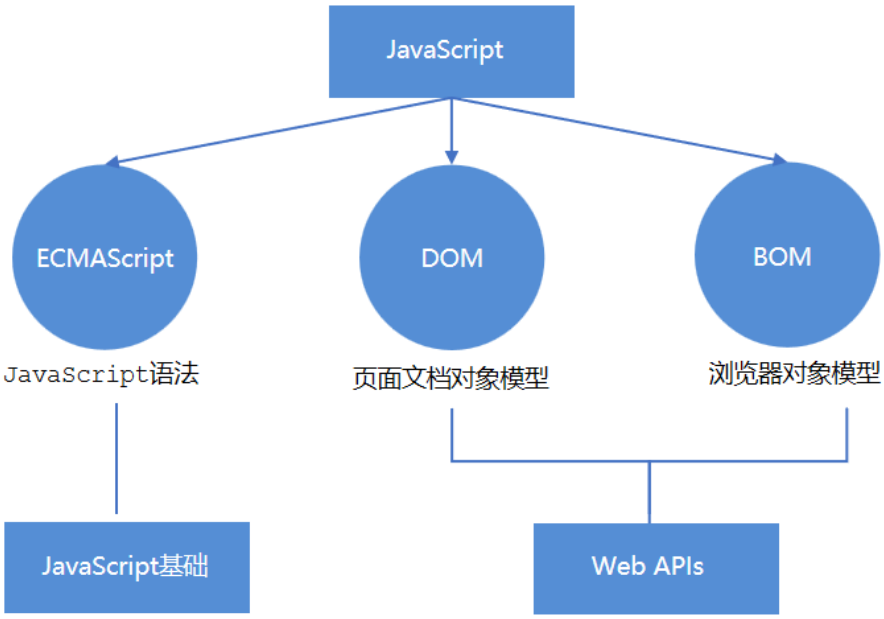
Web APIs 简介

一、Web APIs 和 JS 基础关联性

1、JS的组成



JS 基础阶段

- 我们学习的是 ECMAScript 标准规定的基本语法
- 要求同学们掌握 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

二、API和Web API

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。

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、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

总结

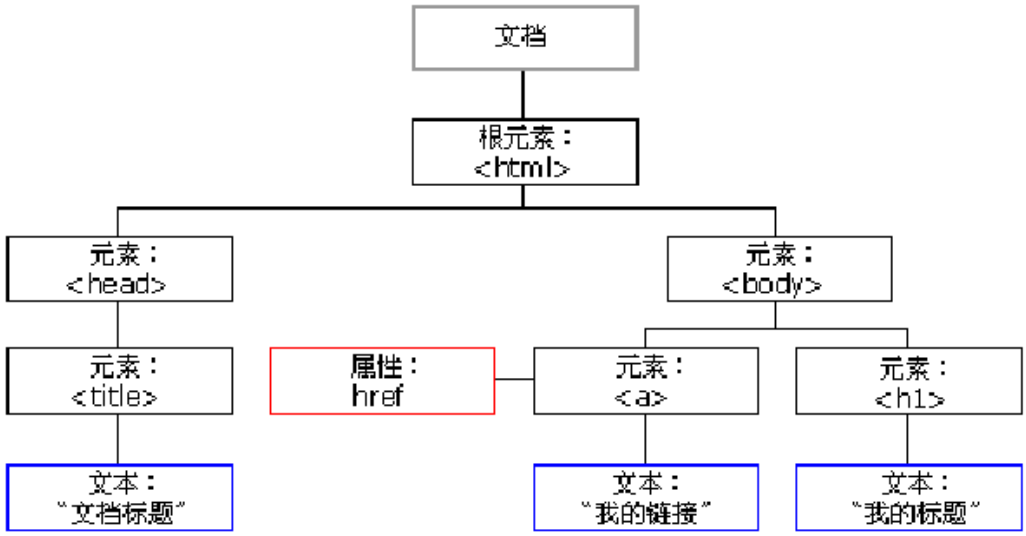
1. API 是为我们程序员提供的一个接口，帮助我们实现某种功能，我们会使用就可以了，不必纠结内部如何实现
2. **Web API 一般都有输入和输出（函数的传参和返回值），Web API 很多都是方法（函数）**
3. Web API 主要是针对于浏览器提供的接口，主要针对于浏览器做交互效果。
4. 学习 Web API 可以结合前面学习内置对象方法的思路学习

