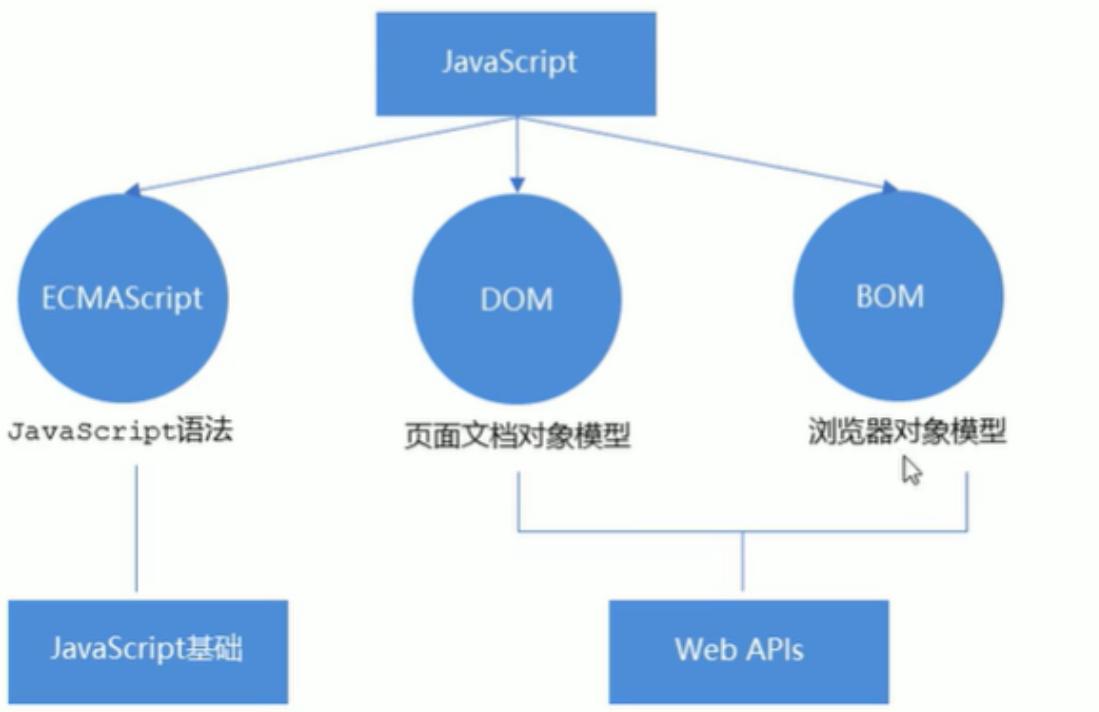


1. Web APIs和JS基础关联性

1.1 JS的组成



1.2 JS基础阶段以及Web APIs阶段

JS基础阶段

- 学习的是ECMAScript标准规定的基本语法
- 要求掌握JS基础语法
- 只学习基本语法，做不了常用的网页交互效果
- 目的是为了JS后面的课程打基础、做铺垫

Web APIs阶段

- web APIs是W3C组织的标准
- Web APIs我们主要学习DOM和BOM
- web APIs是我们JS所独有的部分
- 我们主要学习页面交互功能
- 需要使用JS基础的课程内容做基础

JS基础学习ECMAScript 基础语法为后面作铺垫，web APIs是JS的应用,大量使用JS基础语法做交互效果

2.API和Web API

2.1 API

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, 或理解内部工作机制的细节。

简单理解: API是给程序员提供的一种工具, 以便能更轻松的实现想要完成的功能。

2.2 Web API

Web API是浏览器提供的一套操作浏览器功能和页面元素的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>

因为Web API很多, 所以我们将这个阶段称为Web APIs

2.3 API和Web API总结

1. API是为我们提供的一个接口, 帮助我们实现某种功能, 我们会使用且最好能知道内部如何实现

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)

4. 学习Web API可以结合前面学习内置对象方法的思路学习

3.DOM简介

3.1 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。

3.2 DOM树



• 文档: 一个页面就是一个文档, DOM中使用document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

DOM把以上内容都看做是对象

4. 获取元素

4.1 如何获取页面元素

DOM在我们实际开发中主要用来操作元素。

我们如何来获取页面中的元素呢？

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HTML新增的方法获取
- 特殊元素获取

4.2 根据ID获取元素

访问 HTML 元素最常用的方法是使用元素的 id，**getElementsByID 方法**

语法：

```
document.getElementById()
```

```
//注意点：  
//1.因为文档页面是从上往下加载，所以需要将script写到标签后面  
//2.getElementById() By通过 直译：通过id获取元素，使用驼峰命名法  
//3.参数里面填写id，注意id是大小写敏感的字符串  
//4.返回的是一个元素对象  
var timer = document.getElementById('time');  
console.log(timer);  
console.log(typeof timer);  
//5.使用console.dir打印返回的元素对象，可以更好的查看里面的属性和方法  
console.dir(timer);
```

```
<div id="time">2021-4-23</div>  
object  
  ▼ div#time ⓘ  
    accessKey: ""  
    align: ""  
    ariaAtomic: null  
    ariaAutoComplete: null  
    ...
```

注意：

1. 必须以document开头调用方法，这是规定
2. 参数里面填写id，注意id是大小敏感的字符串
3. 返回的是一个元素对象

4.3 根据标签名获取

1. 使用getElementsByName()方法可以返回带有指定标签名的**对象的集合**。

```
element.getElementsByTagName ('标签名');
```

注意:

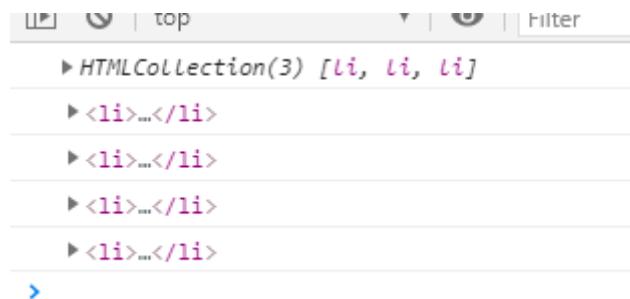
1.因为得到的是一个对象的集合,所以我们想要操作里面的元素就需要遍历。

2.得到元素对象是动态的,会随着li里的内容变化而变化

3.标签名需要是字符串

示例:

```
<ul>
  <li>啦啦啦</li>
  <li>哈哈哈</li>
  <li>呼呼呼</li>
</ul>
<script>
  //1.返回的是获取元素对象的集合，并以伪数组的形式存储
  var lis = document.getElementsByTagName('li');
  console.log(lis);
  console.log(lis[0]); //可以使用索引获取数据
  //2.可以通过遍历依次打印里面的元素
  for (var i = 0; i < lis.length; i++) {
    console.log(lis[i]);
  }
</script>
```



当li只剩一个时

```
> HTMLCollection [li]
```

当没有一个li时

```
> HTMLCollection []
```

返回结果都以伪数组形式存储,没有li时返回的是空的伪数组

2.还可以获取某个元素(父元素)内部所有指定标签名的子元素.

```
element.getElementsByTagName ('标签名');
```

注意:父元素必须是单个对象(必须指明是哪一个元素对象),获取的时候不包括父元素自己.

```
<ol id='ol'>
    <li>11</li>
    <li>22</li>
</ol>

//只想获取ol下的li
var ol = document.getElementById('ol');
console.log(ol.getElementsByTagName('li'));//获取父元素ol下的所有li元素
```

▶ HTMLCollection(2) [li, li]

注意：错误写法

```
var ol = document.getElementsByTagName('ol');//[ol]
console.log(ol.getElementsByTagName('li'));
```

✖ ▶ Uncaught TypeError: ol.getElementsByTagName is not a function
at 02-根据标签获取元素.html:36

▶

因为此时的ol是伪数组，仍是对象的集合，即使只有一个元素，**并不是单个对象**，因此需要根据id获取返回单个ol对象，从而选择ol(父元素)下的li

小结：

- 1.根据**标签名（字符串形式）**获取返回的是**对象的集合**，并以伪数组的形式存储
- 2.可以通过**遍历**依次打印里面的元素
- 3.如果页面中只有一个li时，返回的还是伪数组形式
- 4.如果页面中没有这个元素，返回的是一个空的伪数组
- 5.如果想要获得特定父元素下的元素，可结合**ID**获得**单个对象**，再通过**标签名获取**

4.4通过HTML5新增的方法获取

1.语法：

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

示例：

```
<div class="box">盒子1</div>
<div class="box">盒子2</div>
<ul id="nav">
    <li>1</li>
    <li>2</li>
</ul>
```

```
//1.getElementsByClassName('类名') 根据类名返回元素对象集合
var boxes = document.getElementsByClassName('box');
console.log(boxes);
```

```
▶ HTMLCollection(2) [div.box, div.box]
```

2.语法:

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

示例:

```
//2.querySelector('选择器') 根据指定选择器返回第一个元素对象,注意里面的类和id选择器要加符号 .box #nav
var firstBox = document.querySelector('.box');
console.log(firstBox);
var nav = document.querySelector('#nav')
console.log(nav);
var li = document.querySelector('li');
console.log(li);
```

```
<div class="box">盒子1</div>
```

```
▶ <ul id="nav">...</ul>
```

```
▶ <li>...</li>
```

3.语法:

```
document.querySelectorAll('选择器');//根据指定选择器返回所有元素对象集合
```

示例:

```
//3.querySelectorAll('选择器') 根据指定选择器返回所有元素对象集合
var allBox = document.querySelectorAll('.box');
console.log(allBox);
```

```
▶ NodeList(2) [div.box, div.box]
```

```
>
```

小结:

1.getElementsByName('类名')与querySelectorAll('选择器')方法效果类似, 返回的都是指定**元素对象集合**

2.querySelector ('选择器')与querySelectorAll('选择器')都是根据选择器来返回的, 注意**选择里面的类和id选择器时要加符号 .box #nav**

4.5获取特殊元素(body , html)

获取body元素

```
document.body // 返回body元素对象
```

获取html元素

```
document.documentElement // 返回html元素对象
```

示例:

```
<script>
    //获取body标签
    var bodyEle = document.body;
    console.log(bodyEle);

    //获取html标签
    var htmlEle = document.documentElement;
    console.log(htmlEle);
</script>
```

▶ <body>...</body>

<html lang="en">
▶ <head>...</head>
▶ <body>...</body>
</html>

➤ |

5.事件基础

5.1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。

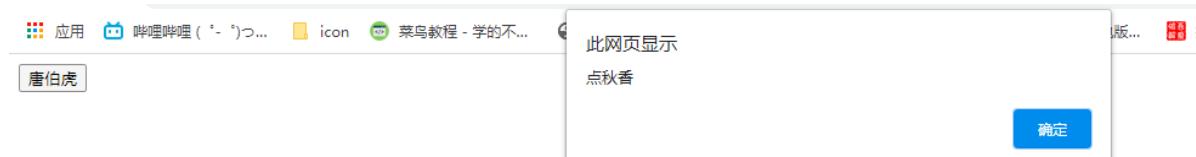
5.2事件三要素

事件由三部分组成: **事件源** (谁) 、 **事件类型** (如何触发) 、 **事件处理程序** (通过赋值完成, 如匿名函数赋值), 又称为**事件三要素**

示例:

```
<button id='btn'>唐伯虎</button>
<script>
    //事件三要素: 事件源、事件类型、事件处理程序
    //1)事件源 谁 获取元素
    var btn = document.getElementById('btn');
    //2)事件类型 如何触发, 如鼠标点击、鼠标经过、键盘按下
    //3)事件处理过程 匿名函数赋值完成
    btn.onclick = function () {
        alert('点秋香');
    }
</script>
```

结果: 鼠标点击按钮后弹出提示框



5.3执行事件的步骤

1.获取事件源

2.注册事件(绑定事件)即添加事件

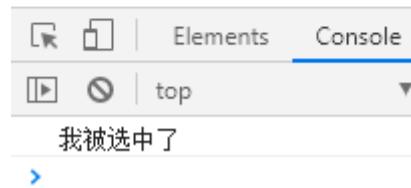
3.添加事件处理程序(采取函数赋值形式)

示例：

```
<div>123</div>
<script>
    //执行事件步骤
    //点击div， 控制台输出'我被选中了'
    //1. 获取事件源
    var div = document.querySelector('div');
    //2. 绑定事件 注册事件
    //div.onclick
    //3. 添加事件处理程序
    div.onclick = function () {
        console.log('我被选中了');
    }
</script>
```



123



5.4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

6.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

6.1改变元素内容

1.语法:

```
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

2.语法:

```
element.innerHTML 更常用
```

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**

共同点:

两者都可以读取和修改元素里面的内容, 读取即上面语法的格式, 修改通过赋值进行修改

案例:

为了美观,可以在样式里设置图片的大小

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

点击按钮显示系统时间,结合之前格式化日期年月日星期

```
<button>显示系统时间</button>
<div>某个时间</div>
<p></p>
<script>
    // 1.事件源
    var btn = document.querySelector('button');
    var div = document.querySelector('div');
    //2.注册事件
    btn.onclick = function () {
        div.innerText = getDate();
    }

    function getDate() {
        var now = new Date();

        //输入2021年3月2日 星期几格式
        year = now.getFullYear();
        month = now.getMonth() + 1;//注意月份+1
        date = now.getDate();
```

```

//由于星期输出的是数字，可以利用数组索引号方式输出文字，注意将周日放在最前面，因为
周日数字为0
var week = ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星
期六'];
day = now.getDay()
return '今天是: ' + year + '年' + month + '月' + date + '日 ' +
week[day];
}

//元素不添加事件，直接修改显示
var p = document.querySelector('p')
p.innerText = getDate();
</script>

```

结果：

某个时间

今天是：2021年3月16日 星期二

点击按钮后：

今天是：2021年3月16日 星期二
今天是：2021年3月16日 星期二

注意：也可以不添加事件，实现刷新就显示系统时间

示例：两者的区别

1.是否识别html标签

```

// 1.innerText不识别html标签
var div = document.querySelector('div');
div.innerText = '<strong>今天是:</strong>2021';

```

今天是:2021

```

// 2.innerHTML识别html标签
div.innerHTML='<strong>今天是:</strong>2021';

```

今天是:2021

2.两者读取元素里面内容的区别

```

//两个属性都是可读写的，可以获取元素里面的内容
var p = document.querySelector('p');
console.log(p.innerText);
console.log(p.innerHTML);

```

我是文字 123

我是文字
123

innerText去除空格和换行, innerHTML保留空格和换行

6.2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

通过案例来了解元素的属性操作

6.2.1案例——图片切换

案例：默认显示刘德华图片，并鼠标经过时显示“刘德华”；点击“张学友”按钮时，显示张学友图片，且鼠标经过时显示“张学友”，点击“刘德华”能显示刘德华的照片



```
<button id='ldh'>刘德华</button>
<button id='zxy'>张学友</button><br>


<script>
    //修改元素属性 src
    // 1. 获取事件源
    var ldh = document.getElementById('ldh');
    var zxy = document.getElementById('zxy');
    var img = document.querySelector('img');
    //2. 注册事件 处理程序
    zxy.onclick = function () {
        img.src = 'images/zxy.jpg';
        img.title = '张学友';
    }
    ldh.onclick = function () {
        img.src = 'images/ldh.jpg';
        img.title = '刘德华';
    }
</script>
```

刘德华 张学友



点击“张学友”后

刘德华 张学友



点击“刘德华”后能返回

6.2.2案例——分时显示不同图片和文字

案例:

根据不同时间，页面显示不同图片,同时显示不同的问候语。

如果上午时间打开页面,显示上午好,显示上午的图片。

如果下午时间打开页面,显示下午好,显示下午的图片。

如果晚上时间打开页面，显示晚上好,显示晚上的图片。

案例分析

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利用多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,就需要用到操作元素src属性

④需要一个div元素，显示不同问候语，修改元素内容即可

```

<div>早上好</div>
<script>
//1. 获取事件源
var img = document.querySelector('img');
var div = document.querySelector('div');
```

```
//2.注册事件 处理程序  
//获得系统时间的小时  
var date = new Date();  
var h = date.getHours();  
//根据小时判断早下晚  
if (h < 12) {  
    img.src = 'images/z.jpg';  
    div.innerHTML = '早上好';  
} else if (h < 18) {  
    img.src = 'images/x.gif';  
    div.innerHTML = '下午好';  
} else {  
    img.src = 'images/w.gif';  
    div.innerHTML = '晚上好';  
}  
</script>
```

早上:



早上好

下午:



下午好

晚上:

6.3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

示例：

```
<button>按钮</button>
<input type="text" value="请输入内容">
<script>
    // 1. 获取事件源
    var btn = document.querySelector('button');
    var input = document.querySelector('input')
    // 2. 注册事件 事件处理
    btn.onclick = function () {
        //innerHTML是普通盒子才有的，如div
        //input.innerHTML='被点击了';
        //表单里的文字内容通过value来修改
        input.value = '被点击了';

