1.1. Web API介绍

1.1.1 API的概念

API（Application Programming Interface，应用程序编程接口）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，无需理解其内部工作机制细节，只需直接调用使用即可。

1.1.2 Web API的概念

Web API 是浏览器提供的一套操作浏览器功能（bom ）和页面元素的 API (BOM 和 DOM)。

现阶段我们主要针对于浏览器讲解常用的 API , 主要针对浏览器做交互效果。比如我们想要浏览器弹出一个警示框， 直接使用 alert('弹出')

MDN 详细 API : https://developer.mozilla.org/zh-CN/docs/Web/API

1.1.3 API 和 Web API 总结

1. API 是为我们程序员提供的一个接口，帮助我们实现某种功能，我们会使用就可以了，不必纠结内部如何实现
2. Web API 主要是针对于浏览器提供的接口，主要针对于浏览器做交互效果。
3. Web API 一般都有输入和输出（函数的传参和返回值），Web API 很多都是方法（函数）

1.2. DOM 介绍

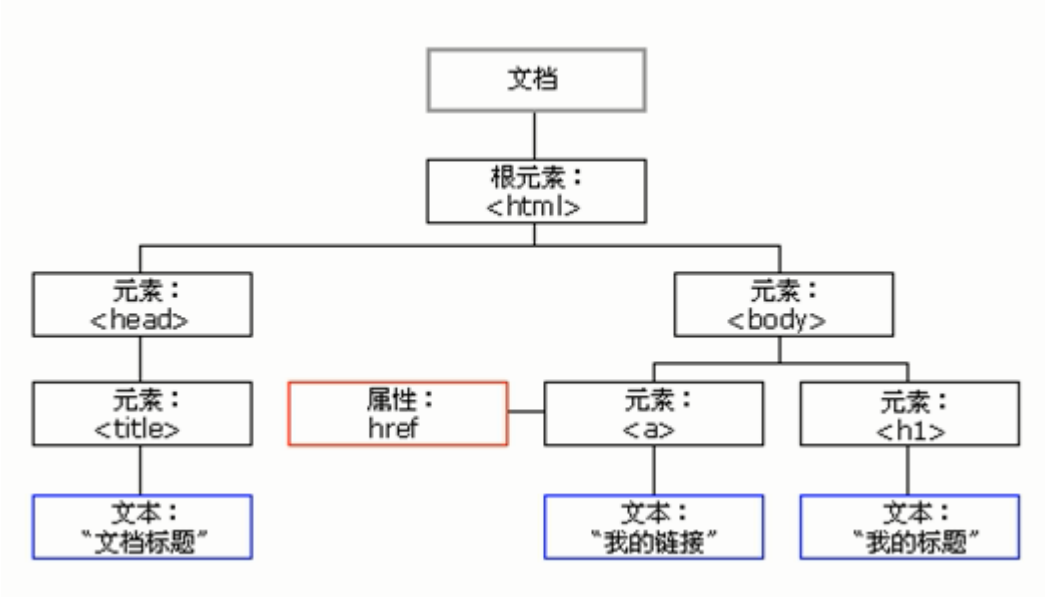
1.2.1 什么是DOM

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xhtml）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。

> DOM是W3C组织制定的一套处理 html和xml文档的规范，所有的浏览器都遵循了这套标准。

1.2.2. DOM树



DOM树 又称为文档树模型，把文档映射成树形结构，通过节点对象对其处理，处理的结果可以加入到当前的页面。

- 文档：一个页面就是一个文档，DOM中使用document表示
- 节点：网页中的所有内容，在文档树中都是节点（标签、属性、文本、注释等），使用node表示
- 标签节点：网页中的所有标签，通常称为元素，又简称为“元素”，使用element表示

DOM 把以上内容都看做是对象

1.3. 获取元素

为什么要获取页面元素？

例如：我们想要操作页面上的某部分(显示/隐藏，动画)，需要先获取到该部分对应的元素，再对其进行操作。

1.3.1. 根据ID获取

语法：`document.getElementById (id)`
作用：根据ID获取元素对象
参数：id值，区分大小写的字符串
返回值：元素对象 或 `null`(就是在获取不到元素的时候就得以null)

```
<div id="dv">2020-08-19</div>
   <script>
       // 1. 因为我们文档页面从上往下加载，所以先得有标签 所以我们script写到标签的下面
       // 2. get 获得 element 元素 by 通过 驼峰命名法
       // 3. 参数 id是大小写敏感的字符串
       // 4. 返回的是一个元素对象
       var dv = document.getElementById('dv');
       console.log(dv);
       console.log(typeof dv);
       // 5. console.dir 打印我们返回的元素对象 更好的查看里面的属性和方法
       console.dir(timer);
```

1.3.2. 根据标签名获取元素

语法：document.getElementsByTagName('标签名') 或者 element.getElementsByTagName('标签名')
作用：根据标签名获取元素对象
参数：标签名
返回值：元素对象集合（伪数组，数组元素是元素对象）

```
    <ul>
       <li>广州太好玩了</li>
       <li>广州太好玩了</li>
       <li>广州太好玩了</li>
       <li>广州太好玩了</li>
   </ul>
   <ol id="ol">
       <li>北京</li>
       <li>上海</li>
       <li>深圳</li>
       <li>广州</li>
   </ol>
   <script>
       // 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的
       var lis = document.getElementsByTagName('li');
       console.log(lis);
       console.log(lis[0]);
       // 2. 我们想要依次打印里面的元素对象我们可以采取遍历的方式
       for (var i = 0; i < lis.length; i++) {
           console.log(lis[i]);
       }
       // 3. 如果页面中只有一个li 返回的还是伪数组的形式
       // 4. 如果页面中没有这个元素 返回的是空的伪数组的形式
       // 5. element.getElementsByTagName('标签名'); 父元素必须是指定的单个元素
       // var ol = document.getElementsByTagName('ol'); // [ol]
       // console.log(ol[0].getElementsByTagName('li'));
       var ol = document.getElementById('ol');
       console.log(ol.getElementsByTagName('li'));
   </script>
```

