基础知识

在文档、浏览器、标签元素等元素在特定状态下触发的行为即为事件，比如用户的单击行为、表单内容的改变行为即为事件，我们可以为不同的事件定义处理程序。JS 使用异步事件驱动的形式管理事件。

#事件类型

JS 为不同的事件定义的类型，也可以称为事件名称。

#事件目标

事件目标指产生事件的对象，比如 a 标签被点击那么 a 标签就是事件目标。元素是可以嵌套的，所以在进行一次点击行为时可能会触发多个事件目标。

#处理程序

事件的目的是要执行一段代码，我们称这类代码为事件处理（监听）程序。当在对象上触发事件时就会执行定义的事件处理程序。

#HTML 绑定

可以在 html 元素上设置事件处理程序，浏览器解析后会绑定到 DOM 属性中

```
<button onclick="alert(`houdunren.com`)">后盾人</button>

```

往往事件处理程序业务比较复杂，所以绑定方法或函数会很常见

- 绑定函数或方法时需要加上括号

```
<button onclick="show()">后盾人</button>
<script>
  function show() {
    alert('houdunren.com')
  }
</script>

```

当然也可以使用方法做为事件处理程序

```
<input type="text" onkeyup="HD.show()" />
<script>
  class HD {
    static show() {
      console.log('houdunren')
    }
  }
</script>

```

可以传递事件源对象与事件对象

```
<button onclick="show(this,'houdunren','hdcms','向军大叔',event)">后盾人</button>
<script>
    function show(...args) {
        console.log(args)
    }
</script>

```

#DOM 绑定

也可以将事件处理程序绑定到 DOM 属性中

- 使用 setAttribute 方法设置事件处理程序无效
- 属性名区分大小写

```
<div id="app">houdunren.com</div>
<script>
  const app = document.querySelector('#app')
  app.onclick = function () {
    this.style.color = 'red'
  }
</script>

```

无法为事件类型绑定多个事件处理程序，下面绑定了多个事件处理程序，因为属性是相同的所以只有最后一个有效

```
<div id="app">houdunren.com</div>
<script>
  const app = document.querySelector('#app')
  app.onclick = function () {
    this.style.color = 'red'
  }
  app.onclick = function () {
    this.style.fontSize = '55px'
  }
</script>

```

#事件监听

通过上面的说明我们知道使用 HTML 与 DOM 绑定事件都有缺陷，建议使用新的事件监听绑定方式

addEventListener 操作事件

使用 addEventListener 添加事件处理程序有以下几个特点

- transtionend / DOMContentLoaded 等事件类型只能使用 addEventListener 处理
- 同一事件类型设置多个事件处理程序，按设置的顺序先后执行
- 也可以对未来添加的元素绑定事件

方法	说明
addEventListener	添加事件处理程序
removeEventListener	移除事件处理程序

addEventListener 的参数说明如下

1. 参数一事件类型
2. 参数二事件处理程序
3. 参数三为定制的选项，可传递 object 或 boolean 类型。后面会详细介绍使用区别

#绑定多个事件

使用 addEventListener 来多个事件处理程序

```
<div id="app">houdunren.com</div>
<script>
  const app = document.querySelector('#app')
  app.addEventListener('click', function () {
```

```
    this.style.color = 'red'
  })
  app.addEventListener('click', function () {
    this.style.fontSize = '55px'
  })
</script>

```

#通过对象绑定

如果事件处理程序可以是对象，对象的 handleEvent 方法会做为事件处理程序执行。下面将元素的事件统一交由对象处理

```
<div id="app">houdunren.com</div>
<script>
  const app = document.querySelector('#app')
  class HD {
    handleEvent(e) {
      this[e.type](e)
    }
    click() {
      console.log('单击事件')
    }
    mouseover() {
      console.log('鼠标移动事件')
    }
  }
  app.addEventListener('click', new HD())
  app.addEventListener('mouseover', new HD())
</script>

```

#移除事件

使用 removeEventListener 删除绑定的事件处理程序

- 事件处理程序单独定义函数或方法，这可以保证事件处理程序是同一个

```
<div id="app">houdunren.com</div>
<button id="hd">删除事件</button>
```

```
<script>
  const app = document.querySelector('#app')
  const hd = document.querySelector('#hd')
  function show() {
    console.log('APP我执行了')
  }
  app.addEventListener('click', show)
  hd.addEventListener('click', function () {
    app.removeEventListener('click', show)
  })
</script>
```

