

变量声明

- 变量声明有三个 `var` `let` 和 `const`
- 我们应该用那个呢?
- 首先var 先排除, 老派写法, 问题很多, 可以淘汰掉...
- `let` or `const`?
- 建议: `const` 优先, 尽量使用 `const`, 原因是:
- `const` 语义化更好
- 很多变量我们声明的时候就知道他不会被更改了, 那为什么不用 `const`呢?
- 实际开发中也是, 比如react框架, 基本`const`
- 如果你还在纠结, 那么我建议:
- 有了变量先给`const`, 如果发现它后面是要被修改的, 再改为`let`

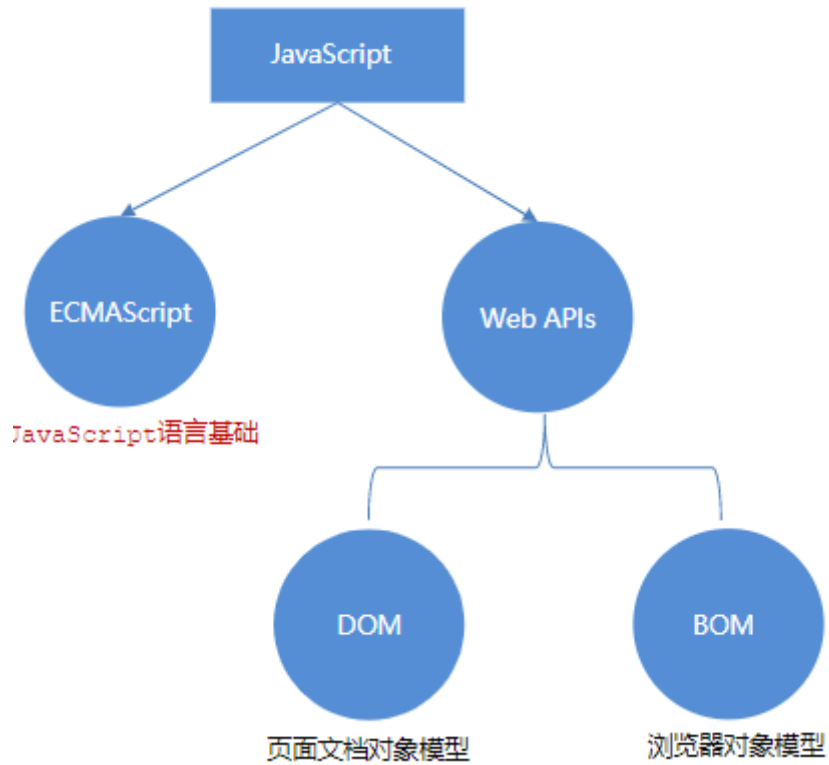
总结

1. 以后声明变量我们优先使用哪个?
 - Ø `const`
 - Ø 有了变量先给`const`, 如果发现它后面是要被修改的, 再改为`let`
2. 为什么`const`声明的对象可以修改里面的属性?
 - Ø 因为对象是引用类型, 里面存储的是地址, 只要地址不变, 就不会报错
 - Ø **建议 数组 和 对象 使用 `const` 来声明**
3. 什么时候使用`let`声明变量?
 - Ø 如果基本数据类型的值或者引用类型的地址发生变化的时候, 需要用`let`
 - Ø 比如 一个变量进行加减运算, 比如 `for`循环中的 `i++`

Web API 基本认知

作用和分类

- 作用: 就是使用 JS 去操作 html 和浏览器
- 分类: DOM (文档对象模型)、BOM (浏览器对象模型)



什么是DOM

1. DOM (Document Object Model —— 文档对象模型) 是用来呈现以及与任意 HTML 或 XML 文档交互的API
2. DOM是浏览器提供的一套专门用来 操作网页内容 的功能
3. DOM作用：开发网页内容特效和实现用户交互

总结

1. Web API阶段我们学习那两部分？
 - DOM
 - BOM
2. DOM 是什么？有什么作用？
 - DOM 是文档对象模型
 - 操作网页内容，可以开发网页内容特效和实现用户交互

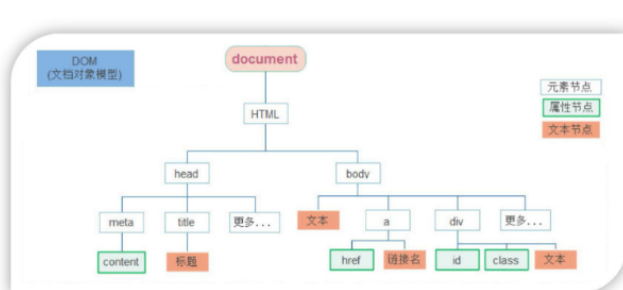
DOM树是什么？

1. 将 HTML 文档以树状结构直观的表现出来，我们称之为文档树或 DOM 树
2. 描述网页内容关系的名词
3. 作用：文档树直观的体现了标签与标签之间的关系

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>标题</title>
</head>
<body>
  文本
  <a href="">链接名</a>
  <div id="" class="">文本</div>
</body>

```



DOM文档对象模型

文档对象模型 (Document Object Model, 简称 **DOM**)，是 W3C 组织推荐的处理可扩展标记语言 (HTML或者XML) 的标准 **编程接口**

W3C 已经定义了一系列的 DOM 接口，通过这些 DOM 接口可以改变网页的内容、结构和样式。

1.2 DOM 树



- 文档：一个页面就是一个文档，DOM中使用document来表示
- 元素：页面中的所有标签都是元素，DOM中使用 `element` 表示
- 节点：网页中的所有内容都是节点（标签，属性，文本，注释等），DOM中使用node表示

获取DOM元素(标签)

获取页面元素

获取页面中的元素可以使用以下几种方式:

- 根据 ID 获取
- 根据标签名获取
- 通过 HTML5 新增的方法获取
- 特殊元素获取

根据ID获取

使用 `getElementById()` 方法可以获取带ID的元素对象

```
1 | document.getElementById('id名')
```

使用 `console.dir()` 可以打印我们获取的元素对象，更好的查看对象里面的属性和方法。

例子：

```
1 | <div id="time">2022-3-20</div>
2 | <script>
3 |     // 1.因为我们文档页面从上往下加载，所以得先有标签，所以script写在标签下面
4 |     // 2.get 获得 element 元素 by 通过 驼峰命名法
5 |     // 3.参数 id是大小写敏感的字符串
6 |     // 4.返回的是一个元素对象
7 |     var timer = document.getElementById('time');
8 |     console.log(timer);
9 |     // 5. console.dir 打印我们的元素对象，更好的查看里面的属性和方法
10 |    console.dir(timer);
11 | </script>
```

根据标签名获取

根据**标签名**获取，使用 `getElementsByTagName()` 方法可以返回带有指定标签名的**对象的集合**

```
1 | document.getElementsByTagName('标签名');
```

- 因为得到的是一个对象的集合，所以我们想要操作里面的元素就需要遍历
- 得到元素对象是动态的
- 返回的是获取过来元素对象的集合，以伪数组的形式存储
- 如果获取不到元素，则返回为空的伪数组(因为获取不到对象)

```
1 | <ul>
2 |     <li>知否知否，应是等你好久</li>
3 |     <li>知否知否，应是等你好久</li>
4 |     <li>知否知否，应是等你好久</li>
5 |     <li>知否知否，应是等你好久</li>
6 |     <li>知否知否，应是等你好久</li>
7 | </ul>
8 | <script>
9 |     // 1.返回的是获取过来元素对象的集合 以伪数组的形式存储
10 |    var lis = document.getElementsByTagName('li');
11 |    console.log(lis);
12 |    console.log(lis[0]);
13 |    // 2.依次打印,遍历
14 |    for (var i = 0; i < lis.length; i++) {
15 |        console.log(lis[i]);
16 |    }
17 | </script>
```

getElementsByName()根据标签中的属性名获取

根据标签中的**属性名**获取元素，返回一个**伪数组**的集合，同样也需要**遍历**

```
1 document.getElementsByName('属性名');
```

例子

```
1 <div name="demo"></div>
2 <div name="demo"></div>
3 <script>
4     var demoList = document.getElementsByName('demo');
5     console.log(demoList);
6 </script>
```

HTML5新增的方法获取

getElementsByClassName，根据类名返回元素对象合集

根据类名返回元素对象合集

```
1 document.getElementsByClassName('类名')
```

例子:

```
1 <div class="box">盒子1</div>
2 <div class="box">盒子2</div>
3 <div id="nav">
4     <ul>
5         <li>首页</li>
6         <li>产品</li>
7     </ul>
8 </div>
9
10 <script>
11     // getElementsByClassName ，根据类名返回元素对象合集
12     var boxes = document.getElementsByClassName('box');
13     console.log(boxes);
14 </script>
```

document.querySelector，根据指定选择器返回第一个元素对象

根据指定选择器返回 第一个元素 对象

```
1 document.querySelector('选择器');
```

例子:

```
1 // 切记里面的选择器需要加符号
2 // 类选择器 .box
3 // id选择器 #nav
4
5 <div class="box">盒子1</div>
6 <div class="box">盒子2</div>
7 <div id="nav">
8     <ul>
9         <li>首页</li>
10        <li>产品</li>
11    </ul>
12 </div>
13
14 <script>
15     // document.querySelector , 根据指定选择器返回第一个元素对象
16     var a = document.querySelector('.box');
17     var b = document.querySelector('#nav');
18     var c = document.querySelector('li');
19     console.log(a);
20     console.log(b);
21     console.log(c);
22 </script>
```

document.querySelectorAll , 根据指定选择器返回所有元素对象

根据指定选择器返回所有元素对象