1.3.3. H5新增获取元素方式

```
1. document.getElementsByClassName('类名'); // 根据类名返回元素对象集合

2. document.querySelector('选择器');          // 根据指定选择器返回第一个元素对象

3. document.querySelectorAll('选择器');      // 根据指定选择器返回
```

```
<body>
    <div class="box">中国</div>
    <div class="box">韩国</div>
    <div id="nav">
        <ul>
            <li>首页</li>
            <li>产品</li>
        </ul>
    </div>
    <script>
        // 1. getElementsByClassName 根据类名获得某些元素集合
        var boxs = document.getElementsByClassName('box');
        console.log(boxs);
        // 2. querySelector 返回指定选择器的第一个元素对象  切记 里面的选择器需要加符号 .box  #nav
        var box = document.querySelector('.box');
        console.log(box);
        var nav = document.querySelector('#nav');
        console.log(nav);
        var li = document.querySelector('li');
        console.log(li);
        // 3. querySelectorAll()返回指定选择器的所有元素对象集合
        var boxs = document.querySelectorAll('.box');
        console.log(boxs);
        var lis = document.querySelectorAll('li');
        console.log(lis);
    </script>
</body>
```

1.3.4 获取特殊元素（body，html）

```
        //获取特殊元素
        // 1.获取body 元素
         var body = document.body;//返回body元素对象
         console.log(body);
         console.dir(body);
         // 2.获取html 元素
         // var html = document.html;//返回html元素对象
         var html = document.documentElement;
         console.log(html);
```

1.4. 事件基础

1.4.1. 事件概述

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。

简单理解： **触发--- 响应机制。**

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。

1.4.2. 事件三要素

- 事件源（谁）：触发事件的元素
- 事件类型（什么事件）： 例如 click 点击事件
- 事件处理程序（做啥）：事件触发后要执行的代码(函数形式)，事件处理函数

```
//事件三要素
<body>
    <button id="btn">在一起</button>
    <script>
        // 点击一个按钮，弹出对话框
        // 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
        //(1) 事件源 事件被触发的对象   谁  按钮
        var btn = document.getElementById('btn');
        //(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
        //(3) 事件处理程序  通过一个函数赋值的方式 完成
        btn.onclick = function() {
            alert('太好了');
        }
    </script>
</body>
```

1.4.3. 执行事件的步骤

```
<body>
    <div>我是一个好人</div>
    <script>
        // 执行事件步骤
```

```
        // 点击div 控制台输出
        // 1. 获取事件源
        var div = document.querySelector('div');
        // 2. 注册事件（绑定事件）
        // div.onclick
        // 3.添加事件处理程序 （采取函数赋值的形式）
        div.onclick = function() {
            console.log('88888');
        }
    </script>
</body>
```

1.4.4. 常见的鼠标事件

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	获得鼠标焦点触发
onblur	失去鼠标焦点触发
onmousemove	鼠标移动触发
onmouseup	鼠标弹起触发
onmousedown	鼠标按下触发

1.5. 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：这些操作都是通过元素对象的属性实现的）

1.5.1. 改变元素内容（获取或设置）

```
element.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

```
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

```
<body>
```

```
    <button>显示当前系统时间</button>
    <div></div>
    <p></p>
    <script>
        // 操作元素之改变元素内容
        // 1. 获取元素
        var btn = document.querySelector('button');
        var div = document.querySelector('div');
        // 2.注册事件
        btn.onclick = function() {
            // div.innerText = '2019-6-6';
            div.innerHTML = getDate();
        }
        function getDate() {
            var date = new Date();
            // 我们写一个 2019年 5月 1日 星期三
            var year = date.getFullYear();
            var month = date.getMonth() + 1;
            var dates = date.getDate();
            var arr = ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期六'];
            var day = date.getDay();
            return '今天是：' + year + '年' + month + '月' + dates + '日 ' + arr[day];
        }
        // 我们元素可以不用添加事件
        var p = document.querySelector('p');
        p.innerHTML = getDate();
    </script>
</body>
```

innerText和innerHTML的区别

- 获取内容时的区别：

  innerText会去除空格和换行，而innerHTML会保留空格和换行

- 设置内容时的区别：

  innerText不会识别html，而innerHTML会识别

```
<body>
    <div></div>
    <p>
        中国人
        <span>国庆</span>
    </p>
    <script>
        // innerText 和 innerHTML的区别
        // 1. innerText 不识别html标签 非标准  去除空格和换行
        var div = document.querySelector('div');
        // div.innerText = '<strong>今天是：</strong> 2019';
        // 2. innerHTML 识别html标签 W3C标准 保留空格和换行的
        div.innerHTML = '<strong>今天是：</strong> 2019';

        // 这两个属性是可读写的  可以获取元素里面的内容
```

```
        var p = document.querySelector('p');
        console.log(p.innerText);
        console.log(p.innerHTML);
    </script>
</body>
```

1.5.2. 常用元素的属性操作

1. innerText、innerHTML 改变元素内容
2. src、href
3. id、alt、title

获取属性的值

> 元素对象.属性名

设置属性的值

> 元素对象.属性名 = 值

```
<body>
    <button id="zhang">张大大</button>
    <button id="li">李小公</button> <br>
    <img src="images/01.jpg" alt="" title="太漂亮了">
    <script>
        // 操作元素之修改元素属性
        // 1. 获取元素
        var zhang = document.getElementById('zhang');
        var li = document.getElementById('li');
        var img = document.querySelector('img');
        // 2. 注册事件  处理程序
        li.onclick = function() {
            img.src = 'images/02.jpg';
            img.title = '太好了';
        }
        zhang.onclick = function() {
            img.src = 'images/02.jpg';
            img.title = '真的很帅';
        }
    </script>
</body>
```

```
    <style>
        img {
            width: 300px;
        }

    </style>
```

```
</head>
<body>
    <img src="images/s.gif" alt="">
    <div>上午好</div>
    <script>
        // 分时问候并显示不同图片
        // 利用多分支语句来设置不同的图片
        // 需要一个图片，并且根据时间修改图片，就需要用到操作元素src属性
        // 需要一个div元素，显示不同问候语，修改元素内容即可
        // 1.获取元素
        var img = document.querySelector('img');
        var div = document.querySelector('div');
        // 2. 得到当前的小时数
        var date = new Date();
        var h = date.getHours();
        // 3. 判断小时数改变图片和文字信息
        if (h < 12) {
            img.src = 'images/s.gif';
            div.innerHTML = '亲，上午好';
        } else if (h < 18) {
            img.src = 'images/x.gif';
            div.innerHTML = '亲，下午好';
        } else {
            img.src = 'images/w.gif';
            div.innerHTML = '亲，晚上好';
        }
    </script>
</body>
```