#事件选项

addEventListener 的第三个参数为定制的选项，可传递 object 或 boolean 类型

下面是传递对象时的说明

选项	可选参数	
once	true/false	只执行一次事件
capture	true/false	事件是在捕获/冒泡哪个阶段执行，true:捕获阶段 false:冒泡阶段
passive	true/false	声明事件里不会调用 preventDefault()，可以减少系统默认行为的等待

下面使用 once:true 来指定事件只执行一次

```
<button id="app">houdunren.com</button>
<script>
    const app = document.querySelector('#app')
    app.addEventListener(
        'click',
        function () {
            alert('houdunren@向军大叔')
        },
        { once: true }
```

```
    )
</script>

```

设置 `{ capture: true }` 或直接设置第三个参数为 true 用来在捕获阶段执行事件

> addEventListener 的第三个参数传递 true/false 和设置 {capture:true/false}是一样

```
<div id="app" style="background-color: red">
    <button id="bt">houdunren.com</button>
</div>
<script>
    const app = document.querySelector('#app')
    const bt = document.querySelector('#bt')
    app.addEventListener(
        'click',
        function () {
            alert('这是div事件 ')
        },
        { capture: true }
    )

    bt.addEventListener(
        'click',
        function () {
            alert('这是按钮事件 ')
        },
        { capture: true }
    )
</script>

```

设置 `{ capture: false }` 或直接设置第三个参数为 false 用来在冒泡阶段执行事件

```
<div id="app" style="background-color: red">
    <button id="bt">houdunren.com</button>
</div>
<script>
    const app = document.querySelector('#app')
    const bt = document.querySelector('#bt')
```

```
    app.addEventListener(
        'click',
        function () {
            alert('这是div事件 ')
        },
        { capture: false }
    )

    bt.addEventListener(
        'click',
        function () {
            alert('这是按钮事件 ')
        },
        { capture: false }
    )
</script>

```

#事件对象

执行事件处理程序时，会产生当前事件相关信息的对象，即为事件对事。系统会自动做为参数传递给事件处理程序。

- 大部分浏览器将事件对象保存到 window.event 中
- 有些浏览器会将事件对象做为事件处理程序的参数传递

事件对象常用属性如下：

属性	说明
type	事件类型
target	事件目标对象，冒泡方式时父级对象可以通过该属性找到在哪个子元素上最终执行事件
currentTarget	当前执行事件的对象
timeStamp	事件发生时间
x	相对窗口的 X 坐标
y	相对窗口的 Y 坐标
clientX	相对窗口的 X 坐标
clientY	相对窗口的 Y 坐标

screenX	相对计算机屏幕的 X 坐标
screenY	相对计算机屏幕的 Y 坐标
pageX	相对于文档的 X 坐标
pageY	相对于文档的 Y 坐标
offsetX	相对于事件对象的 X 坐标
offsetY	相对于事件对象的 Y 坐标
layerX	相对于父级定位的 X 坐标
layerY	相对于父级定位的 Y 坐标
path	冒泡的路径
altKey	是否按了 alt 键
shiftKey	是否按了 shift 键
metaKey	是否按了媒体键
window.pageXOffset	文档参考窗口水平滚动的距离
window.pageYOffset	文档参考窗口垂直滚动的距离

#冒泡捕获

#冒泡行为

标签元素是嵌套的，在一个元素上触发的事件，同时也会向上执行父级元素的事件处理程序，一直到 HTML 标签元素。

- 大部分事件都会冒泡，但像 focus 事件则不会
- event.target 可以在事件链中最底层的定义事件的对象
- event.currentTarget == this 即当前执行事件的对象

以下示例有标签的嵌套，并且父子标签都设置了事件，当在子标签上触发事件事会冒泡执行父级标签的事件

houdunren.com

```
<style>
  #app {
    background: #34495e;
    width: 300px;
    padding: 30px;
  }
  #app h2 {
    background-color: #f1c40f;
    margin-right: -100px;
  }
</style>
<div id="app">
  <h2>houdunren.com</h2>
</div>
<script>
  const app = document.querySelector('#app')
  const h2 = document.querySelector('h2')
  app.addEventListener('click', (event) => {
    console.log(`event.currentTarget:${event.currentTarget.nodeName}`)
    console.log(`event.target:${event.target.nodeName}`)
    console.log('app event')
  })
  h2.addEventListener('click', () => {
    console.log(`event.currentTarget:${event.currentTarget.nodeName}`)
    console.log(`event.target:${event.target.nodeName}`)
    console.log(`h2 event`)
  })
</script>
```

```
29
```