```
1 document.querySelectorAll('选择器');
```

例子:

```
1 <div class="box">盒子1</div>
2 <div class="box">盒子2</div>
3 <div id="nav">
4     <ul>
5         <li>首页</li>
6         <li>产品</li>
7     </ul>
8 </div>
9
10 <script>
11     // document.querySelectorAll , 根据指定选择器返回所有元素对象
12     var a = document.querySelectorAll('.box');
13     console.log(a);
14     var b = document.querySelectorAll('#nav');
15     console.log(b);
16     var c = document.querySelectorAll('li');
17     console.log(c);
18 </script>
```

注意: `querySelector`和`querySelectorAll`里面的选择器需要加符号,比如:`document.querySelector('#nav');`

获取特殊元素

获取body元素, 返回body元素对象

```
1 | document.body;
```

```
1 |     <script>
2 |         // 获取body 元素
3 |         var bodyEle = document.body;
4 |         console.log(bodyEle);
5 |     </script>
```

获取html元素, 返回html元素对象

```
1 | document.documentElement;
```

```
1 |     <script>
2 |         // 获取HTML元素
3 |         var HtmlEle = document.documentElement;
4 |         console.log(HtmlEle);
5 |     </script>
```

事件基础

概念: JavaScript 使我们有能力创建动态页面, 而事件是可以被 JavaScript 侦测到的行为。

网页中的每个元素都可以产生某些可以触发 JavaScript 的事件, 例如, 我们可以在用户点击某按钮时产生一个事件, 然后去执行某些操作。

事件三要素

1. 事件源(谁)
2. 事件类型(什么事件)
3. 事件处理程序(做啥)

```
1      <button id="btn">点击按钮</button>
2      <script>
3          // 点击一个按钮，弹出对话框
4          // 1. 事件是有三部分组成 事件源 事件类型 事件处理程序 我们也称为事件三要素
5          //(1) 事件源 事件被触发的对象 谁 按钮
6          var btn = document.getElementById('btn');
7          //(2) 事件类型 如何触发 什么事件 比如鼠标点击(onclick) 还是鼠标经过 还是键盘
      按下
8          //(3) 事件处理程序 通过一个函数赋值的方式 完成
9          btn.onclick = function() {
10              alert('出现弹窗');
11          }
12      </script>
```

执行事件的步骤

- 1. 获取事件源
- 2. 注册事件(绑定事件)
- 3. 添加事件处理程序(采取函数赋值形式)

```
1      <div>点我</div>
2      <script>
3          // 执行事件步骤
4          // 点击div 控制台输出 我被选中了
5          // 1. 获取事件源
6          var div = document.querySelector('div');
7          // 2.绑定事件 注册事件
8          // div.onclick
9          // 3.添加事件处理程序
10         div.onclick = function() {
11             console.log('我被选中了');
12         }
13     </script>
```

鼠标事件

鼠标事件	触发条件
onclick	鼠标点击左键触发
onmouseover	鼠标经过触发
onmouseout	鼠标离开触发
onfocus	获得表单焦点触发
onblur	失去表单焦点触发
onmousemove	鼠标移动触发
onmouseup	鼠标弹起触发

鼠标事件	触发条件
onmousedown	鼠标按下触发

表单事件 **重要**

onsubmit	onsubmit 事件在表单提交时触发
----------	---------------------

例子

```

1      <form action="a.html" method="get" id="sub">
2          账号<input type="text" value="测试文字">
3          <input type="submit" value="提交信息" id="sub">
4      </form>
5      <script>
6          /* 通过HTML属性的方式添加 */
7          function a(){
8              alert('测试');
9          }
10
11          /* 通过js事件的方式添加，注意：此时不需要 form 标签中的 onsubmit属性 */
12          var form1 = document.getElementById('sub');
13
14          form1.onsubmit = function a(){
15              alert('测试');
16          }
17      </script>

```

操作元素

JavaScript 的 DOM 操作可以改变网页内容、结构和样式，我们可以利用 DOM 操作元素来改变元素里面的内容、属性等。

注意：以下都是属性

改变元素（标签）内容

innerText

从起始位置到终止位置的内容，但它去除html标签，同时空格和换行也会去掉。

```
1 element.innerText
```

element.innerHTML

起始位置到终止位置的全部内容，包括HTML标签，同时保留空格和换行

```
1 element.innerHTML
```

例子:

```
1
2     <div></div>
3     <p>
4         我是文字
5         <span>123</span>
6     </p>
7
8     <script>
9         // innerText 和 innerHTML的区别
10        // 1. innerText 不识别html标签,去除空格和换行
11        var div = document.querySelector('div');
12        div.innerText = '<strong>今天是: </strong> 2019';
13        // 2. innerHTML 识别html标签 保留空格和换行的
14        div.innerHTML = '<strong>今天是: </strong> 2019';
15        // 这两个属性是可读写的  可以获取元素里面的内容
16        var p = document.querySelector('p');
17        console.log(p.innerText);
18        console.log(p.innerHTML);
19    </script>
20
```

案例: 当鼠标点击按钮后, 网页中出现现在的时间

```
1     <button>显示当前系统时间</button>
2     <div>某个时间</div>
3     <p>123</p>
4     <script>
5         // 当点击显示当前系统时间按钮, div里面的文字需要发生变化
6         // 1. 获取元素
7         var btn = document.querySelector('button');
8         var div = document.querySelector('div');
9         // 2. 绑定时间
10        btn.onclick = function(){
11            // div.innerText = '2022-3-20';
12            div.innerText = getTimes();
13        }
14
15        function getTimes() {
16            var now = new Date();
17            var year = now.getFullYear();
18            var month = now.getMonth() + 1;
19            var day = now.getDate();
20            var hours = now.getHours();
21            var minutes = now.getMinutes();
22            var seconds = now.getSeconds();
23            // console.log('今天
是'+year+'年'+month+'月'+day+'日'+hours+'点'+minutes+'分'+seconds+'秒');
24            return '今天
是'+year+'年'+month+'月'+day+'日'+hours+'点'+minutes+'分'+seconds+'秒';
25        }
26        // 元素也可以不添加事件
```

```

27     var p = document.querySelector('p');
28     p.innerText = getTimes();
29 </script>

```

案例：年会抽奖

需求：从数组随机抽取一等奖、二等奖和三等奖，显示到对应的标签里面。

素材：

html文件结构

数组名单 '周杰伦', '刘德华', '周星驰', 'Pink老师', '张学友'



分析：

- ①：声明数组: `const personArr = ['周杰伦', '刘德华', '周星驰', 'Pink老师', '张学友']`
- ②：一等奖:随机生成一个数字（0~数组长度），找到对应数组的名字
- ③：通过 `innerText` 或者 `innerHTML` 将名字写入 `span` 元素内部
- ④：二等奖依次类推

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8" />
6    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
7    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
8    <title>年会抽奖</title>
9    <style>
10     .wrapper {
11       width: 840px;
12       height: 420px;
13       background: url(./bg01.jpg) no-repeat center / cover;
14       padding: 100px 250px;
15       box-sizing: border-box;
16     }
17   </style>
18 </head>
19
20 <body>
21   <div class="wrapper">
22     <strong>抽奖</strong>
23     <h1>一等奖: <span id="one">???</span></h1>
24     <h3>二等奖: <span id="two">???</span></h3>
25     <h5>三等奖: <span id="three">???</span></h5>

```

```

26     </div>
27
28     <script>
29         // 声明数组
30         const personArr = [ '周杰伦', '刘德华', '周星驰', 'Pink老师', '张学友'];
31         // 一等奖，使用随机数作为数组的下标
32         const random = Math.floor(Math.random() * personArr.length);
33         // 绑定one对应的标签
34         const one = document.querySelector('#one');
35         // 更改元素内容
36         one.innerHTML = personArr[random];
37         // 删除一等奖对应的用户，避免二等奖重复
38         personArr.splice(random,1);
39
40         // 二等奖
41         const random1 = Math.floor(Math.random() * personArr.length);
42         // 绑定two对应的标签
43         const two = document.querySelector('#two');
44         // 更改元素内容
45         two.innerHTML = personArr[random1];
46         // 删除一等奖对应的用户，避免二等奖重复
47         personArr.splice(random1,1);
48
49         // 三等奖
50         const random2 = Math.floor(Math.random() * personArr.length);
51         // 绑定three对应的标签
52         const three = document.querySelector('#three');
53         // 更改元素内容
54         three.innerHTML = personArr[random2];
55         // 删除一等奖对应的用户，避免二等奖重复
56         personArr.splice(random2,1);
57     </script>
58 </body>
59
60 </html>

```

改变常用元素（标签）操作属性

- 1、src、href
- 2、alt、title

例子：img.属性 img.src = "xxx", 利用img属性的src元素，进行图片切换

```

1     <button id="a">点击为a图片</button>
2     <button id="b">点击为b图片</button>
3     
4
5     <script>
6         // 修改元素属性src的路径，已替换照片
7         // 1、获取元素
8         var a = document.getElementById('a');
9         var b = document.getElementById('b');
10        var img = document.querySelector('img')

