JavaScript的事件处理

王红元 coderwhy

目录
content

认识事件（Event）

- **Web页面需要经常和用户之间进行交互，而交互的过程中我们可能想要捕捉这个交互的过程：**
  - 比如用户点击了某个按钮、用户在输入框里面输入了某个文本、用户鼠标经过了某个位置；
  - 浏览器需要搭建一条JavaScript代码和事件之间的桥梁；
  - 当某个事件发生时，让JavaScript可以相应（执行某个函数），所以我们需要针对事件编写处理程序（handler）；
- **如何进行事件监听呢?**
  - 事件监听方式一：在script中直接监听（很少使用）；
  - 事件监听方式二：DOM属性，通过元素的on来监听事件；
  - 事件监听方式三：通过EventTarget中的addEventListener来监听；

```
<div id="box" onclick="alert('box点击')">我是box</div>
<script>
  box.onclick = function() {
    alert("box点击2")
  }
  box.addEventListener("click", function() {
    console.log("box点击3")
  })
</script>
```

常见的事件列表

- **鼠标事件:**
  - click —— 当鼠标点击一个元素时（触摸屏设备会在点击时生成）。
  - mouseover / mouseout —— 当鼠标指针移入/离开一个元素时。
  - mousedown / mouseup —— 当在元素上按下/释放鼠标按钮时。
  - mousemove —— 当鼠标移动时。
- **键盘事件:**
  - keydown 和 keyup —— 当按下和松开一个按键时。
- **表单（form）元素事件:**
  - submit —— 当访问者提交了一个 <form> 时。
  - focus —— 当访问者聚焦于一个元素时，例如聚焦于一个 <input>。
- **Document 事件:**
  - DOMContentLoaded —— 当 HTML 的加载和处理均完成，DOM 被完全构建完成时。
- **CSS 事件:**
  - transitionend —— 当一个 CSS 动画完成时。

认识事件流

- **事实上对于事件有一个概念叫做事件流，为什么会产生事件流呢?**
  - 我们可以想到一个问题：当我们在浏览器上对着一个元素点击时，你点击的不仅仅是这个元素本身;
  - 这是因为我们的HTML元素是存在父子元素叠加层级的;
  - 比如一个span元素是放在div元素上的，div元素是放在body元素上的，body元素是放在html元素上的;

```html
<div class="box">
  <span class="word">哈哈哈哈</span>
</div>
```

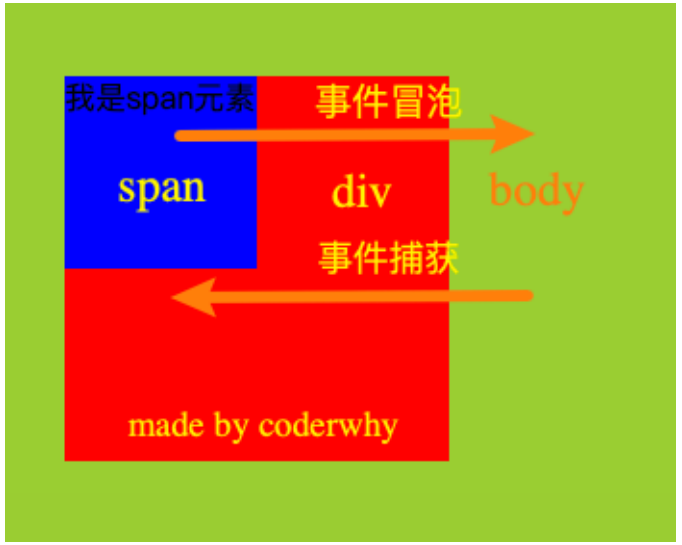
```js
// 1.获取元素
var spanEl = document.querySelector(".word")
var divEl = document.querySelector(".box")
var bodyEl = document.body

// 2.添加监听
spanEl.addEventListener("click", function() {
  console.log("span被点击~")
})
divEl.addEventListener("click", function() {
  console.log("div被点击~")
})
bodyEl.addEventListener("click", function() {
  console.log("body被点击~")
})
```