#阻止冒泡

冒泡过程中的任何事件处理程序中，都可以执行 `event.stopPropagation()` 方法阻止继续进行冒泡传递

- event.stopPropagation() 用于阻止冒泡
- 如果同一类型事件绑定多个事件处理程序 event.stopPropagation() 只阻止当前的事件处理程序
- event.stopImmediatePropagation() 阻止事件冒泡并且阻止相同事件的其他事件处理程序被调用

下例中为 h2 的事件处理程序添加了阻止冒泡动作，将不会产生冒泡，也就不会执行父级中的事件处理程序了。

```
<style>
  #app {
    background: #34495e;
    width: 300px;
    padding: 30px;
  }
  #app h2 {
    background-color: #f1c40f;
    margin-right: -100px;
  }
</style>
<div id="app">
  <h2>houdunren.com</h2>
</div>
<script>
  const app = document.querySelector('#app')
  const h2 = document.querySelector('h2')
  app.addEventListener('click', (event) => {
    console.log(`event.currentTarget:${event.currentTarget.nodeName}`)
    console.log(`event.target:${event.target.nodeName}`)
    console.log('app event')
  })
  h2.addEventListener('click', (event) => {
    event.stopPropagation()
    console.log(`event.currentTarget:${event.currentTarget.nodeName}`)
    console.log(`event.target:${event.target.nodeName}`)
    console.log(`h2 event`)
```

```
  })
   h2.addEventListener('click', (event) => {
        console.log('h2 的第二个事件处理程序')
   })
</script>

```

以上代码如果将 event.stopPropagation() 替换为 event.stopPropagation() ，那么 h2 的其他同类型的事件处理程序将不执行，同时阻止冒泡。

#事件捕获

事件执行顺序为 捕获 > 事件目标 > 冒泡，在向下传递到目标对象的过程即为事件捕获。事件捕获在实际使用中频率不高。

- 通过设置第三个参数为 true 或{ capture: true } 在捕获阶段执行事件处理程序

```
const app = document.querySelector('#app')
const h2 = document.querySelector('h2')
app.addEventListener(
  'click',
  (event) => {
    console.log('app event')
  },
  { capture: true }
)
h2.addEventListener('click', (event) => {
  console.log(`h2 event`)
})

```

#事件代理

借助冒泡思路，我们可以不为子元素设置事件，而将事件设置在父级。然后通过父级事件对象的 event.target 查找子元素，并对他做出处理。

- 这在为多个元素添加相同事件时很方便
- 会使添加事件变得非常容易

下面是为父级 UL 设置事件来控制子元素 LI 的样式切换

```
<style>
```

```
  .hd {
    border: solid 2px #ddd;
    background-color: red;
    color: white;
  }
</style>
<ul>
  <li>houdunren.com</li>
  <li>hdcms.com</li>
</ul>

<script>
  'use strict'
  const ul = document.querySelector('ul')
  ul.addEventListener('click', () => {
    if (event.target.tagName === 'LI') event.target.classList.toggle('hd')
  })
</script>

```

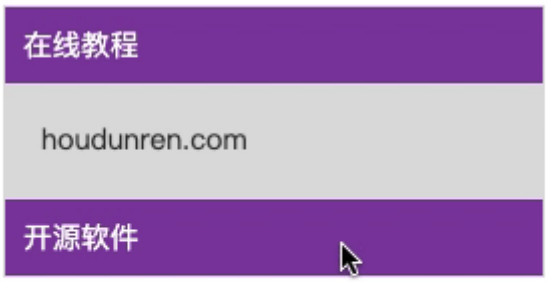
可以使用事件代理来共享事件处理程序，不用为每个元素单独绑定事件

```
<ul>
  <li data-action="hidden">houdunren.com</li>
  <li data-action="color" data-color="red">hdcms.com</li>
</ul>
<script>
  class HD {
    constructor(el) {
      el.addEventListener('click', (e) => {
        const action = e.target.dataset.action
        this[action](e)
      })
    }
    hidden() {
      event.target.hidden = true
    }
    color() {
      event.target.style.color = event.target.dataset.color
```

```
    }
  }
  new HD(document.querySelector('ul'))
</script>

