

# [笔记][李立超 JavaScript 基础]

JavaScript

---

## [笔记][李立超 JavaScript 基础]

- 001. JS简介
- 002. JS的HelloWorld
- 003. js编写位置
- 004. 基本语法
- 005. 字面量和变量
- 006. 标识符
- 007. 字符串
- 008. Number
- 117. 事件的传播
- 118. 事件练习—拖拽（一）
- 119. 事件练习—拖拽（二）
- 120. 事件练习—拖拽（三）
- 121. 滚轮事件
- 122. 键盘事件
- 123. 键盘移动div
- 124. Navigator
- 125. History
- 126. Location
- 127. 定时器简介
- 128. 切换图片练习
- 129. 修改div移动练习
- 130. 延时调用
- 131. 定时器的应用（一）
- 132. 定时器的应用（二）
- 133. 定时器的应用（三）
- 134. 完成轮播图界面
- 135. 完成点击按钮切换图片
- 136. 完成轮播图
- 137. 类的操作
- 138. 二级菜单基本功能
- 139. 二级菜单过渡效果
- 140. JSON

---

## 001. JS简介

## 什么是语言

我们要学习的语言就是人和计算机交流的工具，人类通过语言来控制、操作计算机。编程语言和我们说的中文、英文本质上没有区别，只是语法比较特殊。

## 语言的发展

- 纸带机：机器语言
- 汇编语言：符号语言
- 现代语言：高级语言

## 起源

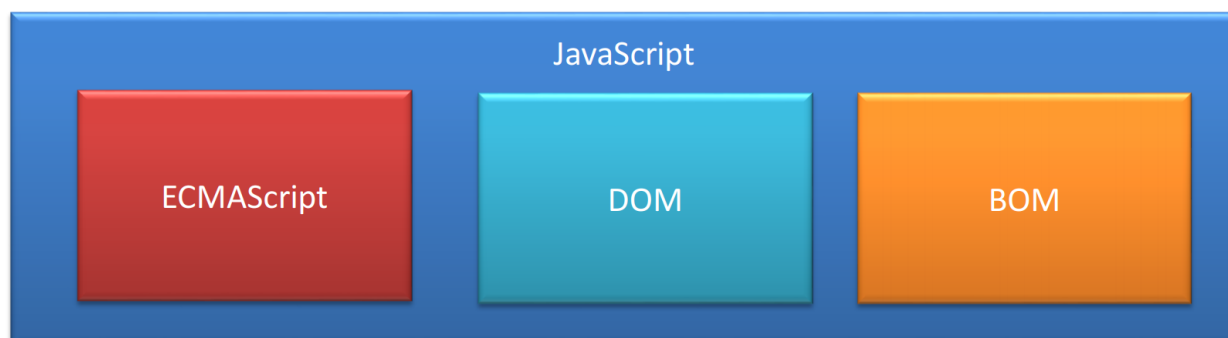
- JavaScript 诞生于 1995 年，它的出现主要是用于处理网页中的前端验证。
- 所谓的前端验证，就是指检查用户输入的内容是否符合一定的规则。
- JavaScript 是由网景公司发明，起初命名为 LiveScript，后来由于 SUN 公司的介入更名为 JavaScript。
- 1996 年微软公司在其最新的 IE3 浏览器中引入了自己对 JavaScript 的实现 JScript。
- 于是在市面上存在两个版本的 JavaScript，一个网景公司的 JavaScript 和微软的 JScript。
- 为了确保不同的浏览器上运行的 JavaScript 标准一致，所以几个公司共同定制了 JS 的标准命名为 ECMAScript。

1997 年出现了 ECMAScript 第 1 版 ( ECMA-262 )

ECMAScript 是一个标准，而这个标准需要由各个厂商去实现。

浏览器	JavaScript实现方式
FireFox	SpiderMonkey
Internet Explorer	JScript/Chakra
Safari	JavaScriptCore
Chrome	v8
Carakan	Carakan

一般认为 ECMAScript 等于 JavaScript，但其实 JavaScript 的范畴更大：



## JS 的特点

- 解释型语言
- 类似于 C 和 Java 的语法结构
- 动态语言
- 基于原型的面向对象

### 解释型语言

- **JavaScript** 是一门解释型语言，所谓解释型语言不需要被编译为机器码在执行，而是直接执行。
- 由于少了编译这一步骤，所以解释型语言开发起来尤为轻松，但是解释型语言运行较慢也是它的劣势。
- 不过解释型语言中使用了 **JIT** 技术，使得运行速度得以改善。

## 002. JS的HelloWorld

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <!--JS代码需要编写到script标签中-->
    <script type="text/javascript">

      /*
       * 控制浏览器弹出一个警告框
       * alert("哥，你真帅啊！！");
       */

      /*
       * 让计算机在页面中输出一个内容
       * document.write()可以向body中输出一个内容
       * document.write("看我出不出来~~~");
       */

      /*
       * 向控制台输出一个内容
       * console.log()的作用是向控制台输出一个内容
       * console.log("你猜我在哪出来呢？");
       */

      alert("哥，你真帅啊！！");
      document.write("看我出不出来~~~");
      console.log("你猜我在哪出来呢？");
    </script>
  </head>
```

```
<body>
</body>
</html>
```

## 003. js编写位置

### 编写位置

- 推荐：编写为外部的一个单独 `js` 文件，通过 `script` 标签的 `src` 属性引入。
- 编写在 `script` 标签中。
- 编写在 `html` 标签的 `onclick` 属性中，形式为 `onclick="要执行的语句;"`。
- 编写在 `a` 标签的 `href` 属性中，形式为 `href="javascript:要执行的语句;"`。

### 注意

- 当 `script` 标签通过指定 `src` 引入外部 `js` 文件后，该标签内部的任何 `js` 代码都不起作用！
- `js` 代码按照上下顺序一行一行执行，所以在前面的会先执行。

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

    <!--
      可以将js代码编写到外部js文件中，然后通过script标签引入
      写到外部文件中可以在不同的页面中同时引用，也可以利用到浏览器的缓存机制
      推荐使用的方式
    -->
    <!--
      script标签一旦用于引入外部文件了，就不能在编写代码了，即使编写了浏览器也
      会忽略
      如果需要则可以在创建一个新的script标签用于编写内部代码
    -->
    <script type="text/javascript" src="js/script.js"></script>
    <script type="text/javascript">
      alert("我是内部的JS代码");
    </script>

    <!--
      可以将js代码编写到script标签
    <script type="text/javascript">

      alert("我是script标签中的代码!!");

    </script>
```

```

-->
</head>
<body>

<!--
    可以将js代码编写到标签的onclick属性中
    当我们点击按钮时，js代码才会执行

    虽然可以写在标签的属性中，但是他们属于结构与行为耦合，不方便维护，不推荐使用
-->
<button onclick="alert('讨厌，你点我干嘛~~');">点我一下</button>

<!--
    可以将js代码写在超链接的href属性中，这样当点击超链接时，会执行js代码
-->
<a href="javascript:alert('让你点你就点！！');">你也点我一下</a>
<a href="javascript:;">你也点我一下</a>
</body>
</html>

```

## 004. 基本语法

day05/08. 基本语法.html

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">

      /*
        多行注释
        JS注释
        多行注释，注释中的内容不会被执行，但是可以在源代码中查看
        要养成良好的编写注释的习惯，也可以通过注释来对代码进行一些简单的
        调试
      */

      //单行注释
      //alert("hello");
      //document.write("hello");
      console.log("hello"); //该语句用来在控制台输出一个日志

    /*

```

化

```
* 1.JS中严格区分大小写
* 2.JS中每一条语句以分号(;)结尾
*     - 如果不写分号，浏览器会自动添加，但是会消耗一些系统资源，
*       而且有些时候，浏览器会加错分号，所以在开发中分号必须写
* 3.JS中会忽略多个空格和换行，所以我们可以利用空格和换行对代码进行格式

