前端基础进阶-JavaScript核心 DOM BOM操作

事件高级

注册事件（绑定事件）

- 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
    - 方法监听注册方式

传统注册方式	方法监听注册方式
• 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	• w3c 标准 推荐方式 • addEventListener() 它是一个方法 • 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

```
<body>
    <button>传统注册事件</button>
    <button>方法监听注册事件</button>
    <button>ie9 attachEvent</button>
    <script>
        var btns = document.querySelectorAll('button');
        // 1. 传统方式注册事件
        btns[0].onclick = function() {
            alert('hi');
        }
        btns[0].onclick = function() {
                alert('hao a u');
            }
            // 2. 事件侦听注册事件 addEventListener
            // (1) 里面的事件类型是字符串 必定加引号 而且不带on
            // (2) 同一个元素 同一个事件可以添加多个侦听器（事件处理程序）
        btns[1].addEventListener('click', function() {
            alert(22);
        })
        btns[1].addEventListener('click', function() {
                alert(33);
            })
            // 3. attachEvent ie9以前的版本支持
        btns[2].attachEvent('onclick', function() {
            alert(11);
        })
    </script>
</body>
```

- 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，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，我们再进一步学习
- 1.3 attachEvent 事件监听方式
  - eventTarget.attachEvent(eventNameWithOn, callback)
  - 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
- 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;
      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

删除事件（解绑事件）

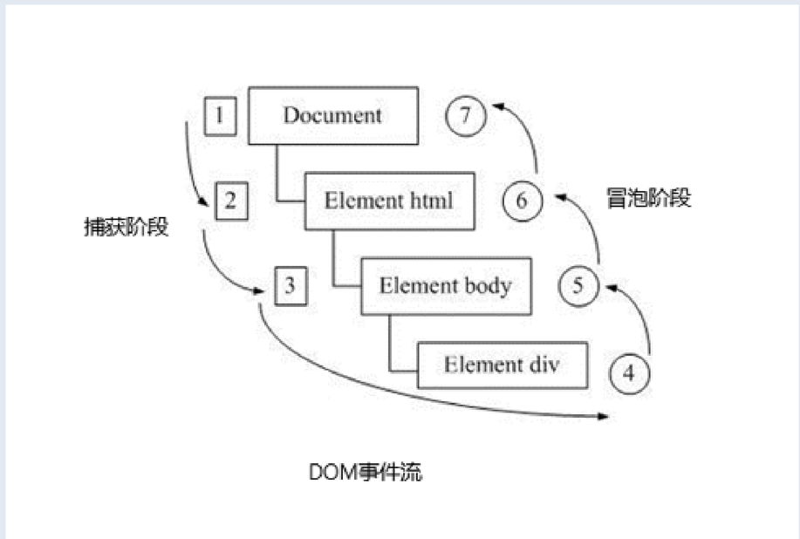
- 2.1 删除事件的方式
  - 1. 传统注册方式 — eventTarget.onclick = null;
  - 2. 方法监听注册方式
    - eventTarget.removeEventListener(type, listener[, useCapture]);
    - eventTarget.detachEvent(eventNameWithOn, callback);
- 2.2 删除事件兼容性解决方案

```
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;
 }
```

```
    <title>Document</title>
    <style>
        div {
            width: 100px;
            height: 100px;
            background-color: pink;
        }
    </style>
</head>

<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <script>
        var divs = document.querySelectorAll('div');
        divs[0].onclick = function() {
                alert(11);
                // 1. 传统方式删除事件
                divs[0].onclick = null;
            }
            // 2. removeEventListener 删除事件
        divs[1].addEventListener('click', fn) // 里面的fn 不需要调用加小括号

        function fn() {
            alert(22);
            divs[1].removeEventListener('click', fn);
        }
        // 3. detachEvent
        divs[2].attachEvent('onclick', fn1);

        function fn1() {
            alert(33);
            divs[2].detachEvent('onclick', fn1);
        }
    </script>
</body>
```