        //想要按钮使用一次后就禁用，通过修改disabled实现
        // btn.disabled = true;
        this.disabled = true;//上面语句的改进，更简单
        //this 指的是事件函数的调用者 btn
    }
</script>
```

按钮 请输入内容

鼠标点击按钮后：按钮被禁用

按钮 被点击了

注意：

1.innerHTML是普通盒子才有的，如div； 表单里的文字内容通过value来修改

2.想要按钮使用一次后就禁用，通过修改disabled实现

this.disabled = true;//this 指的是事件函数的调用者 btn

案例：仿京东显示密码

点击按钮将密码框切换为文本框，并可以查看密码明文。



案例分析

①核心思路:点击眼睛按钮,把密码框类型改为文本框就可以看见里面的密码

②一个按钮两个状态,点击一次, 切换为文本框,继续点击一次切换为密码框

③算法:利用一个**flag**变量,通过判断**flag**的值切换密码框和文本框 (常用算法) ,如果是1就切换为文本框, flag设置为0,如果是0就切换为密码框, flag设置为1

此次案例将html、css也用上, 并了解三个是如何配合使用的

```
<style>
    .box {
        position: relative;
        width: 300px;
        border-bottom: 1px solid #ccc;
        margin: 100px auto;
    }

    .box input {
        width: 270px;
        height: 30px;
        border: 0;
        outline: none;
    }

    .box img {
        position: absolute;
        top: 2px;
        right: 2px;
        width: 24px;
        height: 24px;
    }
</style>
```

```
<div class="box">
    
    <input type="password" name="" id="pwd">
</div>
<script>
    // 1.获取事件源 由于点击图片会更改图片及文本框属性, 所以有两个事件源
    var eye = document.getElementById('eye');
    var pwd = document.getElementById('pwd');

    //2.注册事件 处理程序
    var flag = 0; //通过flag变量来切换文本框和密码框, 默认0为密码框
    eye.onclick = function () {
        if (flag == 0) {
            pwd.type = 'text';
            flag = 1;
            eye.src = 'images/open.png';
        } else {
            pwd.type = 'password';
            flag = 0;
            eye.src = 'images/close.png';
        }
    }
</script>
```

6.4 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。

1. element.style行内样式操作

```
this.style.backgroundColor = 'purple';
```

2. element.className类名样式操作

```
this.className = 'change'; //注意不加点，change类需要在style内定义
```

注意：

1.JS里面的样式采取驼峰命名法比如fontSize、backgroundColor(S、C大写)

2.JS修改style样式操作,产生的是行内样式，权重比较高

3.两者的不同点：element.style适用于样式较少或功能简单的情况下使用

element.className类名样式操作适用于样式较多或功能复杂的情况下使用

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。想保留原先的类名，使用并集选择器。

示例1：

使用style行内样式操作

鼠标点击盒子，盒子颜色变化，且宽度变大

```
<style>
    div {
        width: 200px;
        height: 200px;
        background-color: pink;
    }
</style>
```

```
<div></div>
<script>
    // 1. 获取事件源
    var div = document.querySelector('div');
    // 2. 注册事件 处理程序
    div.onclick = function () {
        // .style 属性
        this.style.backgroundColor = 'purple';
        this.style.width = '300px'; // 注意加单位
    }
</script>
```

结果：



鼠标点击后：



示例2：

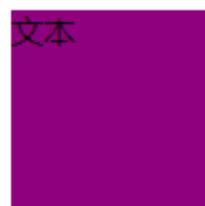
使用**className**类名样式操作，将新样式放入新定义的类内

鼠标点击盒子，盒子颜色变化，宽度变大，字体颜色变化，增加上边距

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

  .change {
    width: 200px;
    color: #fff;
    background-color: pink;
    margin-top: 50px;
  }
</style>
```

```
<div class='first'>文本</div>
<script>
  var test = document.querySelector('div');
  test.onclick = function () {
    //className适合于样式较多或功能较复杂的情况
    // this.className = 'change';//注意不加点，会覆盖原有的类
    //若想保留类，使用并集选择器
    this.className = 'first change';
  }
</script>
```



Screenshot of the Chrome DevTools Elements tab. The DOM tree shows the following structure:

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    ... <div class="first">文本</div> == $0
    <script>...</script>
  </body>
</html>
```

The element `<div class="first">文本</div>` is highlighted with a red border.

鼠标点击后



Screenshot of the Chrome DevTools Elements tab after a click. The DOM tree now shows:

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    ... <div class="first change">文本</div> == $0
    <script>...</script>
  </body>
</html>
```

The element `<div class="first change">文本</div>` is highlighted with a blue background and a red border.

6.5案例

6.5.1案例1:淘宝点击关闭二维码

当鼠标点击二维码关闭按钮的时候,则关闭整个二维码。

案例分析

- ①核心思路:利用样式的显示和隐藏完成, display:none 隐藏元素display:block显示元素
- ②点击按钮,就让这个二维码盒子隐藏起来即可

```
<style>
    .box {
        position: relative;
        width: 200px;
        height: 200px;
        font-size: 14px;
        margin: 100px auto;
        text-align: center;
        border: 1px solid #ccc;
        color: rgb(255, 63, 0);
    }

    .box img {
        width: 160px;
        height: 160px;
        margin-top: 5px;
    }

    .close-btn {
        position: absolute;
        top: -1px;
        left: -16px;
        width: 14px;
        height: 14px;
        font-style: normal;
        border: 1px solid #ccc;
        line-height: 14px;
        cursor: pointer;
    }
</style>

<div class="box">
    淘宝二维码
    
    <i class="close-btn">x</i>
</div>
```

```
<script>
    //1.获取元素
    var btn = document.querySelector('.close-btn');
    var box = document.querySelector('.box');
    //2.注册事件 程序处理
    btn.onclick = function () {
        box.style.display = 'none';
    }
</script>
```



点击x后消失

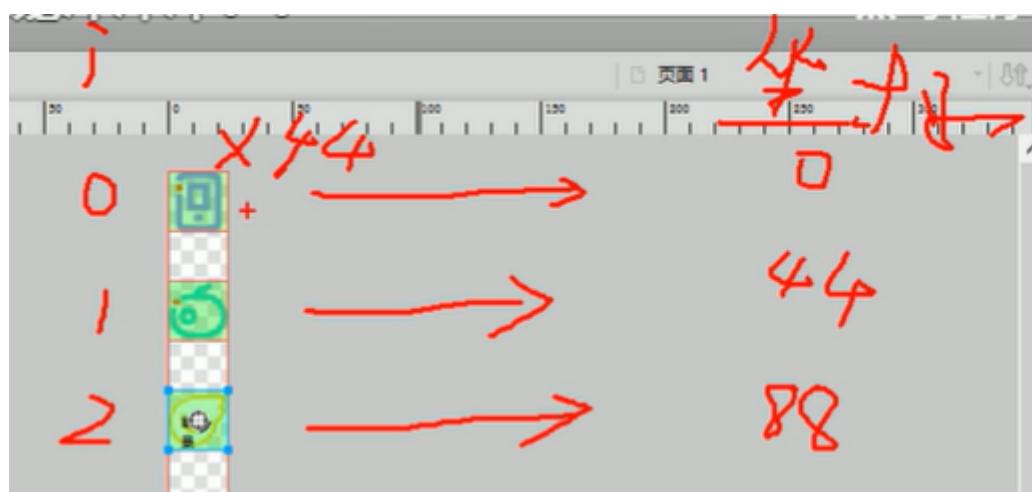
6.5.2案例:循环精灵图背景

可以利用for循环设置一组元素的精灵图背景



案例分析

- ①首先精灵图图片排列有规律的，如下图
- ②核心思路:利用for循环修改精灵图片的背景位置background-position
- ③剩下的就是考验你的数学功底了
- ④让循环里面的i索引号* 44就是每个图片的y坐标， * (-44) 即移动y坐标的距离



```
* {  
    padding: 0;  
    margin: 0;  
}
```

```

.box {
    width: 250px;
    margin: 100px auto;
    /* border: 1px solid pink; */
}

ul li {
    float: left;
    width: 24px;
    height: 24px;
    background-color: purple;
    margin: 15px;
    list-style: none;
    background: url(images/sprite.png);
}

```

```

<script>
    // 1. 获取元素 所有的li
    var lis = document.querySelectorAll('li');
    // 2. 注册事件 处理程序
    for (var i = 0; i < lis.length; i++) {
        // 让索引号乘-44就是每个li的背景y坐标。注意是乘-44
        var index = i * (-44);
        lis[i].style.backgroundPosition = '0 ' + index + 'px'; // 注意：x坐标后要加一个空格，末尾要加px单位
    }
</script>

```

注意：

1.让索引号乘-44就是每个li的背景y坐标。注意是乘-44而不是44，否则后面需要添加负号

2.使用字符串拼接方式形成背景图片坐标，因此注意：x坐标后要加一个空格，末尾要加px单位。

lis[i].style.backgroundPosition = '0 ' + index + 'px'

6.5.3案例:显示隐藏文本框内容

当鼠标点击文本框时,里面的默认文字隐藏,当鼠标离开文本框时,里面的文字显示。



案例分析

①首先表单需要2个新事件,获得焦点onfocus, 失去焦点onblur

②如果获得焦点, 判断表单里面内容是否为默认文字,如果是默认文字,就清空表单内容,且字体颜色变黑

③如果失去焦点, 判断表单内容是否为空,如果为空,则表单内容改为默认文字,且字体颜色变灰

示例：获得焦点onfocus, 失去焦点onblur使用

```

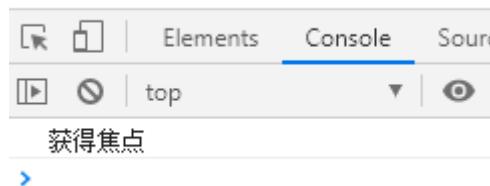
<input type="text" name="" id="" value="手机">

//1. 获取元素
var text = document.querySelector('input');
//2. 注册事件 获得焦点
text.onfocus = function () {
    console.log('获得焦点');
}

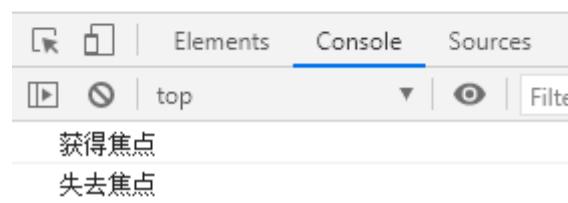
//失去焦点
text.onblur = function () {
    console.log('失去焦点');
}

```

鼠标在表单框内点击，获得焦点



鼠标在表单外点击，失去焦点



```

<input type="text" name="" id="" value="手机">
<script>
//1. 获取元素
var text = document.querySelector('input');
//2. 注册事件 获得焦点
text.onfocus = function () {
    // console.log('获得焦点');
    if (this.value === '手机') {
        this.value = '';
    }
}

```

```

        }
        //获得焦点，文本框字体颜色变黑
        this.style.color = '#fff';
    }

    //失去焦点
    text.onblur = function () {
        // console.log('失去焦点');
        if (this.value === '') {
            this.value = '手机';
        }
        //失去焦点，文本框字体颜色变灰
        this.style.color = '#999';
    }
</script>

```



6.5.4案例: 密码框格式提示错误信息

用户如果离开密码框,里面输入个数不是6~16位 ,则提示错误信息,否则提示输入正确信息



案例分析

- ①首先判断的事件是表单失去焦点onblur
- ②如果输入正确则提示正确的信息颜色，为绿色，小图标变化
- ③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
- ④因为里面变化样式较多,我们采取className修改样式

```

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

.register {
    width: 400px;
    height: 20px;
    margin: 100px auto;
}

.message {
    /* float: left; */
    display: inline-block;
}

```

```

        color: #999;
        font-size: 12px;
        line-height: 20px;
        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>

```

<div class="register">
    <input type="password" name="" id="ipt">
    <p class='message'>请输入6~16位密码</p>
</div>
<script>
    var ipt = document.getElementById('ipt');
    var message = document.querySelector('.message');
    ipt.onblur = function () {
        //根据表单里面的长度 this.value.length
        if (this.value.length < 6 || this.value.length > 16) {
            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 = '您输入正确';
        }
    }
</script>

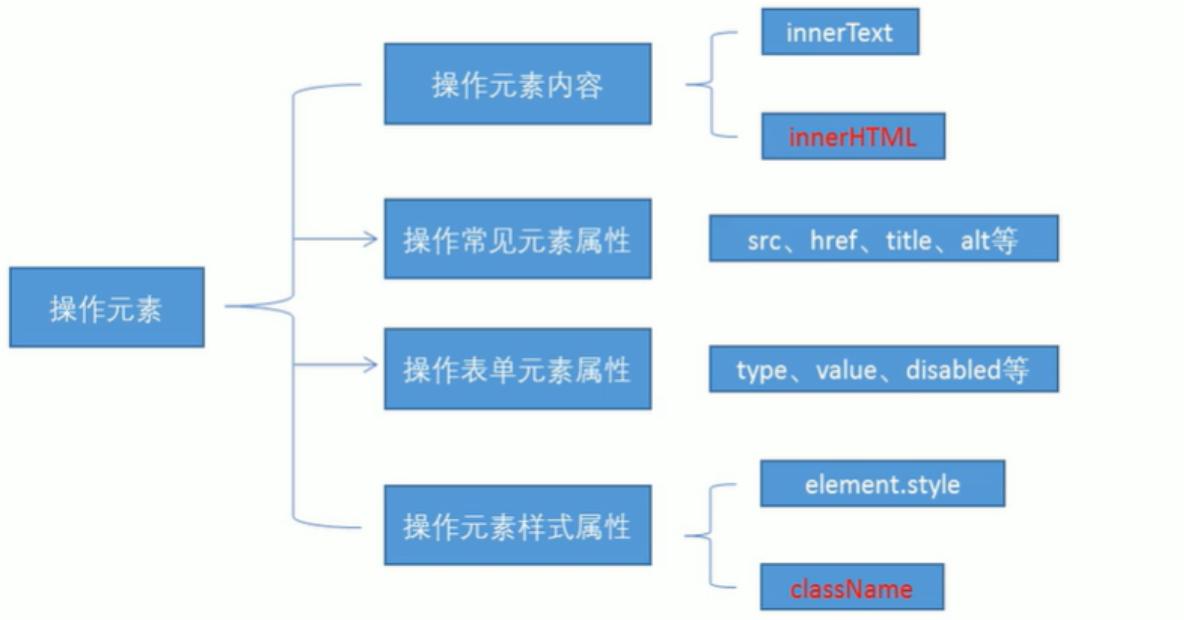
```

注意：

- 1.该案例表单右侧css样式的实现方式
- 2.表单里面的长度 this.value.length
- 3.案例分析思路

6.6操作元素总结

操作元素是DOM核心内容



6.7作业

- 1.世纪佳缘用户名显示隐藏内容
- 2.京东关闭广告(直接隐藏即可)
- 3.新浪下拉菜单(微博即可)
- 4.开关灯案例(见素材)

6.8排他思想



要求: 仅当前点击的按钮背景颜色变粉，其他按钮（包括先前点击的按钮）背景颜色不变。这时候需要排他思想

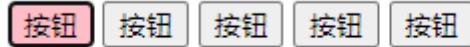
如果有同一组元素,我们想要某一个元素实现某种样式， 需要用到循环的**排他思想算法**:

- 1.所有元素全部清除样式(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

```

<button>按钮</button>
<button>按钮</button>
<button>按钮</button>
<button>按钮</button>
<button>按钮</button>
<script>
  //获取所有按钮元素
  var btns = document.getElementsByTagName('button');
  //遍历对象数组给每个按钮添加事件
  for (var i = 0; i < btns.length; i++) {
    btns[i].onclick = function () {
      //再次遍历，让所有按钮的背景颜色变为默认，干掉所有人
      for (var i = 0; i < btns.length; i++) {
        btns[i].style.backgroundColor = '';
      }
      //只设置自己
    }
  }
</script>
  
```

```
        this.style.backgroundColor = 'pink';
    }
}
</script>
```



案例：百度换肤

要求点击一张图片，浏览器背景图片就成这张图片；然后点击其他，换其他的。如下图，简易版



案例分析

- ①这个案例练习的是给一组元素注册事件
- ②给4个小图片利用循环注册点击事件
- ③当我们点击了这个图片,让我们页面背景改为当前的图片
- ④核心算法:把当前图片的src路径取过来,给body做为背景即可

```
* {
    margin: 0;
    padding: 0;
}

body {
    background: url(images/1.jpg) no-repeat center;
}

.baidu {
    width: 410px;
    background-color: #fff;
    margin: 100px auto;
    /* ?若无下句话, 背景颜色无法显示 */
    overflow: hidden;
    padding-top: 3px;
}

.ul li {
    float: left;
    list-style: none;
    margin: 0 1px;
    cursor: pointer;
}
```

