

Javascript 事件

陈一帅

实务学堂

Events 事件

- JQuery中我们学过 Click 事件
- 还有很多事件
 - 单击或点击链接
 - 悬停或在元素上滑动
 - 调整浏览器窗口的大小
 - 网页加载
- 用JavaScript函数来响应事件

各种事件

- 每个事件都有一个标识它的类型
 - 按键会触发“keydown”和“keyup”事件
 - 按下鼠标按钮会触发“鼠标向下mousedown”，“鼠标向上mouseup”和“单击click”事件。移动鼠标会触发“mousemove”事件
 - 触摸屏交互将导致“触摸开始touchstart”，“触摸移动touchmove”和“触摸结束touchend”事件。
 - 滚动屏幕将导致“滚动scroll”事件，“focus焦点”和“blur模糊”事件可以检测光标焦点变化
 - 文档加载完成后，窗口触发“加载load”事件

将事件绑定到元素

- 三步
 - 选择脚本要响应的元素
 - 指定发生在元素上的哪个事件将触发响应
 - `addEventListener`方法用于注册此类处理程序
 - 运行事件的代码

```
元素.addEventListener('事件', 函数);
```

事件示例：鼠标点击变色

Code

event.target 目标属性

- 可以在父元素用 `event.target.nodeName` 判断是哪种元素发生了事件
 - `event.target` 指向引发这个响应的元素

```
<button>A</button>
<script>
  document.body.addEventListener("click",
    event => {
      if (event.target.nodeName == "BUTTON") {
        console.log("Clicked", event.target.textContent);
      }
    });
</script>
```

按钮示例

鼠标事件

- 按下鼠标按钮会触发“鼠标向下mousedown”，“鼠标向上mouseup”和“单击click”事件。移动鼠标会触发“mousemove”事件
 - click 点击
 - dblclick 双击
 - mousedown 鼠标按下
 - mouseup 鼠标弹上
 - mousemove 鼠标移动
 - mouseover 鼠标滑过
 - mouseout 鼠标移出

鼠标点击事件

- click 点击
 - 在“mouseup”事件之后，将在同时包含按下和释放按钮的最特定节点上触发“click”事件。
 - 例如，如果在一个段落上按下鼠标按钮，然后将指针移到另一段落并释放按钮，则包含两个段落的元素会发生“click”事件
- dblclick 双击
 - 如果两次单击同时发生，则在第二次单击事件之后也会触发“dblclick”（双击）事件。

鼠标点击示例：变色

鼠标键

- 调用事件处理程序时，会将事件的一些信息传递给函数
 - 比如：按下的是鼠标的哪个键

```
let 按钮 = document.querySelector("button");
按钮.addEventListener("mousedown", event => {
  if (event.button == 0) {
    console.log("左键");
  } else if (event.button == 1)
  { console.log("中键");
  } else if (event.button == 2)
  { console.log("右键"); }
});
```

鼠标键示例

鼠标位置

- clientX和clientY属性
 - 包含事件相对于窗口左上角的坐标（以像素为单位）
- pageX和pageY
 - 相对于整个文档的左上角的坐标（当窗口已滚动）

鼠标位置示例

示例：鼠标留痕

```
body {  
    height: 200px;  
    background: beige;  
}  
.dot {  
    height: 8px; width: 8px;  
    border-radius: 4px;  
    background: blue;  
    position: absolute;  
}
```

示例：鼠标留痕

```
window.addEventListener("click",
  event => {
    let dot = document.createElement("div");
    dot.className = "dot";
    dot.style.left = (event.pageX - 4) + "px";
    dot.style.top = (event.pageY - 4) + "px";
    document.body.appendChild(dot);
  });
```

鼠标点击画点示例

根据鼠标位置设置背景色

```
let xPos = event.clientX;  
let hue = Math.ceil(xPos / hueBrowserRatio);
```

动态背景色示例

用户交互事件

- load
 - 文档加载完成后，窗口触发“加载完成”
- unload
- scroll
 - 滚动屏幕引发“滚动scroll”事件
- error
- resize

load 加载完成

- 页面加载完成后，触发
- 用于需要整个文档的初始化操作
 - HTML遇到JS代码，就会立刻运行它们
 - 那么，如果这些代码访问了它后面的HTML内容，就为时过早
 - 放到load里，最安全，保证加载完成后，再执行这些代码