DOM

一、DOM简介

> 文档对象模型（Document Object Model，简称 DOM），是 W3C 组织推荐的处理可扩展标记语言（HTML或者XML）的标准编程接口。
>
> W3C 已经定义了一系列的 DOM 接口，通过这些 DOM 接口可以改变网页的内容、结构和样式。

1、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

二、获取元素

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:
>
> 1. 根据 ID 获取
> 2. 根据标签名获取
> 3. 通过 HTML5 新增的方法获取
> 4. 特殊元素获取

2、根据ID获取

> 使用 getElementById() 方法可以获取带有 ID 的元素对象。

```
document.getElementById('id');
```

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

3、根据标签名获取

> 使用 getElementsByTagName() 方法可以返回带有指定标签名的对象的集合。

```
document.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
> 3. 如果获取不到元素,则返回为空的伪数组(因为获取不到对象)

> 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。

4、通过 HTML5 新增的方法获取

```
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
```

注意:

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

5、获取特殊元素（body，html）

获取body元素

```
1. doucumnet.body  // 返回body元素对象
```

获取html元素

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

三、事件基础

> 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、事件三要素

1. 事件源 （谁）
2. 事件类型 （什么事件）
3. 事件处理程序 （做啥）

2、执行事件的步骤

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

3、常见鼠标事件

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

四、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：

1、改变元素内容