*
*/

    alert("hello");
</script>
</head>
<body>
</body>
</html>
```

## 005. 字面量和变量

day05/09.字面量和变量.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">

      /*
      * 字面量，都是一些不可改变的值
      *     比如：1 2 3 4 5
      *     字面量都是可以直接使用，但是我们一般都不会直接使用字面量
      *
      * 变量      变量可以用来保存字面量，而且变量的值是可以任意改变的
      *     变量更加方便我们使用，所以在开发中都是通过变量去保存一个字面量，
      *     而很少直接使用字面量
      *     可以通过变量对字面量进行描述
      */

      //声明变量
      //在js中使用var关键字来声明一个变量
      var a;

      //为变量赋值
      a = 123;
      a = 456;
      a = 123124223423424;
```

```

        //声明和赋值同时进行
        var b = 789;
        var c = 0;

        var age = 80;
        console.log(age);
    </script>
</head>
<body>
</body>
</html>

```

## 006. 标识符

day05/10. 标识符.html

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">
      /*
       * 标识符
       * - 在JS中所有的可以由我们自主命名的都可以称为是标识符
       * - 例如：变量名、函数名、属性名都属于标识符
       * - 命名一个标识符时需要遵守如下的规则：
       *   1. 标识符中可以含有字母、数字、_、$
       *   2. 标识符不能以数字开头
       *   3. 标识符不能是ES中的关键字或保留字
       *   4. 标识符一般都采用驼峰命名法
       *     - 首字母小写，每个单词的开头字母大写，其余字母小写
       *       helloWorld xxxYyyZzz
       *
       * - JS底层保存标识符时实际上是采用的Unicode编码，
       *   所以理论上讲，所有的utf-8中含有的内容都可以作为标识符
       */
      /*var if = 123;

      console.log(if);*/

      //千万不要这么用
      var 锄禾日当午 = 789;
      console.log(锄禾日当午);
    </script>

```

```
</head>
<body>
</body>
</html>
```

底层采用 **Unicode** 编码保存标识符，所以只要是 **Unicode** 字符都可以用来作标识符，但是最好不要用奇奇怪怪的字符或者中文。

团子注：**Python** 中也一样。

## 007. 字符串

day06/01.数据类型.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
    <script type="text/javascript">

      /*
       * 数据类型指的就是字面量的类型
       * 在JS中一共有六种数据类型
       *   String 字符串
       *   Number 数值
       *   Boolean 布尔值
       *   Null 空值
       *   Undefined 未定义
       *   Object 对象
       *
       * 其中String Number Boolean Null Undefined属于基本数据类型
       * 而Object属于引用数据类型
       */

      /*
       * String字符串
       * - 在JS中字符串需要使用引号引起来
       * - 使用双引号或单引号都可以，但是不要混着用
       * - 引号不能嵌套，双引号不能放双引号，单引号不能放单引号
       */
      var str = 'hello';

      str = '我说:"今天天气真不错! "';
```



```

    /*
    在字符串中我们可以使用\作为转义字符，
    当表示一些特殊符号时可以使用\进行转义

    \" 表示 "
    \' 表示 '
    \n 表示换行
    \t 制表符
    \\ 表示\

    * */
    str = "我说:\"今天\t天气真不错! \"";

    str = "\\\"\\\"\\\"";

    //输出字面量 字符串str
    //alert("str");

    //输出变量str
    //alert(str);

    var str2 = "hello";
    str2 = "你好";
    str2 = 3;

    //alert("hello, 你好");
    //console.log("我就是不出来~~~");
</script>
</head>
<body>
</body>
</html>

```

## 008. Number

day06/02.Number.html

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">
      /*
      * 在JS中所有的数值都是Number类型，
      * 包括整数和浮点数（小数）
      *
      * JS中可以表示的数字的最大值

```

```

    *   Number.MAX_VALUE
    *       1.7976931348623157e+308
    *
    *   Number.MIN_VALUE 大于0的最小值
    *       5e-324
    *
    *   如果使用Number表示的数字超过了最大值，则会返回一个
    *       Infinity 表示正无穷
    *       -Infinity 表示负无穷
    *       使用typeof检查Infinity也会返回number
    *   NaN 是一个特殊的数字，表示Not A Number
    *       使用typeof检查一个NaN也会返回number
    */
    //数字123
    var a = 123;
    //字符串123
    var b = "123";
    /*
        可以使用一个运算符 typeof
        来检查一个变量的类型
        语法: typeof 变量
        检查字符串时，会返回string
        检查数值时，会返回number
    */
    //console.log(typeof b);

    a = -Number.MAX_VALUE * Number.MAX_VALUE; // -Infinity
    a = "abc" * "bcd"; // NaN
    a = NaN;

    //console.log(typeof a);

    a = Number.MIN_VALUE;

    //console.log(a);

    /*
        *   在JS中整数的运算基本可以保证精确
    */
    var c = 1865789 + 7654321;

    /*
        *   如果使用JS进行浮点运算，可能得到一个不精确的结果
        *   所以千万不要使用JS进行对精确度要求比较高的运算
    */
    var c = 0.1 + 0.2;

    console.log(c);
</script>
</head>
<body>
</body>

```

```
</html>
```

### 团子注

- `typeof` 是一个关键字，后面的东西不用放到小括号里，语法是：`typeof 变量`。
- 如果小数点前面是 `0` 的话还可以省略，比如 `0.3` 写成 `.3`。
- `typeof NaN` 得到的是 `"number"`。

## 117. 事件的传播



### 03. 事件的传播.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">

      #box1{
        width: 300px;
```

```
height: 300px;
background-color: yellowgreen;
}
```

```
#box2{
width: 200px;
height: 200px;
background-color: yellow;
}
```

```
#box3{
width: 150px;
height: 150px;
background-color: skyblue;
}
```

```
</style>
```

```
<script type="text/javascript">
```

```
window.onload = function(){
```

```
/*
```

```
 * 分别为三个div绑定单击响应函数
```

```
*/
```

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

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

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

```
/*
```

```
 * 事件的传播
```

```
 * - 关于事件的传播网景公司和微软公司有不同的理解（不同的设计）
```

```
 * - 微软公司认为事件应该是由内向外传播，也就是当事件触发时，应该先触发当前元素上的事件，
```

```
 *          然后再向当前元素的祖先元素上传播，也就是说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的，也就是当前事件触发时，应该先触发当前元素的最外层的祖先元素的事件，
```

```
 *          然后在向内传播给后代元素
```

```
 * - W3C综合了两个公司的方案，将事件传播分成了三个阶段
```

```
 *          1. 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
```

```
 *          2. 目标阶段
```

```
 *          - 事件捕获到目标元素，捕获结束开始在目标元素上触发事件
```

```
 *          3. 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
```

一般都是false

\* 一般情况下我们不会希望在捕获阶段触发事件，所以这个参数

\*

\* - IE8及以下的浏览器中没有捕获阶段

\*/

```
bind(box1,"click",function(){
    alert("我是box1的响应函数")
});
```

```
bind(box2,"click",function(){
    alert("我是box2的响应函数")
});
```

```
bind(box3,"click",function(){
    alert("我是box3的响应函数")
});
```

```
};
```

```
function bind(obj , eventStr , callback){
    if(obj.addEventListener){
        //大部分浏览器兼容的方式
        obj.addEventListener(eventStr , callback , true);
    }else{
        /*
        * this是谁由调用方式决定
        * callback.call(obj)
        */
        //IE8及以下
        obj.attachEvent("on"+eventStr , function(){
            //在匿名函数中调用回调函数
            callback.call(obj);
        });
    }
}
```

```
</script>
```