1.5.4. 表单元素的属性操作

利用 DOM 可以操作如下表单元素的属性：

```
type、value、checked、selected、disabled
```

获取属性的值

> 元素对象.属性名

设置属性的值

> 元素对象.属性名 = 值
>
> 表单元素中有一些属性如：disabled、checked、selected，元素对象的这些属性的值是布尔型。

```
<body>
    <button>按钮</button>
    <input type="text" value="输入内容">
    <script>
```

```
        //操作元素之表单属性设置
        // 1. 获取元素
        var btn = document.querySelector('button');
        var input = document.querySelector('input');
        // 2. 注册事件 处理程序
        btn.onclick = function() {
            // input.innerHTML = '点击了';  这个是 普通盒子 比如 div 标签里面的内容
            // 表单里面的值 文字内容是通过 value 来修改的
            input.value = '被点击了';
            // 如果想要某个表单被禁用 不能再点击 disabled  我们想要这个按钮 button禁用
            // btn.disabled = true;
            this.disabled = true;
            // this 指向的是事件函数的调用者 btn
        }
    </script>
</body>
```

```
    <style>
        .box {
            position: relative;
            width: 400px;
            border-bottom: 1px solid #ccc;
            margin: 100px auto;
        }
        .box input {
            width: 370px;
            height: 30px;
            border: 0;
            outline: none;
        }
        .box img {
            position: absolute;
            top: 2px;
            right: 2px;
            width: 24px;
        }
    </style>
</head>
<body>
    <div class="box">
        <label for="">
            <img src="images/close.png" alt="" id="eye">
        </label>
        <input type="password" name="" id="pwd">
    </div>
    <script>
        //仿京东显示密码
        // 1. 获取元素
        var eye = document.getElementById('eye');
        var pwd = document.getElementById('pwd');
        // 2. 注册事件 处理程序
```

```
        var flag = 0;
        eye.onclick = function() {
            // 点击一次之后， flag 一定要变化
            if (flag == 0) {
                pwd.type = 'text';
                eye.src = 'images/open.png';
                flag = 1; // 赋值操作
            } else {
                pwd.type = 'password';
                eye.src = 'images/close.png';
                flag = 0;
            }
        }
    </script>
</body>
```

1.5.6. 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。

常用方式

```
1. element.style      行内样式操作
2. element.className  类名样式操作
```

方式1：通过操作style属性

> 元素对象的style属性也是一个对象！
>
> 元素对象.style.样式属性 = 值;

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，产生的是行内样式，CSS 权重比较高

```
<body>
    <div></div>
    <script>
        //操作元素之修改样式属性
        // 1. 获取元素
        var div = document.querySelector('div');
        // 2. 注册事件 处理程序
        div.onclick = function() {
            // div.style里面的属性 采取驼峰命名法
            this.style.backgroundColor = 'purple';
            this.style.width = '250px';
```

```
        }
    </script>
</body>
```

```
    <style>
        .box {
            position: relative;
            width: 74px;
            height: 88px;
            border: 1px solid #ccc;
            margin: 100px auto;
            font-size: 12px;
            text-align: center;
            color: #f40;
            /* display: block; */
        }
        .box img {
            width: 60px;
            margin-top: 5px;
        }
        .close-btn {
            position: absolute;
            top: -1px;
            left: -16px;
            width: 14px;
            height: 14px;
            border: 1px solid #ccc;
            line-height: 14px;
            font-family: Arial, Helvetica, sans-serif;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <div class="box">
        淘宝二维码
        <img src="images/tao.png" alt="">
        <i class="closeBtn">x</i>
    </div>
    <script>
        //关闭淘宝二维码
        // 1. 获取元素
        var btn = document.querySelector('.closeBtn');
        var box = document.querySelector('.box');
        // 2.注册事件 程序处理
        btn.onclick = function() {
            box.style.display = 'none';
        }
    </script>
</body>
```

```
    <style>
        * {
            margin: 0;
            padding: 0;
        }
        li {
            list-style-type: none;
        }
        .box {
            width: 250px;
            margin: 100px auto;
        }
        .box li {
            float: left;
            width: 24px;
            height: 24px;
            background-color: pink;
            margin: 15px;
            background: url(images/sprite.png) no-repeat;
        }
    </style>
</head>
<body>
    <div class="box">
        <ul>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
        </ul>
    </div>
    <script>
        //循环精灵图
        // 1. 获取元素 所有的小li
        var lis = document.querySelectorAll('li');
        for (var i = 0; i < lis.length; i++) {
            // 让索引号 乘以 44 就是每个li 的背景y坐标  index就是我们的y坐标
            var index = i * 44;
            lis[i].style.backgroundPosition = '0 -' + index + 'px';
        }
    </script>
</body>
```

```
    <style>
        input {
            color: #999;
        }
    </style>
</head>
<body>
    <input type="text" value="手机">
    <script>
        //显示隐藏文本框内容
        // 1.获取元素
        var input = document.querySelector('input');
        // 2.注册事件 获得焦点事件 onfocus
        input.onfocus = function() {
                // console.log('得到了焦点');
                if (this.value === '手机') {
                    this.value = '';
                }
                // 获得焦点需要把文本框里面的文字颜色变黑
                this.style.color = '#333';
            }
            // 3. 注册事件 失去焦点事件 onblur
        input.onblur = function() {
            // console.log('失去了焦点');
            if (this.value === '') {
                this.value = '手机';
            }
            // 失去焦点需要把文本框里面的文字颜色变浅色
            this.style.color = '#999';
        }
    </script>
</body>
```