```
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**

2、常用属性操作

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

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
```

4、样式属性操作

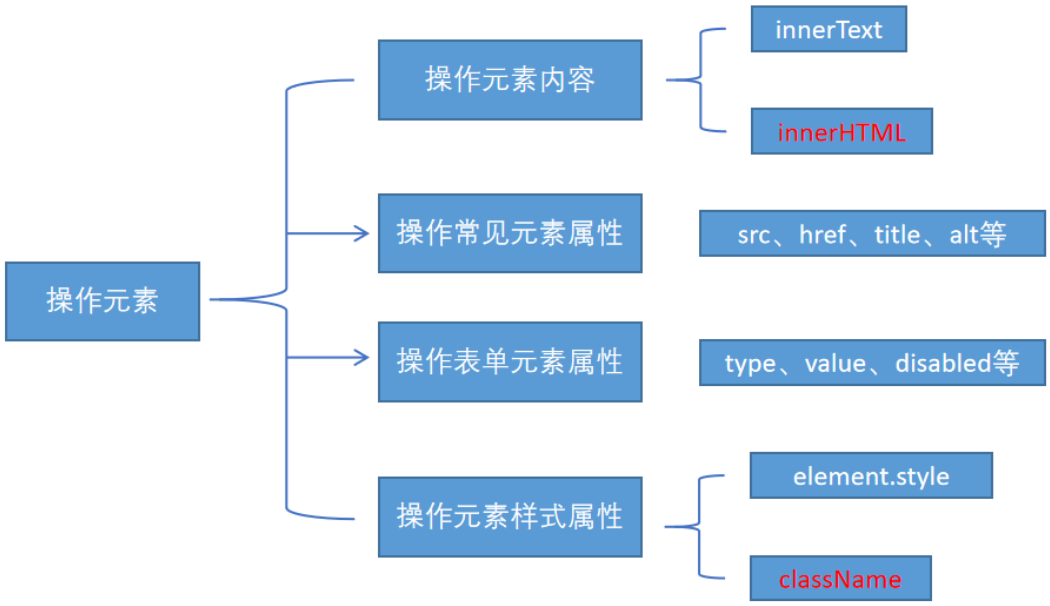
我们可以通过 JS 修改元素的大小、颜色、位置等样式。

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

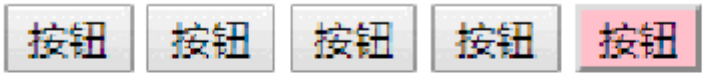
> 注意：
>
> 1. JS 里面的样式采取驼峰命名法 比如 fontSize、 backgroundColor
> 2. JS 修改 style 样式操作，产生的是行内样式，CSS 权重比较高

> 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。**

操作元素是 DOM 核心内容



5、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

6、自定义属性的操作

6.1 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

6.2 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）

6.3 移除属性

> element.removeAttribute('属性');

7、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：

7.1 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)

7.2 获取自定义属性

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

五、节点操作

为什么需要节点操作

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

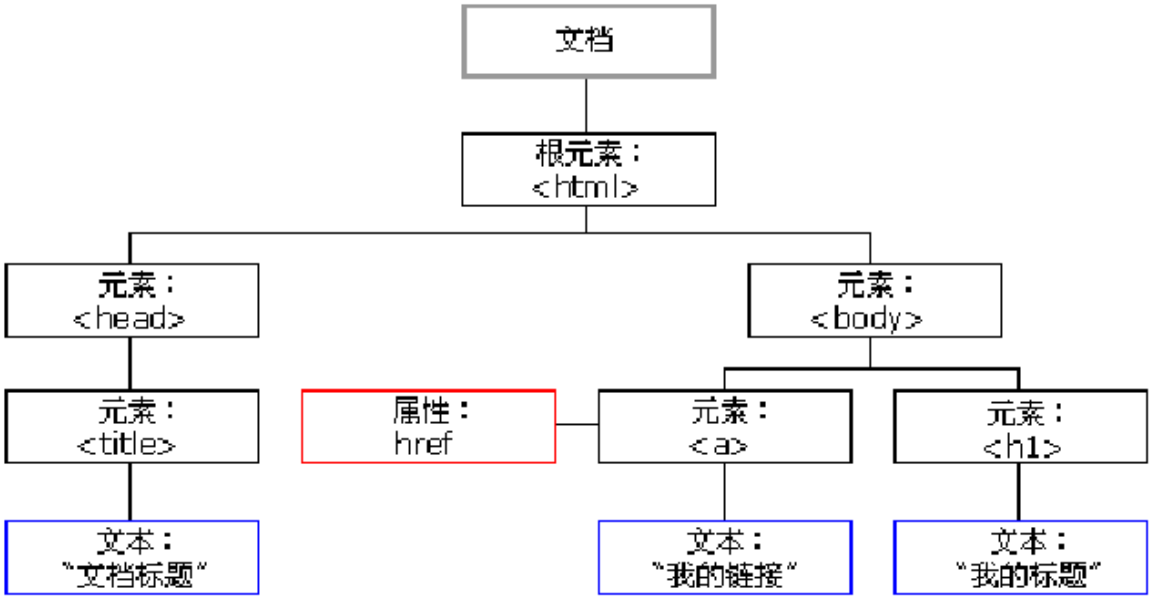
- 利用父子兄节点关系获取元素
- 逻辑性强， 但是兼容性稍差

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

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。

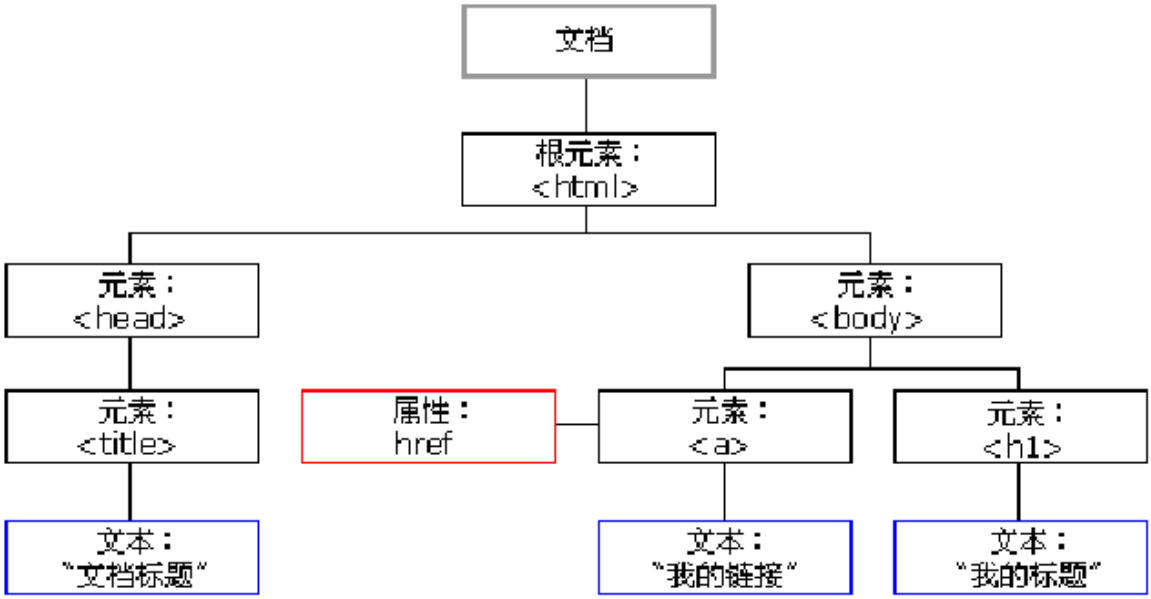


> 一般地，**节点至少拥有nodeType（节点类型）、nodeName（节点名称）和nodeValue（节点值）这三个基本属性。**

> 1. **元素节点 nodeType 为 1**
> 2. **属性节点 nodeType 为 2**
> 3. **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

2、节点层次

> 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。



2.1 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

2.2 子节点

（1）childNodes