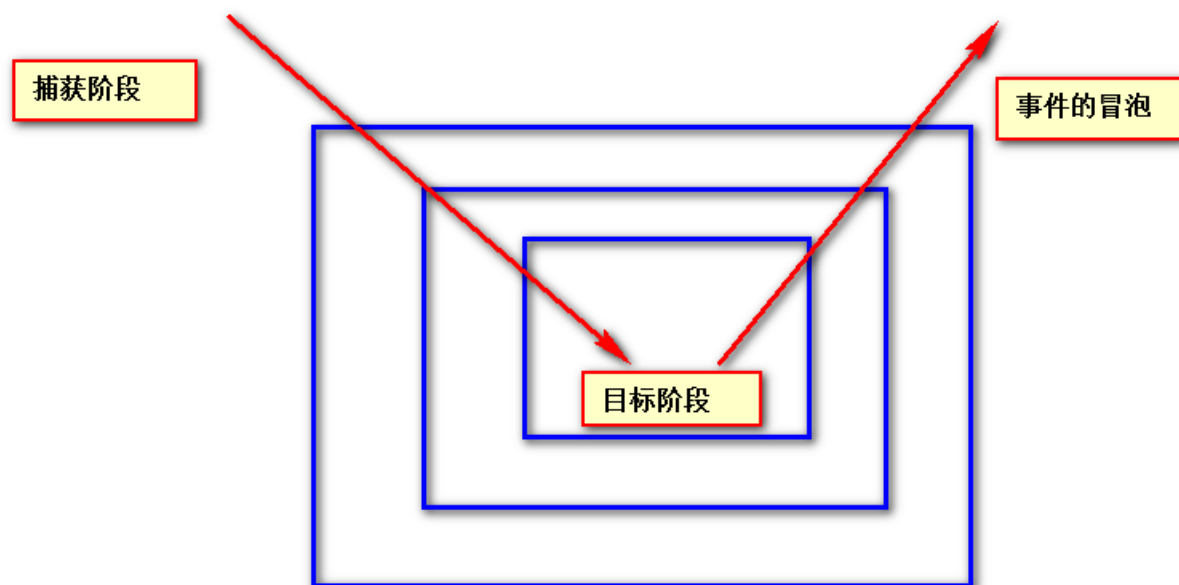
```
</head>
```

```
<body>
```

```
<div id="box1">
    <div id="box2">
        <div id="box3"></div>
    </div>
</div>
```

```
</body>
```

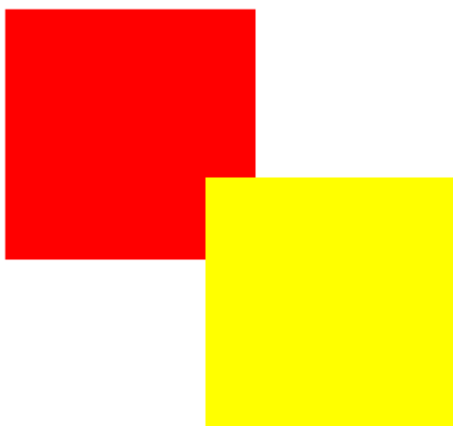
```
</html>
```



**W3C** 默认设置从document开始捕获，但是大部分浏览器设置从window开始捕获。

**IE8** 及以下只有冒泡阶段。

## 118. 事件练习—拖拽（一）



04. 拖拽.html

```
<!DOCTYPE html>
<html>
  <head>
```

```

<meta charset="UTF-8">
<title></title>
<style type="text/css">

    #box1{
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
    }

    #box2{
        width: 100px;
        height: 100px;
        background-color: yellow;
        position: absolute;

        left: 200px;
        top: 200px;
    }

</style>

<script type="text/javascript">

    window.onload = function(){
        /*
        * 拖拽box1元素
        * - 拖拽的流程
        *     1.当鼠标在被拖拽元素上按下时，开始拖拽  onmousedown
        *     2.当鼠标移动时被拖拽元素跟随鼠标移动  onmousemove
        *     3.当鼠标松开时，被拖拽元素固定在当前位置  onmouseup
        */

        //获取box1
        var box1 = document.getElementById("box1");
        //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  onmousedown
        box1.onmousedown = function(event){
            event = event || window.event;
            //div的偏移量 鼠标.clientX - 元素.offsetLeft
            //div的偏移量 鼠标.clientY - 元素.offsetTop
            var ol = event.clientX - box1.offsetLeft;
            var ot = event.clientY - box1.offsetTop;

            //为document绑定一个onmousemove事件
            document.onmousemove = function(event){
                event = event || window.event;
                //当鼠标移动时被拖拽元素跟随鼠标移动  onmousemove
                //获取鼠标的坐标
                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";

    };

    //为document绑定一个鼠标松开事件
    document.onmouseup = function(){
        //当鼠标松开时，被拖拽元素固定在当前位置    onmouseup
        //取消document的onmousemove事件
        document.onmousemove = null;
        //取消document的onmouseup事件
        document.onmouseup = null;
    };
};

</script>
</head>
<body>
    <div id="box1"></div>
    <div id="box2"></div>
</body>
</html>

```

### 注意

- `document.onmousemove` 必须放在 `box1.onmousedown` 里面，否则不按鼠标红方块也会跟随移动
- `document.onmouseup` 不能绑定到 `box1` 上面，否则在黄方块内松开鼠标无效。
- 仍然存在的问题，如何让鼠标和红方块的相对位置保持一致？（上面的代码已经修正这个问题）

## 119. 事件练习—拖拽（二）

略

## 120. 事件练习—拖拽（三）

问题：`ctrl + a` 拖拽的时候会出现问题。



我是一段文字



我是一段文字



解决：在 `onmousedown` 最后 `return false;`，但是对 `IE8` 无效。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">

      #box1{
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
      }

      #box2{
        width: 100px;
        height: 100px;
        background-color: yellow;
        position: absolute;

        left: 200px;
        top: 200px;
      }

    </style>

    <script type="text/javascript">

      window.onload = function(){
```

```

/*
 * 拖拽box1元素
 * - 拖拽的流程
 *     1.当鼠标在被拖拽元素上按下时，开始拖拽  onmousedown
 *     2.当鼠标移动时被拖拽元素跟随鼠标移动  onmousemove
 *     3.当鼠标松开时，被拖拽元素固定在当前位置  onmouseup
 */

//获取box1
var box1 = document.getElementById("box1");
var box2 = document.getElementById("box2");
//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  onmousedown
box1.onmousedown = function(event){

    //设置box1捕获所有鼠标按下的事件
    /*
     * setCapture()
     * - 只有IE支持，但是在火狐中调用时不会报错，
     *     而如果使用chrome调用，会报错
     */
    /*if(box1.setCapture){
        box1.setCapture();
    }*/
    box1.setCapture && box1.setCapture();

    event = event || window.event;
    //div的偏移量 鼠标.clientX - 元素.offsetLeft
    //div的偏移量 鼠标.clientY - 元素.offsetTop
    var ol = event.clientX - box1.offsetLeft;
    var ot = event.clientY - box1.offsetTop;

    //为document绑定一个onmousemove事件
    document.onmousemove = function(event){
        event = event || window.event;
        //当鼠标移动时被拖拽元素跟随鼠标移动  onmousemove
        //获取鼠标的坐标
        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";

    };

    //为document绑定一个鼠标松开事件
    document.onmouseup = function(){
        //当鼠标松开时，被拖拽元素固定在当前位置  onmouseup
        //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的异常，这个是浏览器提供的默认行为，
    * 如果不希望发生这个行为，则可以通过return false来取消默认
行为

    *
    * 但是这招对IE8不起作用
    */
    return false;
};
};
</script>
</head>
<body>
    我是一段文字
    <div id="box1"></div>
    <div id="box2"></div>
</body>
</html>

```

IE8 需要使用 `setCapture()` 和 `releaseCapture()`

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script type="text/javascript">

            window.onload = function(){
                //分别为两个按钮绑定单击响应函数
                var btn01 = document.getElementById("btn01");
                var btn02 = document.getElementById("btn02");

                btn01.onclick = function(){
                    alert(1);
                };

                btn02.onclick = function(){
                    alert(2);
                };

                //设置btn01对鼠标按下相关的事件进行捕获
            }
        </script>
    </head>
    <body>
        我是一段文字
        <div id="box1"></div>
        <div id="box2"></div>
    </body>
</html>

```

//当调用一个元素的setCapture()方法以后，这个元素将会把下一次所有的鼠标按下相关的事件捕获到自身上

```
        btn01.setCapture();
    };

</script>
</head>
<body>
    <button id="btn01">按钮01</button>
    <button id="btn02">按钮02</button>
</body>
</html>
```

## 最终代码

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <style type="text/css">

            #box1{
                width: 100px;
                height: 100px;
                background-color: red;
                position: absolute;
            }

            #box2{
                width: 100px;
                height: 100px;
                background-color: yellow;
                position: absolute;

                left: 200px;
                top: 200px;
            }

        </style>

        <script type="text/javascript">

            window.onload = function(){
                /*
                 * 拖拽box1元素
                 * - 拖拽的流程
                 *     1. 当鼠标在被拖拽元素上按下时，开始拖拽    onmousedown
                 *     2. 当鼠标移动时被拖拽元素跟随鼠标移动    onmousemove
                 *     3. 当鼠标松开时，被拖拽元素固定在当前位置    onmouseup
                */
            }
        </script>
    </head>
</html>
```