例：页面加载时，随机背景色

```
window.addEventListener('load', randomColor);  
  
const 页面 = document.querySelector('body');  
let 随机数 = Math.floor(Math.random() * 360);  
页面.style.backgroundColor  
    = "hsl(" + 随机数 + ", 100%, 50%)";
```

动态背景色示例

beforeunload 关闭或离开前

- 当页面关闭或离开页面（例如，通过链接）时，将触发“beforeunload”事件
- 主要用途是防止用户因关闭文档而意外丢失工作
- 如果您阻止此事件的默认行为，并将事件对象的returnValue属性设置为字符串，浏览器将向用户显示一个对话框，询问他们是否真的要离开该页面。
- 该对话框可能包含您的字符串，但是由于某些恶意网站试图使用这些对话框来使人们迷惑在他们的页面上看不起眼的减肥广告，因此大多数浏览器目前可能不再显示它们。

resize

- 变化窗口大小
- 例：根据窗口大小，设置背景色

```
window.addEventListener('resize', colorScale);
```

动态背景色示例

键盘事件

- 按键会触发“keydown”和“keyup”事件
- keydown 按下
- keyup 键弹起来
- keypress
 - keypress 事件与 keydown 事件类似。当按钮被按下时发生该事件。
 - 然而，有些键不会触发 keypress，比如 ALT、CTRL、SHIFT、ESC。此时，请使用 keydown

键key

- 可以在event.key里检查是按的哪个键
 - 如果按住不放，每出现一个字母，就会响应一次
 - 回车键是 "Enter"
- 下面代码的功能是？

```
window.addEventListener("keydown",
  event => {
    if (event.key == "v") {
      document.body.style.background = "violet"; }
  });
```

按v换背景色

MDN

特殊键

- Ctrl键

```
if (event.key == " " && event.ctrlKey) {  
    console.log("继续!"); }  
}
```

MDN

键码

- 包括了键的编码
- 检查是否是按键d，切换显示模式为暗模式
 - l是亮模式

```
function whichKey(event) {  
  let key = event.code;  
  if (key == 'KeyD') {  
    darkMode();  
  }  
}
```

按D进暗模式示例

MDN

表单事件

- input 输入
- change 改变
- submit 提交按钮
- reset 重置按钮
- cut 剪切
- copy 复制
- paste 粘贴
- select 选择

input事件

- 对 input, textarea 等输入文本框
- 每当用户更改其内容时, 都会触发“input”事件

焦点事件

- 检测焦点变化
- “focus焦点”
 - 元素获得焦点时，浏览器在其上触发“focus”事件
- “blur模糊”
 - 元素失去焦点时，该元素发生“blur”事件
- 与前面讨论的事件不同，这两个事件不会传播
 - 当子元素获得焦点或失去焦点时，不会通知父元素的处理程序。
- 当用户从显示文档的浏览器选项卡或窗口中移入或移出时，该窗口对象将接收“焦点”和“模糊”事件

表单示例

- HTML

```
<p>姓名:<input type="text" data-help="名称"></p>  
<p>年龄:<input type="text" data-help="年龄"></p>  
<p id="help"></p>
```

表单示例

- `event.target` 是引发这个响应的元素

```
let help = document.querySelector("#help");
let fields = document.querySelectorAll("input");
for (let field of Array.from(fields)) {
  field.addEventListener("focus",
    event => {
      let text = event.target
        .getAttribute("data-help");
      help.textContent = text;
    });
}
```

用户输入焦点示例

触摸事件

- 触摸屏交互将导致
 - 触摸开始touchstart
 - 触摸移动touchmove
 - 触摸结束touchend
 - touchcancel

触摸点

- 许多触摸屏可同时检测到多个手指
- 因此这些事件没有与之关联的一组坐标，而是具有一个 touches 属性
 - 该属性包含一个数组，数组里是很多点
 - 每个点都有自己的 clientX, clientY, pageX 和 pageY 属性

事件传递

- 事件会沿着DOM Tree向上传播
 - 大多数事件都是在特定的DOM元素上调用的，然后传播到该元素的祖先，从而使与这些元素相关联的处理程序能够处理它们
- 可以编程，阻止事件往上进一步传播（stopPropagation）

```
button.addEventListener("mousedown",
  event => { console.log("发现鼠标按钮被按下");
    if (event.button == 2)
      event.stopPropagation();
  });
```