事件冒泡和事件捕获

- 我们会发现默认情况下事件是**从最内层的span向外依次传递的顺序**，这个顺序我们称之为**事件冒泡（Event Bubble）**；
- 事实上，还有另外一种监听事件流的方式就是**从外层到内层（body -> span）**，这种称之为**事件捕获（Event Capture）**；
- **为什么会产生两种不同的处理流呢?**
  - 这是因为早期浏览器开发时，不管是IE还是Netscape公司都发现了这个问题;
  - 但是他们采用了完全相反的事件流来对事件进行了传递;
  - IE采用了事件冒泡的方式，Netscape采用了事件捕获的方式;
- **那么我们如何去监听事件捕获的过程呢?**

```
spanEl.addEventListener("click", function() {
  console.log("span被点击~")
}, true)
divEl.addEventListener("click", function() {
  console.log("div被点击~")
}, true)
bodyEl.addEventListener("click", function() {
  console.log("body被点击~")
}, true)
```

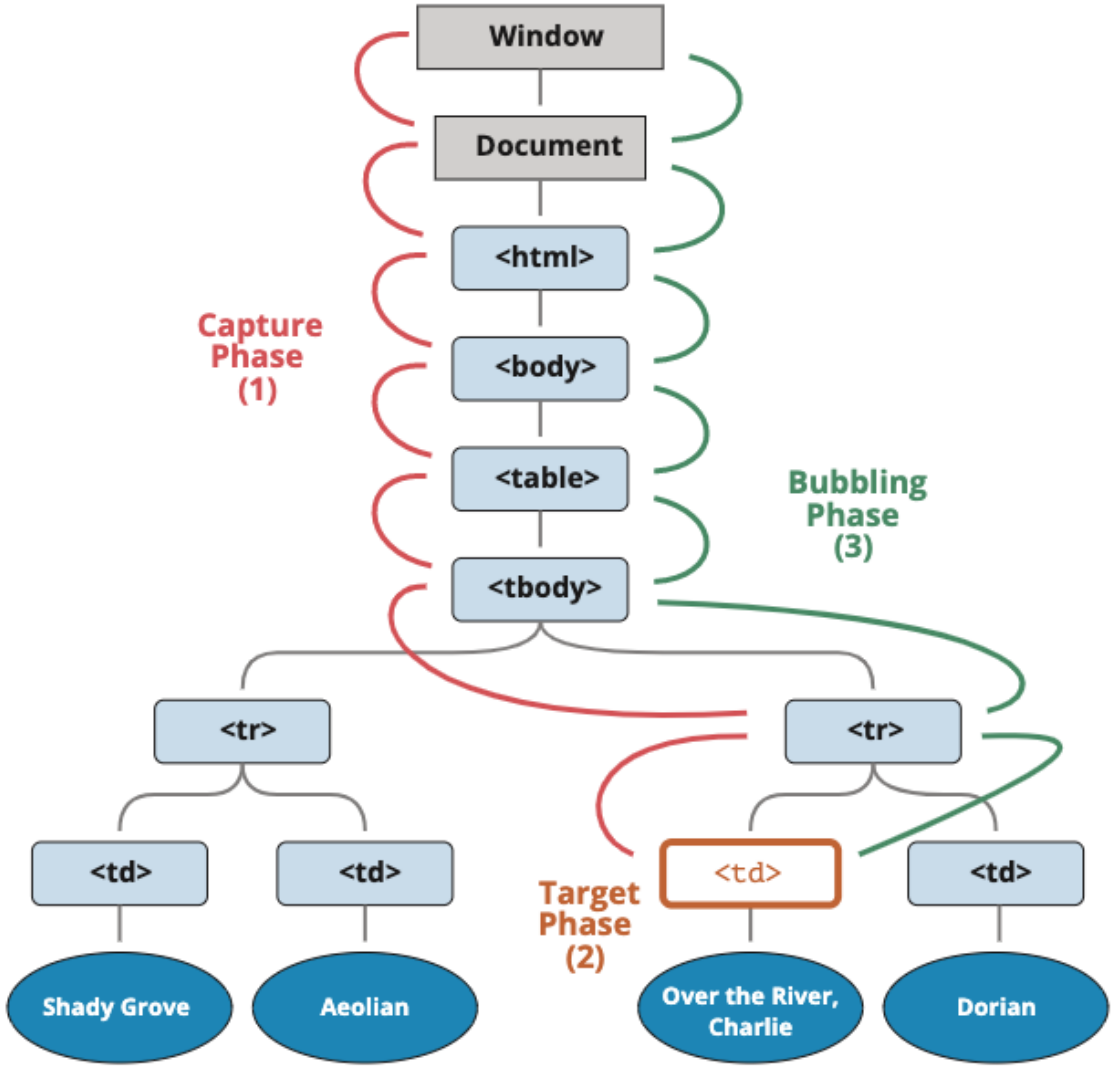


事件捕获和冒泡的过程

- **如果我们都监听，那么会按照如下顺序来执行：**
- 捕获阶段（Capturing phase）：
  - 事件（从 Window）向下走近元素。
- 目标阶段（Target phase）：
  - 事件到达目标元素。
- 冒泡阶段（Bubbling phase）：
  - 事件从元素上开始冒泡。

- **事实上，我们可以通过event对象来获取当前的阶段：**
  - eventPhase

- **开发中通常会使用事件冒泡，所以事件捕获了解即可。**



事件对象

- **当一个事件发生时，就会有和这个事件相关的很多信息：**
  - 比如事件的类型是什么，你点击的是哪一个元素，点击的位置是哪里等等相关的信息；
  - 那么这些信息会被封装到一个Event对象中，这个对象由浏览器创建，称之为event对象；
  - 该对象给我们提供了想要的一些属性，以及可以通过该对象进行某些操作；
- **如何获取这个event对象呢?**
  - event对象会在传入的事件处理（event handler）函数回调时，被系统传入；
  - 我们可以在回调函数中拿到这个event对象；

```
spanEl.onclick = function(event) {
  console.log("事件对象:", event)
}
spanEl.addEventListener("click", function(event) {
  console.log("事件对象:", event)
})
```

- **这个对象中都有哪些常见的属性和操作呢?**

event常见的属性和方法

- **常见的属性：**
  - type：事件的类型；
  - target：当前事件发生的元素；
  - currentTarget：当前处理事件的元素；
  - eventPhase：事件所处的阶段；
  - offsetX、offsetY：事件发生在元素内的位置；
  - clientX、clientY：事件发生在客户端内的位置；
  - pageX、pageY：事件发生在客户端相对于document的位置；