```

*/

//获取box1
var box1 = document.getElementById("box1");
var box2 = document.getElementById("box2");
var img1 = document.getElementById("img1");

//开启box1的拖拽
drag(box1);
//开启box2的
drag(box2);
drag(img1);
};

/*
* 提取一个专门用来设置拖拽的函数
* 参数: 开启拖拽的元素
*/
function drag(obj){
    //当鼠标在被拖拽元素上按下时, 开始拖拽 onmousedown
    obj.onmousedown = function(event){

        //设置box1捕获所有鼠标按下的事件
        /*
        * setCapture()
        * - 只有IE支持, 但是在火狐中调用时不会报错,
        *   而如果使用chrome调用, 会报错
        */
        /*if(box1.setCapture){
            box1.setCapture();
        }*/
        obj.setCapture && obj.setCapture();

        event = event || window.event;
        //div的偏移量 鼠标.clientX - 元素.offsetLeft
        //div的偏移量 鼠标.clientY - 元素.offsetTop
        var ol = event.clientX - obj.offsetLeft;
        var ot = event.clientY - obj.offsetTop;

        //为document绑定一个onmousemove事件
        document.onmousemove = function(event){
            event = event || window.event;
            //当鼠标移动时被拖拽元素跟随鼠标移动 onmousemove
            //获取鼠标的坐标
            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";
        };
    };
}

```

容，

行为

```
//为document绑定一个鼠标松开事件
document.onmouseup = function(){
    //当鼠标松开时，被拖拽元素固定在当前位置    onmouseup
    //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
 * 此时会导致拖拽功能的异常，这个是浏览器提供的默认行为，
 * 如果不希望发生这个行为，则可以通过return false来取消默认
 *
 * 但是这招对IE8不起作用
 */
return false;
};
}
</script>
</head>
<body>
    我是一段文字
    <div id="box1"></div>
    <div id="box2"></div>
    
</body>
</html>
```

我是一段文字



## 121. 滚轮事件

滚轮向上滚变短，向下滚变长

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">
      #box1{
        width: 100px;
        height: 100px;
        background-color: red;
      }
    </style>
    <script type="text/javascript">
      window.onload = function(){
        //获取id为box1的div
        var box1 = document.getElementById("box1");

        //为box1绑定一个鼠标滚轮滚动的事件
        /*
        * onmousewheel鼠标滚轮滚动的事件，会在滚轮滚动时触发，
        * 但是火狐不支持该属性
        */
      }
    </script>
  </head>
</html>
```

```

    *
    * 在火狐中需要使用 DOMMouseScroll 来绑定滚动事件
    * 注意该事件需要通过addEventListener()函数来绑定
    */

box1.onmousewheel = function(event){

    event = event || window.event;

    //event.wheelDelta 可以获取鼠标滚轮滚动的方向
    //向上滚 120  向下滚 -120
    //wheelDelta这个值我们不看大小，只看正负

    //alert(event.wheelDelta);

    //wheelDelta这个属性火狐中不支持
    //在火狐中使用event.detail来获取滚动的方向
    //向上滚 -3  向下滚 3
    //alert(event.detail);

    /*
    * 当鼠标滚轮向下滚动时，box1变长
    * 当滚轮向上滚动时，box1变短
    */
    //判断鼠标滚轮滚动的方向
    if(event.wheelDelta > 0 || event.detail < 0){
        //向上滚，box1变短
        box1.style.height = box1.clientHeight - 10 +
"px";

    }else{
        //向下滚，box1变长
        box1.style.height = box1.clientHeight + 10 +
"px";
    }

    /*
    * 使用addEventListener()方法绑定响应函数，取消默认行为时不
    能使用return false

    * 需要使用event来取消默认行为event.preventDefault();
    * 但是IE8不支持event.preventDefault();这个玩意，如果直接
    调用会报错

    */
    event.preventDefault && event.preventDefault();

    /*
    * 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，如果不希望发生，则可以取消默认行为
    */
    return false;
};

```



```

        //为火狐绑定滚轮事件
        bind(box1,"DOMMouseScroll",box1.onmousewheel);
    };






    function bind(obj , eventStr , callback){
        if(obj.addEventListener){
            //大部分浏览器兼容的方式
            obj.addEventListener(eventStr , callback , false);
        }else{
            /*
            * this是谁由调用方式决定
            * callback.call(obj)
            */
            //IE8及以下
            obj.attachEvent("on"+eventStr , function(){
                //在匿名函数中调用回调函数
                callback.call(obj);
            });
        }
    }
</script>
</head>
<body style="height: 2000px;">
    <div id="box1"></div>
</body>
</html>

```

## 补充

- Opera9.5 之前的 wheelDelta 正负号相反，即向上滚 -120，向下滚 +120。
- onmousewheel 已废弃。使用 onwheel 事件替代。
- onwheel 支持 IE8 以上浏览器，只能通过 addEventListener 来绑定事件，在 DOM 对象中没有 onwheel 属性。
- onwheel 的浏览器支持

## 浏览器支持

事件					
onwheel	31.0	9.0	17.0	不支持	18.0

注意：在 IE 浏览器中，只能通过 addEventListener() 方法支持 wheel 事件。在 DOM 对象中没有 onwheel 属性。

- 参考链接：<https://stackoverflow.com/questions/25204282/mousewheel-wheel-and-dommousewheel-in-javascript>

# 122. 键盘事件

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">

      window.onload = function(){

        /*
        * 键盘事件:
        *   onkeydown
        *     - 按键被按下
        *     - 对于onkeydown来说如果一直按着某个按键不松手，则事件会一
直触发
        *     - 当onkeydown连续触发时，第一次和第二次之间会间隔稍微长一
点，其他的会非常的快
        *       这种设计是为了防止误操作的发生。
        *   onkeyup
        *     - 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
        */

        document.onkeydown = function(event){
          event = event || window.event;

          /*
          * 可以通过keyCode来获取按键的编码
          * 通过它可以判断哪个按键被按下
          * 除了keyCode，事件对象中还提供了几个属性
          *   altKey
          *   ctrlKey
          *   shiftKey
          *     - 这三个用来判断alt ctrl 和 shift是否被按下
          *       如果按下则返回true，否则返回false
          */

          //console.log(event.keyCode);

          //判断一个y是否被按下
          //判断y和ctrl是否同时被按下
          if(event.keyCode === 89 && event.ctrlKey){
            console.log("ctrl和y都被按下了");
          }

        };

        /*document.onkeyup = function(){
          console.log("按键松开了");
        }
      }
    }
  </script>
</head>
</html>

```

```

    };*/

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

    input.onkeydown = function(event){

        event = event || window.event;

        //console.log(event.keyCode);
        //数字 48 - 57
        //使文本框中不能输入数字
        if(event.keyCode >= 48 && event.keyCode <= 57){
            //在文本框中输入内容，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>
</head>
<body>
    <input type="text" />
</body>
</html>

```

## 补充

- 也可以使用 `event.key` 来判断按下的是哪个键，然后使用正则过滤数字，但是兼容性不好。
- 参考网址：<http://www.runoob.com/jsref/event-key-key.html>

## 定义和使用

key 事件在按下按键时返回按键的标识符。






按键标识符是表示键盘按钮的字符串，该属性的返回值可以是：

- 单个字母 (如 "a", "W", "4", "+" 或 "\$")
- 多个字母 (如 "F1", "Enter", "HOME" 或 "CAPS LOCK")

提示：如果你想查看是否按下了 "ALT", "CTRL", "META" 或 "SHIFT" 键，可使用 [altKey](#), [ctrlKey](#), [metaKey](#) 或 [shiftKey](#) 属性。

## 浏览器支持

表格中的数字表示支持该属性的第一个浏览器的版本号。

属性					
key	不支持	9.0	23.0	不支持	不支持

## 语法