阻止默认行为

- 可以编程，阻止浏览器默认处理事件（preventDefault）的方法
 - 比如：实现快捷键，上下文菜单
- 尽量不这么做，用户会不习惯
- 例：点了以后，阻止默认的点击响应

```
link.addEventListener("click", event => {  
  console.log("Nope.");  
  event.preventDefault();  
});
```

后台工作进程

- 浏览器会等一个响应完成后，再执行下一个响应
 - 因此，如果一个响应耗时太长，浏览器看起来死机了似的，没有响应
- 对于确实要在后台执行一些耗时的操作而不冻结页面的情况，浏览器提供了称为Web Worker的功能
 - Worker是一个JavaScript进程，它和主脚本可以同时运行
- 与Worker通信
 - 为避免多个线程接触同一数据的问题，worker不与主脚本的环境共享全局范围或任何其他数据
 - 必须通过来回发送消息来与他们进行通信
 - 只能将可以表示为JSON的值作为消息发送

后台工作进程

```
let squareWorker
    = new Worker("code/squareworker.js");
squareWorker.addEventListener("message",
    event => { console.log(
        "The worker responded:", event.data);
    });
squareWorker.postMessage(10);
squareWorker.postMessage(24);
```


移走事件响应函数

- 例：按钮只能点一次

```
<button>Act-once button</button>
<script>
  let button = document.querySelector("button");
  function once() {
    console.log("Done.");
    button.removeEventListener("click", once);
  }
  button.addEventListener("click", once);
</script>
```

按钮只能点击一次示例

DOM事件

- DOMSubtreeModified
- DOMNodeInserted
- DOMNodeRemoved
- DOMNodeInsertedIntoDocument
- DOMNodeRemovedFromDocument

在系列事件完成后再做

- "mousemove", "scroll", "input" events 会一路fire
- 不能在里面做太消耗CPU的事
- 可以设置timeout, 延时再做
 - 在这个过程中, 可以不断取消, 直到用户停下来, 再做

`<textarea>Type something here...</textarea>`

在系列事件完成后再做

- 在用户输入过程，不断取消定时器，直到用户停下来，输出“终于敲完了”
- 给clearTimeout提供一个未定义的值，或已触发的定时器，没有关系。因此，不必担心调用它时没有指定合适的定时器。调就行了。

```
let timeout;
texta.addEventListener("input", () => {
  clearTimeout(timeout);
  timeout = setTimeout(
    () => console.log("终于敲完了!"), 500);
});
```

用户连续输入完成后再响应

在系列事件发生过程中定时做

- 在事件持续过程中，以一定的响应间隔，做些事情
 - 例如，在用户“mousemove”过程中，每250毫秒，显示鼠标的当前坐标

```
let scheduled = null;
window.addEventListener("mousemove", event => {
  if (!scheduled) {
    setTimeout(() => {
      document.body.textContent =
        `鼠标在 ${scheduled.pageX}, ${scheduled.pageY}`;
      scheduled = null;
    }, 250); }
  scheduled = event; });
```

鼠标移动过程中定期响应

练习1

- 编写一个显示气球🎈的页面（使用气球表情符号）。当您按下向上箭头时，它应该膨胀10%，当您按下向下箭头时，它应该缩小10%。
- 提示
 - 可通过在其父元素上设置font-size CSS属性（style.fontSize）来控制文本（注意🎈为文本）的大小。请记住，给值一个单位，例如，像素（10px）。
 - 箭头的键名是“ArrowUp”和“ArrowDown”
- 要求
 - 按键仅更改气球的大小，页面不能滚动

练习1:



- 再添加一个功能，如果气球超过一定大小，它将爆炸 (🔥)
- 提示
 - 爆炸意味着将🔴替换为🔥
- 要求
 - 要删除事件处理程序，这样就无法对🔥进行放大或缩小

起始代码

```
<p>🎈</p>
```

```
<script>  
  // Your code here  
</script>
```


提示

- 为“keydown”事件注册一个处理程序，然后查看event.key以确定是否按下了向上或向下箭头键。
- 将气球的当前大小保留在变量中，以便您可以在它现在大小的基础上再变化
- 定义一个更新气球大小的函数，这样您可以从事件处理程序中调用它，也可以在启动时调用一次以设置初始大小
- 将文本节点替换为另一个文本节点（使用replaceChild）或将其父节点的textContent属性设置为新字符串来将气球更改为爆炸 