```

```

11      // 2、绑定事件
12      b.onclick = function(){
13          img.src = './images/B.png';
14      }
15      a.onclick = function(){
16          img.src = 'images/A.png';
17      }
18  </script>

```

扩展：当鼠标悬浮时，修改图片标签中的 `title` 属性

```

1      <button id="a">点击为a图片</button>
2      <button id="b">点击为b图片</button>
3      
4
5      <script>
6          // 修改元素属性src的路径，已替换照片
7          // 1、获取元素
8          var a = document.getElementById('a');
9          var b = document.getElementById('b');
10         var img = document.querySelector('img');
11         // 2、绑定事件
12         b.onclick = function(){
13             img.src = './images/B.png';
14             img.title = 'B';
15         }
16         a.onclick = function(){
17             img.src = 'images/A.png';
18             img.title = 'A';
19         }
20     </script>

```

案例：分时显示不同的图片，显示不同的问候语

根据不同时间，页面显示不同图片，同时显示不同的问候语

如果是上午时间打开页面时，显示上午好，显示A图片

如果是下午时间打开页面时，显示下午好，显示B图片

如果是晚上时间打开页面时，显示晚上好，显示C图片

思路：

- 1、根据系统不同时间来判断，所以需要用到日期内置对象。
- 2、利用分支语句来设置不同的图片。
- 3、需要一个图片，并且根据时间修改图片，需要用到操作元素src属性。
- 4、需要一个div元素，显示不同的问候语，修改元素内容即可。

```

1  
2  <div>上午</div>

```

```

3      <script>
4          // 1、根据系统不同时间来判断，所以需要用到日期内置对象
5          // 2、利用分支语句来设置不同的图片
6          // 3、需要一个图片，并且根据时间修改图片，需要用到操作元素src属性
7          // 4、需要一个div元素，显示不同的问候语，修改元素内容即可。
8          // 1、获取元素
9          var img = document.querySelector('img');
10         var div = document.querySelector('div');
11         // 2、得到当前的小时数
12         var date = new Date();
13         var h = date.getHours();
14         // 3、判断小时数改变图片和文字信息。
15         if( h < 12 ){
16             img.src = './images/A.png';
17             div.innerHTML = '上午';
18         }else if( h < 18){
19             img.src = './images/B.png';
20             div.innerHTML = '下午';
21         }else{
22             img.src = './images/C.png';
23             div.innerHTML = '晚上';
24         }
25     </script>

```

案例：页面刷新，图片随机更换

需求：当我们刷新页面，页面中的图片随机显示不同的图片

分析：

- ①：随机显示，则需要用到随机函数
- ②：更换图片需要用到图片的 src 属性，进行修改
- ③：核心思路：

```

1      1. 获取图片元素
2      2. 随机得到图片序号
3      3. 图片.src = 图片随机路径

```

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10     
11     <script>
12         // 非函数写法
13         /*          // 获取图片的随机数
14             var random = Math.floor(Math.random() * (6 - 1 + 1) + 1);
15
16             // 绑定元素

```

```

17     const img = document.querySelector('img');
18
19     // 给图片跟换 img 的 src 属性
20     img.src = `./images/${random}.webp`; */
21
22     // 函数写法
23     // 获取图片的随机数
24     function getRandom(N,M){
25         return Math.floor(Math.random() * (M - N + 1) + N);
26     }
27
28     // 绑定元素
29     const img = document.querySelector('img');
30
31     const random = getRandom(1,6);
32
33     // 给图片跟换 img 的 src 属性
34     img.src = `./images/${random}.webp`;
35 </script>
36 </body>
37 </html>

```

修改表单的属性操作

利用DOM可以操作如下表单元素的属性

- 表单很多情况，也需要修改属性，比如点击眼睛，可以看到密码，本质是把表单类型转换为文本框
- 正常的有属性有取值的 跟其他的标签属性没有任何区别

语法：

- 获取：DOM对象.属性名
- 设置：DOM对象.属性名 = 新值

```

表单.value = '用户名'
表单.type = 'password'

```

```
1 type, value, checked, selected, disabled
```

例子: value , disabled

```

1     <button>按钮</button>
2     <input type="text" value="输入内容">
3     <script>
4         // 1、获取元素

```

```

5      var btn = document.querySelector('button');
6      var input = document.querySelector('input');
7      // 2、注册事件，处理程序
8      btn.onclick = function(){
9          // input.innerHTML = '已经惦记了'; innerHTML仅用于普通盒子，如果div
便签里面的内容
10         // input.innerHTML = '已经点击了';
11         input.value = '被点击';
12         // 如果某个表单被禁用，不能再点击 disabled，需要按钮禁用，可以设置为
13         // btn.disabled = true;
14         this.disabled = true; //this 指向的是事件函数的调用者 btn
15     }
16     </script>

```

改变样式属性操作

我们可以通过JS 修改元素的大小、颜色、位置等样式。

行内样式操作

语法：

```

1 // element.style
2 div.style.backgroundColor = 'pink';
3 div.style.width = '250px';

```

行内样式操作——案例1：

CSS

```

1 <style>
2     div{
3         width: 200px;
4         height: 200px;
5         background-color: pink;
6     }
7 </style>

```

html

```

1 <div></div>
2 <script>
3     // 1、获取元素
4     var div1 = document.querySelector('div');
5     div1.onclick = function(){
6         // div.style 里面的属性，采取驼峰命名法
7         this.style.backgroundColor = 'red';
8         this.style.width = '250px';
9     }
10 </script>

```


行内样式操作——案例2:

html

```
1 <div class="box">
2   
3   <i class="b">x</i>
4 </div>
5 <script>
6   // 1、获取元素
7   var btn = document.querySelector('.b');
8   var box = document.querySelector('.box');
9   // 2、注册事件，程序处理
10  btn.onclick = function(){
11    box.style.display = 'none';
12  }
13 </script>
```

行内样式操作——案例3: 显示隐藏文本框内容

当鼠标点击文本框时，默认文字隐藏，鼠标离开文本框时，里面文字显示

CSS

```
1 <style>
2   input{
3     color: #666;
4   }
5 </style>
```

html

```
1 <input type="text" value="手机">
2 <script>
3   // 1、获取元素
4   var text1 = document.querySelector('input');
5   // 2、绑定事件
6   text1.onfocus = function(){
7     // console.log('获得焦点');
8     if(text1.value === '手机'){
9       text1.value = '';
10    }
11    // 获取焦点，需要把文本框的文字颜色变深
12    text1.style.color = 'black';
13  }
14  text1.onblur = function(){
15    // console.log('失去了焦点');
16    if(text1.value === ''){
17      text1.value = '手机';
18    }
19    // 失去焦点，需要把文本框的文字颜色变浅
```

```
20     text1.style.color = 'red';
21   }
22 </script>
```

案例：页面刷新，页面随机更换背景图片

需求：当我们刷新页面，页面中的背景图片随机显示不同的图片

分析：

- ①：随机函数
- ②：css页面背景图片 background-image
- ③：标签选择body，因为body是唯一的标签，可以直接写 document.body.style

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8      <style lang="">
9          body{
10             background: url(./images/desktop_1.jpg) no-repeat top
center/cover;
11         }
12     </style>
13 </head>
14 <body>
15     <script>
16         // console.log(document.body)
17         // 取到 N ~ M 的随机整数
18         function getRandom(N, M) {
19             return Math.floor(Math.random() * (M - N + 1)) + N
20         }
21         // 随机数
22         const random = getRandom(1, 10)
23         document.body.style.backgroundImage =
`url(./images/desktop_${random}.jpg)`
24     </script>
25 </body>
26 </html>
```

类名样式操作

语法：

```
1  // element.className
```

类名样式操作——案例1：比较使用行内样式和类名样式操作的区别

CSS

```
1      <style>
2          div{
3              width: 100px;
4              height: 100px;
5              background-color: pink;
6          }
7          .change{
8              background-color: red;
9              font-size: 25px;
10             color: #fff;
11             margin-top: 100px;
12         }
13     </style>
```

html

```
1  <div>
2      <p>文本</p>
3  </div>
4  <script>
5      var div1 = document.querySelector('div');
6      div1.onclick = function(){
7          // 使用行内样式修改元素样式，但是仅仅适用于样式比较少或者功能简单的情况下使用
8          // this.style.backgroundColor = 'red';
9          // this.style.color = '#fff';
10         // this.style.fontSize = '25px';
11         // this.style.marginTop = '100px';
12         // 使用类名样式操作来看下，代码会更加简洁，适用于样式比较多的或者功能复杂的情
况
13         this.className = 'change';
14     }
15 </script>
```

注意：

- 1、js里面的样式采取驼峰命名法，比如：fontSize、backgroundColor
- 2、js修改style样式操作，产生的行内样式，css权重比较高
- 3、如果样式修改较多，可以采取操作类名方式更改元素样式
- 4、class 因为是个保留字，因此使用 className 来操作元素类名属性
- 5、className 会直接更改元素的类名，会覆盖原先的类名