```
event.key
```

## 技术细节

返回值：	字符串，表示按键按钮。  可能值： <ul style="list-style-type: none"><li>● 单个字母 (如 "a", "W", "4", "+" 或 "\$")</li><li>● 多个字母 (如 "F1", "Enter", "HOME" 或 "CAPS LOCK")</li></ul> 注意：Chrome，Safari 和 Opera浏览器返回 undefined
DOM 版本：	DOM Level 3 Events

# 123. 键盘移动div

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">
      #box1{
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
      }

    </style>

    <script type="text/javascript">
```

```

//使div可以根据不同的方向键向不同的方向移动
/*
 * 按左键，div向左移
 * 按右键，div向右移
 * 。 。 。
 */
window.onload = function(){

    //为document绑定一个按键按下的事件
    document.onkeydown = function(event){
        event = event || window.event;

        //定义一个变量，来表示移动的速度
        var speed = 10;

        //当用户按了ctrl以后，速度加快
        if(event.ctrlKey){
            speed = 500;
        }

        /*
         * 37 左
         * 38 上
         * 39 右
         * 40 下
         */
        switch(event.keyCode){
            case 37:
                //alert("向左"); left值减小
                box1.style.left = box1.offsetLeft - speed +
"px";

                break;
            case 39:
                //alert("向右");
                box1.style.left = box1.offsetLeft + speed +
"px";

                break;
            case 38:
                //alert("向上");
                box1.style.top = box1.offsetTop - speed + "p
x";

                break;
            case 40:
                //alert("向下");
                box1.style.top = box1.offsetTop + speed + "p
x";

                break;
        }

    };
};

```

```
};

</script>
</head>
<body>
  <div id="box1"></div>
</body>
</html>
```

## 124. Navigator

### 补充

- 全局作用域的变量会作为 `window` 对象的属性保存，全局作用域的函数会作为 `window` 对象的方法保存。
- `Screen` 在移动端用的多。
- 描述类型的时候用大写，用的时候用小写。
- `Edge` 为了不被网站区别对待，把自己的 `navigator.appName` 属性伪装成主流浏览器，即 `Netscape`。
- `navigator.userAgent` 等价于浏览器。
- `Gecko` 是 `CSS` 渲染引擎
- `Edge` 会把 `ActiveXObject` 转成 `false`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
    <script type="text/javascript">
      /*
      * BOM
      * - 浏览器对象模型
      * - BOM可以使我们通过JS来操作浏览器
      * - 在BOM中为我们提供了一组对象，用来完成对浏览器的操作
      * - 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，通过该对象可以来识别不同的浏览器
      *   Location
      *     - 代表当前浏览器的地址栏信息，通过Location可以获取地址栏
      *   History
      *     - 代表浏览器的历史记录，通过History可以操作浏览器的历史记录
      */
    </script>
  </head>
</html>
```

```

*           - 代表浏览器的历史记录，可以通过该对象来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
*           而且该操作只在当次访问时有效
*           Screen
*           - 代表用户的屏幕的信息，通过该对象可以获取到用户的显示器的
相关的信息
*
*
*           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
*
*
*/

//console.log(navigator);
//console.log(location);
//console.log(history);

/*
* Navigator
* - 代表的当前浏览器的信息，通过该对象可以来识别不同的浏览器
* - 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
* - 一般我们只会使用userAgent来判断浏览器的信息，
*   userAgent是一个字符串，这个字符串中包含有用来描述浏览器信息的
内容，
*   不同的浏览器会有不同的userAgent
*
* 火狐的userAgent
*  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:50.0) Gecko/201001
01 Firefox/50.0
*
* Chrome的userAgent
*  Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.
36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36
*
* IE8
*  Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.
0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
*
* IE9
*  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64;
Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.
0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
*
* IE10
*  Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW6
4; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

```

```

        *
        * IE11
        * Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .
        NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center P
        C 6.0; .NET4.0C; .NET4.0E; rv:11.0) like Gecko
        * - 在IE11中已经将微软和IE相关的标识都已经去除了，所以我们基本已经不
        能通过UserAgent来识别一个浏览器是否是IE了
        */

        //alert(navigator.appName);

        var ua = navigator.userAgent;

        console.log(ua);

        if(/firefox/i.test(ua)){
            alert("你是火狐!!!");
        }else if(/chrome/i.test(ua)){
            alert("你是Chrome");
        }else if(/msie/i.test(ua)){
            alert("你是IE浏览器~~~");
        }else if("ActiveXObject" in window){
            alert("你是IE11，枪毙了你~~~");
        }
    }

    /*
    * 如果通过UserAgent不能判断，还可以通过一些浏览器中特有的对象，来判断
    浏览器的信息
    * 比如: ActiveXObject
    */
    /*if("ActiveXObject" in window){
        alert("你是IE，我已经抓住你了~~~");
    }else{
        alert("你不是IE~~~");
    }*/

    /*alert("ActiveXObject" in window);*/
</script>
</head>
<body>

</body>
</html>

```

## 125. History



test01.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <h1>TEST01</h1>
    <a href="test02.html">去test02</a>
  </body>
</html>
```

test02.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <h1>TEST02</h1>
    <a href="02.History.html">去02.History.html</a>
  </body>
</html>
```

02.History.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">
      /*
       * History
       * - 对象可以用来操作浏览器向前或向后翻页
       */
      window.onload = function(){

        //获取按钮对象
        var btn = document.getElementById("btn");

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

```

        * length
        *   - 属性，可以获取到当前访问的链接数量
        */
        //alert(history.length);

        /*
        * back()
        *   - 可以用来回退到上一个页面，作用和浏览器的回退按钮一样
        */
        //history.back();

        /*
        * forward()
        *   - 可以跳转下一个页面，作用和浏览器的前进按钮一样
        */
        //history.forward();

        /*
        * go()
        *   - 可以用来跳转到指定的页面
        *   - 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
        *       -2:表示向后跳转两个页面
        */
        history.go(-2);
    };

};

</script>
</head>
<body>

    <button id="btn">点我一下</button>

    <h1>History</h1>

    <a href="01.BOM.html">去BOM</a>
</body>
</html>

```

## 126. Location

```

<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">
      /*
      * Location
      * - 该对象中封装了浏览器的地址栏的信息
      */
      window.onload = function(){

        //获取按钮对象
        var btn = document.getElementById("btn");

        btn.onclick = function(){

          //如果直接打印location，则可以获取到地址栏的信息（当前页面的完整路径）

          //alert(location);

          /*
          * 如果直接将location属性修改为一个完整的路径，或相对路径
          * 则我们页面会自动跳转到该路径，并且会生成相应的历史记录
          */
          //location = "http://www.baidu.com";
          //location = "01.BOM.html";

          /*
          * assign()
          * - 用来跳转到其他的页面，作用和直接修改location一样
          */
          //location.assign("http://www.baidu.com");

          /*
          * reload()
          * - 用于重新加载当前页面，作用和刷新按钮一样
          * - 如果在方法中传递一个true，作为参数，则会强制清空缓存刷新页面

          */
          //location.reload(true);

          /*
          * replace()
          * - 可以使用一个新的页面替换当前页面，调用完毕也会跳转页面
          *       不会生成历史记录，不能使用回退按钮回退
          */
          location.replace("01.BOM.html");

        };

      };
    </script>
  </head>
</html>

```

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

    <button id="btn">点我一下</button>

    <h1>Location</h1>

    <input type="text" />
    <a href="01.BOM.html">去BOM</a>
</body>
</html>
```

## 补充

## Location 对象属性

属性	描述
<u>hash</u>	设置或返回从井号 (#) 开始的 URL (锚)。
<u>host</u>	设置或返回主机名和当前 URL 的端口号。
<u>hostname</u>	设置或返回当前 URL 的主机名。
<u>href</u>	设置或返回完整的 URL。
<u>pathname</u>	设置或返回当前 URL 的路径部分。
<u>port</u>	设置或返回当前 URL 的端口号。
<u>protocol</u>	设置或返回当前 URL 的协议。
<u>search</u>	设置或返回从问号 (?) 开始的 URL (查询部分)。

ctrl + f5 强制清空缓存刷新

# 127. 定时器简介

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">

      window.onload = function(){

        //获取count
        var count = document.getElementById("count");
```

```

//使count中的内容，自动切换
/*
 * JS的程序的执行速度是非常非常快的
 *  如果希望一段程序，可以每间隔一段时间执行一次，可以使用定时调用
 */
/*for(var i=0 ; i<10000 ; i++){
    count.innerHTML = i;

    alert("hello");
}*/

/*
 * setInterval()
 *   - 定时调用
 *   - 可以将一个函数，每隔一段时间执行一次
 *   - 参数：
 *       1.回调函数，该函数会每隔一段时间被调用一次
 *       2.每次调用间隔的时间，单位是毫秒
 *
 *   - 返回值：
 *       返回一个Number类型的数据
 *       这个数字用来作为定时器的唯一标识
 */
var num = 1;

var timer = setInterval(function(){

    count.innerHTML = num++;

    if(num == 11){
        //关闭定时器
        clearInterval(timer);
    }

},1000);