方式2：通过操作className属性

> 元素对象.className = 值;
>
> 因为class是关键字，所有使用className。

注意：

1. 如果样式修改较多，可以采取操作类名方式更改元素样式。
2. class因为是个保留字，因此使用className来操作元素类名属性
3. className 会直接更改元素的类名，会覆盖原先的类名。

案例代码

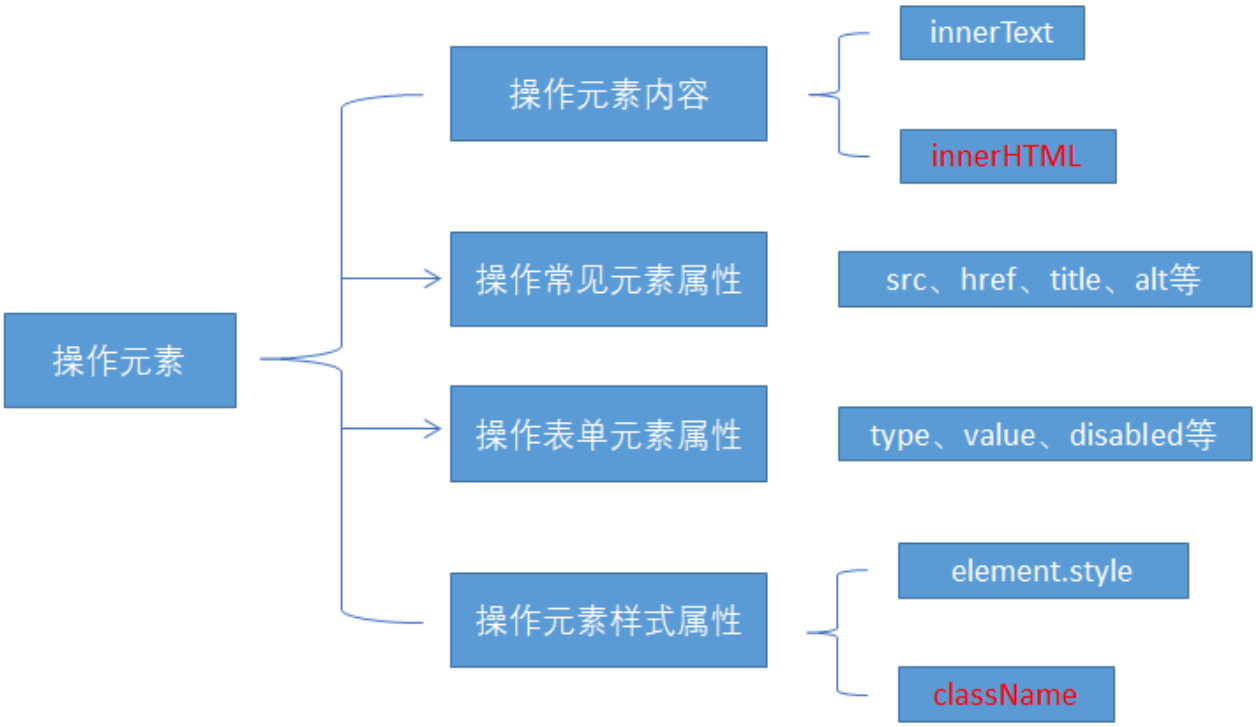
```
 <style>
        div {
            width: 100px;
            height: 100px;
            background-color: pink;
```

```
        }
        .change {
            background-color: purple;
            color: #fff;
            font-size: 25px;
            margin-top: 100px;
        }
    </style>
</head>
<body>
    <div>生活</div>
    <script>
        //通过className更改元素样式
        // 1. 使用 element.style 获得修改元素样式  如果样式比较少 或者 功能简单的情况下使用
        var dv = document.querySelector('div');
        dv.onclick = function() {
            // this.style.backgroundColor = 'purple';
            // this.style.color = '#fff';
            // this.style.fontSize = '25px';
            // this.style.marginTop = '100px';
            // 让我们当前元素的类名改为了 change
            // 2. 我们可以通过 修改元素的className更改元素的样式 适合于样式较多或者功能复杂的情况
            // 3. 如果想要保留原先的类名，我们可以这么做 多类名选择器
            // this.className = 'change';
            this.className = 'first change';
        }
    </script>
```

```
    <style>
        div {
            width: 600px;
            margin: 100px auto;
        }
        .message {
            display: inline-block;
            font-size: 12px;
            color: #999;
            background: url(images/mess.png) no-repeat left center;
            padding-left: 20px;
        }
        .wrong {
            color: red;
            background-image: url(images/wrong.png);
        }
        .right {
            color: green;
            background-image: url(images/right.png);
        }
    </style>
</head>
<body>
```

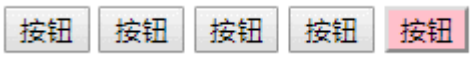
```
    <div class="register">
        <input type="password" class="ipt">
        <p class="message">请输入6~16位密码</p>
    </div>
    <script>
        // 首先判断的事件是表单失去焦点 onblur
        // 如果输入正确则提示正确的信息颜色为绿色小图标变化
        // 如果输入不是6到16位，则提示错误信息颜色为红色 小图标变化
        // 因为里面变化样式较多，我们采取className修改样式
        // 1.获取元素
        var ipt = document.querySelector('.ipt');
        var message = document.querySelector('.message');
        //2. 注册事件 失去焦点
        ipt.onblur = function() {
            // 根据表单里面值的长度 ipt.value.length
            if (this.value.length < 6 || this.value.length > 16) {
                // console.log('错误');
                message.className = 'message wrong';
                message.innerHTML = '您输入的位数不对要求6~16位';
            } else {
                message.className = 'message right';
                message.innerHTML = '您输入的正确';
            }
        }
    </script>
</body>
```

```
<body>
    <button id="btn">开关灯</button>
    <script>
        //修改背景
        var btn = document.getElementById('btn');
        var flag = 0;
        btn.onclick = function() {
            if (flag == 0) {
                document.body.style.backgroundColor = 'black';
                flag = 1;
            } else {
                document.body.style.backgroundColor = '#fff';
                flag = 0;
            }
        }
    </script>
</body>
```



