设置 Date 对象中月份 (0 ~ 11)。
setFullYear()	设置 Date 对象中的年份（四位数字）。
setYear()	请使用 setFullYear() 方法代替。
setHours()	设置 Date 对象中的小时 (0 ~ 23)。
setMinutes()	设置 Date 对象中的分钟 (0 ~ 59)。
setSeconds()	设置 Date 对象中的秒钟 (0 ~ 59)。
setMilliseconds()	设置 Date 对象中的毫秒 (0 ~ 999)。
setTime()	以毫秒设置 Date 对象。
toLocaleString()	根据本地时间格式，把 Date 对象转换为字符串。

3.1.2示例

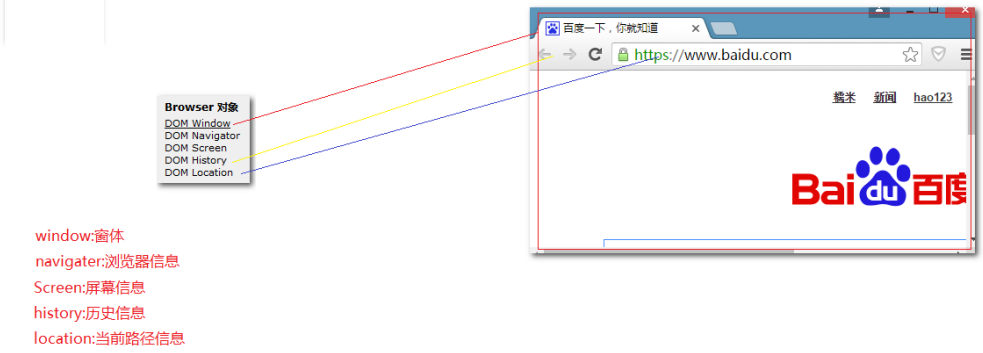
```
<script>
        // 创建日期对象
        var date = new Date();

        console.log("年:"+date.getFullYear());
        console.log("月:"+date.getMonth());
        console.log("日:"+date.getDate());
        console.log("时:"+date.getHours());
        console.log("分:"+date.getMinutes());
        console.log("秒:"+date.getSeconds());
        console.log("毫秒:"+date.getMilliseconds());
        console.log(date);

        console.log(date.toLocaleString());// 日期和时间
        console.log(date.toLocaleDateString());// 日期
        console.log(date.toLocaleTimeString());// 时间


    </script>
```

第二章-JS的BOM

2.1 JS的BOM

概述

Browser Object Model ,浏览器对象模型. 为了便于对浏览器的操作，JavaScript封装了对浏览器中各个对象，使得开发者可以方便的操作浏览器中的各个对象。



BOM里面的五个对象

window: 窗体对象

```
alert() 显示带有一段消息和一个确认按钮的警告框
confirm() 显示带有一段消息以及确认按钮和取消按钮的对话框,返回选择的结果,确定(true),取消(false)
setInterval('函数名()',time) 按照指定的周期（以毫秒计）来调用函数或计算表达式
setTimeout('函数名()',time) 在指定的毫秒数后调用函数或计算表达式
clearInterval() 取消由 setInterval() 设置的 Interval()。
clearTimeout() 取消由 setTimeout() 方法设置的 timeout。
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        /*
            alert() 显示带有一段消息和一个确认按钮的警告框
            confirm() 显示带有一段消息以及确认按钮和取消按钮的对话框,返回选择的结果,确定(true),取消(false)
            setInterval('函数名()',time) 按照指定的周期（以毫秒计）来调用函数或计算表达式
            setTimeout('函数名()',time) 在指定的毫秒数后调用函数或计算表达式
            clearInterval() 取消由 setInterval() 设置的 Interval()。
            clearTimeout() 取消由 setTimeout() 方法设置的 timeout。
         */
        // alert:完整格式
        // window.alert("hello js1...");
        // alert:省略格式
        // alert("hello js2...");

        // confirm: 确认对话框
        /*var flag = confirm("确认删除吗?");
        if (flag){
            console.log(flag + "----确认删除");
        } else {
            console.log(flag + "----取消删除");
        }*/

        // 循环定时器
        /*setInterval("f1()",1000);// 每隔1秒就调用1次f1函数

        function f1() {
            console.log("setInterval...")
        }*/

        // 一次性定时器
        /*setTimeout("f2()",5000);// 1秒以后调用1次f2函数

        function f2() {
            console.log("setTimeout...");
        }*/

    </script>
</head>
<body>
<input type="button" value="开启循环定时器" onclick="f1()">
<input type="button" value="结束循环定时器" onclick="f2()">
```

```
</body>
<script>
    var interval;

    function f1() {
        // 开启循环定时器
        interval = setInterval("f()", 300);
    }

    function f() {
        console.log("开启循环定时器...");
    }

    function f2() {
        // 结束循环定时器
        clearInterval(interval);
    }
</script>

</html>
```