```

.baidu img {
    width: 100px;
}

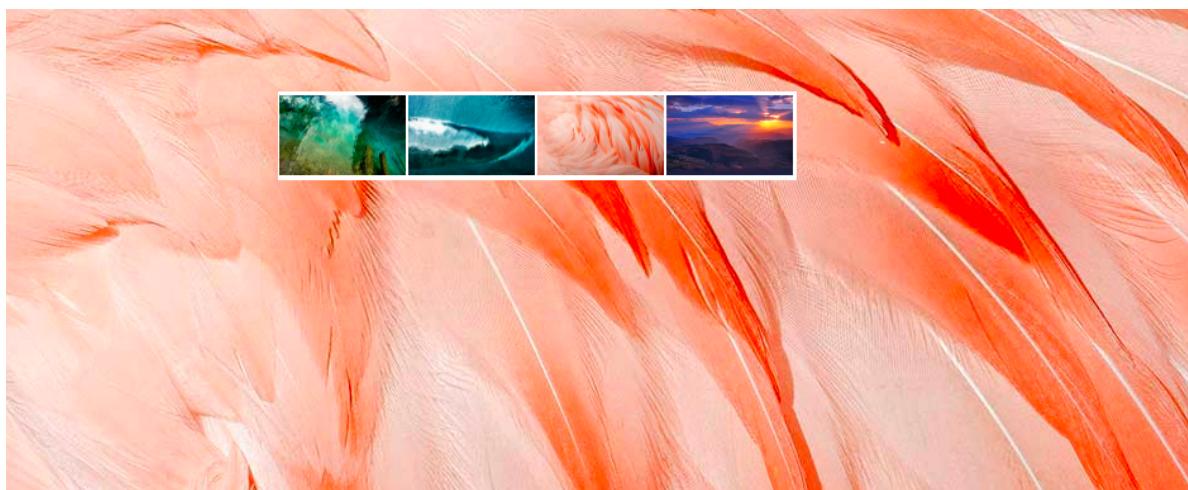
<ul class="baidu">
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>

```

```

<script>
    //获取元素 严格选择是.baidu类的下图片，元素获取方法可以连用
    var imgs = document.querySelector('.baidu').querySelectorAll('img');
    //注册事件 处理程序 循环添加图片事件
    for (var i = 0; i < imgs.length; i++) {
        imgs[i].onclick = function () {
            //将鼠标点击的图片的src路径赋值给body的背景图片
            document.body.style.background = 'url(' + this.src + ')';//注意该语句的连用，以及url()是字符串，使用字符串拼接来赋值路径
        }
    }
</script>

```



注意：

- 1.图片使用ul、li摆放
- 2.严格筛选是.baidu下的img实现方法，这样更严谨
- 3.src的路径需要与'url()'字符串拼接
- 4.document.body.style.background = 'url(' + this.src + ')';注意该语句的连用及赋值的路径

6.9案例

案例1：表格隔行变色

鼠标经过哪行，哪行背景颜色改变

代码	名称	最新公布净值	累计净值	前单位净值	净值增长率	公布日期	基金规模
003526	农银金穗3个月定期开放债券	1.075	1.079	1.074	+0.047%	2019-01-11	708.224
270047	广发理财30天债券B	0.903	3.386	0.000	0.000%	2019-01-16	535.921
163417	兴全合宜混合A	0.860	0.860	0.863	-0.382%	2019-01-16	326.996
003929	中银证券安进债券A	1.034	1.088	1.034	+0.077%	2019-01-16	320.634
360020	光大添天盈月度理财债券B	0.950	3.557	0.000	0.000%	2019-01-16	305.182

案例分析：

- ①用到新的鼠标事件鼠标经过onmouseover鼠标离开 onmouseout
- ②核心思路:鼠标经过tr行,当前的行变背景颜色, 鼠标离开去掉当前的背景颜色
- ③注意:第一行(thead里面的行)不需要变换颜色,因此我们获取的是tbody里面的行

```

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

  table {
    width: 800px;
    text-align: center;
    margin: 100px auto;
    border-collapse: collapse;
  }

  thead tr {
    height: 30px;
    font-size: 14px;
    background-color: skyblue;
  }

  tbody tr {
    height: 30px;
    font-size: 12px;
    color: blue;
    border-bottom: 1px solid #ccc;
  }

  .bg {
    background-color: pink;
  }

```

```

<script>
  // 1. 获取元素 只选择tbody下的tr
  var trs = document.querySelector('tbody').querySelectorAll('tr');
  //2. 循环注册事件, 处理程序
  for (var i = 0; i < trs.length; i++) {
    //鼠标经过事件, onmouseover 行变色
    trs[i].onmouseover = function () {
      this.className = 'bg';
    }
    //鼠标离开事件, onmouseout 行默认颜色
    trs[i].onmouseout = function () {

```

```

        this.className = '';
    }
}
</script>

```

代码	名称	最新公布净值	累计净值	前单位净值	净值增长率	公布日期
163417	兴全合宜混合A	0.860	0.860	0.863	-0.382%	2021-01-12
163417	兴全合宜混合A	0.860	0.860	0.863	-0.382%	2021-01-12
163417	兴全合宜混合A	0.860	0.860	0.863	-0.382%	2021-01-12
163417	兴全合宜混合A	0.860	0.860	0.863	-0.382%	2021-01-12
163417	兴全合宜混合A	0.860	0.860	0.863	-0.382%	2021-01-12

注意：

1. 使用边框合并border-collapse: collapse; 否则设置背景颜色时，单元格直接会有空隙，如下图

代码	名称	最新公布净值	累计净值	前单位净值	净值增长率	公布日期
----	----	--------	------	-------	-------	------

2. 鼠标经过事件，onmouseover；鼠标离开事件，onmouseout

3. 该案例也可通过排他思想完成，但分两个事件完成更语义化。

案例2：表单全选，取消全选

<input checked="" type="checkbox"/>	商品	价钱
<input checked="" type="checkbox"/>	iPhone8	8000
<input checked="" type="checkbox"/> 	iPad Pro	5000
<input checked="" type="checkbox"/>	iPad Air	2000
<input checked="" type="checkbox"/>	Apple Watch	2000

业务需求：

1. 点击上面全选复选框，下面所有的复选框都选中(全选)
2. 再次点击全选复选框，下面所有的复选框都不中选(取消全选)
3. 如果下面复选框全部选中，上面全选按钮就自动选中
4. 如果下面复选框有一个没有选中，上面全选按钮就不选中
5. 所有复选框一开始默认都没选中状态

案例分析：

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

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，如果有任何一个没选中的，上面全选就不选中。

```
<script>
```

```

//1.获取元素
var j_cbAll = document.getElementById('j_cbAll');//全选按钮
var j_tbs =
document.getElementById('j_tb').getElementsByTagName('input');//下面所有的复选框
//2.注册事件
//1) 全选和取消全选：让下面所有复选框的checked属性(选中状态)跟随全选按钮
j_cbAll.onclick = function () {
    //console.log(this.checked); //复选框选中状态为true，否则false未选择
    for (var i = 0; i < j_tbs.length; i++) {
        j_tbs[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。
for (var i = 0; i < j_tbs.length; i++) {
    //循环检查下面复选框的状态
    j_tbs[i].onclick = function () {
        //通过flag变量来控制全选框的状态，默认是勾选状态true，不勾选则为false
        var flag = true;
        for (var i = 0; i < j_tbs.length; i++) {
            if (!j_tbs[i].checked) { //下面复选框有一个没勾选，全选框就不选，并退出循环，提高效率
                flag = false;
                break;
            }
        }
        j_cbAll.checked = flag;
    }
}
</script>

```

	商品名称	价格
<input checked="" type="checkbox"/>	iPad Pro	6000
<input checked="" type="checkbox"/>	iPad Air	5000
<input checked="" type="checkbox"/>	Apple Watch	4000

注意：

1.若勾选了复选框，控制台输出为‘true’

```
console.log(this.checked);
```



同理，让复选框勾选，赋值为true；不让勾选，赋值为false

```
j_cbAll.checked = true;
```

2.让下面所有复选框的checked属性(选中状态)跟随全选按钮（可以一个事件实现全选和取消全选），而不是赋值checked，这样只能全选，无法取消全选。

```
j_tb[i].checked = this.checked;
```

3.在实现下面复选框需要全部选中，上面全选才能选中，通过flag变量来控制全选框的状态

6.10自定义属性的操作

6.10.1.获取属性值

语法：

- element . 属性
- 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

```
<div id="demo" index='2' class="nav" ></div>
```

```
<script>
    var div = document.querySelector('div');
    //获取属性值
    //1.element.属性
    console.log(div.id);
    //2.element.getAttribute('属性') 适用于获取自定义属性，如index
    console.log(div.getAttribute('id'));
    console.log(div.getAttribute('index'));
</script>
```

```
demo
demo
2
```

6.10.2设置属性值

语法：

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

区别：

- element.属性= '值' 设置内置属性值，**设置class类属性值时，写的是className**
- element .setAttribute('属性', '值'); 主要设置**自定义的属性** (标准)，**设置class类元素属性值时，写的是class**

```
//2.设置元素属性值
//1)element.属性='值'
div.id = 'test';
//div.className = 'navs';//注意设置class类属性值时，写的是className
//2)element.setAttribute('属性','值') 主要针对自定义属性
div.setAttribute('index', 1);
div.setAttribute('class', 'footer');//注意设置class类元素属性值时，写的是class
```

```
...<html lang="en"> == $0
  > <head>...</head>
  > <body>
    <div id="test" index="1" class="footer"></div>
      > <script>...</script>
    </body>
  > <script>...
```

6.10.3 移除属性

语法：

- element . removeAttribute('属性');

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

```
<body>
  <div id="test" class="footer"></div>
```

6.10.4 案例：tab栏切换(重点案例)

当鼠标点击上面相应的选项卡(tab)，下面内容跟随变化：



模块划分：

一个大盒子分为上下两个盒子：tab_list、tab_con

tab_list内放置5个li

tab_con放置5个上面标题对应的内容，选中哪个标题，显示相应的模块内容，其他隐藏

案例分析：

① Tab栏切换有2个大的模块

② 上面的模块选项卡，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

③ 下面的模块内容，会跟随上面的选项卡变化。所以下面模块变化写到点击事件里面。

④ 规律：下面的模块显示内容和上面的选项卡一一对应，相匹配。

⑤ 核心思路：给上面的tab_list里面的所有小li添加自定义属性index，属性值从0开始编号。

```

//实现点击某个模块选项卡，其底色变红，其余不变（排他思想）
var lis = document.querySelector('.tab_list').querySelectorAll('li');
var items = document.querySelectorAll('.item');
//循环注册事件
for (var i = 0; i < lis.length; i++) {
    lis[i].setAttribute('index', i); //开始就给上面模块设置索引号
    lis[i].onclick = function () {
        //1.上面的点击显示底色模块
        //干掉其他人，只设置自己
        for (var i = 0; i < lis.length; i++) {
            lis[i].className = ''; //清除class类来达到默认底色的效果
        }
        this.className = 'current';

        //2.下面显示选项卡内容模块，也是排他思想
        //得到当前点击模块的索引值
        var index = this.getAttribute('index');
        //干掉其他人
        for (var i = 0; i < lis.length; i++) {
            items[i].style.display = 'none';
        }
        // console.log(index);
        //只设置自己
        items[index].style.display = 'block';
    }
}

```

商品介绍 规格与包装 售后保障 商品评价(1.1万+) 手机社区
规格与包装模块

注意：

1.默认选中第一个模块，显示第一个模块内容。可以将所有模块内容隐藏，在行内样式中设置第一个模块内容显示。

```

.item {
    display: none;
}

<div class="item" style="display: block;">
    商品介绍模块
</div>

```

2.点击后模块设置底色，显示对应模块内容都应用到了**排他思想**

3.给上面模块添加自定义属性**index**来对应下面的模块内容，从而实现对应模块内容的隐藏和显示。注意不能使用变量*i*来对应，**for**循环里的计数器*i*的作用范围并不能影响点击事件匿名函数里，会报错。

```

    style="display: none;">
}
// console.log(index);
//只设置自己
items[i].style.display = 'block';✖
,

```

Uncaught TypeError: Cannot read property 'style' of undefined

```
... ▼ <body> == $0
  ▼ <div class="tab">
    ▼ <div class="tab_list">
      ▼ <ul>
        <li class index="0">商品介绍</li>
        <li index="1" class="current">规格与包装</li> 成功的示例
        <li index="2" class="disabled">售后保障</li>
        <li index="3" class="disabled">商品评价(1.1万+)</li>
        <li index="4" class="disabled">手机社区</li>
      </ul>
    </div>
```

6.11 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<div data-index= "1">

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

示例:

```
<div getTime="20" data-index='1' data-list-name='ddd'></div>
<script>
  var div = document.querySelector('div');
  // console.log(div.getTime); //undefined
  console.log(div.getAttribute('getTime'));
  //H5新增获取自定义元素element.dataset.xxxx 这个xxx是data-后面的名字
  //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']);
  //若自定义属性里面有多个-连接的单词, 获取时注意采用驼峰命名法, 去掉-
  console.log(div.dataset.listName);
  console.log(div.dataset['listName']);
</script>
```

```
20
▶ DOMStringMap {index: "1", listName: "ddd"}
1
1
ddd
ddd
>
<div gettime="20" data-index="1" data-list-name="ddd"></div>
```

注意：

- 1.dataset是一个集合，里面存放了所有以data开头的自定义属性
- 2.若自定义属性里面有多个-连接的单词，获取时注意采用驼峰命名法，去掉-

7. 节点操作

7.1 为什么学节点操作

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

- document.getElementById()
- document.getElementsByTagName()
- document.querySelector() 等
- 逻辑性不强、繁琐

2. 利用节点层级关系获取元素

- 利用父子兄节点关系获取元素
- 逻辑性强，但是兼容性稍差

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

7.2 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**

7.3 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**。

1.父节点

node.parentNode 驼峰命名法

示例：

```
<div class="demo">
    <div class="box">
        <span class="erweima"></span>
    </div>
</div>
<script>
//获取erweima和box元素
//以前的方法
var erweima = document.querySelector('.erweima');
// var box = document.querySelector('.box');
//使用父节点parentNode，简单获得box元素，是最近一级的父元素（亲父亲），如果找不到父节点就返回为null
console.log(erweima.parentNode);
</script>
```

2.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（循环遍历返回值，根据节点值来筛选元素节点）。所以一般不使用这个。

示例：

```
//获得ol下的li元素
//DOM下的方法
var ol = document.querySelector('ol');
var lis = ol.querySelectorAll('li');
// console.log(lis);

//子节点
//childNodes
console.log(ol.childNodes); //包括文本节点、元素节点等所有子节点
console.log(ol.childNodes[0].nodeType); //3 文本节点
console.log(ol.childNodes[1].nodeType); //1 元素节点
```

► NodeList(7) [text, li, text, li, text, li, text]

3

1

↙

这个文本节点是li之间的换行

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

```
//children  
console.log(o1.children);
```

▶ HTMLCollection(3) [li, li, li]

3.第一个/最后一个子元素节点

1)

```
1. parentNode.firstChild  
2. parentNode.lastChild
```

firstChild/lastChild返回第一个/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。所以返回的不一定是第一个/最后一个子元素节点

2)

```
3. parentNode.firstElementChild  
4. parentNode.lastElementChild
```

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

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。

3)最常用的,没有兼容性问题。通过children子节点索引获得

```
console.log(u1.children[0]);  
console.log(u1.children[u1.children.length - 1]);
```

示例:

```
var u1 = document.querySelector('ul');  
// 1.firstChild/lastChild 第一个/最后一个子节点 包括文本节点和其他类型节点  
console.log(u1.firstChild);  
console.log(u1.lastChild);  
//2.firstElementChild/lastElementChild 第一个/最后一个子元素节点 有兼容性问题  
console.log(u1.firstElementChild);  
console.log(u1.lastElementChild);  
//3.最常用的 通过子节点children索引得到第一个/最后一个子元素节点 无兼容性问题  
console.log(u1.children[0]);  
console.log(u1.children[u1.children.length - 1]);
```

```

▶ #text
▶ #text
▼ <li>
  ::marker
  "1"
</li>
▼ <li>
  ::marker
  "4"
</li>
▼ <li>
  ::marker
  "1"
</li>
▼ <li>
  ::marker
  "4"
</li>

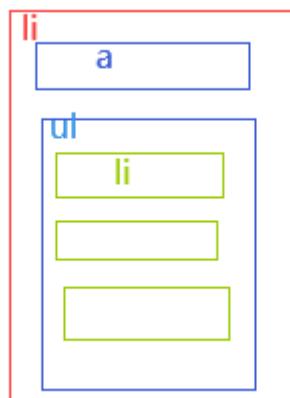
```

案例——新浪下拉菜单

实现：



模块划分：



导航栏整体使用ul、li做，下拉菜单模块是li中包含上下两个模块，上模块使用链接a制作，下模块使用ul、li制作，通过绝对定位定位到a下方。

实现：鼠标经过时，ul显示；鼠标离开时，ul隐藏。

```

<ul class="nav">
  <li>
    <a href="#">登录</a>
  </li>
  <li>
    <a href="#">微博</a>
  </li>

```

```
<ul>
    <li>评论</li>
    <li>私信</li>
    <li>@我</li>
</ul>
</li>
<li>
    <a href="#">博客</a>
    <ul>
        <li>评论</li>
        <li>私信</li>
        <li>@我</li>
    </ul>
</li>
<li>
    <a href="#">邮箱</a>
    <ul>
        <li>评论</li>
        <li>私信</li>
        <li>@我</li>
    </ul>
</li>
</ul>
```

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

li {
    list-style: none;
}

.nav>li {
    position: relative;
    float: left;
    text-align: center;
}

.nav>li a {
    display: block;
    font-size: 18px;
    height: 41px;
    line-height: 41px;
    text-decoration: none;
    color: #333;
    padding: 0 10px;
    /* background-color: skyblue; */
}

.nav>li a:hover {
    background-color: #eee;
}

.nav ul {
    display: none;
```

```

        position: absolute;
        top: 41px;
        left: 0;
        width: 100%;
        font-size: 16px;
    }

    .nav ul li {
        height: 30px;
        border: 1px solid #FECC5B;
        border-top: 0;
        /* background-color: pink; */
    }

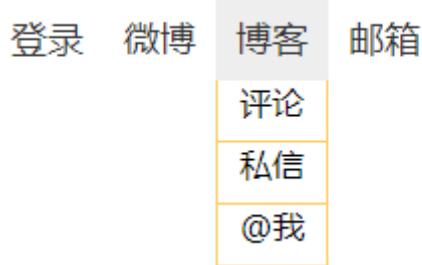
    .nav ul li:hover {
        background-color: pink;
        cursor: pointer;
    }

```

```

<script>
    // 1.获取元素
    var nav = document.querySelector('.nav');
    var lis = nav.children;//获得4个li
    //2.for循环注册事件
    for (var i = 1; 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>

```



注意：为了实现鼠标经过时ul显示，ul是li的第二个孩子，因此通过此来设置隐藏还是显示，红色框是li的第一个孩子，黄色框是第二个孩子。

```
this.children[1].style.display = 'block';
```



3. 兄弟节点

1. `node.nextSibling`

`nextSibling`返回当前元素的下一个兄弟节点,找不到则返回null.同样,也是**包含所有的节点**.