1.1. 排他操作

1.1.1 排他思想



如果有同一组元素，我们想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
2. 给当前元素设置样式

```
    <button>深圳</button>
    <button>深圳</button>
    <button>深圳</button>
    <button>深圳</button>
    <button>深圳</button>
    <script>
        //排他思想(算法)
        // 1. 获取所有按钮元素
        var btns = document.getElementsByTagName('button');
        // btns得到的是伪数组  里面的每一个元素 btns[i]
        for (var i = 0; i < btns.length; i++) {
            btns[i].onclick = function() {
                // (1) 我们先把所有的按钮背景颜色去掉  干掉所有人

                for (var i = 0; i < btns.length; i++) {
```

```
                    btns[i].style.backgroundColor = '';
                }
                // (2) 然后才让当前的元素背景颜色为pink 留下我自己
                this.style.backgroundColor = 'pink';
            }
        }
    </script>
```

```
<style>
        *{
            margin: 0;
            padding: 0;
        }
        body {
            background: url(images/1.jpg) no-repeat center top;
        }
        li {
            list-style: none;
        }
        .baidu {
            overflow: hidden;
            margin: 100px auto;
            background-color: #fff;
            width: 410px;
            padding-top: 3px;
        }
        .baidu li {
            float: left;
            margin: 0 1px;
            cursor: pointer;
        }
        .baidu img {
            width: 100px;
        }
    </style>
</head>
<body>
    <ul class="baidu">
        <li><img src="images/1.jpg"></li>
        <li><img src="images/2.jpg"></li>
        <li><img src="images/3.jpg"></li>
        <li><img src="images/4.jpg"></li>
    </ul>
    <script>
        // 1. 获取元素
        var imgs = document.querySelector('.baidu').querySelectorAll('img');
        // console.log(imgs);
        // 2. 循环注册事件
        for (var i = 0; i < imgs.length; i++) {
            imgs[i].onclick = function() {
                // this.src 就是我们点击图片的路径   images/2.jpg
```

```
                // console.log(this.src);
                // 把这个路径 this.src 给body 就可以了
                document.body.style.backgroundImage = 'url(' + this.src + ')';
            }
        }
    </script>
</body>
```

```
 <style>
        table {
            width: 800px;
            margin: 100px auto;
            text-align: center;
            border-collapse: collapse;
            font-size: 14px;
        }
        thead tr {
            height: 30px;
            background-color: skyblue;
        }
        tbody tr {
            height: 30px;
        }
        tbody td {
            border-bottom: 1px solid #d7d7d7;
            font-size: 12px;
            color: blue;
        }
        .bg {
            background-color: pink;
        }
    </style>
</head>
<body>
    <table>
        <thead>
            <tr>
                <th>学号</th>
                <th>姓名</th>
                <th>数学</th>
                <th>语文</th>
                <th>英语</th>
                <th>音乐</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>001</td>
                <td>何小凡</td>
                <td>98</td>
                <td>96</td>
```

```
                <td>58</td>
                <td>96</td>
            </tr>
            <tr>
                <td>002</td>
                <td>叶小风</td>
                <td>98</td>
                <td>96</td>
                <td>58</td>
                <td>96</td>
            </tr>
            <tr>
                <td>003</td>
                <td>郑成成</td>
                <td>38</td>
                <td>76</td>
                <td>58</td>
                <td>96</td>
            </tr>
            <tr>
                <td>004</td>
                <td>张小气</td>
                <td>98</td>
                <td>69</td>
                <td>58</td>
                <td>96</td>
            </tr>
            <tr>
                <td>005</td>
                <td>李芬</td>
                <td>98</td>
                <td>35</td>
                <td>58</td>
                <td>98</td>
            </tr>
            <tr>
                <td>006</td>
                <td>郑小小</td>
                <td>81</td>
                <td>96</td>
                <td>93</td>
                <td>96</td>
            </tr>
        </tbody>
    </table>
    <script>
        //表格隔行变色
        // 1.获取元素 获取的是 tbody 里面所有的行
        var trs = document.querySelector('tbody').querySelectorAll('tr');
        // 2. 利用循环绑定注册事件
        for (var i = 0; i < trs.length; i++) {
            // 3. 鼠标经过事件 onmouseover

            trs[i].onmouseover = function() {
```

```
                    // console.log(11);
                    this.className = 'bg';
                }
                // 4. 鼠标离开事件 onmouseout
            trs[i].onmouseout = function() {
                this.className = '';
            }
        }
    </script>
</body>
```

```
    <body>
     <div class="wrap">
         <table>
             <thead>
                 <tr>
                     <th>
                         <input type="checkbox" id="all" />
                     </th>
                     <th>商品</th>
                     <th>价钱</th>
                 </tr>
             </thead>
             <tbody id="tabs">
                 <tr>
                     <td>
                         <input type="checkbox" />
                     </td>
                     <td>电视机</td>
                     <td>8000</td>
                 </tr>
                 <tr>
                     <td>
                         <input type="checkbox" />
                     </td>
                     <td>手机</td>
                     <td>5000</td>
                 </tr>
                 <tr>
                     <td>
                         <input type="checkbox" />
                     </td>
                     <td>电脑</td>
                     <td>2000</td>
                 </tr>
                 <tr>
                     <td>
                         <input type="checkbox" />
                     </td>
                     <td>电动车</td>
                     <td>2000</td>
```

```
                </tr>
            </tbody>
        </table>
    </div>
    <script>
        // 1. 全选和取消全选做法：  让下面所有复选框的checked属性（选中状态） 跟随 全选按钮即可
        // 获取元素
        var all = document.getElementById('all'); // 全选按钮
        var tabs = document.getElementById('tabs').getElementsByTagName('input'); // 下面所有的复选框
        // 注册事件
        all.onclick = function() {
                // this.checked 它可以得到当前复选框的选中状态如果是true 就是选中，如果是false 就是未选中
                console.log(this.checked);
                for (var i = 0; i < tabs.length; i++) {
                    tabs[i].checked = this.checked;
                }
            }
            // 2. 下面复选框需要全部选中， 上面全选才能选中做法： 给下面所有复选框绑定点击事件，每次点击，都要循环查看下面所有的复选框是否有没选中的，如果有一个没选中的， 上面全选就不选中。
        for (var i = 0; i < tabs.length; i++) {
            tabs[i].onclick = function() {
                // flag 控制全选按钮是否选中
                var flag = true;
                // 每次点击下面的复选框都要循环检查者4个小按钮是否全被选中
                for (var i = 0; i < tabs.length; i++) {
                    if (!tabs[i].checked) {
                        flag = false;
                        break; // 退出for循环 这样可以提高执行效率 因为只要有一个没有选中，剩下的就无需循环判断了
                    }
                }
                all.checked = flag;
            }
        }
    </script>
</body>
```