navigator:浏览器导航对象【了解】

属性	作用
appName	返回浏览器的名称
appVersion	返回浏览器的平台和版本信息

screen:屏幕对象【了解】

方法	作用
width	返回显示器屏幕的分辨率宽度
height	返回显示屏幕的分辨率高度

history:历史对象【了解】

方法	作用
back()	加载 history 列表中的前一个 URL
forword()	加载 history 列表中的下一个 URL
go()	加载 history 列表中的某个具体页面，(-1上一个页面，1前进一个页面)

```
<body>

<a href="b.html">从a.html进入b.html</a>
<input type="button" value="进入下一页" onclick="f1()">


</body>

<script>
```

```
    /*
        history:历史对象【了解】
        back() 加载 history 列表中的前一个 URL
        forword() 加载 history 列表中的下一个 URL
        go() 加载 history 列表中的某个具体页面,（-1上一个页面，1前进一个页面)
     */
    function f1() {
        history.forward();
    }

</script>
```

```
<body>
<a href="a.html">从b.html回退到a.html</a>
<input type="button" value="回退到上一页" onclick="f1()">

</body>
<script>
    function f1() {
        history.back();
    }
</script>
```

location:当前路径信息

属性	作用
host	设置或返回主机名和当前 URL 的端口号
==href==	设置或返回完整的 URL
port	设置或返回当前 URL 的端口号

location.href; 获得路径

location.href = "http://www.baidu.com"; 设置路径,跳转到百度页面

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

<input type="button" value="跳转" onclick="f1()">

</body>
<script>
    console.log("ip和端口号:"+location.host);
    console.log("url路径:"+location.href);
    console.log("端口号"+location.port);

    function f1() {
```

```
        // 跳转到百度页面
        // location.href = "http://www.baidu.com";
        // 跳转到本项目的其他页面
        location.href = "03_screen对象.html";
    }
</script>
</html>
```

第三章-JS的DOM

3.1 DOM介绍

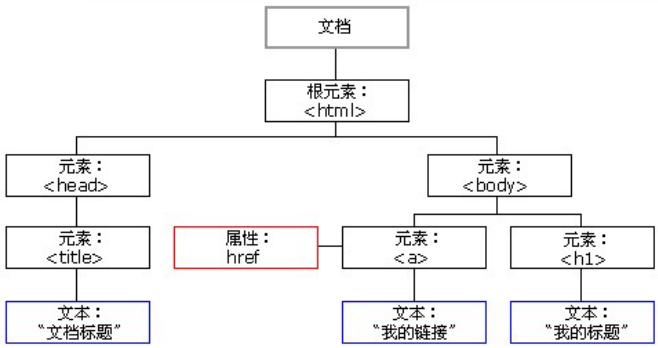
1. 什么是dom

- DOM：**D**ocument **O**bject **M**odel，文档对象模型。是js提供的，用来访问网页里所有内容的(标签,属性,标签的内容)

2. 什么是dom树

- 当网页被加载时，浏览器会创建页面的DOM对象。DOM对象模型是一棵树形结构：网页里所有的标签、属性、文本都会转换成节点对象，按照层级结构组织成一棵树形结构。
    - 整个网页封装成的对象叫`document`
    - 标签封装成的对象叫`Element`
    - 属性封装成的对象叫`Attribute`
    - 文本封装成的对象叫`Text`