//console.log(timer);

//clearInterval()可以用来关闭一个定时器
//方法中需要一个定时器的标识作为参数，这样将关闭标识对应的定时器
//clearInterval(timer);

};
</script>
</head>
<body>
    <h1 id="count"></h1>
</body>
</html>

```

## 128. 切换图片练习



开始

停止

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">

      window.onload = function(){

        /*
         * 使图片可以自动切换
         */

        //获取img标签
        var img1 = document.getElementById("img1");

        //创建一个数组来保存图片的路径
        var imgArr = ["img/1.jpg","img/2.jpg","img/3.jpg","img/4.
jpg","img/5.jpg"];

        //创建一个变量，用来保存当前图片的索引
        var index = 0;

        //定义一个变量，用来保存定时器的标识
```

```

var timer;

//为btn01绑定一个单击响应函数
var btn01 = document.getElementById("btn01");
btn01.onclick = function(){

    /*
    * 目前，我们每点击一次按钮，就会开启一个定时器，
    * 点击多次就会开启多个定时器，这就导致图片的切换速度过快，
    * 并且我们只能关闭最后一次开启的定时器
    */

    //在开启定时器之前，需要将当前元素上的其他定时器关闭
    clearInterval(timer);

    /*
    * 开启一个定时器，来自动切换图片
    */
    timer = setInterval(function(){
        //使索引自增
        index++;
        //判断索引是否超过最大索引
        /*if(index >= imgArr.length){
            //则将index设置为0
            index = 0;
        }*/
        index %= imgArr.length;
        //修改img1的src属性
        img1.src = imgArr[index];

    },1000);
};

//为btn02绑定一个单击响应函数
var btn02 = document.getElementById("btn02");
btn02.onclick = function(){
    //点击按钮以后，停止图片的自动切换，关闭定时器
    /*
    * clearInterval()可以接收任意参数，
    * 如果参数是一个有效的定时器的标识，则停止对应的定时器
    * 如果参数不是一个有效的标识，则什么也不做
    */
    clearInterval(timer);
};

};

</script>
</head>
<body>
    
    <br /><br />
    <button id="btn01">开始</button>

```

```
<button id="btn02">停止</button>
</body>
</html>
```

## 129. 修改div移动练习

修正第一下的卡顿问题

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">
      #box1{
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
      }
    </style>

    <script type="text/javascript">

      //使div可以根据不同的方向键向不同的方向移动
      /*
      * 按左键，div向左移
      * 按右键，div向右移
      * 。 。 。
      */
      window.onload = function(){

        //定义一个变量，来表示移动的速度
        var speed = 10;

        //创建一个变量表示方向
        //通过修改dir来影响移动的方向
        var dir = 0;

        //开启一个定时器，来控制div的移动
        setInterval(function(){
          /*
          * 37 左
          * 38 上
          * 39 右
          */
```



```

        * 40 下
        */
        switch(dir){
            case 37:
                //alert("向左"); left值减小
                box1.style.left = box1.offsetLeft - speed +
"px";

                break;
            case 39:
                //alert("向右");
                box1.style.left = box1.offsetLeft + speed +
"px";

                break;
            case 38:
                //alert("向上");
                box1.style.top = box1.offsetTop - speed + "p
x";

                break;
            case 40:
                //alert("向下");
                box1.style.top = box1.offsetTop + speed + "p
x";

                break;
        }
    },30);

    //为document绑定一个按键按下的事件
    document.onkeydown = function(event){
        event = event || window.event;

        //当用户按了ctrl以后，速度加快
        if(event.ctrlKey){
            speed = 500;
        }else{
            speed = 10;
        }
        //使dir等于按键的值
        dir = event.keyCode;
    };

    //当按键松开时，div不再移动
    document.onkeyup = function(){
        //设置方向为0
        dir = 0;
    };
};

</script>
</head>
<body>
    <div id="box1"></div>
</body>
</html>

```

## 130. 延时调用

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">

      var num = 1;

      //开启一个定时器
      /*setInterval(function(){
        console.log(num++);
      },3000);*/

      /*
      * 延时调用，
      * 延时调用一个函数不马上执行，而是隔一段时间以后在执行，而且只会执行一
      次

      *
      * 延时调用和定时调用的区别，定时调用会执行多次，而延时调用只会执行一次
      *
      * 延时调用和定时调用实际上是可以互相代替的，在开发中可以根据自己需要去选
      择

      */
      var timer = setTimeout(function(){
        console.log(num++);
      },3000);

      //使用clearTimeout()来关闭一个延时调用
      clearTimeout(timer);

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

# 131. 定时器的应用（一）

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">

      *{
        margin: 0;
        padding: 0;
      }

      #box1{
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
        left: 0;
      }

    </style>

    <script type="text/javascript">

      window.onload = function(){
        //获取box1
        var box1 = document.getElementById("box1");
        //获取btn01
        var btn01 = document.getElementById("btn01");

        //定义一个变量，用来保存定时器的标识
        var timer;

        //点击按钮以后，使box1向右移动（left值增大）
        btn01.onclick = function(){

          //关闭上一个定时器
          clearInterval(timer);

          //开启一个定时器，用来执行动画效果
          timer = setInterval(function(){

            //获取box1的原来的left值
            var oldValue = parseInt(getStyle(box1,"left"));

            //在旧值的基础上增加
            var newValue = oldValue + 1;
```

```

        //判断newValue是否大于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，使其停止执行动画
        if(newValue == 800){
            //达到目标，关闭定时器
            clearInterval(timer);
        }
    },30);
};

};

/*
 * 定义一个函数，用来获取指定元素的当前的样式
 * 参数:
 *     obj 要获取样式的元素
 *     name 要获取的样式名
 */
function getStyle(obj , name){
    if(window.getComputedStyle){
        //正常浏览器的方式，具有getComputedStyle()方法
        return getComputedStyle(obj , null)[name];
    }else{
        //IE8的方式，没有getComputedStyle()方法
        return obj.currentStyle[name];
    }
}
</script>
</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>
</html>

```

## 注意

IE 中要指定 `left: 0;` , 否则使用 `currentStyle` 会返回默认值 `auto` 。

## 132. 定时器的应用（二）

点击按钮以后box1向右移动

点击按钮以后box1向左移动



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">

      *{
        margin: 0;
        padding: 0;
      }

      #box1{
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
        left: 0;
      }

    </style>

    <script type="text/javascript">
      window.onload = function(){
        //获取box1
        var box1 = document.getElementById("box1");
        //获取btn01
        var btn01 = document.getElementById("btn01");

        //获取btn02
        var btn02 = document.getElementById("btn02");

        //点击按钮以后，使box1向右移动（left值增大）
        btn01.onclick = function(){
          move(box1 , 800 , 10);
        };
      };
    </script>
  </head>
  <body>
    <div>
      <div id="box1">
      </div>
      <div id="btn01">向右移动</div>
      <div id="btn02">向左移动</div>
    </div>
  </body>
</html>
```

```

        //点击按钮以后，使box1向左移动（left值减小）
        btn02.onclick = function(){
            move(box1 , 0 , 10);
        };
};

//定义一个变量，用来保存定时器的标识
var timer;

//尝试创建一个可以执行简单动画的函数
/*
 * 参数:
 *   obj:要执行动画的对象
 *   target:执行动画的目标位置
 *   speed:移动的速度(正数向右移动，负数向左移动)
 */
function move(obj , target ,speed){
    //关闭上一个定时器
    clearInterval(timer);

    //获取元素目前的位置
    var current = parseInt(getStyle(obj,"left"));