2. `node.previousSibling`

`previousSibling`返回当前元素上一个兄弟节点,找不到则返回null.同样,也是**包含所有的节点**.

3. `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,

4. `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,

注意:这两个方法有兼容性问题, IE9以上才支持。

问:如何解决兼容性问题?

答:自己封装一个兼容性的函数, 会调用即可

```
function getNextElementSibling (element) {  
    var e1 = element ;  
    while (e1 = e1.nextSibling) {  
        if (e1.nodeType === 1) {  
            return e1;  
        }  
    }  
    return null;  
}
```

兄弟节点实际工作中不太使用, 主要是父子节点

示例:

```
<div>111</div>
<span>222</span>
<script>
    var div = document.querySelector('div');
    //1.nextSibling下一个兄弟节点 包括所有节点
    console.log(div.nextSibling);
    console.log(div.previousSibling); //上一个
    //2.nextElementSibling下一个兄弟元素节点
    console.log(div.nextElementSibling);
    console.log(div.previousElementSibling); //上一个
</script>
```

```
▶ #text
▶ #text
<span>222</span>
null
>
```

7.4 创建节点

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

document.createElement()方法创建由**tagName** (如li等标签名字)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**。

7.5 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)
```

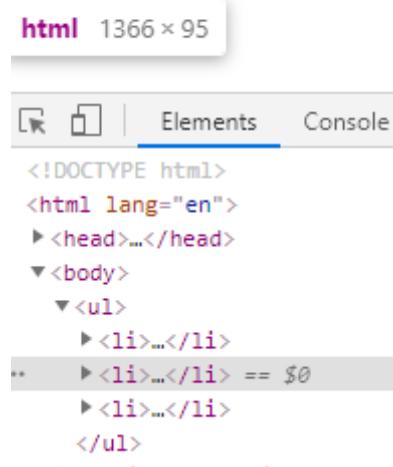
node.insertBefore ()方法将一个节点**添加到父节点的指定子节点前面**。类似于css里面的before伪元素。

示例:

```

<ul>
    <li>111</li>
</ul>
<script>
    //1. 创建元素节点 创建一个li节点
    var li = document.createElement('li')
    //2. 添加节点 node.appendChild child是父级 child是子级，后面添加元素
    var ul = document.querySelector('ul');
    ul.appendChild(li);
    //3. 添加节点 node.insertBefore(child,指定元素) 在指定元素前添加child节点 node是父级
    var lili = document.createElement('li');
    ul.insertBefore(lili, ul.children[0]);
</script>

```



小结：

- 1.想要页面添加一个新的元素需要两步：1) 创建元素2) 添加元素
- 2.创建元素document.createElement ('tagName')， tagName为标签元素名字，如li，需要加引号
- 3.**添加元素有两种**， 1) node.appendChild (child)， 在node父节点末尾添加child子元素节点2) node.insertBefore (child,指定元素)， 在node父节点**指定元素的前面添加**child节点
- 4.这可应用于发布评论， 每一条评论的发布都是在页面中添加一个元素

案例：简单版发布留言案例

案例分析

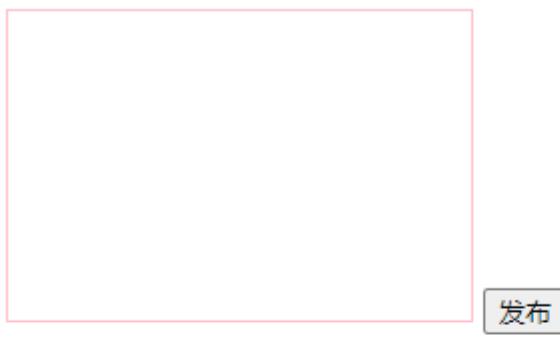
- ①核心思路:点击按钮之后,就动态创建一个li,添加到ul里面。
- ②创建li的同时,把文本域里面的值通过li.innerHTML赋值给li
- ③如果想要新的留言后面显示就用appendChild；如果想要前面显示就用insertBefore
增加未输入内容时发布的友好提示，增加发布内容完毕后留言板清空

```
//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 点击按钮添加一个元素节点
btn.onclick = function () {
    //首先判断是否输入为空，更完善
    if (text.value == '') {
        alert('您未输入内容');
        return false; //终止操作
    } else {
        // console.log(text.value);
        //1.创建元素节点
        var li = document.createElement('li');
        li.innerHTML = text.value //将内容赋值给li里面，先创建后赋值再添加
        //2.添加节点
        ul.appendChild(li);
        text.value = '' //发布内容后留言板内清空
    }
}

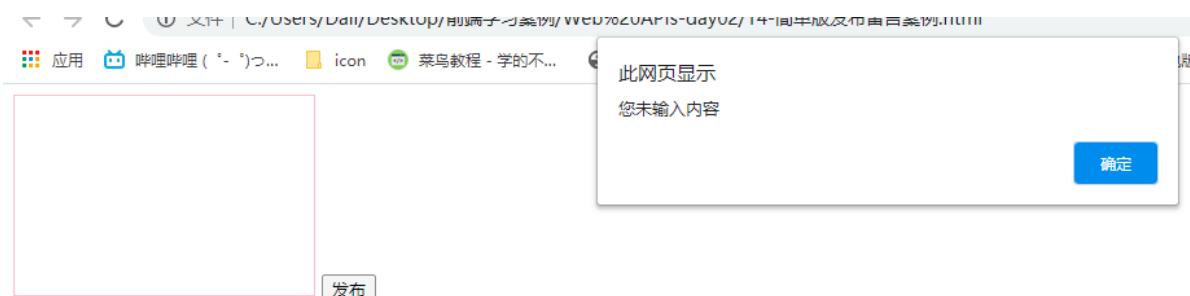
```



1

2

未输入就点击发布按钮

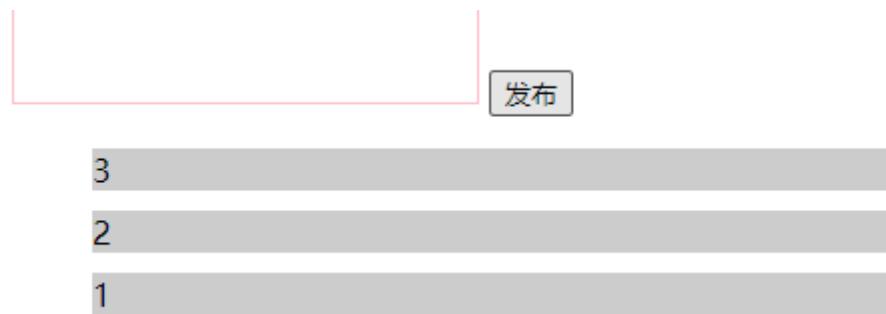


让新发布的内容第一条显示

```

ul.insertBefore(li,ul.children[0]); //让新发布的内容第一条显示，永远在第一个孩子前面

```



7.6删除节点

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

node.removeChild()方法从DOM中删除一个node父节点下的child子节点,返回删除的节点。

示例:

实现点击删除按钮,删除第一个节点, 删完了就不能再删了 (通过ul的孩子长度是否为0判断), 禁用按钮

```
var ul = document.querySelector('ul');
var btn = document.querySelector('button');
//实现点击删除按钮, 删除第一个节点
btn.onclick = function () {
    //完善判断, 没有孩子时, 按钮禁用
    if (ul.children.length > 0) {
        ul.removeChild(ul.children[0]);
    } else {
        this.disabled = true;
    }
}
```

删除

- 熊大
- 熊二
- 光头强

删除

案例：删除留言板留言

案例分析:

- ①当我们把文本域里面的值赋值给li的时候,**多添加一个删除的链接**
- ②需要把所有的链接获取过来,当我们点击当前的链接的时候,删除**当前链接所在的li** (a是li的子节点, 可根据a的父节点找到当前链接所在的li)
- ③**阻止链接跳转**需要在href里添加javascript:void(0);或者javascript::;

```
li.innerHTML = text.value + "<a href='javascript:;'>删除</a>" //将内容赋值给li里面，先创建后赋值再添加，并添加链接
```

```
//3.删除节点 删除评论 循环给链接a添加删除事件
var as = document.querySelectorAll('a');
for (var i = 0; i < as.length; i++) {
    as[i].onclick = function () {
        //删除当前链接a所在的li
        //删除节点 node.removeChild(child) child是当前链接a所在的li, node是删除节点的父节点, li的父节点是ul
        ul.removeChild(this.parentNode);
    }
}
```

3	删除
2	删除
1	删除

删除评论'2'

3	删除
1	删除

注意点：

1.每条评论的li里面包括评论内容和**删除键（链接实现）**，注意其语句书写

```
li.innerHTML = text.value + "<a href='javascript:;'>删除</a>"
```

2.'删除'链接右浮动，放到css内实现

```
ul li a {
    float: right;
}
```

3.**删除事件仍在按钮发布评论事件内**，事件嵌套，不能单独放在发布事件外，否则不能实现删除。猜测原因：只有评论发布了才能删除评论，事件之间具有先后性

4.通过a的父节点来找到当前链接所在的li,并删除

4.删除节点 node.removeChild(child) , child是当前链接a所在的li, node应是删除节点的父节点, li的父节点是ul

```
ul.removeChild(this.parentNode);
```

7.7复制节点(克隆节点)

```
node.cloneNode()
```

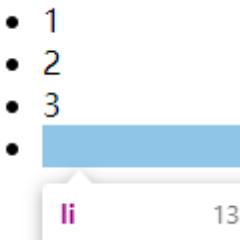
node.cloneNode()方法返回调用该方法的节点的一个副本。也称为**克隆节点/拷贝节点**

注意:

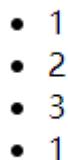
- 1.如果括号参数为空或者为false ,则是浅拷贝,即只克隆复制节点本身,不克隆里面的子节点(内容)。
- 2.如果括号参数为true ,则是深度拷贝,会复制节点本身以及里面所有的子节点。
- 3.复制节点后仍需添加节点才能在页面中显示。 (与创建+添加一样)

示例:

```
var ul = document.querySelector('ul');
//1.node.cloneNode() 参数为空或false, 浅拷贝, 只复制标签不复制内容
var lili = ul.children[0].cloneNode();
ul.appendChild(lili);
```



```
//2.node.cloneNode(true) 参数为true, 深拷贝, 复制标签和复制内容
var lili = ul.children[0].cloneNode(true);
ul.appendChild(lili);
```



克隆了ul的第一个孩子节点

案例：动态生成表格

姓名	科目	成绩	操作
魏璎珞	JavaScript	100	删除
弘历	JavaScript	90	删除
傅恒	JavaScript	99	删除
明玉	JavaScript	89	删除

案例分析:

①因为里面的学生数据都是动态的,我们需要js动态生成。这里我们先模拟数据,自己定义好数据。**数据我们采取对象形式存储, 并将对象放入数组内**

②所有的数据都是放到tbody里面的行里面。

③因为行很多,我们需要循环创建多个行(对应多少人)

④每个行里面又有很多单元格(对应里面的数据), 我们还继续使用循环创建多个单元格, 并且把数据存入里面(双重for循环)

⑤最后一列单元格是删除, 需要单独创建单元格。

分为5步

- 1) 数组形式存储放有学生数据的对象
- 2) 创建行, 行数取决于学生人数, 即数组的长度
- 3) 在行内创建单元格, 单元格数取决于对象的属性数量, 并使用双重for循环填入对象的属性值
- 4) 单独创建删除单元格
- 5) 给删除单元格添加删除链接及删除操作

```
//1.先准备数据
//通过数组的形式存储数据对象, 注意不包括操作
var datas = [
    {
        name: '张三',
        subject: '数学',
        score: 100,
    }, {
        name: '李思',
        subject: '数学',
        score: 90,
    }, {
        name: '王武',
        subject: '数学',
        score: 80,
    }, {
        name: '刘柳',
        subject: '数学',
        score: 89,
    }
]
var tbody = document.querySelector('tbody');
//2.创建行, 行数与人数有关(取决于数组长度)
for (var i = 0; i < datas.length; i++) {//外层循环管行
    //创建元素节点行tr
    var tr = document.createElement('tr');
    //添加节点
    tbody.appendChild(tr);
    //3.行里创建单元格td(与数据有关, 取决于对象里的属性数量, 循环对象属性)
    for (var k in datas[i]) {//内层循环管单元格
        //创建单元格
        var td = document.createElement('td');
        //把对象里面的属性值放入单元格内
        td.innerHTML = datas[i][k];//obj[k] 属性值
        //添加单元格, 放入行内
        tr.appendChild(td);
    }
    //4.创建操作单元格, 含删除操作
    var td = document.createElement('td');
    td.innerHTML = '<a href=javascript: ;>删除</a>';
    tr.appendChild(td);
}
//5.点击删除操作 添加完单元格后再进行删除, 放在for循环创建的外面
var as = document.querySelectorAll('a');
```

```
for (var i = 0; i < as.length; i++) {
    as[i].onclick = function () {
        //删除链接a所在的行tr node.removeChild a的爸爸是td 再爸爸是tr
        //tr的爸爸是tbody
        tbody.removeChild(this.parentNode.parentNode);
    }
}
```

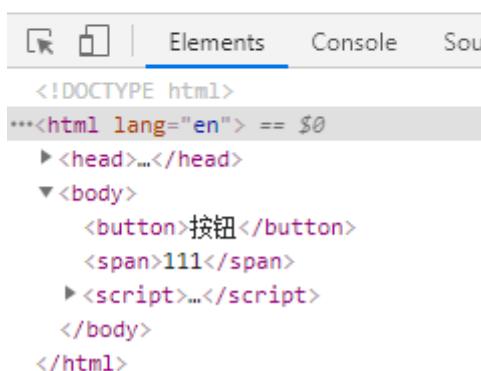
7.8三种动态创建元素区别

- `document.write()` 了解即可，很少使用
- `element.innerHTML`
- `document.createElement()`

`document.write()` 示例：

```
<button>按钮</button>
<span>111</span>
<script>
    //1. document.write() 创建元素，如果页面文档流加载完毕，再调用这句话会导致页面重绘
    var btn = document.querySelector('button');
    btn.onclick = function () {
        document.write('<div>234</div>');
    }
</script>
```

未调用前：



调用后：