```
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

```
var ul = document. querySelector('ul');
for(var i = 0; i < ul.childNodes.length;i++) {
    if (ul.childNodes[i].nodeType == 1) {
        // ul.childNodes[i] 是元素节点
        console.log(ul.childNodes[i]);
    }
}
```

（2）children

```
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

```
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。**

```
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。**

```
5. parentNode.firstElementChild
```

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

```
6. parentNode.lastElementChild
```

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

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：

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

2.3 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。**

存在兼容性

问：如何**解决兼容性问题？**

答：自己封装一个兼容性的函数

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

3、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。**

4、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。

5、删除节点

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

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

6、复制节点（克隆）

```
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。**
> 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。**

7、三种动态创建元素的方式

> 1. document.write()
> 2. element.innerHTML
> 3. document.createElement()

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，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 creatElement 高

DOM总结：

1、创建

> - document.write
> - innerHTML
> - createElement

2、增

> - appendChild
> - insertBefore

3、删

> - removeChild

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

- 修改元素属性： src、href、title等
- 修改普通元素内容： innerHTML 、innerText
- 修改表单元素： value、type、disabled等
- 修改元素样式： style、className

5、查

主要获取查询dom的元素

- DOM提供的API 方法： getElementById、getElementsByTagName 古老用法 不太推荐
- H5提供的新方法： querySelector、querySelectorAll 提倡
- 利用节点操作获取元素： 父(parentNode)、子(children)、兄(previousElementSibling、nextElementSibling) 提倡

6、属性操作

主要针对于自定义属性。

- setAttribute：设置dom的属性值
- getAttribute：得到dom的属性值
- removeAttribute移除属性

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**

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

事件高级

一、注册事件（绑定事件）

1、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
- 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

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：
>
> 1. type：事件类型字符串，比如 click 、mouseover ，注意这里不要带 on
> 2. listener：事件处理函数，事件发生时，会调用该监听函数
> 3. 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

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：
>
> 1. eventNameWithOn：事件类型字符串，比如 onclick 、onmouseover ，这里要带 on
> 2. 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

```
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;
}
```

兼容性解决：

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

二、删除事件（解绑事件）

1、删除事件的方式

1.1 传统注册方式

> eventTarget.onclick = null;

1.2 方法监听注册方式

> 1. eventTarget.**removeEventListener**(type, listener[, useCapture]);
> 2. eventTarget.**detachEvent**(eventNameWithOn, callback);

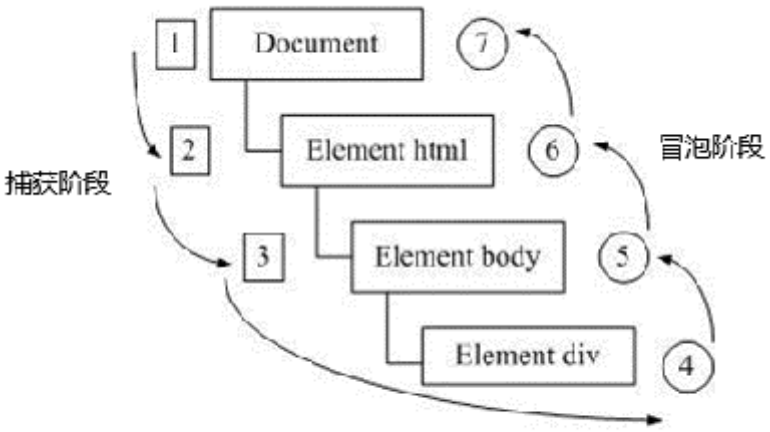
兼容性解决方案：

```
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;
}
```