    //判断速度的正负值
    //如果从0 向 800移动，则speed为正
    //如果从800向0移动，则speed为负
    if(current > target){
        //此时速度应为负值
        speed = -speed;
    }

    //开启一个定时器，用来执行动画效果
    timer = setInterval(function(){

        //获取box1的原来的left值
        var oldValue = parseInt(getStyle(obj,"left"));

        //在旧值的基础上增加
        var newValue = oldValue + speed;

        //判断newValue是否大于800
        //从800 向 0移动
        //向左移动时，需要判断newValue是否小于target
        //向右移动时，需要判断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";
    }, 10);
}

```

```

        //当元素移动到0px时，使其停止执行动画
        if(newValue == target){
            //达到目标，关闭定时器
            clearInterval(timer);
        }
    },30);
}

/*
 * 定义一个函数，用来获取指定元素的当前的样式
 * 参数：
 *     obj 要获取样式的元素
 *     name 要获取的样式名
 */
function getStyle(obj , name){
    if(window.getComputedStyle){
        //正常浏览器的方式，具有getComputedStyle()方法
        return getComputedStyle(obj , null)[name];
    }else{
        //IE8的方式，没有getComputedStyle()方法
        return obj.currentStyle[name];
    }
}
</script>
</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>
</html>

```

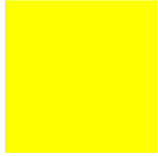
## 133. 定时器的应用（三）

点击按钮以后box1向右移动

点击按钮以后box1向左移动

点击按钮以后box2向右移动

测试按钮



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">

      *{
        margin: 0;
        padding: 0;
      }

      #box1{
        width: 100px;
        height: 100px;
        background-color: red;
        position: absolute;
        left: 0;
      }

      #box2{
        width: 100px;
        height: 100px;
        background-color: yellow;
        position: absolute;
        left: 0;
        top: 200px;
      }

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

      window.onload = function(){

        //获取box1
        var box1 = document.getElementById("box1");
        //获取btn01
```



```

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

//获取btn02
var btn02 = document.getElementById("btn02");

//点击按钮以后, 使box1向右移动 (left值增大)
btn01.onclick = function(){
    move(box1 , "left", 800 , 20);
};

//点击按钮以后, 使box1向左移动 (left值减小)
btn02.onclick = function(){
    move(box1 , "left", 0 , 10);
};

//获取btn03
var btn03 = document.getElementById("btn03");
btn03.onclick = function(){
    move(box2 , "left", 800 , 10);
};

//测试按钮
var btn04 = document.getElementById("btn04");
btn04.onclick = function(){
    //move(box2 , "width", 800 , 10);
    //move(box2 , "top", 800 , 10);
    //move(box2 , "height", 800 , 10);
    move(box2 , "width" , 800 , 10 , function(){
        move(box2 , "height" , 400 , 10 , function(){
            move(box2 , "top" , 0 , 10 , function(){
                move(box2 , "width" , 100 , 10 ,
function(){

                });
            });
        });
    });
};

};

//定义一个变量, 用来保存定时器的标识
/*
 * 目前我们的定时器的标识由全局变量timer保存,
 * 所有的执行正在执行的定时器都在这个变量中保存
 */
//var timer;

</script>
</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>
</html>

```

js/tools.js

```

//尝试创建一个可以执行简单动画的函数
/*
 * 参数:
 *  obj:要执行动画的对象
 *  attr:要执行动画的样式, 比如: left top width height
 *  target:执行动画的目标位置
 *  speed:移动的速度(正数向右移动, 负数向左移动)
 *  callback:回调函数, 这个函数将会在动画执行完毕以后执行
 */
function move(obj, attr, target, speed, callback) {
    //关闭上一个定时器
    clearInterval(obj.timer);

    //获取元素目前的位置
    var current = parseInt(getStyle(obj, attr));

    //判断速度的正负值
    //如果从0 向 800移动, 则speed为正
    //如果从800向0移动, 则speed为负
    if(current > target) {
        //此时速度应为负值
        speed = -speed;
    }

    //开启一个定时器, 用来执行动画效果
    //向执行动画的对象中添加一个timer属性, 用来保存它自己的定时器的标识
    obj.timer = setInterval(function() {

        //获取box1的原来的left值
        var oldValue = parseInt(getStyle(obj, attr));

```

```

        //在旧值的基础上增加
        var newValue = oldValue + speed;

        //判断newValue是否大于800
        //从800 向 0移动
        //向左移动时，需要判断newValue是否小于target
        //向右移动时，需要判断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0px时，使其停止执行动画
        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();
        }

    }, 30);
}

/*
 * 定义一个函数，用来获取指定元素的当前的样式
 * 参数:
 *      obj 要获取样式的元素
 *      name 要获取的样式名
 */
function getStyle(obj, name) {

    if(window.getComputedStyle) {
        //正常浏览器的方式，具有getComputedStyle()方法
        return getComputedStyle(obj, null)[name];
    } else {
        //IE8的方式，没有getComputedStyle()方法
        return obj.currentStyle[name];
    }
}
}

```

## 134. 完成轮播图界面

- 浮动以后内联元素变成块元素

- 文件名是中文无法在 `ie` 显示时，可以拖动文件到浏览器

## 135. 完成点击按钮切换图片

弹幕补充：`let` 解决了 `for` 循环绑定事件的问题，但是 `ie8` 不支持。

注意：通过 `js` 设置的是内联样式，会覆盖掉 `a:hover`，但是把它设置为空字符串，则会采用默认的风格。

## 136. 完成轮播图



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

    <style type="text/css">
      *{
        margin: 0;
        padding: 0;
      }
    </style>
  </head>
  <body>
    <div>
      <img alt="Carousel of three cartoon ice cream bars" data-bbox="136 352 856 699"/>
    </div>
  </body>
</html>
```

```

/*
 * 设置outer的样式
 */
#outer{
    /*设置宽和高*/
    width: 520px;
    height: 333px;
    /*居中*/
    margin: 50px auto;
    /*设置背景颜色*/
    background-color: greenyellow;
    /*设置padding*/
    padding: 10px 0;
    /*开启相对定位*/
    position: relative;
    /*裁剪溢出的内容*/
    overflow: hidden;
}

/*设置imgList*/
#imgList{
    /*去除项目符号*/
    list-style: none;
    /*设置ul的宽度*/
    /*width: 2600px;*/
    /*开启绝对定位*/
    position: absolute;
    /*设置偏移量*/
    /*
     * 每向左移动520px, 就会显示到下一张图片
     */
    left: 0px;
}

/*设置图片中的li*/
#imgList li{
    /*设置浮动*/
    float: left;
    /*设置左右外边距*/
    margin: 0 10px;
}

/*设置导航按钮*/
#navDiv{
    /*开启绝对定位*/
    position: absolute;
    /*设置位置*/
    bottom: 15px;
    /*设置left值
     outer宽度 520
     navDiv宽度 25*5 = 125
     520 - 125 = 395/2 = 197.5
    */

```

```

        * */
        /*left: 197px;*/
    }

    #navDiv a{
        /*设置超链接浮动*/
        float: left;
        /*设置超链接的宽和高*/
        width: 15px;
        height: 15px;
        /*设置背景颜色*/
        background-color: red;
        /*设置左右外边距*/
        margin: 0 5px;
        /*设置透明*/
        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>

<!--引用工具-->
<script type="text/javascript" src="js/tools.js"></script>
<script type="text/javascript">
    window.onload = function(){
        //获取imgList
        var imgList = document.getElementById("imgList");
        //获取页面中所有的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*/
        //获取navDiv
        var navDiv = document.getElementById("navDiv");
        //获取outer
        var outer = document.getElementById("outer");
        //设置navDiv的left值
        navDiv.style.left = (outer.offsetWidth - navDiv.offsetWid
th)/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");
        //设置默认选中的效果

```

```

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

/*
    点击超链接切换到指定的图片
        点击第一个超链接，显示第一个图片
        点击第二个超链接，显示第二个图片
    */

//为所有的超链接都绑定单击响应函数
for(var i=0; i<allA.length ; i++){

    //为每一个超链接都添加一个num属性
    allA[i].num = i;

    //为超链接绑定单击响应函数
    allA[i].onclick = function(){

        //关闭自动切换的定时器
        clearInterval(timer);
        //获取点击超链接的索引,并将其设置为index
        index = this.num;