```
<html>
  <head></head>
  <body>
    <div>234</div>
  </body>
</html>
```

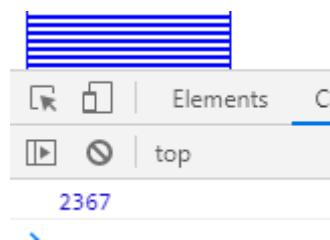
区别

1. 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**
2. innerHTML 是将内容写入某个DOM节点,不会导致页面全部重绘
3. **innerHTML 创建多个元素效率更高(不要拼接字符串, 拼接字符串占用内存, 采取数组形式拼接)**, 结构稍微复杂
4. createElement() 创建多个元素效率稍低一点点,但是结构更清晰

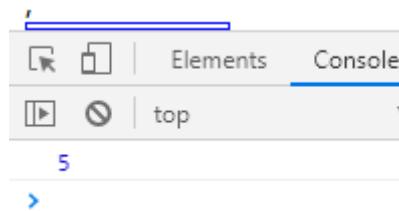
总结:不同浏览器下, innerHTML 效率要比 createElement 高

下面是测试结果, 创建1000个100*2的盒子所花费的毫秒数

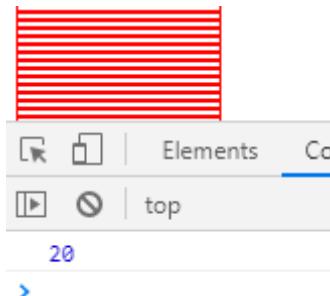
innerHTML字符拼接效率测试



innerHTML数组效率测试



createElement效率测试



8.DOM重点核心

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使得html形成一棵dom树. 包含文档、元素、节点

我们获取过来的DOM元素是一个对象(object) ,所以称为文档对象模型

关于dom操作,我们主要针对于元素的操作。主要有创建、增、删、改、查、属性操作、事件操作。

8.1创建

1. document.write
2. innerHTML
3. createElement

8.2增

1. appendChild
2. insertBefore

8.3删

1. removeChild

8.4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等
- 4.修改元素样式: style. className

8.5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, 不太推荐

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

主要是后两种方法

8.6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

8.7事件操作

给元素注册事件，采取事件源事件类型=事件处理程序

9.事件高级

9.1注册事件(绑定事件)

9.1.1注册事件概述

给元素添加事件,称为注册事件或者绑定事件。

注册事件有两种方式:传统方式和方法监听注册方式

传统注册方式

- 利用on开头的事件onclick

```
<button onclick= "alert(hi~)" ></button>
```

- btn.onclick = function(){}

特点:注册事件的唯一性

同一个元素同一个事件只能设置一个处理函数,最后注册的处理函数将会覆盖前面注册的处理函数

方法监听注册方式

- w3c 标准推荐方式

- addEventListener()它是一个方法,更常用

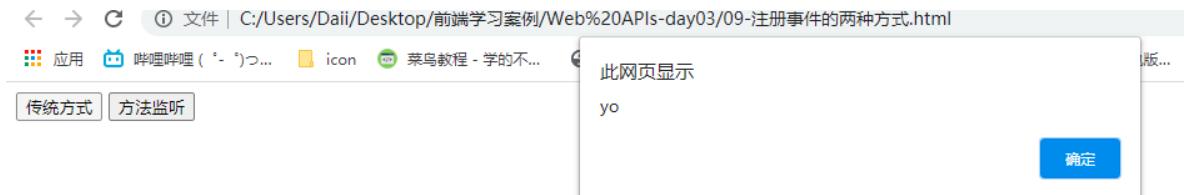
- IE9之前的IE不支持此方法,可使用attachEvent()代替

- 特点:同一个元素同一个事件可以注册多个监听器,按注册顺序依次执行

示例:

传统注册方式,唯一性

```
var btns = document.querySelectorAll('button');
btns[0].onclick = function () {
    alert('ho');
}
btns[0].onclick = function () {
    alert('yo');
}
```



添加两个点击事件,只会显示最后一个注册事件

9.1.2 addEventListener事件监听方式

```
eventTarget.addEventListener (type, listener[, useCapture])
```

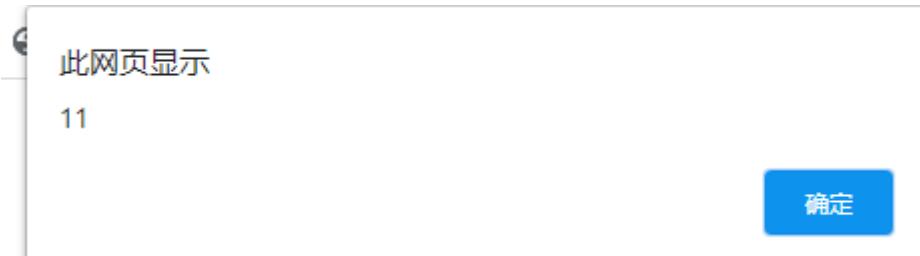
eventTarget.addEventListener ()方法将指定的监听器注册到eventTarget (目标对象)上,当该对象触发指定的事件时,就会执行事件处理函数。

该方法接收三个参数:

- type** :事件类型**字符串, 必须加引号**, 比如click. mouseover ,注意这里不要带on
- listener** :事件处理函数,事件发生时,会调用该监听函数
- useCapture** :可选参数,是一个布尔值,默认是false. 学完DOM事件流后,再学习

示例:

```
//监听注册事件
//1) 事件类型是字符串, 必须加引号, 且不用加on
//2) 同一个元素 同一个事件可以添加多个监听器
bttns[1].addEventListener('click', function () {
    alert(11);
})
bttns[1].addEventListener('click', function () {
    alert(22);
})
```



点击确定后



9.1.3 attachEvent事件监听方式

```
eventTarget.attachEvent (eventNamewithOn, callback)
//如
btn.attachEvent('onclick', funcation(){
    alert('1');
})
```

eventTarget .attachEvent ()方法将指定的监听器注册到eventtarget(目标对象)上,当该对象触发指定的事件时,指定的回调函数就会被执行。

该方法接收两个参数:

- eventNameWithOn** :事件类型**字符串**,比如onclick. onmouseover , 这里要**带on**
- callback**: 事件处理函数,当目标触发事件时回调函数被调用

注意: IE8及早期版本支持

9.1.4注册事件兼容性解决方案

```
function addEventListener (element, eventName, fn) {  
    //判断当前浏览器是否支持addEventListener方法  
    if (element.addEventListener) {  
        element.addEventListener (eventName,fn); // 第三个参数默认是false  
    } else if (element.attachEvent) {  
        element.attachEvent('on' + eventName, fn) ;  
    } else {  
        //相当于element.onclick= fn;传统方法  
        element['on' + eventName] = fn;  
    }  
}
```

兼容性处理的原则:首先照顾大多数浏览器,再处理特殊浏览器

9.2删除事件(解绑事件)

9.2.1删除事件的方式

1.传统注册方式

```
eventTarget.onclick = null;
```

示例:

```
var divs = document.querySelectorAll('div');  
//1.传统方法删除事件 只让事件触发一次  
divs[0].onclick = function () {  
    alert('1');  
    divs[0].onclick = null;  
}
```

2.方法监听注册方式

```
@eventTarget. removeEventListener (type, listener[, useCapture]);
```

注意为了移除, 监听处理程序函数需要外写以带名字

```
@eventTarget. detachEvent (eventName, callback);
```

示例:

```
//2.方法监听注册方式  
divs[1].addEventListener('click', fn); //fn不要调用加小括号  
function fn() {  
    alert('2');  
    divs[1].removeEventListener('click', fn); //由于需要输入监听器名字, 而使用匿名函数没有名字, 所以需要监听器外写  
}  
//3.ie版本的同理
```

9.2.2 删除事件兼容性解决方案

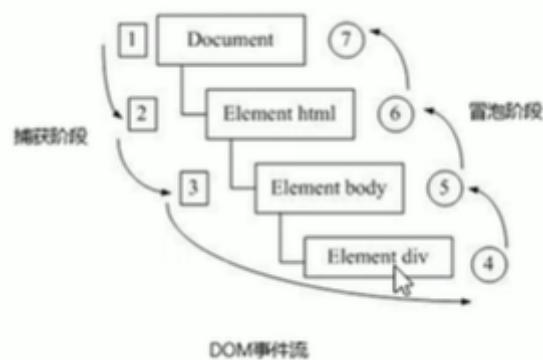
```
function removeEventListener (element, eventName, fn) {  
    // 判断当前浏览器是否支持removeEventListener方法  
    if (element.removeEventListener) {  
        element.removeEventListener (eventName, fn); // 第三个参数默认是false  
    } else if (element.detachEvent) {  
        element.detachEvent('on' + eventName, fn) ;  
    } else {  
        element['on' + eventName] = null ;  
    }  
}
```

9.3 DOM事件流

事件流描述的是从页面中接收事件的顺序。

事件发生时会在元素节点之间按照特定的顺序传播,这个传播过程即DOM事件流。

DOM事件流分为3个阶段:



DOM事件流分为3个阶段:

1.捕获阶段 从外到里

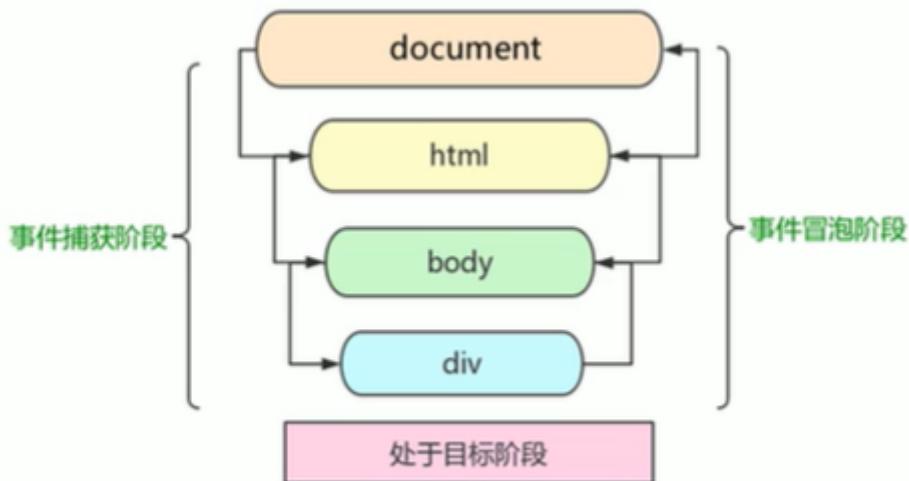
2.当前目标阶段

3.冒泡阶段 从里到外

●事件冒泡: IE最早提出,事件开始时由最具体的元素接收,然后逐级向上传播到到DOM最顶层节点的过程。

●事件捕获:网景最早提出,由DOM最顶层节点开始,然后逐级向下传播到到最具体的元素接收的过程。

简单理解: 我们向水里面扔一块石头,首先它会有一个下降的过程,这个过程就可以理解为从最顶层向事件发生的最具体元素(目标点)的捕获过程;之后会产生泡泡,会在最低点(最具体元素)之后漂浮到水面上,这个过程相当于事件冒泡。



注意

- 1.JS代码中只能执行捕获或者冒泡其中的一个阶段。
- 2.onclick和attachEvent只能得到冒泡阶段。
- 3.addEventListener(type, listener[, usecapture])第三个参数如果是true ,表示在事件捕获阶段调用事件处理程序;如果是false (不写默认就是false) , 表示在事件冒泡阶段调用事件处理程序。
- 4.实际开发中我们很少使用事件捕获, 我们更关注事件冒泡。
- 5.有些事件是没有冒泡的, 比如onblur. onfocus. onmouseenter. onmouseleave
- 6.事件冒泡有时候会带来麻烦, 有时候又会帮助很巧妙的做某些事件, 后面讲解。

示例:

```
//捕获阶段 addEventListener 第三个参数为true 从外到里 document->html->body->father->son
var son = document.querySelector('.son');
var father = document.querySelector('.father')
son.addEventListener('click', function () {
    alert('son');
}, true)
father.addEventListener('click', function () {
    alert('father');
}, true)
```



点击son盒子，先提示father 再提示son

```
//冒泡阶段 addEventListener 第三个参数为false或默认 从里到外 son->father->body->html->document
var son = document.querySelector('.son');
var father = document.querySelector('.father')
son.addEventListener('click', function () {
    alert('son');
})
father.addEventListener('click', function () {
    alert('father');
}, false)
```

点击son盒子,先提示son 再提示father

此网页显示

son

确定

此网页显示

father

确定

9.4事件对象

9.4.1什么是事件对象

```
eventTarget.onclick = function (event) {}
eventTarget.addEventListener('click', function(event) {})
//这个event就是事件对象，我们还喜欢的写成e或者evt
```

官方解释: event对象代表事件的状态,比如键盘按键的状态鼠标的位置、鼠标按钮的状态。

简单理解:事件发生后, 跟**事件相关的一系列信息数据的集合**都放到这个对象里面, 这个对象就是**事件对象event**,它有很多属性和方法。有了事件才有事件对象。

比如:

- 1.谁绑定了这个事件。
- 2.鼠标触发事件的话,会得到鼠标的相关信息,如鼠标位置。
- 3.键盘触发事件的话,会得到键盘的相关信息,如按了哪个键。

9.4.2事件对象的使用语法

```
eventTarget.onclick = function (event) {}
//这个event就是事件对象，我们还喜欢的写成e或者evt
eventTarget.addEventListener ('click', function (event) {
    //这个event就是事件对象，我们还喜欢的写成e或者evt
})
```

这个event是个形参,系统帮我们设定为事件对象,不需要传递实参过去。

当我们注册事件时, event 对象就会被系统自动创建,并依次传递给事件监听器(事件处理函数) .

9.4.3事件对象的兼容性方案

事件对象本身的获取存在兼容问题:

- 1.标准浏览器中是浏览器给方法传递的参数,只需要定义形参e就可以获取到。
- 2.在IE6~8中,浏览器不会给方法传递参数,如果需要的话,需要到window.event中获取查找。

解决:

```
e=e||window.event;
```

9.4.4事件对象的常见属性和方法

事件对象属性方法	说明
e.target	返回触发事件的对象 标准
e.srcElement	返回触发事件的对象 非标准 ie6-8使用
e.type	返回事件的类型, 比如click mouseover 不带on
e.stopPropagation()	阻止冒泡 标准 方法
e.cancelBubble	该属性阻止冒泡 非标准 ie6-8使用 属性
e.preventDefault()	该方法阻止默认事件(默认行为) 标准 比如不让链接跳转
e.returnValue	该属性阻止默认事件(默认行为) 非标准 ie6-8使用 比如不让链接跳转

e.target使用,返回的是触发事件的对象 (元素) 简单理解为点击的元素

兼容性处理

```
div.onclick = function(e) {  
    e = e || window.event;  
    var target = e.target || e.srcElement;  
    console.log(target);  
}
```

示例: 点击“111”

```
//1.e.target返回的是触发事件的对象 (元素) 简单理解为点击的元素  
//this返回的是绑定事件的对象  
var ul = document.querySelector('ul');  
ul.addEventListener('click', function (e) {  
    console.log(e.target); //li  
    console.log(this); //ul  
})
```

- 111
- 222



```
//currentTarget效果和this一致，返回的是绑定事件的对象，但有兼容性问题，不常用
    console.log(e.currentTarget);
    console.log(this);
```

▶ ...
▶ ...
➤

e.preventDefault()使用,阻止默认行为 让链接不跳转, 提交按钮不提交等

示例:

```
<div>123</div>
<a href="www.baidu.com">百度</a>
<form action="www.baidu.com">
    <input type="button" value="提交">
</form>
<script>
    //常见事件对象的属性和方法
    //1.返回事件类型
    var div = document.querySelector('div');
    div.addEventListener('click', fn);
    div.addEventListener('mouseover', fn);
    div.addEventListener('mouseout', fn);

    function fn(e) {
        console.log(e.type);
    }
    //2.阻止默认行为 让链接不跳转, 提交按钮不提交
    var a = document.querySelector('a')
    //监听
    a.addEventListener('click', function (e) {
        e.preventDefault(); //方法
    })
    //传统
    a.onclick = function (e) {
        // e.preventDefault();
        //也可以利用return false实现, 没有兼容性问题, 但return后面代码无法执行, 且只限
        //于传统的注册方式
        return false;
    }
</script>
```

```
    alert('1');
}
</script>
```

9.5阻止事件冒泡的两种方式

事件冒泡:开始时由最具体的元素接收,然后逐级向上传播到到DOM最顶层节点。

事件冒泡本身的特性,会带来的坏处,也会带来的好处,需要我们灵活掌握。

阻止事件冒泡

- 标准写法:利用事件对象里面的stopPropagation()方法,Propagation意为传播

```
e.stopPropagation()
```

- 非标准写法:IE6-8 利用事件对象cancelBubble属性

示例:

```
var son = document.querySelector('.son');
var father = document.querySelector('.father')
son.addEventListener('click', function (e) {
    alert('son');
    e.stopPropagation();
})
father.addEventListener('click', function () {
    alert('father');
}, false)
```

在son的监听处理程序函数内添加e.stopPropagation();



结果只提示son。注意给谁添加停止冒泡,谁就停止冒泡

阻止事件冒泡的兼容性解决方案