类名样式操作——案例1：密码框格式提示错误信息

用户如果离开密码框，里面输入的个数不是6~16，则提示错误信息，否则提示输入信息正确思路；

①先判断的事件，需要用到表单失去焦点

②如果输入正确则提示正确的信息，颜色使用为绿色

③如果输入不是6~16位，我们采取 `className` 修改样式

④因为样式可能变化较多，我们采取 `className` 进行样式修改

CSS

```
1      <style>
2          .msg{
3              color: blue;
4          }
5          .re{
6              color: red;
7          }
8          .rg{
9              color: green;
10         }
11     </style>
```

html

```
1  <input type="text" class="ipt">
2      <span class="msg">请输入6~16为字母</span>
3      <script>
4          // 1、获取元素
5          var ipt1 = document.querySelector('.ipt');
6          var msg1 = document.querySelector('.msg');
7          // 2、绑定失去焦点事件
8          ipt1.onblur = function(){
9              // 过去表单里面值的长度 ipt1.value.length
10             if(this.value.length < 6 || this.value.length >16){
11                 msg1.className = 're';
12                 msg1.innerHTML = '你输入的位数不正确，要求6~16位哦!';
13             }else{
14                 msg1.className = "rg";
15                 msg1.innerHTML = '恭喜，你终于输入对了.';
16             }
17         }
18     </script>
```

操作元素样式属性

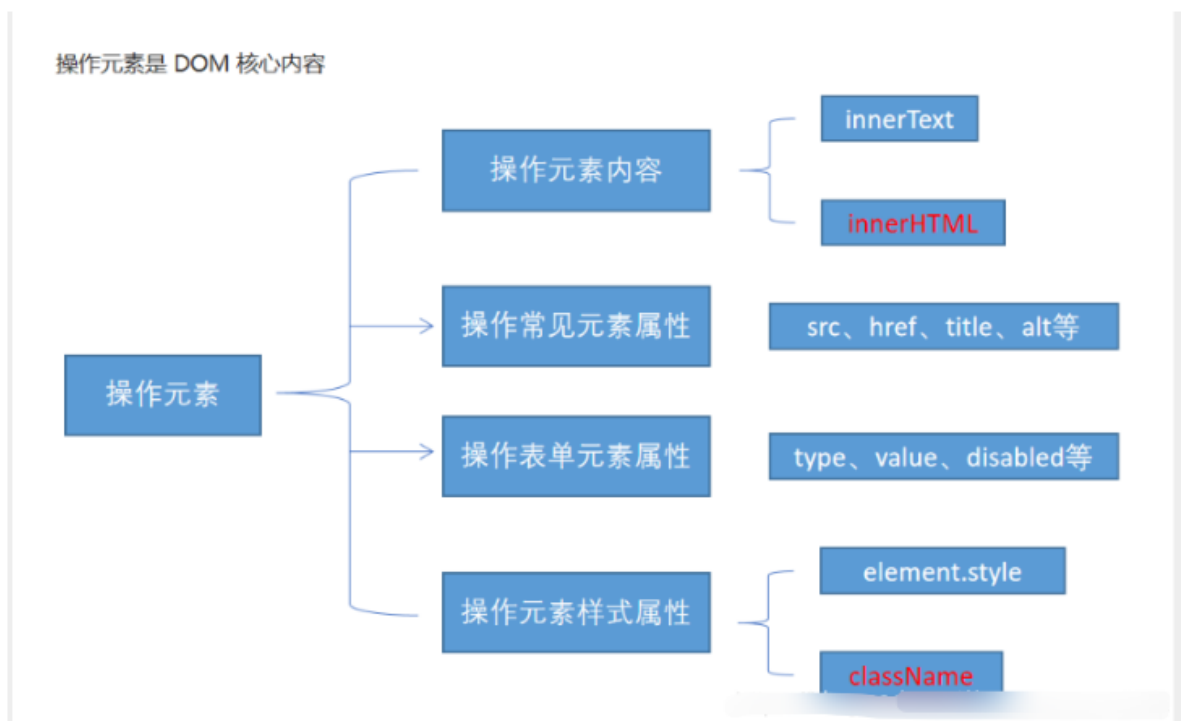
- 通过 `classList` 操作类控制 CSS
- 为了解决 `className` 容易覆盖以前的类名，我们可以通过 `classList` 方式追加和删除类名
- 语法：

```
// 追加一个类
元素.classList.add('类名')
// 删除一个类
元素.classList.remove('类名')
// 切换一个类
元素.classList.toggle('类名')
```

操作元素小结

使用 `className` 和 `classList` 的区别？

- Ø 修改大量样式的更方便
- Ø 修改不多样式的时候方便
- Ø `classList` 是追加和删除不影响以前类名



案例：轮播图

需求：当我们刷新页面，页面中的轮播图会显示不同图片以及样式



模块：

1. 图片会随机变换
2. 底部盒子背景颜色和文字内容会变换
3. 小圆点随机一个高亮显示

案例：轮播图随机版

需求：当我们刷新页面，页面中的轮播图会显示不同图片以及样式

分析：

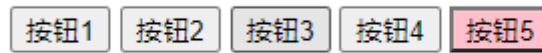
1. 准备一个数组对象，里面包含详细信息（素材包含）
2. 随机选择一个数字，选出数组对应的对象，更换图片，底部盒子背景颜色，以及文字内容
3. 利用这个随机数字，让小圆点添加高亮的类（addClass） 利用css 结构伪类选择器

```
// 1. 初始数据
const sliderData = [
  { url: './images/slider01.jpg', title: '对人类来说会不会太超前了?', color: 'rgb(100, 67, 68)' },
  { url: './images/slider02.jpg', title: '开启剑与雪的黑暗传说!', color: 'rgb(43, 35, 26)' },
  { url: './images/slider03.jpg', title: '真正的jo厨出现了!', color: 'rgb(36, 31, 33)' },
  { url: './images/slider04.jpg', title: '李玉刚：让世界通过B站看到东方大国文化', color: 'rgb(139, 98, 66)' },
  { url: './images/slider05.jpg', title: '快来分享你的寒假日常吧~', color: 'rgb(67, 90, 92)' },
  { url: './images/slider06.jpg', title: '哔哩哔哩小年YEAH', color: 'rgb(166, 131, 143)' },
  { url: './images/slider07.jpg', title: '一站式解决你的电脑配置问题!!!', color: 'rgb(53, 29, 25)' },
  { url: './images/slider08.jpg', title: '谁不想和小猫咪贴贴呢!', color: 'rgb(99, 72, 114)' },
]
```

排他思想

案例：思考，使用操作元素所学内容，完成以下操作

点击第一个按钮为粉色，其他按钮默认无颜色，点击第二个按钮时，其他按钮默认也是无颜色的状态，依次类推



如果有同一组元素，我们相要某一个元素实现某种样式，需要用到循环的排他思想算法：

1. 所有元素全部清除样式（干掉其他人）
2. 给当前元素设置样式（留下我自己）
3. 注意顺序不能颠倒，首先干掉其他人，再设置自己

```
1      <button>按钮1</button>
2      <button>按钮2</button>
3      <button>按钮3</button>
4      <button>按钮4</button>
5      <button>按钮5</button>
6      <script>
7          // 1. 获取所有按钮元素
8          var btns = document.getElementsByTagName('button');
9          // btns得到的是伪数组 里面的每一个元素 btns[i]
10         for (var i = 0; i < btns.length; i++) {
11             btns[i].onclick = function() {
12                 // (1) 我们先把所有的按钮背景颜色去掉 干掉所有人
13                 for (var i = 0; i < btns.length; i++) {
14                     btns[i].style.backgroundColor = '';
15                 }
16                 // (2) 然后才让当前的元素背景颜色为pink 留下我自己
17                 this.style.backgroundColor = 'pink';
18             }
19         }
20     }
21     //2. 首先排除其他人，然后才设置自己的样式 这种排除其他人的思想我们成为排他思想
22 </script>
```

案例：换肤效果

CSS

```

1 <style>
2     body{
3         background: url(./images/A.png) repeat;
4     }
5     ul{
6         display: inline-block;
7         border: 3px solid aqua;
8     }
9 </style>

```

html

```

1 <ul class="box">
2     <li>
3         
4     </li>
5     <li>
6         
7     </li>
8     <li>
9         
10    </li>
11 </ul>
12 <script>
13     // 1、获取元素
14     var img1 = document.querySelector('.box').querySelectorAll('img');
15     // console.log(img1);
16     for(var i = 0;i<img1.length;i++){
17         img1[i].onclick = function(){
18             // this.src 就是我们图片的路径， ./images/B.png
19             // console.log(this.src);
20             // 将this.src 路径设置给 body 上
21             document.body.style.backgroundImage = 'url('+this.src+')';
22         }
23     }
24 </script>

```

自定义属性操作

获取属性值的两种方法

```

1 <div id="demo"></div>
2 <script>
3     var div1 = document.querySelector('div');
4     // 1.获取元素的属性值
5     // (1) element.属性
6     console.log(div1.id); //demo
7     // (2) element.getAttribute('属性') get得到获取 attribute 属性的意思，自
    定义属性，即编程人员自定的属性
8     console.log(div1.getAttribute('id')); //demo
9 </script>

```


(1) element.属性

(2) element.getAttribute('属性')

获取属性值两种方法的区别

- element.属性 获取内置属性值(元素本身自带的属性)
- element.getAttribute('属性'): 主要用于获取自定义属性的。自定义属性: 即编程人员自定的属性。

```
1 <div id="demo" index="1" ></div>
2 <script>
3     var div1 = document.querySelector('div');
4     console.log(div1.index);    //undefined
5     console.log(div1.getAttribute('index'));    //1
6 </script>
```