```
<html>
    <head>
        <title>文档标题</title>
    </head>
    <body>
        <a href="#">我的链接</a>
        <h1>我的标题</h1>
    </body>
</html>
```



一切皆节点, 一切皆对象

3.2 操作标签

获取标签

```
document.getElementById(id) 根据id获取标签 返回值:Element对象
document.getElementsByTagName(tagName) 根据标签名称获取一批标签 返回值:Element类数组
document.getElementsByClassName(className) 根据类名获取一批标签 返回值:Element类数组
document.getElementsByName(name) 根据标签name获取一批标签 返回值:Element类数组
```

```
<body>
```

```
<input id="inputId1" type="text" value="admin"><br/>
<input id="inputId2" type="password" value="123456"><br/>
<input type="text" class="c1"><br/>
<input type="text" class="c1"><br/>
<input type="text" class="c1"><br/>
<input type="text" class="c1"><br/>
<input type="checkbox" name="hobby"> 篮球<br/>
<input type="checkbox" name="hobby"> 足球<br/>
<input type="checkbox" name="hobby"> 敲代码<br/>


</body>
<script>
    /*
        document.getElementById(id) 根据id获取标签 返回值:Element对象
        document.getElementsByTagName(tagName) 根据标签名称获取一批标签 返回
值:Element类数组
        document.getElementsByClassName(className) 根据类名获取一批标签 返回
值:Element类数组
        document.getElementsByName(name) 根据标签name获取一批标签 返回值:Element类数
组
     */
    // 获取文本输入框标签对象
    var textE = document.getElementById("inputId1");
    // 获取密码输入框标签对象
    var passwordE = document.getElementById("inputId2");
    console.log("默认用户名:"+textE.value);
    console.log("默认密码:"+passwordE.value);

    // 获取所有的input标签
    var inputArr = document.getElementsByTagName("input");
    console.log(inputArr.length);// 9

    // 获取所有class属性值为c1的input标签
    var inputArr2 = document.getElementsByClassName("c1");
    console.log(inputArr2.length);// 4

    // 获取所有name属性值为hobby的标签
    var inputArr3 = document.getElementsByName("hobby");
    console.log(inputArr3.length);// 3
</script>
```

操作标签

方法	描述	返回值
`document.createElement(tagName)`	创建标签	`Element`对象
`parentElement.appendChild(sonElement)`	插入标签	
`element.remove()`	删除标签	
`document.createTextNode()`	创建文本	

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<p id="pId">段落标签</p>
<input type="button" value="添加font标签到p标签中" onclick="addElement()">
<input type="button" value="删除p标签" onclick="removeElement()">
</body>
<script>
    /*
        document.createElement(tagName) 创建标签 Element对象
        parentElement.appendChild(sonElement) 插入标签
        document.createTextNode()创建文本
        element.remove() 删除标签
     */
    // 需求: 往p标签中添加一个font标签,font标签的文本内容是: 字体标签
    // 1.获取p标签对象  <p></p>
    var pElement = document.getElementById("pId");

    function addElement() {
        // 1.获取p标签对象  <p></p>
        // var pElement = document.getElementById("pId");

        // 2.创建font标签   <font></font>
        var fontElement = document.createElement("font");

        // 3.创建font的文本   字体标签
        var textElement = document.createTextNode("字体标签");

        // 4.把font文本添加到font标签 <font>字体标签</font>
        fontElement.appendChild(textElement);

        // 5.把font标签插入到p标签中  <p><font>字体标签</font></p>
        pElement.appendChild(fontElement);

    }

    function removeElement() {
        pElement.remove();
    }

</script>
</html>
```