三、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**

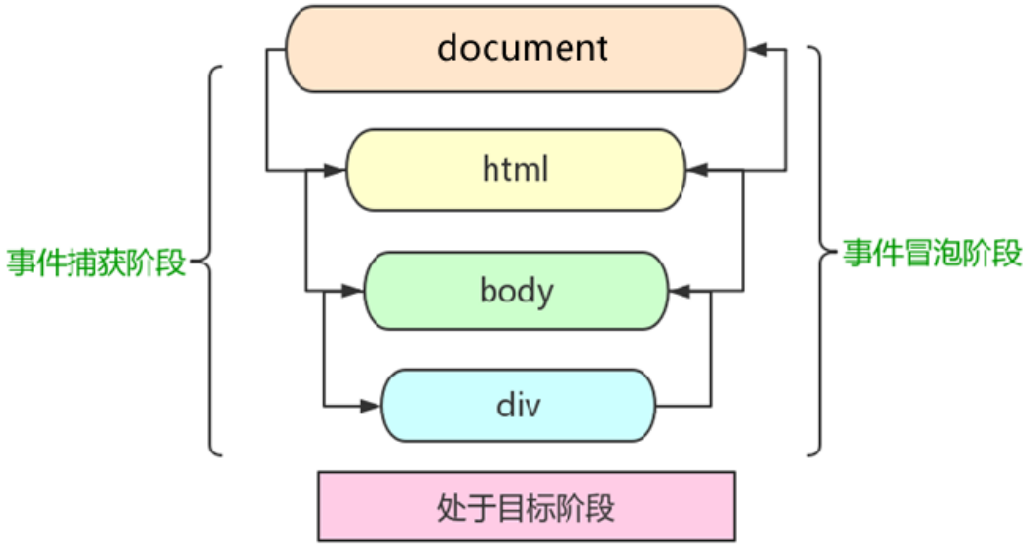


DOM事件流

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。



> 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，我们后面讲解。

四、事件对象

1、概述

```
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。**
>
> 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。

2、使用

```
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）。

> 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;**

3、事件对象的常见属性和方法

> **e.target 和 this 的区别：**
>
> 1. 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）
> 2. 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

五、阻止事件冒泡

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。

1、阻止事件冒泡

1.1 标准写法

> 标准写法：利用事件对象里面的 stopPropagation()方法

```
e.stopPropagation()
```

1.2 非标准写法

> 非标准写法：IE 6-8 利用事件对象 cancelBubble 属性

```
e.cancelBubble = true;
```

兼容性解决：

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

六、事件委托（代理、委派）

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，何如?

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。

```
<ul>
    <li>知否知否，应该有弹框在手</li>
    <li>知否知否，应该有弹框在手</li>
    <li>知否知否，应该有弹框在手</li>
    <li>知否知否，应该有弹框在手</li>
    <li>知否知否，应该有弹框在手</li>
</ul>
```

点击每个 li 都会弹出对话框，以前需要给每个 li 注册事件，是非常辛苦的，而且访问 DOM 的次数越多，这就会延长整个页面的交互就绪时间。

1、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。

2、事件委托原理

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。

以上案例：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。

3、作用

我们只操作了一次 DOM ， 提高了程序的性能。

七、常用鼠标事件

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、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();
})
```

2、禁止鼠标选中（selectstart 开始选中）

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

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

八、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 等。**
> 3. 三个事件的执行顺序是： keydown -- keypress --- keyup

键盘事件对象

键盘事件对象 属性	说明
keyCode	返回该键的ASCII 值

注意： onkeydown 和 onkeyup 不区分字母大小写，onkeypress 区分字母大小写。

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