1.5. 自定义属性操作

1.5.1 获取属性值

- element.属性　获取属性值。
- element.getAttribute('属性');

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');　主要获得自定义的属性　（标准）　我们程序员自定义的属性

```
<body>
    <div id="demo" index="1" class="nav"></div>
    <script>
        //自定义属性
        var div = document.querySelector('div');
        // 1. 获取元素的属性值
        // (1) element.属性
        console.log(div.id);
        //(2) element.getAttribute('属性')  get得到获取 attribute 属性的意思 我们程序员自己添加的属性我们称为自定义属性 index
        console.log(div.getAttribute('id'));
        console.log(div.getAttribute('index'));

    </script>
</body>
```

1.5.2. 设置属性值

2. 设置属性值

- element.属性 = '值'　设置内置属性值。
- element.setAttribute('属性', '值');

区别：

- element.属性　设置内置属性值
- element.setAttribute('属性');　主要设置自定义的属性　（标准）

```
    // 2. 设置元素属性值
        // (1) element.属性= '值'
        div.id = 'test';
        div.className = 'navs';
        // (2) element.setAttribute('属性', '值');  主要针对于自定义属性
        div.setAttribute('index', 2);
        div.setAttribute('class', 'footer'); // class 特殊  这里面写的就是class 不是className
        // 3 移除属性 removeAttribute(属性)
        div.removeAttribute('index');
```

1.5.3. 移出属性

- element.removeAttribute('属性');

```
        // class 不是className
        // 3 移除属性 removeAttribute(属性)
        div.removeAttribute('index');
```

```
<style>
        * {
            margin: 0;
            padding: 0;
        }

        li {
            list-style-type: none;
        }

        .tab {
            width: 978px;
            margin: 100px auto;
        }

        .tab_list {
            height: 39px;
            border: 1px solid #ccc;
            background-color: #f1f1f1;
        }

        .tab_list li {
            float: left;
            height: 39px;
            line-height: 39px;
            padding: 0 20px;
            text-align: center;

            cursor: pointer;
```

```
        }

        .tab_list .current {
            background-color: #c81623;
            color: #fff;
        }

        .item_info {
            padding: 20px 0 0 20px;
        }

        .item {
            display: none;
        }
    </style>
</head>
<body>
    <div class="tab">
        <div class="tab_list">
            <ul>
                <li class="current">商品介绍</li>
                <li>规格与包装</li>
                <li>售后保障</li>
                <li>商品评价（50000）</li>
                <li>手机社区</li>
            </ul>
        </div>
        <div class="tab_con">
            <div class="item" style="display: block;">
                商品介绍模块内容
            </div>
            <div class="item">
                规格与包装模块内容
            </div>
            <div class="item">
                售后保障模块内容
            </div>
            <div class="item">
                商品评价（50000）模块内容
            </div>
            <div class="item">
                手机社区模块内容
            </div>
        </div>
    </div>
    <script>
        // 获取元素
        var tab_list = document.querySelector('.tab_list');
        var lis = tab_list.querySelectorAll('li');
        var items = document.querySelectorAll('.item');
        // for循环绑定点击事件
        for (var i = 0; i < lis.length; i++) {

            // 开始给5个小li 设置索引号
```

```
            lis[i].setAttribute('index', i);
            lis[i].onclick = function() {
                // 1. 上的模块选项卡，点击某一个，当前这一个底色会是红色，其余不变（排他思想） 修改类名的方式
                //  其余的li清除 class 这个类
                for (var i = 0; i < lis.length; i++) {
                    lis[i].className = '';
                }
               //留下自己
                this.className = 'current';
                // 2. 下面的显示内容模块
                var index = this.getAttribute('index');
                console.log(index);
                // 干掉所有人 让其余的item 这些div 隐藏
                for (var i = 0; i < items.length; i++) {
                    items[i].style.display = 'none';
                }
                // 留下我自己 让对应的item 显示出来
                items[index].style.display = 'block';
            }
        }
    </script>
```

1.5.5. H5自定义属性

自定义属性目的：是为了保存并使用数据。有些数据可以保存到页面中而不用保存到数据库中。

自定义属性获取是通过getAttribute('属性') 获取。

但是有些自定义属性很容易引起歧义，不容易判断是元素的内置属性还是自定义属性。

H5给我们新增了自定义属性：

1. 设置H5自定义属性

H5规定自定义属性data-开头做为属性名并且赋值。

比如

或者使用 JS 设置

element.setAttribute('data-index',2)