```
if(e && e.stopPropagation) {
    e.stopPropagation();
} else {
    window.event.cancelBubble = true;
}
```

9.6事件委托(代理、委派)

事件冒泡本身的特性,会带来坏处,也会带来好处,需要我们灵活掌握。生活中有如下场景:

咱们班有100个学生,快递员有100个快递,如果一个个的送花费时间较长。同时每个学生领取的时候,也需要排队领取,也花费时间较长,怎么办?

解决方案:快递员把100个快递,委托给班主任,班主任把这些快递放到办公室,同学们下课自行领取即可。

优势:快递员省事,委托给班主任就可以走了。同学们领取也方便,因为相信班主任。

事件委托

事件委托也称为事件代理,在jQuery里面称为事件委派。

事件委托的原理

不是每个子节点单独设置事件监听器,而是事件监听器设置在其父节点上,然后利用冒泡原理影响设置每个子节点。

想要点击ul下的li弹出对话框:给ul注册点击事件,然后利用事件对象的target来找到当前点击的li,因为点击li,事件会冒泡到ul上,ul有注册事件,就会触发事件监听器。

事件委托的作用

我们只操作了一次DOM,提高了程序的性能。

示例:

点击li弹框

```
<ul>
    <li>点击有弹框</li>
    <li>点击有弹框</li>
    <li>点击有弹框</li>
    <li>点击有弹框</li>
</ul>
<script>
    var ul = document.querySelector('ul');
    ul.addEventListener('click', function (e) {
        alert('tan');
    })
</script>
```

无论点击哪个li,都



9.7常用的鼠标事件

9.7.1禁止鼠标右键菜单

contextmenu主要控制应该何时显示上下文菜单,主要用于程序员取消默认的上下文菜单

```
document.addEventListener ('contextmenu', function(e) {
    e.preventDefault();
})
```

9.7.2禁止鼠标选中(selectstart开始选中)

```
document.addEventListener('selectstart', function(e) {  
    e.preventDefault();  
})
```

示例：

我是一段不愿意分享的文字

```
<script>  
//contextmenu 禁用右键菜单  
document.addEventListener('contextmenu', function (e) {  
    e.preventDefault();  
})  
</script>
```

应用 哔哩哔哩 (^-^)つ...

我是一段不愿意分享的文字

只能选中，右键不能出现菜单

```
//selectstart禁止选中  
document.addEventListener('selectstart', function (e) {  
    e.preventDefault();  
})
```

我是一段不愿意分享的文字

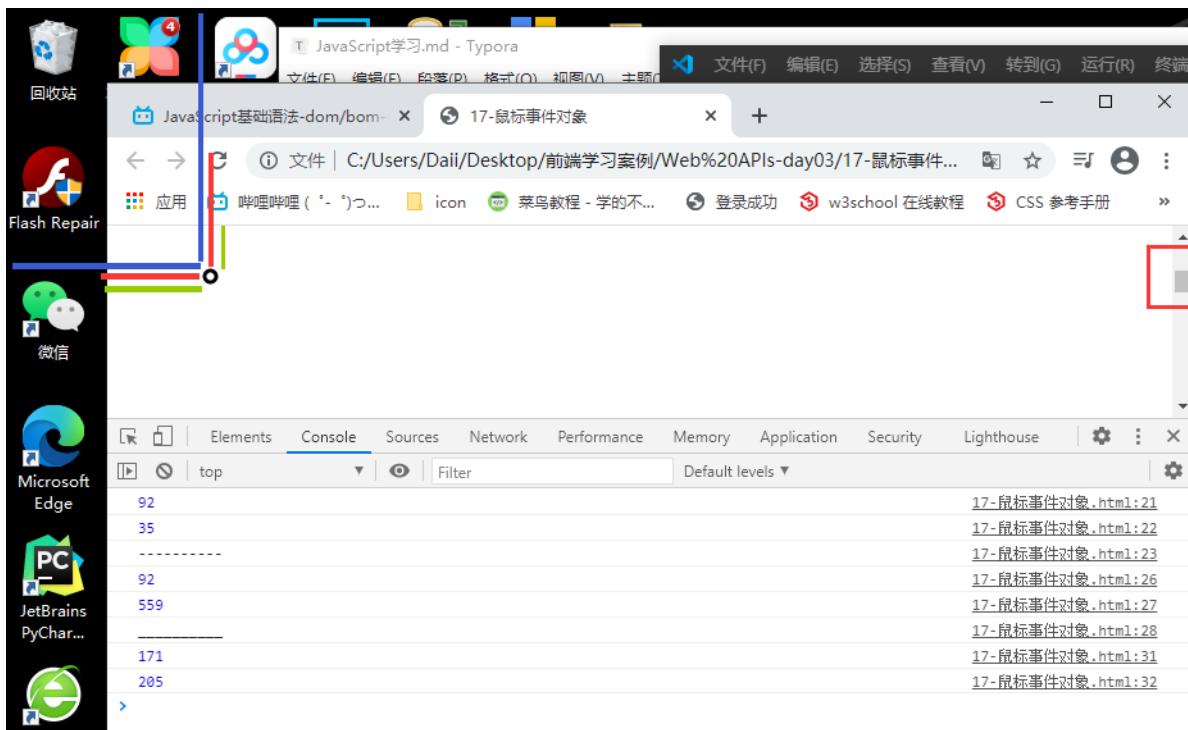
无法选中文字

9.7.3鼠标事件对象

event对象代表事件的状态,跟事件相关的一系列信息的集合。现阶段我们主要是用**鼠标事件对象**
MouseEvent和**键盘事件对象****KeyboardEvent**.

鼠标事件对象	说明
e.clientX	返回鼠标相对于浏览器窗口可视区的X坐标
e.clientY	返回鼠标相对于浏览器窗口可视区的Y坐标
e.pageX	返回鼠标相对于文档页面的X坐标IE9+支持
e.pageY	返回鼠标相对于文档页面的Y坐标IE9+支持
e.screenX	返回鼠标相对于电脑屏幕的X坐标
e.screenY	返回鼠标相对于电脑屏幕的Y坐标

三种坐标点区别



蓝色框为电脑屏幕，绿色框为浏览器，红色框为页面

screenX/Y即点击位置距离电脑屏幕的坐标

clientX/Y即点击位置距离浏览器的坐标

pageX /Y即距离文档页面的坐标，文档页面若有滚动条，会很长，**较常用**

案例：随鼠标移动的小天使

案例分析：

①鼠标不断的移动,使用鼠标移动事件: mousemove

②在页面中移动,给document注册事件

③图片要移动距离,而且不占位置,我们使用绝对定位即可

④核心原理:每次鼠标移动,我们都会获得最新的鼠标坐标,**把这个x和y坐标做为图片的top和left值就可以移动图片**

```

<script>
    var img = document.querySelector('img');
    //核心原理: 每次鼠标移动, 会获得新的鼠标坐标, 将这个x、y坐标作为图片的top和left值
    document.addEventListener('mousemove', function (e) {
        var x = e.pageX;
        var y = e.pageY;
        // console.log(x, y);
        //注意一定要加单位px
        img.style.top = y - 50 + 'px';
        img.style.left = x - 40 + 'px';
    })
</script>
```



注意：

- 1.将获取的鼠标x、y坐标赋值给图片的top和left值时，一定要加单位px
- 2.为了让图片位于移动鼠标点中央，对x、y各减去图片宽度、高度的一半。
- 3.top对应是y， left对应是x，不要弄反
- 3.该案例原理可应用于淘宝商品pc端放大镜看商品图片效果。

9.7.4mouseenter和mouseover的区别

mouseenter鼠标事件

- 当鼠标移动到元素上时就会触发mouseenter事件
- 类似mouseover,它们两者之间的差别是：

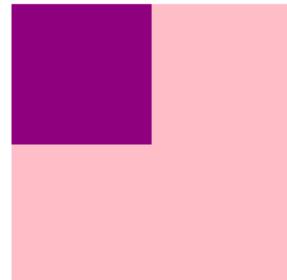
mouseover 鼠标经过自身盒子会触发,经过子盒子还会触发。mouseenter 只会经过自身盒子触发

原因：**mouseenter不会冒泡**，跟mouseenter搭配，**鼠标离开mouseleave同样不会冒泡**

示例：

```
var father = document.querySelector('.father');
father.addEventListener('mouseover', function () {
    console.log(11);
})
```

mouseover，鼠标经过父盒子打印“11”；再次经过子盒子，再打印“11”，因为冒泡触发率父级盒子事件



```
2 11
```

mouseenter，鼠标经过父盒子打印“11”，再次经过子盒子，不打印，因为不会冒泡



9.8 常用键盘事件

9.8.1 常用键盘事件

事件除了使用鼠标触发，还可以使用键盘触发。

键盘事件	触发条件
onkeyup	某个键盘按键被松开时触发
onkeydown	某个键盘按键被按下时触发
onkeypress	某个键盘按键被按下时触发，但是它不识别功能键，比如ctrl shift箭头等

注意：

1. 如果使用`addEventListener`不需要加`on`

2. `onkeypress`和前面2个的区别是，它不识别功能键，比如左右箭头、`shift`、`backspace`等。

3. 三个事件的执行顺序是：`keydown-- keypress --- keyup`

示例：

```
//监听器 1.keyup键盘弹起
document.addEventListener('keyup', function () {
    console.log('keyup');
})
//2.keypress 键盘按下 不包括一些功能键 如ctrl shfit 上下箭头
document.addEventListener('keypress', function () {
    console.log('keypress');
})
//3.keyup 键盘按下 包括一些功能键 如ctrl shfit 上下箭头
document.addEventListener('keydown', function () {
    console.log('keydown');
})
//执行顺序 keydown---keypress---keyup
```

键盘按下“1”

keydown
keypress
keyup

键盘按下“Ctrl”

keydown
keyup

9.8.2 键盘事件对象

键盘事件对象 属性	说明
<code>keyCode</code>	返回该键的ASCII值

注意：

1.onkeydown和onkeyup不区分字母大小写，onkeypress区分字母大小写。

2.在我们实际开发中，我们更多的使用keydown和keyup，它能识别所有的键(包括功能键)

Keypress不识别功能键，但是keyCode属性能区分大小写，返回不同的ASCII值

示例：

```
//keyup keydown事件不区分字母大小写 a和A得到的都是65
document.addEventListener('keydown', function (e) {
    // console.log(e);
    console.log('down:' + e.keyCode); //注意C大写
    //我们可以通过keyCode返回的ASCII码值来判断用户按下了哪个键
    if (e.keyCode == 65) {
        alert('您按下了a键');
    } else {
        alert('您没有按下a键');

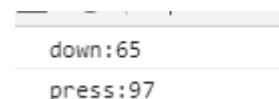
    }
})
//keypress事件区分字母大小写, a 97 A 65
document.addEventListener('keypress', function (e) {
    console.log('press:' + e.keyCode);
})
```

打印鼠标事件对象



```
▼ KeyboardEvent {isTrusted: true, key: "1", code: "Digit1",
  altKey: false
  bubbles: true
  cancelBubble: false
  cancelable: true
  charCode: 0
  code: "Digit1"
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 0
  eventPhase: 0
  isComposing: false
  isTrusted: true
  key: "1"
  keyCode: 49
  location: 0
  metaKey: false}
```

按下a键后 keyCode



```
down:65
press:97
```

案例：模拟京东按键输入内容

当我们按下s键，光标就自动定位到搜索框

案例分析：

①核心思路：检测用户是否按下了s键，如果按下了s键，就把光标定位到搜索框里面

②使用键盘事件对象里面的keyCode判断用户按下的是否是s键

③搜索框获得焦点:使用js里面的focus()方法

```
<input type="text">
<script>
    var search = document.querySelector('input');
    document.addEventListener('keydown', function (e) {
        // console.log(e.keyCode); //可以通过输出s键的keyCode来获得ASCII码, 不用死记ASCII码表
        if (e.keyCode === 83) {
            //focus() 让搜索框获得光标
            search.focus();
        }
    })
</script>
```



问题：搜索框获得了焦点，但将s输入在搜索框中。因为按下键的同时获得焦点，然后将文字输入。

解决方法：将keydown改为keyup 事件，让键盘弹起再获得焦点，这样就不会将字母输入了。

```
document.addEventListener('keyup', function (e){})
```



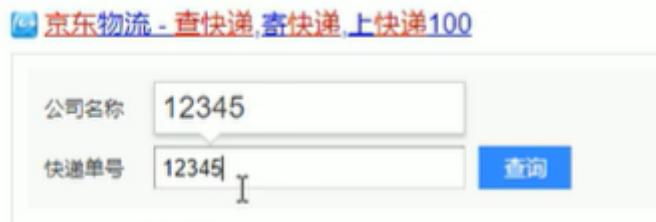
只获得焦点，不输入文字

注意：

- 1.搜索框获得焦点:使用js里面的focus()方法
- 2.可以通过输出s键的keyCode来获得ASCII码，不用死记ASCII码表
- 3.使用keyup事件，让键盘弹起再获得焦点，这样就不会将字母输入了。

案例:模拟京东快递单号查询

要求:当我们在文本框中输入内容时,文本框上面自动显示大字号的内容。



案例分析

- ①快递单号输入内容时，上面的大号字体盒子(con)显示(这里面的字号更大)
- ②表单检测用户输入:给表单添加键盘事件
- ③同时把快递单号里面的值(value)获取过来赋值给con盒子(innerText)做为内容

④如果快递单号里面内容为空，则隐藏大号字体盒子(con)盒子

④注意：keydown和keypress在文本框里面的特点：他们两个事件触发的时候，文字还没有落入文本框中，先触发事件后落入文本框。

⑤keyup事件触发的时候，文字已经落入文本框里面了

⑥当我们失去焦点，就隐藏这个con盒子

⑦当我们获得焦点，并且文本框内容不为空，就显示这个con盒子

```
<div class="search">
    <div class="con"></div>
    <input type="text" value="" placeholder="请输入您的快递单号">
</div>
```

```
<style>
    .search {
        position: relative;
        width: 150px;
        height: 80px;
        margin: 100px;
        /* background-color: pink; */
    }

    .con {
        display: none;
        position: absolute;
        top: -17px;
        left: 0px;
        width: 155px;
        font-size: 18px;
        line-height: 23px;
        border: 1px solid rgba(0, 0, 0, .2);
        box-shadow: 0 2px 4px rgba(0, 0, 0, .2);
    }

    input {
        width: 100%;
        outline: none;
        margin-top: 15px;
    }
```

```
.con::after {
    position: absolute;
    top: 20px;
    left: 10px;
    content: '';
    width: 0;
    height: 0;
    border: 8px solid transparent;
    border-top: 8px solid #fff;
}
```