3.4 操作标签体[重点]

语法

- 获取标签体内容：`标签对象.innerHTML`
- 设置标签体内容：
    - `innerHTML`是覆盖式设置，原本的标签体内容会被覆盖掉；`标签对象.innerHTML = "新的HTML代码";`

- 支持标签的 可以插入标签, 设置的html代码会生效; `标签对象.innerHTML += "新的HTML代码";`

示例

```
<body>
<div id="divId1">
    div1
</div>
<input type="button" value="修改div内容" onclick="f1()">
</body>
<script>
    // 点击按钮,修改div中的标签体为: itheima    并且itheima要为红色
    function f1() {
        // 1.获得div标签对象
        var divE = document.getElementById("divId1");

        // 2.修改标签体内容
        // divE.innerHTML = "<font color='red'>itheima</font>";// 覆盖
        divE.innerHTML += "<font color='red'>itheima</font>";// 追加
    }

</script>
```

3.5 操作属性

- 每个标签 `Element` 对象提供了操作属性的方法

方法名	描述	参数
`getAttribute(attrName)`	获取属性值	属性名称
`setAttribute(attrName, attrValue)`	设置属性值	属性名称, 属性值
`removeAttribute(attrName)`	删除属性	属性名称

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<img id="imgId" src="../img/b.jpg">
<input type="button" value="获取src属性" onclick="getSrc()">
<input type="button" value="修改src属性" onclick="setSrc()">
<input type="button" value="删除src属性" onclick="removeSrc()">
</body>
<script>
    function getSrc() {
        // 1.根据标签名获取标签对象
        // var e = document.getElementsByTagName("img");// 返回的是数组(长度为1的数组)
        // console.log(e[0].getAttribute("src"));
        var imgElement = document.getElementById("imgId");
```

```
        // 2.根据img标签对象获取src属性
        console.log(imgElement.getAttribute("src"));
    }

    function setSrc() {
        // 1.根据标签名获取标签对象
        var imgElement = document.getElementById("imgId");

        // 2.根据img标签对象修改src属性
        imgElement.setAttribute("src","../img/girl.jpg");
    }

    function removeSrc() {
        // 1.根据标签名获取标签对象
        var imgElement = document.getElementById("imgId");

        // 2.根据img标签对象删除src属性
        imgElement.removeAttribute("src");
    }
</script>
</html>
```

第四章-JS案例

案例-使用JS完成表单的校验

1. 案例需求

- 课堂完成: 用户名校验
- 课后完成: 手机号码,电子邮箱校验

2.思路分析

```
1.为用户名输入框绑定获得焦点事件和失去焦点事件
2.完成获得焦点事件函数:
    2.1 获得用户名输入框后面的span标签
    2.2 把提示信息作为标签体设置到span标签中

3.完成失去焦点事件函数:
    3.1 获得用户输入的用户名
    3.2 创建正则表达式
    3.3 使用正则表达式校验用户输入的用户名
    3.4 获得用户名输入框后面的span标签
    3.5 根据判断结果,显示提示信息(设置到span标签中)
```

3.代码实现

```
<script>
    function showTips() {
        // 2.完成获得焦点事件函数:
        // 2.1 获得用户名输入框后面的span标签
        var spanE = document.getElementById("usernamespan");

        // 2.2 把提示信息作为标签体设置到span标签中
        spanE.innerHTML = "以字母开头,由字母和数字组成4-16位";
    }

    function checkUsername(obj) {
        // 3.1 获得用户输入的用户名
        var username = obj.value;

        // 3.2 创建正则表达式
        var reg = /^[a-zA-Z][a-zA-Z0-9]{3,15}$/;

        // 3.3 使用正则表达式校验用户输入的用户名
        var flag = reg.test(username);