2. 获取H5自定义属性

1. 兼容性获取 element.getAttribute('data-index');
2. H5新增 element.dataset.index 或者 element.dataset['index'] ie 11才开始支持

```
<body>
```

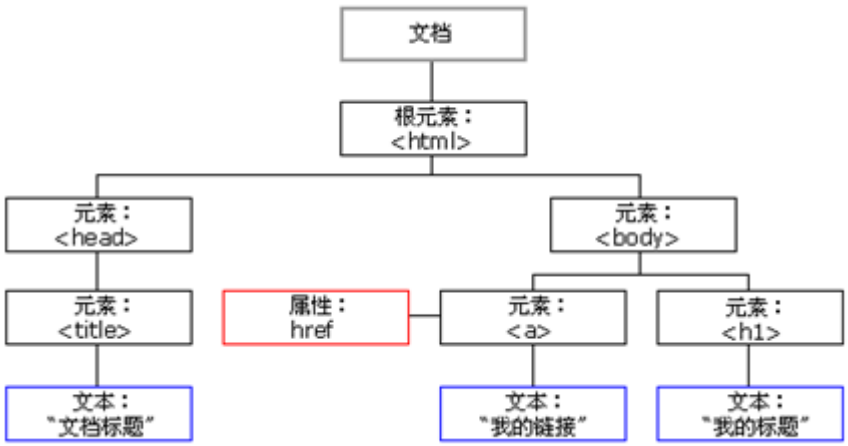
```
    <div getTime="20" data-index="2" data-list-name="andy"></div>
    <script>
        var div = document.querySelector('div');
        // console.log(div.getTime);
        console.log(div.getAttribute('getTime'));
        div.setAttribute('data-time', 20);
        console.log(div.getAttribute('data-index'));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

1.6. 节点操作

1.6.1. 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。



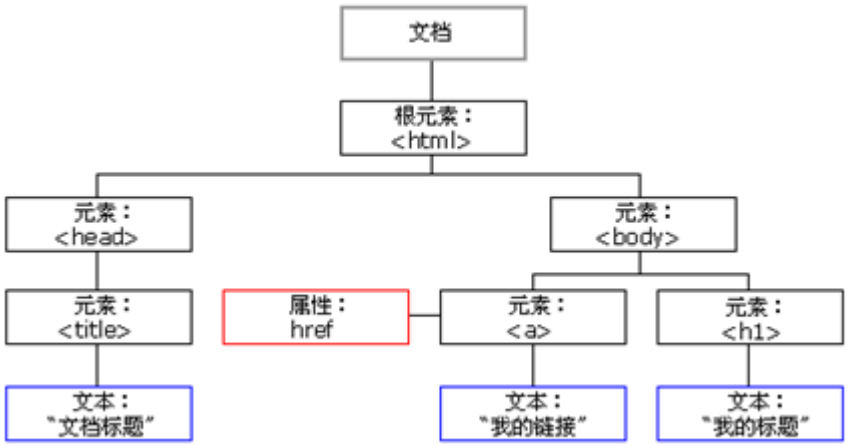
一般地，节点至少拥有nodeType（节点类型）、nodeName（节点名称）和nodeValue（节点值）这三个基本属性。

- 元素节点 nodeType 为 1
- 属性节点 nodeType 为 2
- 文本节点 nodeType 为 3 （文本节点包含文字、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**

1.6.2. 节点层级

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**。




```
<body>
    <!-- 节点的优点 -->
    <div>我是div</div>
    <span>我是span</span>
    <ul>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
    </ul>
    <div class="box">
        <span class="erweima">×</span>
    </div>
    <script>
        var box = document.querySelector('.box');
        console.dir(box);
    </script>
</body>
```

1.6.3. 父级节点

```
node.parentNode
```

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

```
<body>
    <!-- 节点的优点 -->
    <div>我是div</div>
    <span>我是span</span>
    <ul>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
    </ul>
    <div class="demo">
        <div class="box">
            <span class="erweima">×</span>
        </div>
    </div>
    <script>
         //父节点操作
        // 1. 父节点 parentNode
        var erweima = document.querySelector('.erweima');
        // var box = document.querySelector('.box');
        // 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
        console.log(erweima.parentNode);
    </script>
</body>
```

1.6.4. 子节点

所有子节点

```
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。

注意：返回值里面包含了所有的子节点，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

子元素节点

2. 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

```
<body>
    <!-- 节点的优点 -->
    <div>我是div</div>
    <span>我是span</span>
    <ul>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
    </ul>
    <ol>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
        <li>我是li</li>
    </ol>
    <div class="demo">
        <div class="box">
            <span class="erweima">×</span>
        </div>
    </div>
    <script>
        // DOM 提供的方法（API）获取
        var ul = document.querySelector('ul');
        var lis = ul.querySelectorAll('li');
        // 1. 子节点  childNodes 所有的子节点 包含 元素节点 文本节点等等
        console.log(ul.childNodes);
        console.log(ul.childNodes[0].nodeType);
        console.log(ul.childNodes[1].nodeType);
        // 2. children 获取所有的子元素节点 也是我们实际开发常用的
        console.log(ul.children);
    </script>
</body>
```