  - screenX、screenY：事件发生相对于屏幕的位置；
- **常见的方法：**
  - preventDefault：取消事件的默认行为；
  - stopPropagation：阻止事件的进一步传递（冒泡或者捕获都可以阻止）；

事件处理中的this

- 在函数中，我们也可以通过this来获取当前的发生元素：

```
boxEl.addEventListener("click", function(event) {
  console.log(this === event.target) // true
})
```

- 这是因为在浏览器内部，调用event handler是绑定到当前的target上的

EventTarget类

- **我们会发现，所有的节点、元素都继承自EventTarget**
  - 事实上Window也继承自EventTarget；

- **那么这个EventTarget是什么呢?**
  - EventTarget是一个DOM接口，主要用于添加、删除、派发Event事件；
- **EventTarget常见的方法：**
  - addEventListener：注册某个事件类型以及事件处理函数；
  - removeEventListener：移除某个事件类型以及事件处理函数；
  - dispatchEvent：派发某个事件类型到EventTarget上；

```
var boxEl = document.querySelector(".box")
boxEl.addEventListener("click", function() {
  console.log("点击了box")
})
```

```
boxEl.addEventListener("click", function() {
  window.dispatchEvent(new Event("coderwhy"))
})
window.addEventListener("coderwhy", function(event) {
  console.log("监听到coderwhy事件:", event)
})
```

事件委托（event delegation）

- 事件冒泡在某种情况下可以帮助我们实现强大的事件处理模式 – **事件委托模式**（也是一种设计模式）
- 那么这个模式是怎么样的呢?
  - 因为当子元素被点击时，父元素可以通过冒泡可以监听到子元素的点击；
  - 并且可以通过event.target获取到当前监听的元素；
- 案例：一个ul中存放多个li，点击某一个li会变成红色
  - 方案一：监听每一个li的点击，并且做出相应；
  - 方案二：在ul中监听点击，并且通过event.target拿到对应的li进行处理；
    - ✓ 因为这种方案并不需要遍历后给每一个li上添加事件监听，所以它更加高效；

```
var listEl = document.querySelector(".list")
var currentActive = null
listEl.addEventListener("click", function(event) {
  if (currentActive) currentActive.classList.remove("active")
  event.target.classList.add("active")
  currentActive = event.target
})
```