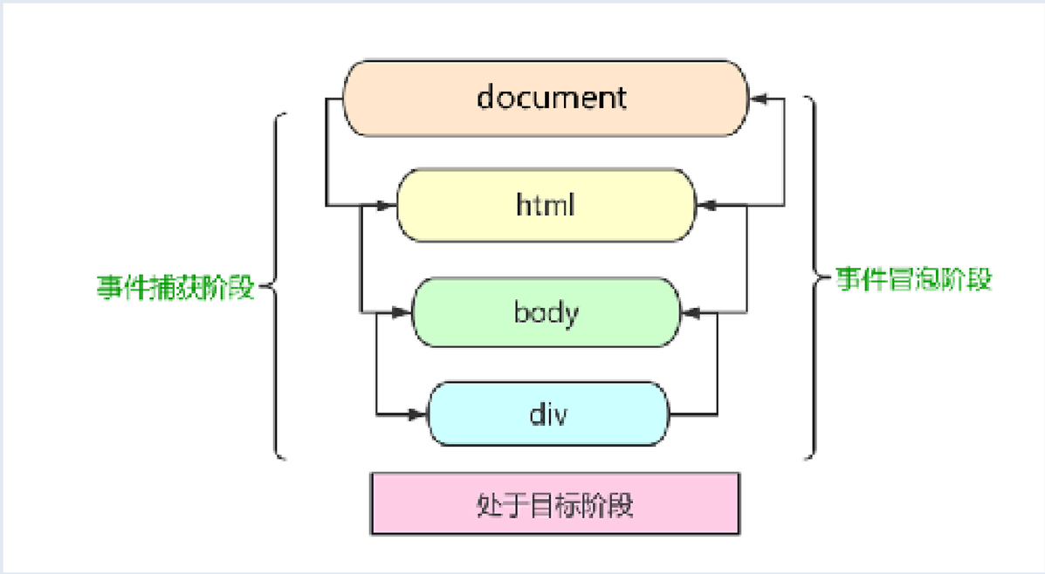
DOM事件流

- 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。



比如我们给一个div 注册了点击事件

- DOM 事件流分为3个阶段：
  - 1. 捕获阶段
  - 2. 当前目标阶段
  - 3. 冒泡阶段
- 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。
- 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。



- 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
  - 注意
    - JS 代码中只能执行捕获或者冒泡其中的一个阶段。
    - onclick 和 attachEvent 只能得到冒泡阶段。
    - 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。
    - 实际开发中我们很少使用事件捕获，我们更关注事件冒泡。
    - 有些事件是没有冒泡的，比如 onblur、onfocus、onmouseenter、onmouseleave
    - 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，我们后面讲解

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        .father {
            overflow: hidden;
            width: 300px;
            height: 300px;
            margin: 100px auto;
            background-color: pink;
            text-align: center;
        }

        .son {
            width: 200px;
            height: 200px;
            margin: 50px;
            background-color: purple;
            line-height: 200px;
            color: #fff;
        }
    </style>
</head>

<body>
    <div class="father">
        <div class="son">son盒子</div>
    </div>
    <script>
        // dom 事件流 三个阶段
        // 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（ie） 只能得到冒泡阶段。
        // 3. 捕获阶段 如果addEventListener 第三个参数是 true 那么则处于捕获阶段  document -> html -> body -> father -> son
        // var son = document.querySelector('.son');
        // son.addEventListener('click', function() {
        //     alert('son');
        // }, true);
        // var father = document.querySelector('.father');
        // father.addEventListener('click', function() {
        //     alert('father');
        // }, true);
        // 4. 冒泡阶段 如果addEventListener 第三个参数是 false 或者 省略 那么则处于冒泡阶段  son -> father ->body -> html -> document
        var son = document.querySelector('.son');
        son.addEventListener('click', function() {
            alert('son');
        }, false);
        var father = document.querySelector('.father');
        father.addEventListener('click', function() {
            alert('father');
        }, false);
        document.addEventListener('click', function() {
            alert('document');
        })
    </script>
</body>
```

概要

事件对象

- 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。
    - 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。
- 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）。
- 4.3 事件对象的兼容性方案
  - 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;

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

<body>
    <div>123</div>
    <script>
        // 事件对象
        var div = document.querySelector('div');
        div.onclick = function(e) {
                // console.log(e);
                // console.log(window.event);
                // e = e || window.event;
                console.log(e);


            }
            // div.addEventListener('click', function(e) {
            //         console.log(e);

        //     })
        // 1. event 就是一个事件对象 写到我们侦听函数的 小括号里面 当形参来看
        // 2. 事件对象只有有了事件才会存在，它是系统给我们自动创建的，不需要我们传递参数
        // 3. 事件对象 是 我们事件的一系列相关数据的集合 跟事件相关的 比如鼠标点击里面就包含了鼠标的相关信息，鼠标坐标啊，如果是键盘事件里面就包含的键盘事件的信息 比如 判断用户按下了那个键
        // 4. 这个事件对象我们可以自己命名 比如 event 、 evt、 e
        // 5. 事件对象也有兼容性问题 ie678 通过 window.event 兼容性的写法  e = e || window.event;
    </script>
</body>
```

- 4.4 事件对象的常见属性和方法
  - e.target 和 this 的区别：
    this 是事件绑定的元素， 这个函数的调用者（绑定这个事件的元素）
    e.target 是事件触发的元素

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.cancelBubble	该属性阻止冒泡 非标准 ie6-8使用
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.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.stopPropagation()	阻止冒泡 标准

阻止事件冒泡

- 5.1 阻止事件冒泡的两种方式
- 5.2 阻止事件冒泡的兼容性解决方案

事件委托（代理、委派）

常用的鼠标事件

常用的键盘事件