第1个子节点

3. 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。

最后1个子节点

4. 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。

第1个子元素节点

5. parentNode.firstElementChild

firstElementChild 返回第一个子元素节点，找不到则返回null。

最后1个子元素节点

6. parentNode.lastElementChild

lastElementChild返回最后一个子元素节点，找不到则返回null。

注意：这两个方法有兼容性问题，IE9 以上才支持。

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？

解决方案：

1. 如果想要第一个子元素节点，可以使用parentNode.children[0]
2. 如果想要最后一个子元素节点，可以使用parentNode.children[parentNode.children.length - 1]

```
<body>
    <ol>
        <li>我是li1</li>
        <li>我是li2</li>
        <li>我是li3</li>
        <li>我是li4</li>
        <li>我是li5</li>
    </ol>
    <script>
    //子节点-第一个子元素和最后一个子元素
        var ol = document.querySelector('ol');
        // 1. firstChild 第一个子节点 不管是文本节点还是元素节点
        console.log(ol.firstChild);
        console.log(ol.lastChild);
        // 2. firstElementChild 返回第一个子元素节点 ie9才支持
        console.log(ol.firstElementChild);
        console.log(ol.lastElementChild);
        // 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
        console.log(ol.children[0]);
        console.log(ol.children[ol.children.length - 1]);
    </script>
```

```
</body>
```

```
<style>
    * {
        margin: 0;
        padding: 0;
    }
    li {
        list-style-type: none;
    }
    a {
        text-decoration: none;
        font-size: 14px;
    }
    .nav {
        margin: 100px;
    }
    .nav>li {
        position: relative;
        float: left;
        width: 80px;
        height: 41px;
        text-align: center;
    }
    .nav li a {
        display: block;
        width: 100%;
        height: 100%;
        line-height: 41px;
        color: #333;
    }
    .nav>li>a:hover {
        background-color: #eee;
    }
    .nav ul {
        display: none;
        position: absolute;
        top: 41px;
        left: 0;
        width: 100%;
        border-left: 1px solid #FECC5B;
        border-right: 1px solid #FECC5B;
    }

    .nav ul li {
        border-bottom: 1px solid #FECC5B;
    }

    .nav ul li a:hover {

        background-color: #FFF5DA;
```

```
        }
    </style>
</head>
<body>
    <ul class="nav">
        <li>
            <a href="#">微博</a>
            <ul>
                <li>
                    <a href="">私信</a>
                </li>
                <li>
                    <a href="">评论</a>
                </li>
                <li>
                    <a href="">@我</a>
                </li>
            </ul>
        </li>
        <li>
            <a href="#">微博</a>
            <ul>
                <li>
                    <a href="">私信</a>
                </li>
                <li>
                    <a href="">评论</a>
                </li>
                <li>
                    <a href="">@我</a>
                </li>
            </ul>
        </li>
        <li>
            <a href="#">微博</a>
            <ul>
                <li>
                    <a href="">私信</a>
                </li>
                <li>
                    <a href="">评论</a>
                </li>
                <li>
                    <a href="">@我</a>
                </li>
            </ul>
        </li>
        <li>
            <a href="#">微博</a>
            <ul>
                <li>
                    <a href="">私信</a>

                </li>
```

```
                <li>
                    <a href="">评论</a>
                </li>
                <li>
                    <a href="">@我</a>
                </li>
            </ul>
        </li>
    </ul>
    <script>
         //新浪下拉菜单
        // 1. 获取元素
        var nav = document.querySelector('.nav');
        var lis = nav.children; // 得到4个小li
        // 2.循环注册事件
        for (var i = 0; i < lis.length; i++) {
            lis[i].onmouseover = function() {
                this.children[1].style.display = 'block';
            }
            lis[i].onmouseout = function() {
                this.children[1].style.display = 'none';
            }
        }
    </script>
</body>
```

1.6.6. 兄弟节点

下一个兄弟节点

```
nextSibling
```

上一个兄弟节点

```
previousSibling
```

```
<body>
    <div>我是div</div>
    <span>我是span</span>
    <script>
    //兄弟节点
        var div = document.querySelector('div');
        // 1.nextSibling 下一个兄弟节点 包含元素节点或者 文本节点等等
        console.log(div.nextSibling);
        console.log(div.previousSibling);
        // 2. nextElementSibling 得到下一个兄弟元素节点
        console.log(div.nextElementSibling);
        console.log(div.previousElementSibling);
    </script>
</body>
```

下一个兄弟元素节点（有兼容性问题）

```
nextSibling
```

上一个兄弟元素节点（有兼容性问题）

```

```

```
function getNextElementSibling(element) {
   var el = element;
   while (el = el.nextSibling) {
     if (el.nodeType === 1) {
         return el;
     }
   }
   return null;
 }
```

1.6.7. 创建节点

```
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。

1.6.8. 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。

```
2. node.insertBefore(child, 指定元素)
```

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。

```
<ul>
    <li>123</li>
</ul>
```

```
    <script>
        // 1. 创建节点元素节点
        var li = document.createElement('li');
        // 2. 添加节点 node.appendChild(child)  node 父级  child 是子级 后面追加元素
        var ul = document.querySelector('ul');
        ul.appendChild(li);
        // 3. 添加节点 node.insertBefore(child, 指定元素);
        var lili = document.createElement('li');
        ul.insertBefore(lili, ul.children[0]);
        // 4. 我们想要页面添加一个新的元素 ： 1. 创建元素 2. 添加元素
    </script>
```

1.6.9. 案例：简单版发布留言

```
<style>
        * {
            margin: 0;
            padding: 0;
        }
        body {
            padding: 100px;
        }
        textarea {
            width: 200px;
            height: 100px;
            border: 1px solid pink;
            outline: none;
            resize: none;
        }
        ul {
            margin-top: 50px;
        }
        li {
            width: 300px;
            padding: 5px;
            background-color: rgb(245, 209, 243);
            color: red;
            font-size: 14px;
            margin: 15px 0;
        }
    </style>
</head>
<body>
    <textarea name="" id=""></textarea>
    <button>发布</button>
    <ul>
    </ul>
    <script>
        // 1. 获取元素
        var btn = document.querySelector('button');
        var text = document.querySelector('textarea');

        var ul = document.querySelector('ul');
```

```
        // 2. 注册事件
        btn.onclick = function() {
            if (text.value == '') {
                alert('您没有输入内容');
                return false;
            } else {
                // console.log(text.value);
                // (1) 创建元素
                var li = document.createElement('li');
                // 先有li 才能赋值
                li.innerHTML = text.value;
                // (2) 添加元素
                // ul.appendChild(li);
                ul.insertBefore(li, ul.children[0]);
            }
        }
    </script>
</body>
```