        // 3.4 获得用户名输入框后面的span标签
        var spanE = document.getElementById("usernamespan");

        // 3.5 根据判断结果,显示提示信息(设置到span标签中)
        if (flag){
            spanE.innerHTML = "<img src='img/gou.png' width='10px'
height='10px'/>";
        } else {
            spanE.innerHTML = "用户名格式不对!";
        }
    }
</script>
```

> 做一下验证手机号码,email格式 var reg = /^([a-zA-Z]|[0-9])(\w|-)+@[a-zA-Z0-9]+.([a-zA-Z]{2,4})$/;

4,小结

- 事件,函数,正则表达式...

案例-使用JS完成图片轮播效果

1.需求分析

- 实现每过3秒中切换一张图片的效果，一共3张图片，当显示到最后1张的时候，再次显示第1张。

2.思路分析

```
1.设置一个循环定时器(每隔3s调用一个函数changeImg)
2.在changeImg函数中,获得img标签,然后修改img标签的src属性
```

3.代码实现

```
<script>
    // 1.设置一个循环定时器(每隔3s调用一个函数changeImg)
    setInterval("changeImg()", 500);

    // 2.在changeImg函数中,获得img标签,然后修改img标签的src属性
    var i = 0;

    function changeImg() {
        // 2.1 获得img标签对象
        var imgElement = document.getElementById("imgId");

        // 2.2 修改img标签的src属性
        i++;
        // 一轮完毕之后,重置i的值
        if (i == 4) {
            i = 1;
        }
        imgElement.setAttribute("src", "../img/banner_" + i + ".jpg")
    }
</script>
```

4.小结

1. 定时任务

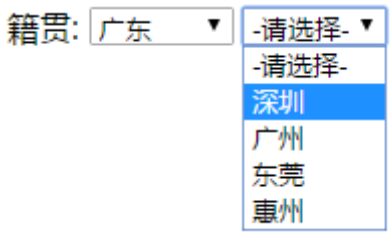
```
setInterval("函数名()", time);
```

2. 操作属性

```
setAttribute("属性名","属性值")
```

案例-JS控制二级联动

1.需求分析



- 在注册页面添加籍贯,左边是省份的下拉列表,右边是城市的下拉列表.右边的select根据左边的改变而更新数据

2.思路分析

```
1.创建页面(含有2个下拉框,分别是省份下拉框,城市下拉框)
2.为省份下拉框添加内容改变事件
3.在内容改变事件对应的函数中:
3.1 获取选择的省份数据
3.2 根据选择的省份数据获取对应的城市数据(eg:广州,深圳,东莞,惠州)
3.3 获得城市下拉框对象
3.4 循环遍历城市数据
3.5 创建option标签对象
3.6 创建城市数据文本对象
3.7 把城市文本对象添加到option标签中 (<option>广州</option>)
3.8 把option标签对象添加到城市下拉框对象中
```

3.代码实现

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>案例-JS控制二级联动</title>
</head>
<body>
籍贯:
<select onchange="changeCitys(this)">
    <option value="-1">-请选择-</option>
    <option value="0">广东</option>
    <option value="1">山东</option>
    <option value="2">湖北</option>
```

```
</select>
<select id="citys">
    <option>-请选择-</option>
</select>

</body>
<script>
    // 三个省份对应的城市
    var arr = [
        ["深圳", "广州", "惠州", "东莞"],
        ["济南", "青岛", "烟台", "淄博"],
        ["武汉", "荆门", "黄冈", "十堰"]
    ];

    // 1.创建页面(含有2个下拉框,分别是省份下拉框,城市下拉框)
    // 2.为省份下拉框添加内容改变事件
    // 3.在内容改变事件对应的函数中:
    function changeCitys(obj) {
        // 3.1 获取选择的省份数据
        var value = obj.value;// -1

        //  获得城市下拉框对象
        var citysElement = document.getElementById("citys");

        // 清空城市下拉框中的选项
        // citysElement.innerHTML = "";
        citysElement.innerHTML = "<option>-请选择-</option>";


        // 只有值大于等于0,才去改变城市下拉框
        if (value >= 0) {
            // 3.2 根据选择的省份数据获取对应的城市数据(eg:广州,深圳,东莞,惠州)
            var cityNamesArr = arr[value];

            // 3.4 循环遍历城市数据
            for (var i = 0; i < cityNamesArr.length; i++) {
                // 3.5 创建option标签对象
                var optionE = document.createElement("option");

                // 3.6 创建城市数据文本对象
                var text = document.createTextNode(cityNamesArr[i]);

                // 3.7 把城市文本对象添加到option标签中 (eg: <option>广州</option>)
                optionE.appendChild(text);

                // 3.8 把option标签对象添加到城市下拉框对象中
                citysElement.appendChild(optionE);
            }
        }

    }
</script>
</html>
```