```
</style>
```

```
<script>
    // ①快递单号输入内容时，上面的大号字体盒子(con)显示(这里面的字号更大)
```

```

// ①表单检测用户输入：给表单添加键盘事件
// ②同时把快递单号里面的值( value )获取过来赋值给con盒子( innerText )做为内容
// ③如果快递单号里面内容为空，则隐藏大号字体盒子(con)盒子
var input = document.querySelector('input');
var con = document.querySelector('.con')
input.addEventListener('keyup', function () {
    // console.log('1');
    // 内容为空，则隐藏大号字体盒子(con)盒子
    if (this.value == '') {
        con.style.display = 'none';
    } else {
        con.style.display = 'block';
        con.innerHTML = this.value;
    }
})
//失去焦点，就隐藏这个con盒子
input.addEventListener('blur', function () {
    con.style.display = 'none';
})
//获得焦点，并且文本框内容不为空，就显示这个con盒子
input.addEventListener('focus', function () {
    if (input.value != '') {
        con.style.display = 'block';
    }
})
</script>

```

请输入您的快递单号

不输入内容和删除完内容时con隐藏

注意：

1.con一定要使用**绝对定位**方式定位到搜索框上面，绝对定位不占用位置，如果只是div默认的垂直流摆放，当con隐藏时会影响后面盒子布局

2.con里面的内容是根据input表单里的value走的，所以进行赋值就好，会随着value动态变化

3.表单的**键盘事件应为keyup**，键盘弹起时触发，此时文本已落入文本框内；

若为keydown键盘按下，刚按下时就触发事件，文本没有落入文本框内，输入第一个数字时，不现实con，输入第二个数字时才会显示第一个数字内容。

若为keypress同样有keydown的问题，还有删除键无法起作用文本，因为无法识别删除键backspace。

4.让con显示前都需要注意判断，input里面的内容是否为空，不为空才显示。

10.BOM简介

10.1什么是BOM

BOM (Browser Object Model)即**浏览器对象模型**,它提供了独立于内容而与**浏览器窗口进行交互**的对象,其核心对象是window。

BOM由一系列相关的对象构成,并且每个对象都提供了很多方法与属性。

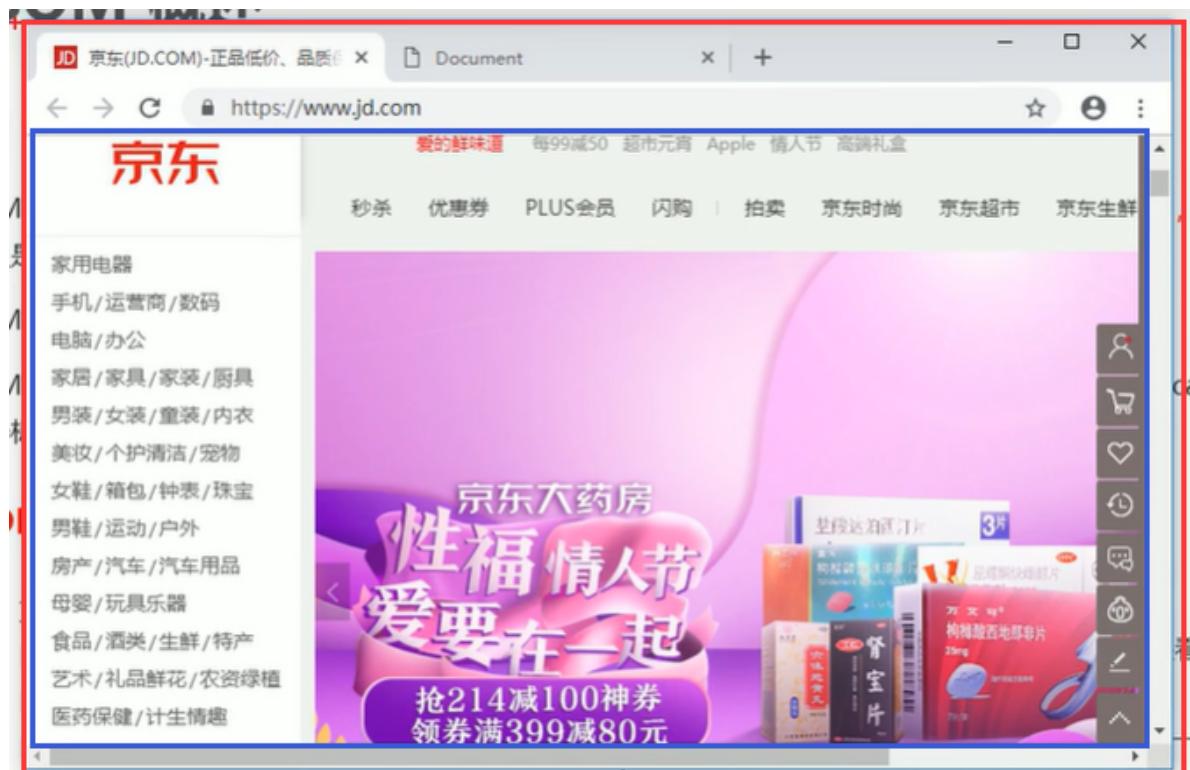
BOM缺乏标准, JavaScript语法的标准化组织是ECMA, DOM的标准化组织是W3C , BOM最初是Netscape 浏览器标准的一部分。

DOM

- 文档对象模型
- DOM就是把「**文档**」当做一个「**对象**」来看待
- DOM的顶级对象是**document**
- DOM 主要学习的是操作页面元素
- DOM 是W3C标准规范

BOM

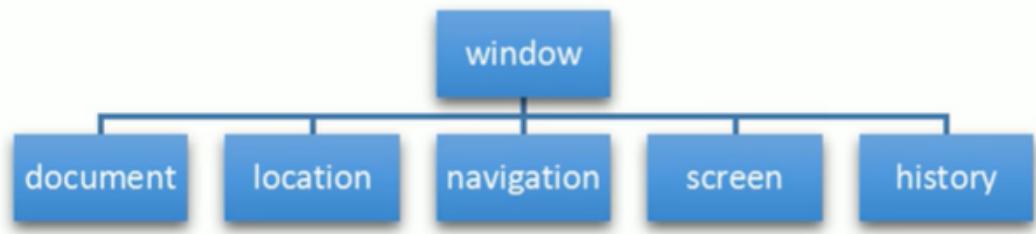
- 浏览器对象模型
- 把「**浏览器**」当做一个「**对象**」来看待
- BOM的顶级对象是**window**
- BOM 学习的是浏览器窗口交互的一些对象
- BOM 是浏览器厂商在各自浏览器上定义的,兼容性较差



红色框为BOM，蓝色框为DOM， BOM包含DOM

10.2 BOM的构成

BOM比DOM更大,它包含DOM。



window对象是浏览器的顶级对象,它具有双重角色。

1.它是JS访问浏览器窗口的一个接口。

2.它是一个全局对象。定义在全局作用域中的变量、函数都会变成window对象的属性和方法。如
window.alert()、window.fn()

在调用的时候可以省略window ,前面学习的对话框都属于window对象方法,如alert()、prompt()等.

注意: **window下的一个特殊属性window.name**, 输出为空

11.window对象的常见事件

11.1 窗口加载事件

```

window.onload = function() {}
或者
window.addEventListener ("load",function(){});
```

window.onload是窗口(页面)加载事件,当文档内容完全加载完成会触发该事件(包括图像、脚本文件、CSS文件等),就调用的处理函数

注意:

1.有了window.onload就可以把JS代码写到页面元素的上方,因为onload是等页面内容全部加载完毕,再去执行处理函数。

2.window.onload传统注册事件方式只能写一次,如果有多个,会以最后一个window.onload为准。唯一性

3.如果使用addEventListener则没有限制

```
document.addEventListener ('DOMContentLoaded',function(){})
```

DOMContentLoaded事件触发时,仅当DOM加载完成,不包括样式表,图片,flash等等。**加载速度比load更快些**。ie9以上才支持

如果页面的图片很多的话,从用户访问到onload触发可能需要较长的时间交互效果就不能实现,必然影响用户的体验,此时用DOMContentLoaded事件比较合适。注意字母大小写

```

//1.传统方式 窗口加载
/* window.onload = function () {
    var btn = document.querySelector('button');
    btn.addEventListener('click', function () {
        alert('111');
    })
} */
//2.监听器方式
window.addEventListener('load', function () {
```

```

var btn = document.querySelector('button');
btn.addEventListener('click', function () {
    alert('111');
})
}

// 
document.addEventListener('DOMContentLoaded', function () {
    alert('222');
})

```

先显示222，再点击按钮显示111

11.2 调整窗口大小事件

```

window.onresize = function() {}

window.addEventListener("resize", function() {});

```

window.onresize是调整窗口大小加载事件,当触发时就调用的处理函数。

注意:

1.只要窗口大小发生像素变化,就会触发这个事件。

2.我们经常利用这个事件完成响应式布局,如窗口小于某种程度,就隐藏盒子,大于某种程度,就显示盒子。window.innerWidth 当前屏幕的宽度

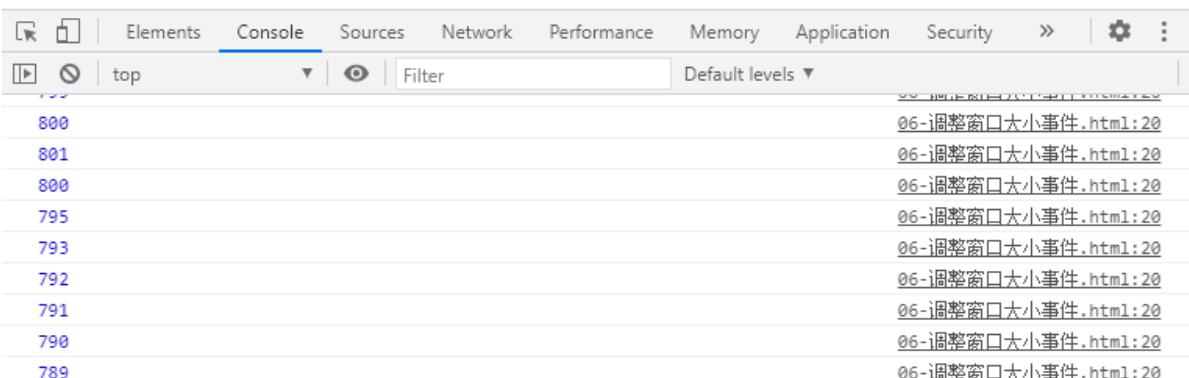
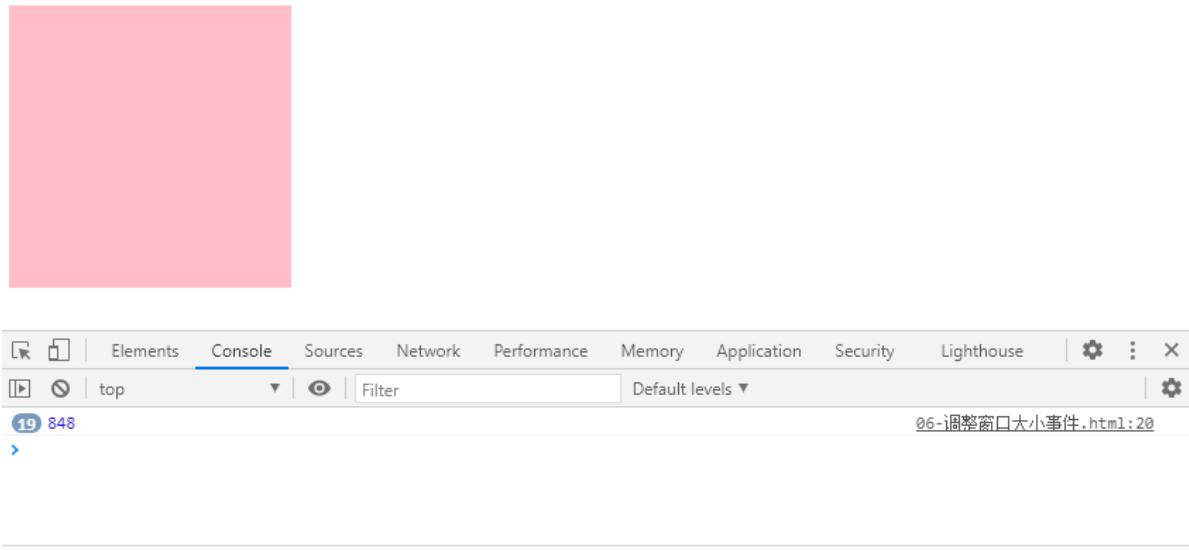
示例:

当浏览器宽度小于等于800px时,隐藏盒子;反之显示

```

<style>
div {
    width: 200px;
    height: 200px;
    background-color: pink;
}
</style>
<script>
window.addEventListener('load', function () {
    window.addEventListener('resize', function () {
        console.log(window.innerWidth);
        var div = document.querySelector('div');
        if (window.innerWidth <= 800) {
            div.style.display = 'none';
        } else {
            div.style.display = 'block';
        }
    })
})
</script>

```



12. 定时器

12.1 两种定时器

window对象给我们提供了2个非常好用的方法定时器。

- setTimeout()
- setInterval()

12.2 setTimeout()定时器

```
window.setTimeout (调用函数, [ 延迟的毫秒数]) ;
```

setTimeout()方法用于设置一个定时器,该定时器在定时器到期后执行调用函数。类似于定时炸弹,一次 性。

setTimeout()这个调用函数我们也称为**回调函数callback**

普通函数是按照代码顺序直接调用。

而这个函数，**需要等待时间**，时间到了才去调用这个函数，因此称为回调函数。

简单理解：回调，就是回头调用的意思。上一件事干完，再**回头再调用**这个函数。

以前我们讲的element.onclick = function(){}或者element.addEventListener("click", fn);里面的函数也是回调函数。

示例：

```
/* setTimeout(function () {
    console.log('时间到了');
}, 2000) */

function callback() {
    console.log('爆炸了');
}
setTimeout(callback, 3000);
// setTimeout('callback()', 4000);

// 起名字
var timer1 = setTimeout(callback, 1000);
var timer2 = setTimeout(callback, 2000);
```

注意：

- 1.window在调用时可以省略
- 2.延时时间单位是毫秒，也可省略，省略默认为0
- 3.调用函数可以是匿名函数也可以写函数名。还有另一种写法'函数名()' 麻烦，不常用
- 4.页面中可能有很多个定时器，我们经常给定时器加标识符(名字)如timer1

案例：5s后关闭广告

案例分析：

①核心思路：5秒之后，就把这个广告隐藏起来

②用定时器setTimeout

```

<script>
    var ad = document.querySelector('img');
    setTimeout(function () {
        ad.style.display = 'none';
    }, 5000)
</script>
```



5s后广告消失

12.3停止setTimeout()定时器

```
window.clearTimeout (timeoutID)
```

clearTimeout ()方法取消了先前通过调用setTimeout ()建立的定时器。

注意:

1. window可以省略。
2. timerID即我们给定时器取的名字，如timer

示例:

```
<button>停止定时器</button>
<script>
    var timer = setTimeout(function () {
        console.log('爆炸了');
    }, 5000);
    var btn = document.querySelector('button');
    btn.addEventListener('click', function () {
        clearTimeout(timer);
    })
</script>
```

在定时器倒计时完毕前，点击定时器按钮后，就不再“爆炸了”

12.4setInterval()定时器

```
window.setInterval (回调函数, [间隔的毫秒数]);
```

setInterval()方法重复调用一个函数,每隔这个时间,就去调用一次回调函数。

注意:

- 1.window可以省略。
- 2.这个调用函数可以**直接写函数,或者写函数名**或者采取字符串'函数名()'三种形式。
- 3.间隔的毫秒数省略默认是0,如果写,必须是毫秒,表示每隔多少毫秒就自动调用这个函数。
- 4.因为定时器可能有很多,所以我们经常给定时器赋值一个标识符。

与setTimeout()最大的区别:

- 1.setTimeout()是**一次性的**,延时时间到了就调用函数,只调用一次,这个定时器就结束了
- 2.setInterval()是**可重复利用的**,可重复多次调用

示例:

```
setInterval(function () {
    console.log('继续输出');
}, 1000)
```



案例：时分秒倒计时效果

案例分析:

- ①这个倒计时是不断变化的,因此需要定时器来自动变化(setInterval)
- ②三个黑色盒子里面分别存放时分秒
- ③三个黑色盒子利用innerHTML放入计算的小时分钟秒数
- ④第一次执行也是间隔毫秒数,因此刚刷新页面会有**空白**
- ⑤最好采取封装函数的方式,这样**可以先调用一次这个函数,防止刚开始刷新页面有空白问题**

```
<div class="t">
    <span class="h">1</span>
    <span class="m">2</span>
    <span class="s">3</span>
</div>
<script>
    var hour = document.querySelector('.h');
    var minute = document.querySelector('.m');
    var second = document.querySelector('.s');
    var inputTime = +new Date('2021-3-19 18:00:00');//将目标时间放在外面,在外面
修改时间,尽量保持函数不变
    countDown(); //先调用一次函数,防止刷新页面出现1s的空白
    setInterval(countDown, 1000); //每个1s执行一次
    function countDown() {
        var nowTime = +new Date();
        hour.textContent = nowTime.getHours();
        minute.textContent = nowTime.getMinutes();
        second.textContent = nowTime.getSeconds();
    }
</script>
```

```

var times = (inputTime - nowTime) / 1000;
var h = parseInt(times / 60 / 60 % 24); //时
h = h < 10 ? '0' + h : h; //补零
hour.innerHTML = h;
var m = parseInt(times / 60 % 60); //分
m = m < 10 ? '0' + m : m; //补零
minute.innerHTML = m;
var s = parseInt(times % 60); //秒
s = s < 10 ? '0' + s : s; //补零
second.innerHTML = s;
}
</script>

```



注意：

- 1.将目标时间放在外面，在外面修改时间，尽量保持函数不变
- 2.通过定时器setInterval每隔1s执行一次格式化时分秒函数实现倒计时效果。
- 3.第一次执行也是间隔毫秒数，因此刚刷新页面会有空白。解决方法：在定时器执行前，先调用一次格式化时分秒函数。

12.5停止setInterval()定时器

```
window.clearInterval (intervalID);
```

clearInterval ()方法取消了先前通过调用setInterval ()建立的定时器。

注意：

- 1.window可以省略。
- 2.里面的参数intervalID就是定时器的标识符。
- 3.定时器名字一般需要外写。

示例：

```

<button class="start">开始计时器</button>
<button class="end">停止计时器</button>
<script>
    var start = document.querySelector('.start');
    var end = document.querySelector('.end')
    start.addEventListener('click', function () {
        var timer = setInterval(function () {
            console.log('你好');
        }, 1000)
    })

    end.addEventListener('click', function () {
        clearInterval(timer);
    })
</script>

```

```
</script>
```

5 你好

② ► Uncaught ReferenceError: timer is not defined
at HTMLButtonElement.<a...erval().html:24:27)

4 你好

出现该问题的原因：timer定义在开始计时器的函数内，是局部变量。

解决方法：**将timer放在外面，作为全局变量**。这样就能实现停止定时器效果了。