```

下面是使用事件代理实现的 TAB 面板效果




```
<div class="tab">
  <dl>
    <dt data-action="toggle">在线教程</dt>
    <dd data-action="hidden">houdunren.com</dd>
  </dl>
  <dl>
    <dt data-action="toggle">开源软件</dt>
    <dd data-action="hidden">hdcms.com</dd>
  </dl>
</div>

<script>
  class HD {
    constructor(el) {
      this.el = el
      el.addEventListener('click', this.handle.bind(this))
    }
    handle() {
      const action = event.target.dataset.action
      if (action) this[action]()
```

```
    }
    hidden() {
      event.target.hidden = true
    }
    toggle() {
      this.el.querySelectorAll(`[data-action='hidden']`).forEach((e) => (e.hidden = true))
      event.target.nextElementSibling.hidden = ''
    }
  }
  new HD(document.querySelector('.tab'))
</script>
```

下面实现通过代理事件行为，在表单提交时禁用提交按钮，并记录提示次数

```
<form>
  <input type="text" />
  <button type="button" data-submit-disabled data-action="submit,counter">提交表单</button>
</form>
<script>
  class FORM {
    constructor(form) {
      this.form = form
      this.form.counter = 0
      this.form.addEventListener('click', this.handle.bind(this))
    }
    handle() {
      const actions = event.target.dataset.action
      actions.split(',').forEach((action) => {
        if (this[action]) this[action]()
      })
    }
    submit() {
      event.preventDefault()
      this.disableButton(true)
      console.log('提交中')
      setTimeout(() => {
        this.disableButton(false)
```

```
        console.log('提交结束')
      }, 1000)
    }
    disableButton(state) {
      this.form.querySelectorAll(`[data-submit-disabled]`).forEach((bt) => {
        bt.disabled = state
      })
    }
    counter() {
      this.form.counter++
      console.log(this.form.counter)
    }
  }
  document.querySelectorAll(`form`).forEach((form) => {
    new FORM(form)
  })
</script>

```

#未来元素

下面使用事件代理来对未来元素进行事件绑定

```
<div id="app">
  <h2>houdunren.com</h2>
</div>

<script>
  function show() {
    console.log(this.textContent)
  }
  const app = document.querySelector('#app')
  const h2 = document.querySelectorAll('h2')
  app.addEventListener('click', () => {
    show.call(event.target)
  })
  let newH2 = document.createElement('h2')
  newH2.textContent = 'hdcms.com'
  app.append(newH2)
```

```
</script>

```

#默认行为

JS 中有些对象会设置默认事件处理程序，比如 A 链接在点击时会进行跳转。
一般默认处理程序会在用户定义的处理程序后执行，所以我们可以在我们定义的事件处理中取消默认事件处理程序的执行。

- 使用 onclick 绑定的事件处理程序，return false 可以阻止默认行为
- 推荐使用

event.preventDefault()阻止默认行为
下面阻止超链接的默认行为

```
<a href="
https://www.houdunren.com
<a href="
<script>
  document.querySelector('a').addEventListener('click', () => {
    event.preventDefault()
    alert(event.target.innerText)
  })
</script>

```

#窗口文档

下面来学习针对窗口和文档的事件的处理。

#事件类型

事件名	说明
window.onload	文档解析及外部资源加载后
DOMContentLoaded	文档解析后执行，不需要等待图片/样式文件等外部资源加载，该事件只能通过 addEventListener 设置
window.beforeunload	文档刷新或关闭时
window.unload	文档卸载时

scroll	页面滚动时

#onload

window.onload 事件在文档解析后及图片、外部样式文件等资源加载完后执行

```
<script>
  window.onload = function () {
    alert('houdunren.com')
  }
</script>
<div id="app">houdunren.com</div>

```

#DOMContentLoaded

DOMContentLoaded 事件在文档标签解析后执行，不需要等外部图片、样式文件、JS 文件等资源加载

```
<script>
  window.addEventListener('DOMContentLoaded', (event) => {
    console.log('houdunren.com')
  })
</script>
<div id="app">houdunren.com</div>

```

#beforeunload

当浏览器窗口关闭或者刷新时，会触发 beforeunload 事件，可以取消关闭或刷新页面。

- 返回值为非空字符串时，有些浏览器会做为弹出的提示信息内容
- 部分浏览器使用 addEventListener 无法绑定事件

```
window.onbeforeunload = function (e) {
  return '真的要离开吗? '
}