设置属性值的两种方法

```
1 <div id="demo" index="1"></div>
2 <script>
3     var div1 = document.querySelector('div');
4     // 1、设置元素的属性值
5     // (1) element.属性名 = '值';
6     div1.id = 'test';
7     // (2) element.setAttribute('属性名','属性值'),只要针对自定义属性,可以修
    改,也可以新增。
8     div1.setAttribute('index',2);
9     div1.setAttribute('class','footer');    //class 比较特殊,必须写class,
    否则css样式不生效。
10 </script>
```

(1) element.属性名 = '值';

(2) element.setAttribute('属性名','属性值'),只要针对自定义属性,可以修改,也可以新增。

设置属性值两种方法的区别

- element.属性
- element.setAttribute('属性'); 主要用于设置自定义属性。

移出属性值

(1)移除属性值 removeAttribute('属性');

```
1 <div id="demo" index="1"></div>
2 <script>
3     var div1 = document.querySelector('div');
4     // 1、移除属性值 removeAttribute('属性');
5     div1.removeAttribute('index');
6 </script>
```

案例：S10-tab栏切换

显示效果：[S10-table栏切换](#)

思路：

代码：

CSS

```
1      <style>
2          .tab_list ul,.tab_list li{
3              margin: 0;
4              padding: 0;
5          }
6          .tab ul li{
7              list-style: none;
8              float: left;
9              border: 1px solid black;
10             border-left: none;
11             padding: 20px;
12         }
13         .current{
14             color: white;
15             background-color: red;
16         }
17         .item{
18             display: none;
19         }
20
21     </style>
```

HTML

```
1  <div class="tab">
2      <div class="tab_list">
3          <ul>
4              <li class="current" >商品介绍</li>
5              <li>规格与包装</li>
6              <li>售后保障</li>
7              <li>商品评价(50000)</li>
8              <li>手机社区</li>
9          </ul>
10     </div>
11     <div style="clear: both;"></div>
12     <div class="tab_con">
13         <div class="item" style="display: block;">商品介绍模块</div>
14         <div class="item">规格与包装模块</div>
15         <div class="item">售后保障模块</div>
16         <div class="item">商品评价模块</div>
17         <div class="item">手机社区模块</div>
18     </div>
```

```

19     </div>
20
21     <script>
22         // 获取元素
23         var tab_list = document.querySelector('.tab_list');
24         var list = tab_list.querySelectorAll('li');
25         var items = document.querySelectorAll('.item');
26         // 利用for循环绑定点击事件
27         for(var i = 0;i < list.length;i++){
28             // 开始给5个li标签设置index属性和值
29             list[i].setAttribute('index',i);
30             list[i].onclick = function(){
31                 // 1.模块选项卡，点击某一个，默认第一个li标签的中的底色是红色，其余不变
(利用排他思想)修改类名的方式
32                 // 使用排它思想，将所有li中的class样式current清除
33                 for(var i = 0;i<list.length;i++){
34                     list[i].className = '';
35                 }
36                 // 给自己加样式
37                 this.className = 'current';
38                 // 2、选项卡解决了，需要处理内容模块。当点击对应的选项卡，出现对应的内
容。
39                 // 利用li标签中自定义属性的功能，自定义一个属性。
40                 var index = this.getAttribute('index');
41                 console.log(index);
42                 // 使用排它思想，将所有item中的div全部隐藏
43                 for(var i = 0;i < items.length;i++){
44                     items[i].style.display = 'none';
45                 }
46                 items[index].style.display = 'block'了;
47             }
48         }
49     </script>

```

H5自定义属性

自定义属性目的：

- 保存并保存数据，有些数据可以保存到页面中而不用保存到数据库中
- 有些自定义属性很容易引起歧义，不容易判断到底是内置属性还是自定义的，所以H5有了规定

设置H5自定义属性

H5规定自定义属性 `data-` 开头作为属性名并赋值

```

1 <!-- 自定义属性 -->
2 <div data-index = "1"></div>
3 <script>
4     var div1 = document.querySelector('div');
5     // 或者使用JavaScript设置
6     div1.setAttribute('data-obj',10);
7 </script>

```

获取H5自定义属性

- 兼容性获取 `element.getAttribute('data-index')`
- H5新增的: `element.dataset.index` 或 `element.dataset['index']` [IE11才开始支持]

```

1 <div getTime="20" data-index="2" data-list-name="andy"></div>
2 <script>
3     var div1 = document.querySelector('div');
4     console.log(div1.getAttribute('getTime'));
5     div1.setAttribute('data-time', 20);
6     console.log(div1.getAttribute('data-index'));
7     console.log(div1.getAttribute('data-list-name'));
8     // h5新增的获取自定义属性的方法 它只能获取data-开头的
9     // dataset 是一个集合里面存放了所有以data开头的自定义属性
10    console.log(div1.dataset);
11    console.log(div1.dataset.index);
12    console.log(div1.dataset['index']);
13    // 如果自定义属性里面有多条-链接的单词，我们获取的时候采取 驼峰命名法
14    console.log(div1.dataset.listName);
15    console.log(div1.dataset['listName']);
16 </script>

```

在以后开发中，一定要知道其代码的写法及意义。

定时器——间歇函数

- 网页中经常会需要一种功能：每隔一段时间需要自动执行一段代码，不需要我们手动去触发
- 例如：网页中的倒计时
- 要实现这种需求，需要定时器函数
- 定时器函数有两种，先讲间歇函数

1. 开启定时器

```

1 setInterval(函数,间隔时间);

```

1. 作用：每隔一段时间调用这个函数
2. 间隔时间单位是毫秒

例子:

```
1 <script>
2     function time(){
3         console.log('一秒只触发了一次');
4     }
5     setInterval(time,1000);
6 </script>
```

1. 注意：函数不需要加括号。
2. 定时器返回的是一个 id 数字

2.关闭定时器

```
1 let 变量名 = setInterval(函数, 间隔时间);
2 clearInterval(变量名);
```

一般不会刚创建就停止，而是满足一定条件再停止

例子:

```
1 <script>
2     let timer = setInterval(function(){
3         console.log('一秒只触发了一次');
4     },1000);
5
6     clearInterval(timer);
7 </script>
```

案例：阅读注册协议

需求：按钮60秒之后才可以使用

分析：

- ①：开始先把按钮禁用（disabled 属性）
- ②：一定要获取元素
- ③：函数内处理逻辑

秒数开始减减

按钮里面的文字跟着一起变化

如果秒数等于0 停止定时器 里面文字变为 同意 最后 按钮可以点击

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Document</title>
8 </head>
9 <body>
```