练习2 鼠标路径元素

- 创建一堆（比如10个）固定大小和背景颜色的绝对定位的div元素
- 鼠标移动时，会在鼠标指针之后不断显示它们，以显示鼠标的路径
- 提示
 - 一种简单的解决方案是准备固定数量的元素
 - 每次发生“mousemove”事件时，将最后的一个元素移到到鼠标的当前位置

起始代码

```
<style>
  .trail { /* 鼠标路径元素的className */
    position: absolute;
    height: 6px; width: 6px;
    border-radius: 3px;
    background: teal;
  }
  body {
    height: 300px;
  }
</style>

<script>
  // Your code here.
</script>
```

提示

- 用循环来创建这些元素
- 将它们附加到文档中以使其显示
- 为了以后可以访问它们以更改位置，请将元素存储在数组中
- 通过保留计数器变量并在每次“mousemove”事件触发时对其加1来完成它们之间的循环
- 使用余数运算符（`%elements.length`）获取有效的数组索引，以选择要在给定事件中放置的元素

练习3

- 基于下面的代码片段，完成一个鼠标拖动元素大小的网页
- 阅读、调试和实验代码，理解每一句的功能
- HTML代码

<p>拖动栏以更改其宽度</p>

```
<div style="background: orange;  
    width: 60px; height: 20px"> </div>
```

练习3

- Javascript代码1

```
let lastX;
let bar = document.querySelector("div");

bar.addEventListener("mousedown",
  event => {
    if (event.button == 0) {
      lastX = event.clientX;
      window.addEventListener("mousemove", moved);
      event.preventDefault(); // 禁止默认行为选择
    }
  });
```

练习3

- Javascript代码2

```
function moved(event) {  
  if (event.buttons == 0) {  
    window.removeEventListener  
      ("mousemove", moved); }  
  else {  
    let dist = event.clientX - lastX;  
    let newWidth  
      = Math.max(10, bar.offsetWidth + dist);  
    bar.style.width = newWidth + "px";  
    lastX = event.clientX;  
  } }  
}
```

练习3（提示）

- 这是一个用鼠标把一个元素的宽度拖大的程序
- “mousemove”处理程序在整个窗口上注册，这样的话，即使在调整大小期间，鼠标移到了元素的外面，只要按住按钮，仍然能够更新元素大小
- 松开鼠标按钮时，需停止调整栏的大小，为此，使用了buttons属性（注意是复数形式），该属性告诉我们当前按住的是哪些按钮
 - 当它为零时，没有按钮按下
 - 按住按钮时，其值为这些按钮的代码之和 - 左按钮的代码为1，右按钮代码为2，中间代码为4

练习4：滚动状态条

- 基于下面的代码片段，完成一个浮动滚动条的网页
- 代码
 - 说明，position 设为 fixed; 固定，这个滚动条就会固定在窗口上，不会滚动

```
#progress {  
  border-bottom: 2px solid blue;  
  width: 0;  
  position: fixed;  
  top: 0;  
  left: 0;  
}  
<div id="progress"></div>
```

练习4：浮动滚动条

- 生成很长的文本，使网页变得很长，因此浏览时，就需要滚动了

```
document.body.appendChild(  
  document.createTextNode(  
    "supercalifragilisticexps ".repeat(1000)));
```

练习4： 浮动滚动条

- 设置滚动条的百分比
 - 全局变量 `innerHeight` 由系统提供，其中存着窗口高度
 - 从 `scrollHeight`（总可滚动高度）中减去该高度，就是我们能够滚动的最大数字
 - 将 `pageYOffset`（当前滚动位置）除以最大滚动位置，再乘以100，得到进度条的百分比。

```
let bar = document.querySelector("#progress");
window.addEventListener("scroll", () => {
  let max = document.body.scrollHeight
            - innerHeight;
  bar.style.width
    = `${(pageYOffset / max) * 100}%`; });
```

参考

[Eloquent Javascript 图书 \(英文版\)](#)

[W3school 事件介绍 \(中文版\)](#)