事件委托的标记

- 某些事件委托可能需要对具体的子组件进行区分，这个时候我们可以使用data-*对其进行标记：
- 比如多个按钮的点击，区分点击了哪一个按钮：

```html
<div class="btn-list">
  <button data-action="new">新建</button>
  <button data-action="search">搜索</button>
  <button data-action="delete">删除</button>
</div>
```

```js
var btnListEl = document.querySelector(".btn-list")
btnListEl.addEventListener("click", function(event) {
  var action = event.target.dataset.action
  switch (action) {
    case "new":
      console.log("点击了新建~")
      break
    case "search":
      console.log("点击了搜索~")
      break
    case "delete":
      console.log("点击了删除~")
      break
    default:
      console.log("位置action")
  }
})
```

常见的鼠标事件

- 接下来我们来看一下常见的鼠标事件（不仅仅是鼠标设备，也包括模拟鼠标的设备，比如手机、平板电脑）
- 常见的鼠标事件：

属性	描述
click	当用户点击某个对象时调用的事件句柄。
contextmenu	在用户点击鼠标右键打开上下文菜单时触发
dblclick	当用户双击某个对象时调用的事件句柄。
mousedown	鼠标按钮被按下。
mouseup	鼠标按键被松开。
mouseover	鼠标移到某元素之上。（支持冒泡）
mouseout	鼠标从某元素移开。（支持冒泡）
mouseenter	当鼠标指针移动到元素上时触发。（不支持冒泡）
mouseleave	当鼠标指针移出元素时触发。（不支持冒泡）
mousemove	鼠标被移动。

mouseover和mouseenter的区别

- mouseenter和mouseleave
  - 不支持冒泡
  - 进入子元素依然属于在该元素内，没有任何反应
- mouseover和mouseout
  - 支持冒泡
  - 进入元素的子元素时
    - ✓ 先调用父元素的mouseout
    - ✓ 再调用子元素的mouseover
    - ✓ 因为支持冒泡，所以会将mouseover传递到父元素中；

常见的键盘事件

- **常见的键盘事件：**

属性	描述
onkeydown	某个键盘按键被按下。
onkeypress	某个键盘按键被按下。
onkeyup	某个键盘按键被松开。

- **事件的执行顺序是 onkeydown、onkeypress、onkeyup**
  - down事件先发生；
  - press发生在文本被输入；
  - up发生在文本输入完成；
- **我们可以通过key和code来区分按下的键：**
  - code：“按键代码”（"KeyA"，"ArrowLeft" 等），特定于键盘上按键的物理位置。
  - key：字符（"A"，"a" 等），对于非字符（non-character）的按键，通常具有与 code 相同的值。）

常见的表单事件

- 针对表单也有常见的事件：

属性	描述
onchange	该事件在表单元素的内容改变时触发(<input>, <keygen>, <select>, 和 <textarea>)
oninput	元素获取用户输入时触发
onfocus	元素获取焦点时触发
onblur	元素失去焦点时触发
onreset	表单重置时触发
onsubmit	表单提交时触发

文档加载事件

- **DOMContentLoaded**：浏览器已完全加载 HTML，并构建了 DOM 树，但像 <img> 和样式表之类的外部资源可能尚未加载完成。
- **load**：浏览器不仅加载完成了 HTML，还加载完成了所有外部资源：图片，样式等。

```
<div>哈哈哈</div>
<img src="https://ossweb-img.qq.com/upload/webplat/info/yxzj/20220510/283456868994485.jpg" alt="">

<script>
  window.addEventListener("DOMContentLoaded", function() {
    var imgEl = document.querySelector("img")
    console.log("页面内容加载完毕", imgEl.offsetWidth, imgEl.offsetHeight)
  })
  window.addEventListener("load", function() {
    var imgEl = document.querySelector("img")
    console.log("页面所有内容加载完毕", imgEl.offsetWidth, imgEl.offsetHeight)
  })
</script>
```

- 事件类型：https://developer.mozilla.org/zh-CN/docs/Web/Events

轮播图的基本切换

- 实现轮播图的基本切换