```

10     <textarea name="" id="" cols="30" rows="10">
11         用户注册协议
12         欢迎注册成为京东用户！在您注册过程中，您需要完成我们的注册流程并通过点击同意的形式
        在线签署以下协议，请您务必仔细阅读、充分理解协议中的条款内容后再点击同意（尤其是以粗体或下划
        线标识的条款，因为这些条款可能会明确您应履行的义务或对您的权利有所限制）。
13         【请您注意】如果您不同意以下协议全部或任何条款约定，请您停止注册。您停止注册后将仅
        可以浏览我们的商品信息但无法享受我们的产品或服务。如您按照注册流程提示填写信息，阅读并点击同
        意上述协议且完成全部注册流程后，即表示您已充分阅读、理解并接受协议的全部内容，并表明您同意我
        们可以依据协议内容来处理您的个人信息，并同意我们将您的订单信息共享给为完成此订单所必须的第三
        方合作方（详情查看
14     </textarea>
15     <br>
16     <button class="btn" disabled>我已经阅读用户协议(5)秒</button>
17
18     <script>
19         // 获取按钮元素
20         const btn = document.querySelector('.btn');
21
22         let i = 5;
23
24         let n = setInterval(function(){
25             i--;
26             btn.innerHTML = `我已经阅读用户协议${i}秒`;
27             if(i === 0){
28                 // 关闭定时器
29                 clearInterval(n);
30                 // 时间到，关闭定时器，开启按钮
31                 btn.disabled = false;
32                 btn.innerHTML = '同意';
33             }
34         },1000)
35     </script>
36 </body>
37 </html>

```

综合练习：轮播图

需求：每隔一秒钟切换一个图片

分析：

- ①：准备一个数组对象，里面包含详细信息（素材包含）
- ②：获取元素
- ③：设置定时器函数

设置一个变量++

找到变量对应的对象

更改图片、文字信息

激活小圆点：移除上一个高亮的类名，当前变量对应的小圆点添加类

- ④：处理图片自动复原从头播放（放到变量++后面，紧挨）

如果图片播放到最后一张，就是大于等于数组的长度

则把变量重置为0

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8" />
6   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
7   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
8   <title>轮播图点击切换</title>
9   <style>
10     * {
11       box-sizing: border-box;
12     }
13
14     .slider {
15       width: 560px;
16       height: 400px;
17       overflow: hidden;
18     }
19
20     .slider-wrapper {
21       width: 100%;
22       height: 320px;
23     }
24
25     .slider-wrapper img {
26       width: 100%;
27       height: 100%;
28       display: block;
29     }
30
31     .slider-footer {
32       height: 80px;
33       background-color: rgb(100, 67, 68);
34       padding: 12px 12px 0 12px;
35       position: relative;
36     }
37
38     .slider-footer .toggle {
39       position: absolute;
40       right: 0;
41       top: 12px;
42       display: flex;
43     }
44
45     .slider-footer .toggle button {
46       margin-right: 12px;
47       width: 28px;
48       height: 28px;
49       appearance: none;
50       border: none;
51       background: rgba(255, 255, 255, 0.1);
52       color: #fff;
53       border-radius: 4px;
```

```
54     cursor: pointer;
55 }
56
57 .slider-footer .toggle button:hover {
58     background: rgba(255, 255, 255, 0.2);
59 }
60
61 .slider-footer p {
62     margin: 0;
63     color: #fff;
64     font-size: 18px;
65     margin-bottom: 10px;
66 }
67
68 .slider-indicator {
69     margin: 0;
70     padding: 0;
71     list-style: none;
72     display: flex;
73     align-items: center;
74 }
75
76 .slider-indicator li {
77     width: 8px;
78     height: 8px;
79     margin: 4px;
80     border-radius: 50%;
81     background: #fff;
82     opacity: 0.4;
83     cursor: pointer;
84 }
85
86 .slider-indicator li.active {
87     width: 12px;
88     height: 12px;
89     opacity: 1;
90 }
91 </style>
92 </head>
93
94 <body>
95     <div class="slider">
96         <div class="slider-wrapper">
97             
98         </div>
99         <div class="slider-footer">
100             <p>对人类来说会不会太超前了? </p>
101             <ul class="slider-indicator">
102                 <li class="active"></li>
103                 <li></li>
104                 <li></li>
105                 <li></li>
106                 <li></li>
107                 <li></li>
108                 <li></li>
```



```

109     </li></li>
110 </ul>
111 </div>
112 </div>
113 <script>
114     // 1. 初始数据
115     const sliderData = [
116         { url: './images/slider01.jpg', title: '对人类来说会不会太超前了? ',
117         color: 'rgb(100, 67, 68)' },
118         { url: './images/slider02.jpg', title: '开启剑与雪的黑暗传说! ', color:
119         'rgb(43, 35, 26)' },
120         { url: './images/slider03.jpg', title: '真正的jo厨出现了! ', color:
121         'rgb(36, 31, 33)' },
122         { url: './images/slider04.jpg', title: '李玉刚: 让世界通过B站看到东方大国
123         文化', color: 'rgb(139, 98, 66)' },
124         { url: './images/slider05.jpg', title: '快来分享你的寒假日常吧~', color:
125         'rgb(67, 90, 92)' },
126         { url: './images/slider06.jpg', title: '哔哩哔哩小年YEAH', color:
127         'rgb(166, 131, 143)' },
128         { url: './images/slider07.jpg', title: '一站式解决你的电脑配置问题!!! ',
129         color: 'rgb(53, 29, 25)' },
130         { url: './images/slider08.jpg', title: '谁不想和小猫咪贴贴呢! ', color:
131         'rgb(99, 72, 114)' },
132     ]
133
134     // 获取元素, class为 .slider-warpper 下的图片
135     const img = document.querySelector('.slider-wrapper img');
136     // 获取图片下对应的 p 标签元素
137     const p = document.querySelector('.slider-footer p');
138
139     let i = 0; // 图片的起始量索引
140
141     // 设置定时器
142     setInterval(function(){
143         i++;
144         // 无缝衔接, 总共8张图片, 到最后一张就是 8 , 数量长度为 8
145         if(i >= sliderData.length){
146             i = 0;
147         }
148         // 更换图片路径
149         img.src = sliderData[i].url;
150         // 更换p标签的文字内容
151         p.innerHTML = sliderData[i].title;
152         // 获取小圆点
153         let aq = document.querySelector('.slider-indicator .active');
154         // 移除小圆点的样式
155         aq.classList.remove('active');
156         // 给当前 li 添加 active 样式
157         document.querySelector(`.slider-indicator li:nth-child(${i +
158         1})`).classList.add('active');
159     },1000)
160 </script>
161 </body>
162 </html>

```

节点操作

获取元素通常使用两种方式：

1.利用DOM提供的方法获取元素	2.利用节点层级关系获取元素
<code>document.getElementById()</code>	利用父子兄节点关系获取元素
<code>document.getElementsByTagName()</code>	逻辑性强，但是兼容性较差
<code>document.querySelector</code> 等	
逻辑性不强，繁琐	

这两种方式都可以获取元素节点，我们后面都会使用，但是节点操作更简单

一般的，节点至少拥有三个基本属性

DOM节点概述

网页中的所有内容都是节点（标签、属性、文本、注释等），在DOM中，节点使用 node 来表示。

HTML DOM 树中的所有节点均可通过 JavaScript 进行访问，所有 HTML 元素（节点）均可被修改，也可以创建或删除。

1.2 DOM 树



一般的，节点至少拥有nodeType（节点类型）、nodeName（节点名称）和nodeValue（节点值）这三个基本属性。

- 元素节点：nodeType 为1
- 属性节点：nodeType 为2
- 文本节点：nodeType 为3(文本节点包括文字、空格、换行等)

我们在实际开发中，节点操作主要操作的是**元素节点**。

利用 DOM 树可以把节点划分为不同的层级关系，常见的是**父子兄层级关系**。

查找节点__父级节点

1 | `node.parentNode`

- `parentNode` 属性可以返回某节点的父结点，注意是最近的一个父结点
- 如果指定的节点没有父结点则返回 `null`

例子:

```
1      <!-- 父节点 -->
2      <div class="demo">
3          <div class="box">
4              <span class="erweima">x</span>
5          </div>
6      </div>
7
8      <script>
9          // 1. 父节点 parentNode
10         var erweima = document.querySelector('.erweima');
11         // var box = document.querySelector('.box');
12         // 得到的是离元素最近的父级节点(亲爸爸) 如果找不到父节点就返回为 null
13         console.log(erweima.parentNode);
14     </script>
```

查找节点__子结点

1 | `parentNode.childNodes`(标准)

- `parentNode.childNodes` 返回包含指定节点的子节点的集合，该集合为即时更新的集合
- 返回值包含了所有的子结点，包括元素节点，文本节点等
- 如果只想要获得里面的元素节点，则需要专门处理。所以我们一般不提倡使用 `childNodes`

1 | `parentNode.children`(非标准)

- `parentNode.children` 是一个只读属性，返回所有的子元素节点
- 它只返回子元素节点，其余节点不返回 (**这个是我们重点掌握的**)
- 虽然 `children` 是一个非标准，但是得到了各个浏览器的支持，因此我们可以放心使用

```
1      <ul>
2          <li>我是li</li>
3          <li>我是li</li>
4          <li>我是li</li>
5          <li>我是li</li>
6      </ul>
7      <ol>
8          <li>我是li</li>
9          <li>我是li</li>
```

```

10     <li>我是li</li>
11     <li>我是li</li>
12 </ol>
13 <script>
14     // DOM 提供的方法（API）获取
15     var ul = document.querySelector('ul');
16     var lis = ul.querySelectorAll('li');
17     // 1. 子节点 childNodes 所有的子节点 包含 元素节点 文本节点等等
18     console.log(ul.childNodes);
19     console.log(ul.childNodes[0].nodeType);
20     console.log(ul.childNodes[1].nodeType);
21     // // 2. children 获取所有的子元素节点 也是我们实际开发常用的
22     console.log(ul.children);
23 </script>

```

第一个子结点

```
1 parentNode.firstChild
```

- `firstChild` 返回第一个子节点，找不到则返回null
- 同样，也是包含所有的节点

```
1 parentNode.firstElementChild
```

- 第一个子结点(兼容性)
- `firstElementChild` 返回第一个子节点，找不到则返回null
- 有兼容性 问题，IE9以上才支持

最后一个子结点

```
1 parentNode.lastChild
```

- `lastChild` 返回最后一个子节点，找不到则返回null
- 同样，也是包含 所有的节点

```
1 parentNode.lastElementChild
```

- 最后一个子结点(兼容性)
- `lastElementChild` 返回最后一个子节点，找不到则返回null
- 有兼容性问题，IE9以上才支持

例子：

```

1 <ol>
2     <li>我是li1</li>
3     <li>我是li2</li>
4     <li>我是li3</li>
5     <li>我是li4</li>

```

```

6      <li>我是li5</li>
7  </ol>
8  <script>
9      var ol = document.querySelector('ol');
10     // 1. firstChild 第一个子节点 不管是文本节点还是元素节点
11     console.log(ol.firstChild);
12     console.log(ol.lastChild);
13     // 2. firstElementChild 返回第一个子元素节点 ie9才支持
14     console.log(ol.firstElementChild);
15     console.log(ol.lastElementChild);
16     // 3. 实际开发的写法 既没有兼容性问题又返回第一个子元素
17     console.log(ol.children[0]);          //第一个子元素节点
18     console.log(ol.children[ol.children.length - 1]); //最后一个子元素节点
19 </script>

```

实际开发中，`firstChild` 和 `lastChild` 包含其他节点，操作不方便，而 `firstElementChild` 和 `lastElementChild` 又有兼容性问题，那么我们如何获取第一个子元素节点或最后一个子元素节点呢？

使用children索引来获取元素

```

1 // 数组元素个数减1 就是最后一个元素的索引号
2 parentNode.children[parentNode.children.length - 1]