```
var timer = null; //null是一个对象
start.addEventListener('click', function () {
    timer = setInterval(function () {
        console.log('你好');
    }, 1000)
})
```

注意：

clearInterval (intervalID);中的参数即**定时器的名字一定要设置为全局变量**，否则无法调用。并且**赋值为null**，若不赋值会出现undefined等问题，而null又是个空对象，非常合适。

```
var timer=null;
```

案例:发送短信

点击按钮后,该按钮60秒之内不能再次点击,防止重复发送短信



案例分析：

- ①按钮点击之后，会禁用disabled为true
- ②同时按钮里面的内容会变化,注意button里面的内容通过innerHTML修改
- ③里面秒数是有变化的,因此需要用到定时器
- ④定义一个变量,在定时器里面,不断递减
- ⑤如果变量为0说明到了时间,我们需要停止定时器,并且复原按钮初始状态，并重新定义倒计时秒数。

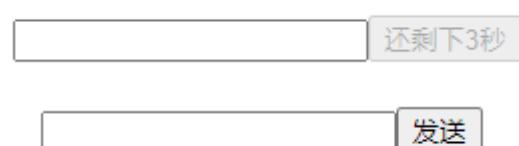
```
var btn = document.querySelector('button');
btn.addEventListener('click', function () {
    btn.disabled = true;//禁用按钮
    var time = 3;//重新计时
    var timer = setInterval(function () {
        if (time == 0) {//倒计时为0, 停止计时, 恢复按钮
            clearInterval(timer);
            btn.disabled = false;
            btn.innerHTML = '发送';
        } else {//倒计时不为0, 开始倒计时
            btn.innerHTML = '还剩下' + time + '秒';
            time--;
        }
    }, 1000)
})
```

```
        }
    }, 1000)
})
```

注意其中逻辑和业务要求：

1.点击发送后，禁用按钮->定时器开始倒计时变化秒数（注意秒数为0时需要停止定时+恢复按钮的功能和文字）->秒数不为0时倒计时变化秒数）

2.停止定时器要注意倒计时时间应变为之前的数字，可以通过将倒计时数放入点击事件后，定时器之前完成，**注意time=3的位置。**



计时完毕后

12.6 this

this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁，一般情况下this的最终指向的是那个调用它的对象

1.全局作用域或者普通函数中this指向全局对象window (注意定时器里面的this指向window)

```
// 1.全局作用域或者普通函数中this指向全局对象window (注意定时器里面的this指向window)
console.log(this); //window
function fn() {
    console.log(this);
}
window.fn(); //window
setTimeout(function() {
    console.log(this); //window
}, 1000);
```

2.方法调用中谁调用this指向谁

```
// 2.方法调用中谁调用this指向谁
var o={
    sayHi: function() {
        console.log(this); // this指向的是o这个对象
    }
}
o.sayHi();

var btn = document.querySelector('button');
btn.onclick = function() {
    console.log(this); // this指向的是btn这个按钮
}
```

3.构造函数中this指向构造函数的实例

```
//3.构造函数中this指向构造函数的实例
function Fun() {
    console.log(this); // this指向的是fun实例对象
}
var fun = new Fun();
```

13.JS执行机制

13.1 JS是单线程

JavaScript语言的一大特点就是**单线程**,也就是说,**同一个时间只能做一件事**。这是因为Javascript这门脚本语言诞生的使命所致——JavaScript 是为处理页面中用户的交互,以及操作DOM而诞生的。比如我们对某个DOM元素进行添加和删除操作,不能同时进行。应该先进行添加,之后再删除。

单线程就意味着,所有任务需要排队,前一个任务结束,才会执行后一个任务。这样所导致的问题是:如果JS执行的时间过长,这样就会造成页面的渲染不连贯,导致页面渲染加载阻塞的感觉。

13.2 同步和异步基本概念

为了解决这个问题,利用多核CPU的计算能力, HTML5提出Web Worker标准,允许JavaScript脚本创建多个线程。于是, JS中出现了**同步**和**异步**。

同步

前一个任务结束后再执行后一个任务,程序的执行顺序与任务的排列顺序是一致的、 同步的。比如做饭的同步做法:我们要烧水煮饭,等水开了(10分钟之后) .再去切菜,炒菜。

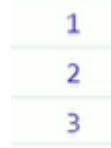
异步

你在做一件事情时,因为这件事情会花费很长时间,在做这件事的同时,你还可以去处理其他事情。比如做饭的异步做法,我们在烧水的同时,利用这10分钟, 去切菜,炒菜。

他们的本质区别:这条流水线上各个流程的执行顺序不同。

示例1:

```
console.log(1);
setTimeout (function() {
    console.log(3);
}, 1000);
console.log(2);
```



第二段代码花费时间比较长, 但后面代码不会等待它运行完再运行, 采取异步, 在等待同时运行第三段代码, 再运行第二段代码

所以注意结果是1 2 3, 不是1 3 2

13.3 同步和异步执行顺序

示例2:

```
console.log(1);
setTimeout (function () {
    console.log (3);
},0);
console.log (2);
```

注意：虽然第二段代码的计时为0毫秒，但输出结果仍为1 2 3

同步任务

同步任务都在主线程上执行,形成一个**执行栈**。

异步任务

JS的异步是通过回调函数实现的。

一般而言,异步任务有以下三种类型

- 1、普通事件,如click, resize等
- 2、资源加载,如load, error等
- 3、定时器,包括setInterval, setTimeout 等

异步任务相关**回调函数**添加到**任务队列**中(任务队列也称为消息队列)。

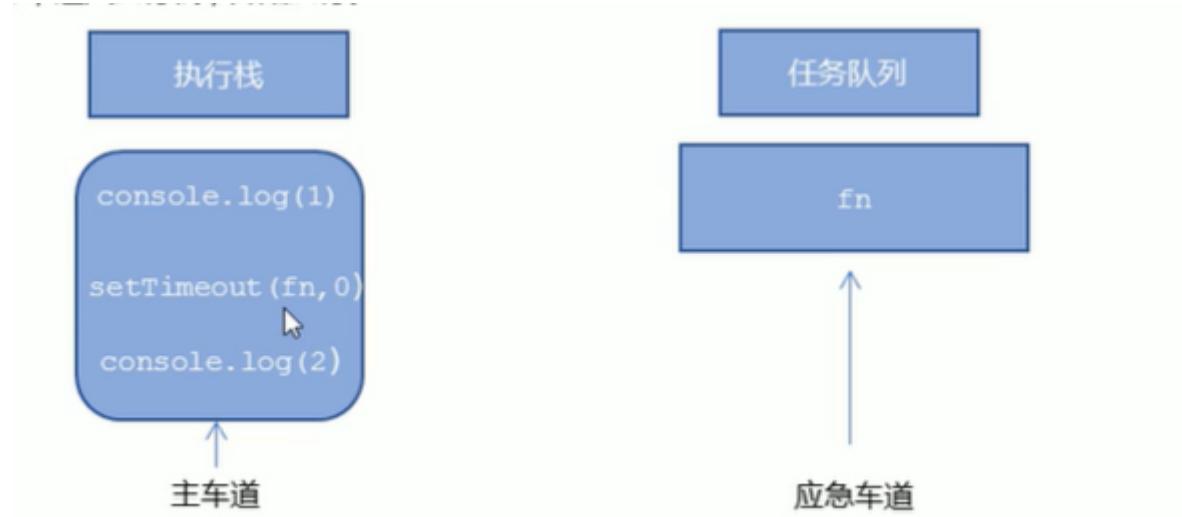
13.4 JS执行机制

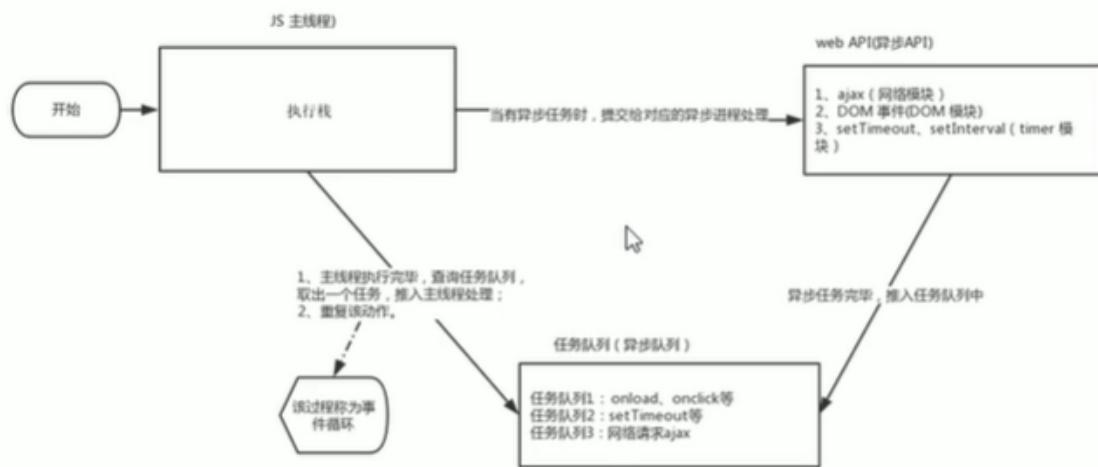
1.先执行**执行栈**中的同步任务。

2.异步任务(回调函数)放入任务队列中。

3.一旦执行栈中的所有同步任务执行完毕,系统就会按次序读取**任务队列**中的异步任务,于是被读取的异步任务结束等待状态,进入执行栈,开始执行。

如示例2, 可以将执行栈看为主车道, 任务队列看作应急车道





由于主线程不断的重复获得任务、执行任务、再获取任务、再执行,所以这种机制被称为**事件循环**(event loop)。

先依次处理执行栈里的任务,若遇到异步并触发了就提交给对应的**异步进程处理**,等待触发条件后将其放入任务队列里。

处理完执行栈里的任务再去查询任务队列里是否有任务,有则**取出一个放入执行栈**,重复。

14.location对象

14.1 什么是location对象

window对象给我们提供了一个**location**属性用于**获取或设置窗体的URL**,并且可以用于**解析URL**.因为这个属性返回的是一个对象,所以我们将这个属性也称为**location对象**。

14.2 URL

统一资源定位符(Uniform Resource Locator, URL)是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的URL,它包含的信息指出文件的位置以及浏览器应该怎么处理它。

URL的一般语法格式为:

```
protocol: //host [:port ]/path/ [ ?query]# fragment
http: //www. itcast.cn/ index . html ?name= andy&age=18#link
```

组成	说明
protocol	通信协议常用的http,ftp,mailto等
host	主机(域名) www.itheima.com
port	端口号可选,省略时使用方案的默认端口如http的默认端口为80
path	路径由零或多个/符号隔开的字符串,一般用来表示主机上的一个目录或文件地址
query	参数以键值对的形式,通过&符号分隔开来
fragment	片段#后面内容,常见于链接锚点

14.3 location对象的属性

location对象属性	返回值
location.href	获取或者设置整个URL
location.host	返回主机(域名) www.itheima.com
location.port	返回端口号, 如果未写返回空字符串
location.pathname	返回路径
location.search	返回参数
location.hash	返回片段, #后面内容, 常见于链接锚点

重点记住: href 和search

案例1：5s后自动跳转链接

案例分析：

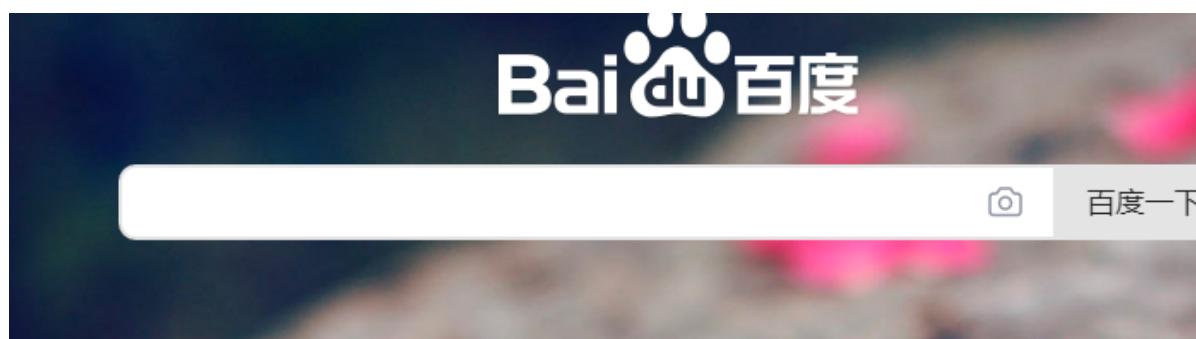
- ①利用定时器做倒计时效果
- ②时间到了,就跳转页面。使用location.href

```

<button>点击</button>
<div></div>
<script>
    var btn = document.querySelector('button');
    var div = document.querySelector('div');
    btn.addEventListener('click', function () {
        location.href = 'https://www.baidu.com';//注意是整个url, 不要忘了通信协议
    })
    var time = 5;//倒计时时间注意得是全局变量
    setInterval(function () {
        if (time == 0) {
            location.href = 'https://www.baidu.com';
        } else {
            div.innerHTML = time + '秒后自动跳转链接';
            time--;
        }
    }, 1000)
</script>

```

点击
4秒后自动跳转链接



注意：

- 1.location.href是整个url，不要忘了通信协议如http
- 2.页面跳转是在当前页面，因为是将当前页面的url更改了

案例2:获取URL参数数据

主要练习数据在不同页面中的传递。

实现：



案例分析：

- ①第一个登录页面,里面有提交表单, action 提交到index.html页面
- ②第二个页面,可以使用第一个页面的参数,这样实现了一个数据不同页面之间的传递效果
- ③第二个页面之所以可以使用第一个页面的数据,是利用了URL里面的location.search参数
- ④在第二个页面中,需要把这个参数提取。
- ⑤第一步去掉?利用substr
- ⑥第二步利用=号分割键和值 split('=')

login.html

```
<form action="index.html">
    <!-- 注意name属性里一定要填, 否则没有参数 -->
    用户名: <input type="text" name="uname" id="">
    <input type="submit" value="登录">
</form>
```

注意：

- 1.form表单
- 2.text文本框name属性里一定要填, 否则无法没有参数

index.html

```

<div></div>
<script>
    var div = document.querySelector('div');
    console.log(location.search); //?uname=aaa
    //1.去掉? substr('起始的索引号',截取几个字符), 第二个参数不填, 默认截取到末尾
    var params = location.search.substr(1);
    console.log(params); //uname=aaa
    //2.split('='),将字符存储为数组, 以=分隔  uname也需要, 所以完整取, 而不是单取aaa
    var arr = params.split('=');
    console.log(arr); //["uname", "aaa"]
    console.log(arr[1]); //aaa
    div.innerHTML = arr[1] + '欢迎你';
</script>

```

注意：

1.去掉?, 使用ubstr('起始的索引号',截取几个字符), 第二个参数不填, 默认截取到末尾

2.split('='),将字符存储为数组, 以=分隔



用户名：

 应用  哔哩哔哩

aaa欢迎你

14.4 location对象的方法

location对象方法	返回值
location.assign()	跟href一样, 可以跳转页面(也称为重定向页面)
location.replace()	替换当前页面, 因为不记录历史, 所以不能后退页面
location.reload()	重新加载页面, 相当于刷新按钮或者f5, 如果参数为true强制刷新ctrl+f5

示例：

```

<button>点击</button>
<script>
    var btn = document.querySelector('button');
    btn.addEventListener('click', function () {
        location.assign('https://www.baidu.com');
        // location.replace('https://www.baidu.com');
        // location.reload();
    })
</script>

```



 应用  哔哩哔哩 (^-^)つ...

location.replace('https://www.baidu.com');

不能后退页面

```
location.reload();
```

同刷新效果

15.navigator对象

navigator对象包含有关浏览器的信息,它有很多属性,我们最常用的是userAgent ,该属性可以返回由客户机发送服务器的user-agent头部的值。

下面前端代码可以判断用户是手机还是电脑终端打开页面,实现跳转。会调用即可

```
if (navigator.userAgent.match(/(phone|pad|pod|iPhone|iPod|ios|iPad|Android|Mobile|BlackBerry|IEMobile|MQQBrowser|JUC|Fennec|wOSBrowser|BrowserNG|webOS|Symbian|Windows Phone)/i)) {
    window.location.href ="";//手机端页面
} else {
    window.location.href ="";//电脑端页面
}
```

16.history对象

window对象给我们提供了一个history对象,与浏览器历史记录进行交互。该对象包含用户(在浏览器窗口中)访问过的URL。

history对象方法	作用	举例
back()	可以后退功能	history.back()
forward()	前进功能	history.forward()
go(参数)	前进后退功能, 参数如果是1前进1个页面; 如果是-1后退1个页面	history.go(1)

history对象一般在实际开发中比较少用,但是会在一些OA办公系统中见到。