4.小结

1. 内容改变事件

```
<select onchange="函数()"></select>
```

2. 二维数组
3. innerHTML
    - 会把前面的内容覆盖掉
    - 支持标签的插入
4. dom
    - 父节点.appendChild(子节点)
    - document.createElement()
    - document.createTextNode()

总结

```
必须练习：
    1.课堂上三个js案例独立完成

- 了解js的作用
    - JS代码可以操作浏览器：进行网址跳转、历史记录切换、浏览器弹窗等等
    - JS代码可以操作网页：操作HTML的标签、标签的属性、样式等等
    注意：JS的是在浏览器内存中运行时操作，并不会修改网页源码，所以刷新页面后网页会还原

- 能够在HTML里引入js
    内联方式:在html里增加<script>标签，把js代码写在标签体里
    外联方式:把js代码写在单独的js文件中，js文件后缀名是.js
             在HTML里使用<script>标签引入外部js文件

- 能够使用浏览器开发者工具调试js代码
      F12

- 掌握js的基本语法
    变量:var 变量名 = 值;
    数据类型: 数值类型,布尔类型,字符串类型,对象类型,未定义类型
    运算符:
        / 会取小数;
        数值和字符串可以-,*,/,%操作,字符串会自动转换为数值;
        == 比较值是否相等
        === 比较值和类型都是否相等
    循环\控制语句
        循环语句,条件判断语句(if,switch)
- 能够定义和调用函数
     普通函数:
            function 函数名(形参列表){
                函数体
                [return 返回值;]
            }
            注意:形参不要写var
      匿名函数:
            function (形参列表){
                函数体
                [return 返回值;]
            }
- 能够使用js的内置对象
    Date日期对象,Array数组对象,常用函数
```

- 能够绑定js事件
    方式一: <标签 属性="js代码，调用函数"></标签>
    方式二: <script>
        标签对象.事件属性 = function(){
            //执行一段代码
        }
    </script>
- 能够使用正则表达式校验字符串格式
    var reg = /^表达式$/;
    test(str);
- 能够使用js的bom操作浏览器
    windows,navigator,screen,history,location
- 能够使用js的dom操作网页
    获取标签
    document.getElementById(id) 根据id获取标签 返回值:Element对象
    document.getElementsByName(name) 根据标签name获取一批标签 返回值:Element类数组
    document.getElementsByTagName(tagName) 根据标签名称获取一批标签 返回值:Element类数组
    document.getElementsByClassName(className) 根据类名获取一批标签 返回值:Element类数组

    操作标签
    document.createElement(tagName) 创建标签 Element对象
     parentElement.appendChild(sonElement) 插入标签
    element.remove() 删除标签
    document.createTextNode()创建文本

    操作标签体
    - 获取标签体内容：标签对象.innerHTML
    - 设置标签体内容：标签对象.innerHTML = "新的HTML代码";

    操作属性
        1. getAttribute(attrName) 获取属性值
        2. setAttribute(attrName, attrValue) 设置属性值
        3. removeAttribute(attrName) 删除属性