```

- 如果想要第一个子元素节点，可以使用 `parentNode.children[0]`
- 如果想要最后一个子元素节点，可以使用

例子：

```

1 <ol>
2     <li>我是li1</li>
3     <li>我是li2</li>
4     <li>我是li3</li>
5     <li>我是li4</li>
6 </ol>
7 <script>
8     var ol = document.querySelector('ol');
9     // 1.firstChild 获取第一个子结点的，包含文本节点和元素节点
10    console.log(ol.firstChild);
11    // 返回的是文本节点 #text(第一个换行结点)
12
13    console.log(ol.lastChild);
14    // 返回的是文本节点 #text(最后一个换行结点)
15    // 2. firstElementChild 返回第一个子元素节点
16    console.log(ol.firstElementChild);
17    // <li>我是li1</li>
18
19    // 第2个方法有兼容性问题，需要IE9以上才支持
20    // 3.实际开发中，既没有兼容性问题，又返回第一个子元素
21    console.log(ol.children[0]);
22    // <li>我是li1</li>
23    console.log(ol.children[3]);

```

```
24 // <li>我是li4</li>
25 // 当里面li个数不唯一时候，需要取到最后一个结点时这么写
26 console.log(ol.children[ol.children.length - 1]);
27 </script>
```

案例：S17案例-新浪下拉菜单（使用技术：子节点，鼠标事件，获取元素）

参考网址链接：[新浪首页](#)

效果预览如下：[S17案例-新浪下拉菜单](#)

CSS

```
1  *{
2      margin: 0;
3      padding:0;
4  }
5  li{
6      list-style: none;
7  }
8  .nav{
9      background: #fcfcfc;
10 }
11 .nav>li{
12     float: left;
13 }
14 .nav>li a{
15     text-decoration: none;
16     color: black;
17 }
18 .nav_a1{
19     display: inline-block;
20     padding: 12px 15px 12px 18px;
21 }
22 .nav_a1:hover{
23     background:#edeef0;
24 }
25 /* 平替方法，但是也有bug，鼠标取消悬浮后，会消失，不符合要求 */
26 /* .nav_a1:hover +.nav_1{
27     display: block;
28 } */
29 .nav_1{
30     border: 1px solid #ebbe7a;
31     display: none;
32 }
33 }
34 .nav_1>li{
35     border-bottom: 1px solid #fecc5b;
36     text-align: center;
37     line-height: 20px;
38     padding: 12px 15px;
39 }
```

HTML

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8      <link rel="stylesheet" href="./s17.css">
9      <style>
10
11      </style>
12 </head>
13 <body>
14     <ul class="nav">
15         <li>
16             <a href="#" class="nav_a1">微博</a>
17             <ul class="nav_1">
18                 <li>
19                     <a href="#">私信</a>
20                 </li>
21                 <li>
22                     <a href="#">私信</a>
23                 </li>
24                 <li>
25                     <a href="#">私信</a>
26                 </li>
27             </ul>
28         </li>
29         <li>
30             <a href="#" class="nav_a1">博客</a>
31             <ul class="nav_1">
32                 <li>
33                     <a href="#">博客评论</a>
34                 </li>
35                 <li>
36                     <a href="#">未读提醒</a>
37                 </li>
38             </ul>
39         </li>
40         <li>
41             <a href="#" class="nav_a1">邮箱</a>
42             <ul class="nav_1">
43                 <li>
44                     <a href="#">免费邮箱</a>
45                 </li>
46                 <li>
47                     <a href="#">VIP邮箱</a>
48                 </li>
49             </ul>
50         </li>
51     </ul>
52 </body>
53 </html>

```

```
1 <script>
2     // 获取元素
3     var nav1 = document.querySelector('.nav');
4     var lis = nav1.children;
5     // console.log(lis);
6     for(var i = 0;i<lis.length;i++){
7         lis[i].onmouseover = function(){
8             this.children[1].style.display = 'block';
9         }
10        lis[i].onmouseout = function(){
11            this.children[1].style.display = 'none';
12        }
13    }
14 </script>
```

查找节点__兄弟节点

下一个兄弟节点

```
1 node.nextSibling
```

- `nextSibling` 返回当前元素的下一个兄弟元素节点，找不到则返回null
- 同样，也是包含所有的节点

上一个兄弟节点

```
1 node.previousSibling
```

- `previousSibling` 返回当前元素上一个兄弟元素节点，找不到则返回null
- 同样，也是包含所有的节点

下一个兄弟节点(兼容性)

```
1 node.nextElementSibling
```

- `nextElementSibling` 返回当前元素下一个兄弟元素节点，找不到则返回null
- 有兼容性问题，IE9才支持

上一个兄弟节点(兼容性)

```
1 node.previousElementSibling
```

- `previousElementSibling` 返回当前元素上一个兄弟元素节点，找不到则返回null
- 有兼容性问题，IE9才支持

例子:

```
1      <div>我是div</div>
2      <span>我是span</span>
3      <script>
4          var div1 = document.querySelector('div');
5          // 1.nextSibling 下一个兄弟节点 包含元素节点或者 文本节点等等
6          console.log(div1.nextSibling);    // #text
7          // previousSibling 上一个兄弟节点，包含元素节点或者 文本节点等等
8          console.log(div1.previousSibling); // #text
9          // 2. nextElementSibling 得到下一个兄弟元素节点
10         console.log(div1.nextElementSibling); //<span>我是span</span>
11         // previousElementSibling 得到上一个兄弟元素节点
12         console.log(div1.previousElementSibling); //null
13     </script>
```

通过封装函数解决兄弟节点的兼容性问题 [了解](#)

```
1  function getNextElementSibling(element) {
2      var e1 = element;
3      while(e1 = e1.nextSibling) {
4          if(e1.nodeType === 1){
5              return e1;
6          }
7      }
8      return null;
9  }
```

增加节点__创建节点

```
1  document.createElement('tagName');
```

- `document.createElement()` 方法创建由 `tagName` 指定的HTML 元素
- 因为这些元素原先不存在，是根据我们的需求动态生成的，所以我们也称为**动态创建元素节点**

添加节点

```
1  node.appendChild(child)
```

- `node.appendChild()` 方法将一个节点添加到指定父节点的子节点列表**末尾**。类似于 CSS 里面的 `after` 伪元素。

```
1  node.insertBefore(child,指定元素的位置)
```

- `node.insertBefore()` 方法将一个节点添加到父节点的指定子节点**前面**。类似于 CSS 里面的 `before` 伪元素。

例子:

```
1      <ul>
2          <li>123</li>
3      </ul>
4      <script>
5          // 1. 创建节点元素节点
6          var li = document.createElement('li');
7          // 2. 添加节点 node.appendChild(child) node 父级 child 是子级 后面追加
元素 类似于数组中的push
8          // 先获取父亲ul
9          var ul = document.querySelector('ul');
10         ul.appendChild(li);
11         // // 3. 添加节点 node.insertBefore(child, 指定元素的位置);
12         var lili = document.createElement('li');
13         ul.insertBefore(lili, ul.children[0]);
14
15         var li1 = document.createElement('li');
16         ul.appendChild(li1);
17         // // 4. 我们想要页面添加一个新的元素分两步: 1. 创建元素 2. 添加元素
18     </script>
```

案例: S1820-简单留言板——发布留言功能

效果预览如下: [S1820-简单留言板——发布留言功能](#)

```
1      <textarea>123</textarea>
2      <button>发布</button>
3      <ul>
4
5      </ul>
6
7      <script>
8          var text1 = document.querySelector('textarea');
9          var btn1 = document.querySelector('button');
10         var ul1 = document.querySelector('ul');
11         // 绑定事件
12         btn1.onclick = function(){
13             if(text1.value == ''){
14                 alert('你没输入内容, 就是在占用资源');
15             }else{
16                 var li1 = document.createElement('li');
17                 li1.innerHTML = text1.value;
18                 // 添加元素, 从后面添加
19                 // ul1.appendChild(li1);
20                 ul1.insertBefore(li1, ul1.children[0]);
21             }
22         }
23     </script>
```

删除节点

```
1 node.removeChild(child)
```

- `node.removeChild()` 方法从 DOM 中删除一个子节点，返回删除的节点

例子：

```
1      <button>删除</button>
2      <ul>
3          <li>甲</li>
4          <li>乙</li>
5          <li>丙</li>
6      </ul>
7      <script>
8          // 获取元素
9          var ul1 = document.querySelector('ul');
10         var btn1 = document.querySelector('button');
11         // 删除元素
12         // ul1.removeChild(ul1.children[0]);
13         // 点击按钮，依次删除li中的文字，当li标签删除没有后，就会报错。
14         // btn1.onclick = function(){
15         //     ul1.removeChild(ul1.children[0]);
16         // }
17         btn1.onclick = function(){
18             if(ul1.children.length == 0){
19                 this.disabled = true;
20             }else{
21                 ul1.removeChild(ul1.children[0]);
22             }
23         }
24     </script>
```

案例：S1822删除留言案例

效果预览如下：[S1822删除留言案例](#)

HTML

```
1      <textarea>123</textarea>
2      <button>发布</button>
3      <ul>
4          <li>
5              这个案例,没有加js功能
6              <a href="#">删除</a>
7          </li>
8      </ul>
9
10     <script>
11         var text1 = document.querySelector('textarea');
```

```

12     var btn1 = document.querySelector('button');
13     var ul1 = document.querySelector('ul');
14     // 绑定事件
15     btn1.onclick = function(){
16         if(text1.value == ''){
17             alert('你没输入内容，就是在占用资源');
18         }else{
19             var li1 = document.createElement('li');
20             li1.innerHTML = text1.value + "<a href='javascript:;'>删除
    </a>";
21             // 添加元素，从后面添加
22             // ul1.appendChild(li1);
23             ul1.insertBefore(li1,ul1.children[0]);
24             // 删除元素，删除当前的li元素
25             // 获取a标签的元素
26             var as = document.querySelectorAll('a');
27             for(var i = 0;i<as.length;i++){
28                 as[i].onclick = function(){
29                     // node.removeChild(child);删除的是li当前a所在的li，
    this.parentNode
30                     ul1.removeChild(this.parentNode);
31                 }
32             }
33         }
34     }
35     </script>

```

复制节点(克隆节点)