        //切换图片
        /*
            * 第一张    0  0
            * 第二张    1 -520
            * 第三张    2 -1040
            */
        //imgList.style.left = -520*index + "px";
        //设置选中的a
        setA();

        //使用move函数来切换图片
        move(imgList , "left" , -520*index , 20 , function()

n(){

        //动画执行完毕，开启自动切换
        autoChange();
    });

};
}

//开启自动切换图片
autoChange();

//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;
    }
}

```

```

        //此时显示的最后张图片，而最后一张图片和第一张是一摸一样
        //通过CSS将最后一张切换成第一张
        imgList.style.left = 0;
    }

    //遍历所有a，并将它们的背景颜色设置为红色
    for(var i=0 ; i<allA.length ; i++){
        allA[i].style.backgroundColor = "red";
    }

    //将选中的a设置为黑色
    allA[index].style.backgroundColor = "black";
};

//定义一个自动切换的定时器的标识
var timer;
//创建一个函数，用来开启自动切换图片
function autoChange(){

    //开启一个定时器，用来定时去切换图片
    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
        move(imgList , "left" , -520*index , 20 , functionio

n(){

        //修改导航按钮
        setA();
    });
    },3000);
}

};

</script>
</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></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></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>
</body>
</html>
```

## 137. 类的操作

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">

      .b1{
        width: 100px;
        height: 100px;
        background-color: red;
      }

      .b2{
        height: 300px;
        background-color: yellow;
      }

    </style>

    <script type="text/javascript">

      window.onload = function(){
        //获取box
        var box = document.getElementById("box");
        //获取btn01
        var btn01 = document.getElementById("btn01");

        //为btn01绑定单击响应函数
        btn01.onclick = function(){
          //修改box的样式
```

重新渲染一次页面

式时，也不太方便

```
/*
 * 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
 * 这样的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

 */
/*box.style.width = "200px";
box.style.height = "200px";
box.style.backgroundColor = "yellow";*/

/*
 * 我希望一行代码，可以同时修改多个样式
 */

//修改box的class属性
/*
 * 我们可以通过修改元素的class属性来间接的修改样式
 * 这样一来，我们只需要修改一次，即可同时修改多个样式，
 * 浏览器只需要重新渲染页面一次，性能比较好，
 * 并且这种方式，可以使表现和行为进一步的分离
 */
//box.className += " b2";
//addClass(box, "b2");

//alert(hasClass(box, "hello"));

//removeClass(box, "b2");

toggleClass(box, "b2");
};

};

//定义一个函数，用来向一个元素中添加指定的class属性值
/*
 * 参数：
 * obj 要添加class属性的元素
 * cn 要添加的class值
 *
 */
function addClass(obj , cn){

    //检查obj中是否含有cn
    if(!hasClass(obj , cn)){
        obj.className += " "+cn;
    }

}

/*
 * 判断一个元素中是否含有指定的class属性值
 * 如果有该class，则返回true，没有则返回false
```

```

    *
    */
    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/;
        var reg = new RegExp("\\b"+cn+"\\b");

        return reg.test(obj.className);

    }

    /*
    * 删除一个元素中的指定的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 , "");

    }

    /*
    * toggleClass可以用来切换一个类
    * 如果元素中具有该类，则删除
    * 如果元素中没有该类，则添加
    */
    function toggleClass(obj , cn){

        //判断obj中是否含有cn
        if(hasClass(obj , cn)){
            //有，则删除
            removeClass(obj , cn);
        }else{
            //没有，则添加
            addClass(obj , cn);
        }
    }

</script>
</head>
<body>

    <button id="btn01">点击按钮以后修改box的样式</button>

    <br /><br />

    <div id="box" class="b1 b2"></div>
</body>

```

```
</html>
```

## 138. 二级菜单基本功能

略

## 139. 二级菜单过渡效果



```
<!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>
<script type="text/javascript">
    window.onload = function(){
        /*
        * 我们的每一个菜单都是一个div
        * 当div具有collapsed这个类时，div就是折叠的状态
        * 当div没有这个类是，div就是展开的状态
        */

        /*
        * 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，来保存当前打开的菜单
        var openDiv = menuSpan[0].parentNode;

        //为span绑定单击响应函数
        for(var i=0 ; i<menuSpan.length ; i++){
            menuSpan[i].onclick = function(){

                //this代表我当前点击的span
                //获取当前span的父元素
                var parentDiv = this.parentNode;

                //切换菜单的显示状态
                toggleMenu(parentDiv);

                //判断openDiv和parentDiv是否相同
                if(openDiv != parentDiv && !hasClass(openDiv ,
"collapsed")){

                    //打开菜单以后，应该关闭之前打开的菜单
                    //为了可以统一处理动画过渡效果，我们希望在这将addClass
s改为toggleClass

                    //addClass(openDiv , "collapsed");
                    //此处toggleClass()不需要有移除的功能
                    //toggleClass(openDiv , "collapsed");
                    //切换菜单的显示状态
                    toggleMenu(openDiv);
                }
                //修改openDiv为当前打开的菜单
                openDiv = parentDiv;
            };
        }

        /*

```

```

        * 用来切换菜单折叠和显示状态
        */
function toggleMenu(obj){
    //在切换类之前，获取元素的高度
    var begin = obj.offsetHeight;

    //切换parentDiv的显示
    toggleClass(obj , "collapsed");

    //在切换类之后获取一个高度
    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
    move(obj,"height",end,10,function(){
        //动画执行完毕，内联样式已经没有存在的意义了，删除之
        obj.style.height = "";
    });
}

};
</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>
</html>

```

## 140. JSON

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>

    <!--
      如果需要兼容IE7及以下的JSON操作，则可以通过引入一个外部的js文件来处理
    -->
    <script type="text/javascript" src="js/json2.js"></script>
    <script type="text/javascript">

      /*
        * JSON
        *   - JS中的对象只有JS自己认识，其他的语言都不认识
        *   - 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
        *       - JavaScript Object Notation JS对象表示法
        *       - JSON和JS对象的格式一样，只不过JSON字符串中的属性名必须加双引
          号

        *       其他的和JS语法一致
        *       JSON分类：
        *         1.对象 {}
        *         2.数组 []
        *
        *       JSON中允许的值：
        *         1.字符串
        *         2.数值
        *         3.布尔值
        *         4.null
        *         5.对象
        *         6.数组
        */

      //创建一个对象

```

```

var arr = '[1,2,3,"hello",true]';
var obj2 = '{"arr":[1,2,3]}';
var arr2 = '[{"name":"孙悟空","age":18,"gender":"男"},{"name":"孙悟空","age":18,"gender":"男"}]';

/*
 * 将JSON字符串转换为JS中的对象
 * 在JS中，为我们提供了一个工具类，就叫JSON
 * 这个对象可以帮助我们将一个JSON转换为JS对象，也可以将一个JS对象转换为JSON

var json = '{"name":"孙悟空","age":18,"gender":"男"}';

/*
 * json --> js对象
 * JSON.parse()
 * - 可以将JSON字符串转换为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
 */

var o = JSON.parse(json);
var o2 = JSON.parse(arr);

//console.log(o.gender);
//console.log(o2[1]);

var obj3 = {name:"猪八戒" , age:28 , gender:"男"};

/*
 * JS对象 ---> JSON
 * JSON.stringify()
 * - 可以将一个JS对象转换为JSON字符串
 * - 需要一个js对象作为参数，会返回一个JSON字符串
 */

var str = JSON.stringify(obj3);
//console.log(str);

/*
 * 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

 */
var str3 = '{"name":"孙悟空","age":18,"gender":"男"}';

JSON.parse(str3);
</script>
</head>
<body>
</body>
</html>

```



兼容 **ie7**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript">

      var str = '{"name":"孙悟空","age":18,"gender":"男"}';

      /*
      * eval()
      * - 这个函数可以用来执行一段字符串形式的JS代码，并将执行结果返回
      * - 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()
      *
      * - 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
      *
      */

      var str2 = "alert('hello');";

      var obj = eval("(" + str + ")");

      //console.log(obj);
    </script>
  </head>
  <body>
  </body>
</html>
```

全隐患

完结于 **2019.04.12**