```

#unload

window.unload 事件在文档资源被卸载时执行，在 beforeunload 后执行

- 不能执行 alert、confirm 等交互指令
- 发生错误也不会阻止页面关闭或刷新

```
window.addEventListener('unload', function (e) {
  localStorage.setItem('name', 'houdunren')
})

```

#鼠标事件

#事件类型

针对鼠标操作的行为有多种事件类型

- 鼠标事件会触发在 Z-INDEX 层级最高的那个元素上

事件名	说明
click	鼠标单击事件，同时触发 mousedown/mouseup
dblclick	鼠标双击事件
contextmenu	点击右键后显示的所在环境的菜单
mousedown	鼠标按下
mouseup	鼠标抬起时
mousemove	鼠标移动时
mouseover	鼠标移动时
mouseout	鼠标从元素上离开时
mouseup	鼠标抬起时
mouseenter	鼠标移入时触发，不产生冒泡行为
mosueleave	鼠标移出时触发，不产生冒泡行为
oncopy	复制内容时触发
scroll	元素滚动时，可以为元素设置 overflow:auto; 产生滚动条来测试

#事件对象

鼠标事件产生的事件对象包含相对应的属性

属性	说明
which	执行 mousedown/mouseup 时，显示所按的键 1 左键，2 中键，3 右键
clientX	相对窗口 X 坐标
clientY	相对窗口 Y 坐标
pageX	相对于文档的 X 坐标
pageY	相对于文档的 Y 坐标
offsetX	目标元素内部的 X 坐标
offsetY	目标元素内部的 Y 坐标
altKey	是否按了 alt 键
ctrlKey	是否按了 ctlr 键
shiftKey	是否按了 shift 键
metaKey	是否按了媒体键
relatedTarget	mouseover 事件时从哪个元素来的，mouseout 事件时指要移动到的元素。当无来源（在自身上移动）或移动到窗口外时值为 null

#禁止复制

```
<body>
  houdunren.com
  <script>
    document.addEventListener('copy', () => {
      event.preventDefault()
      alert('禁止复制内容')
    })
  </script>
</body>

```

#relatedTarget

relatedTarget 是控制鼠标移动事件的来源和目标对象的

- 如果移动过快会跳转中间对象

```
<div id="app">houdunren.com</div>
<div id="cms">hdcms.com</div>
<script>
  const app = document.querySelector(`#app`)
  const cms = document.querySelector(`#cms`)
  app.addEventListener('mouseout', () => {
    console.log(event.target)
    console.log(event.relatedTarget)
  })
</script>

```

#mouseenter 与 mouseleave

mouseenter 与 mouseleave 事件从子元素移动到父元素时不触发父元素事件

```
<style>
  #app {
    background: red;
    padding: 80px;
    width: 500px;
  }
  #cms {
    background: teal;
    padding: 30px;
  }
</style>
<div id="app">
  houdunren.com
  <div id="cms">hdcms.com</div>
</div>

<script>
  const app = document.querySelector(`#app`)
```

```
  const cms = document.querySelector(`#cms`)

  app.addEventListener('mouseenter', () => {
    console.log('app')
  })
  cms.addEventListener('mouseenter', () => {
    console.log('cms')
  })
</script>

```

#键盘事件

针对键盘输入操作的行为有多种事件类型

事件名	说明
Keydown	键盘按下时，一直按键不松开时 keydown 事件会重复触发
keyup	按键抬起时

#事件对象

键盘事件产生的事件对象包含相对应的属性

属性	说明
keyCode	返回键盘的 ASCII 字符数字
code	按键码，字符以 Key 开始，数字以 Digit 开始，特殊字符有专属名子。左右 ALT 键字符不同。 不同布局的键盘值会不同
key	按键的字符含义表示，大小写不同。不能区分左右 ALT 等。不同语言操作系统下值会不同
altKey	是否按了 alt 键
ctrlKey	是否按了 ctlr 键
shiftKey	是否按了 shift 键
metaKey	是否按了媒体键

#表单事件

下面是可以用在表单上的事件类型

事件类型	说明
focus	获取焦点事件
blur	失去焦点事件
element.focus()	让元素强制获取焦点
element.blur()	让元素失去焦点
change	文本框在内容发生改变并失去焦点时触发， select/checkbox/radio 选项改变时触发事件
input	Input、textarea 或 select 元素的 value 被修改时，会触发 input 事件。而 change 是鼠标离开后或选择一个不同 的 option 时触发。
submit	提交表单