```
1 node.cloneNode()
```

- `node.cloneNode()` 方法返回调用该方法的节点的一个副本。也称为克隆节点/拷贝节点
- 如果括号参数为空或者为 `false`，则是浅拷贝，即只克隆复制节点本身，不克隆里面的子节点
- 如果括号参数为 `true`，则是深度拷贝，会复制节点本身以及里面所有的子节点

例子：

```

1     <ul>
2         <li>1</li>
3         <li>2</li>
4         <li>3</li>
5     </ul>
6
7     <script>
8         var ul1 = document.querySelector('ul');
9         var li1 = ul1.children[0].cloneNode(true);
10        // 1、如果括号参数为空或者为FALSE，则是浅拷贝，即只克隆复制节点本身，不克隆里面
    的字节点。
11        // 2、括号参数为 true，则是深度拷贝，会复制节点本身以及里面所有的子节点
12        ul1.appendChild(li1);
13    </script>

```

案例：S1824案例-动态生成表格

效果预览如下：[S1824案例-动态生成表格](#)

HTML

```
1      <table cellspacing="0" border="1">
2          <thead>
3              <td>姓名</td>
4              <td>考试科目</td>
5              <td>考试成绩</td>
6              <td>删除</td>
7          </thead>
8          <tbody>
9              <tr>
10                 <td>我是案例，没有添加</td>
11                 <td>HTML</td>
12                 <td>100</td>
13                 <td>
14                     <a href="#">删除</a>
15                 </td>
16             </tr>
17         </tbody>
18     </table>
19
20     <script>
21         // 1、先去准备好学生的数据，使用数组对象，创建数据
22         var datas = [
23             {
24                 name: '路人甲',
25                 subject: 'JavaScript',
26                 score: 100
27             },
28             {
29                 name: '路人乙',
30                 subject: 'HTML',
31                 score: 90
32             },
33             {
34                 name: '路人丙',
35                 subject: 'CSS',
36                 score: 96
37             },
38             {
39                 name: '路人丁',
40                 subject: 'jQuery',
41                 score: 80
42             }
43         ];
44         // 2、往tbody 里面创建行，有几个人，就创建几个行
45         var tbody1 = document.querySelector('tbody');
46         for(var i = 0; i < datas.length; i++){ //外层for循环管行，tr
47             // 3、创建tr行
48             var tr1 = document.createElement('tr');
```

```

49         tbody1.appendChild(tr1);
50         // 行里面创建单元格(跟数据有关) td 单元格的数量取决于每个对象里面的属性个
数, for循环遍历对象
51         for(var k in datas[i]){ //里面for循环管列数, td
52             // 创建单元格
53             var td = document.createElement('td');
54             // 插入创建的单元格
55             td.innerHTML = datas[i][k];
56             tr1.appendChild(td);
57             // 把对象里面的属性值给 td
58             //获取对象里面的属性值 datas[i][k]
59             // console.log(datas[i][k]);
60         }
61         // 3、创建有删除2个字对应的单元格
62         var td = document.createElement('td');
63         td.innerHTML = '<a href="#">删除</a>';
64         tr1.appendChild(td);
65     }
66     /* for(var k in datas){
67         k 是属性名
68         datas[k] 是属性值
69     } */
70
71     // 3、添加删除操作
72     var as = document.querySelectorAll('a');
73     for(var i = 0;i<as.length;i++){
74         as[i].onclick = function(){
75             // 点击a 删除 当前a 所在的行 (对应的链接) node.removeChild(child)
76             tbody1.removeChild(this.parentNode.parentNode);
77         }
78     }
79     </script>

```

三种动态创建元素的区别

document.write()

```

1     <button>点击</button>
2     <p>abc</p>
3
4     <script>
5         //三种创建元素的区别
6         // 1、 document.write
7         var btn1 = document.querySelector('button');
8         btn1.onclick = function(){
9             document.write('<div>123</div>');
10        }
11    </script>

```

element.innerHTML

创建单个元素

```
1 <div class="inner"></div>
2 <script>
3     // 使用innerHTML创建元素
4     var inner1 = document.querySelector('.inner');
5     inner1.innerHTML = '<a href="#">百度</a>';
6 </script>
```

使用 innerHTML 创建多个元素

```
1 <div class="inner"></div>
2 <script>
3     // 使用innerHTML创建多个元素
4     var inner1 = document.querySelector('.inner');
5     for(var i = 0;i<=100;i++){
6         inner1.innerHTML += '<a href="#">百度</a>';
7     }
8
9     // 使用innerHTML创建多个元素，检测加载时间需要多久，加载时长1500~2000毫秒
10    function fn(){
11        var d1 = +new Date();
12        var str = '';
13        for(var i = 0 ;i < 1000;i++){
14            document.body.innerHTML += '<div style="width:100px;height:
15            10px;border: 1px solid black;"></div>';
16        }
17        var d2 = +new Date();
18        console.log(d2-d1);
19    }
20    fn();
</script>
```

解决使用innerHTML创建多个元素时，加载时间过长的办法，使用数组追加的方式，

```
1 <!-- 解决使用innerHTML创建多个元素时，加载时间过长的办法，使用数组追加的方式 -->
2 <div class="inner"></div>
3 <script>
4     var inner1 = document.querySelector('.inner');
5     var arr = [];
6     for(var i = 0;i<100;i++){
7         arr.push('<a href="#">百度</a>');
8     }
9     inner1.innerHTML = arr.join('');
10
11    // 使用innerHTML创建多个元素，用数组追加的方式，检测加载时间需要多久，加载时长5
    毫秒
12    function fn(){
13        var d1 = +new Date();
14        var array = [];
15        for(var i = 0;i<1000;i++){
```

```

16         array.push('<div style="width:100px;height:10px;border:1px
solid blue;"></div>');
17     }
18     document.body.innerHTML = array.join('');
19     var d2 = +new Date();
20     console.log(d2-d1);
21 }
22 fn();
23 </script>

```

document.createElement()

创建单个元素

```

1     <div class="create"></div>
2     <script>
3         // 使用createElement()创建元素
4         var create1 = document.querySelector('.create');
5         var a = document.createElement('a');
6         create1.appendChild(a);
7     </script>

```

使用 createElement() 创建多个元素

```

1     <div class="create"></div>
2     <script>
3         // 使用createElement()创建元素
4         var create1 = document.querySelector('.create');
5         var a = document.createElement('a');
6         create1.appendChild(a);
7
8         // 使用createElement()创建多个元素，检测加载时间需要多久，加载时间10~13毫秒
9         function fn(){
10             var d1 = +new Date();
11             for(var i = 0;i<1000;i++){
12                 var div1 = document.createElement('div');
13                 div1.style.width = '100px';
14                 div1.style.height = '100px';
15                 div1.style.border = '1px solid red';
16                 document.body.appendChild(div1);
17             }
18             var d2 = +new Date();
19             console.log(d2-d1);
20         }
21         fn();
22     </script>

```

区别:

- `document.write()` 是直接将内容写入页面的内容流，但是文档流执行完毕，则它会导致页面全部重绘

- `innerHTML` 是将内容写入某个 DOM 节点，不会导致页面全部重绘
- `innerHTML` 创建多个元素效率更高（不要拼接字符串，采取数组形式拼接），结构稍微复杂
- `createElement()` 创建多个元素效率稍低一点点，但是结构更清晰

总结：不同浏览器下，`innerHTML` 效率要比 `createElement` 高

DOM重点核心

文档对象模型（Document Object Model，简称 `DOM`），是W3C组织推荐的处理可扩展标记语言的标准编程接口

W3C已经定义了一系列的DOM接口，通过这些DOM接口可以改变网页的内容、结构和样式

- 1、对于JavaScript，为了能够使JavaScript操作HTML，JavaScript就有了一套自己的DOM编程接口
- 2、对于HTML，DOM使HTML形成了一个DOM树，包含文档、元素、节点

1.2 DOM 树



对于DOM操作，我们主要针对子元素的操作，主要有

- 创建
- 增
- 删
- 改
- 查
- 属性操作
- 时间操作

创建

1. `document.write`
2. `innerHTML`
3. `createElement`

增

1. appendChild
2. insertBefore

删

1. removeChild

改

- 主要修改dom的元素属性，dom元素的内容、属性、表单的值等
1. 修改元素属性：src、href、title 等
 2. 修改普通元素内容：innerHTML、innerText
 3. 修改表单元素：value、type、disabled
 4. 修改元素样式：style、className

查

- 主要获取查询dom的元素
- 1、DOM提供的API方法：getElementById、getElementsByTagName (古老用法，不推荐)
 - 2、H5提供的新方法：querySelector、querySelectorAll (提倡)
 - 3、利用节点操作获取元素：父(parentNode)、子(children)、兄(previousElementSibling、nextElementSibling) 提倡

属性操作

- 主要针对于自定义属性
1. setAttribute：设置dom的属性值
 2. getAttribute：得到dom的属性值
 3. removeAttribute：移除属性

DOM节点操作小案例