События

Мікалай Янкойць

Адукар

Повестка дня

1. Что такое события
2. Обработка событий. Event listeners
3. Объект события. События мыши, клавиатуры, тачскрина
4. Всплытие и погружение событий
5. Делегирование. Действия браузера по умолчанию

Что такое события

Событие — это сообщение от браузера о том, что **что-то произошло.** В частности, любое действие пользователя в браузере генерирует событие.

В DOM возникают события самых разных типов, например:

- события мыши (click, mouseover, mouseup, mousemove)
- события форм (submit, reset)
- события элементов управления (focus, blur)
- события клавиатуры (keydown, keyup)
- события изменения дерева DOM (DOMContentLoaded)
- события загрузки ресурсов (load, error)

Обработка событий

Каждому событию можно назначить **обработчик** — функцию, которая автоматически вызывается, как только событие произошло. Благодаря обработчикам наши страницы могут реагировать на действия пользователя.

Существует три способа назначения обработчика:

- через html-атрибут
- используя DOM-свойства
- через функцию addEventListener

Обработка событий

```
<input type="button" onclick="alert('Клик!')" value="Кнопка"/>
```

```
<input type="button" id="button" value="Кнопка" />
<script>
  button.onclick = function() {
    alert( 'Клик!' );
  };
</script>
```

addEventListener

Устанавливать обработчики в атрибутах HTML-тегов — плохая практика, а у установки через DOM-свойства (вроде onclick) есть серьёзный недостаток — невозможность назначить несколько обработчиков на одно событие.
Современный способ добавления обработчика события — метод **element.addEventListener(event, handler[, phase])**, где:

element — любой DOM-элемент, на котором может возникнуть событие
event — название события (строка)
handler — функция-обработчик
phase — фаза события*

*необязательный аргумент, будет рассмотрен позже

addEventListener

```
<body>
    <div>на меня можно кликнуть</div>
    <div>а на меня можно кликнуть дважды</div>
    <script>
        let clickables = document.body.getElementsByTagName('div');
        let firstDiv = clickables[0];
        let secondDiv = clickables[1]; // нашли оба DIV, уложили в переменные

        function reaction() {
            alert('привет! ты кликнул по диву!');
        }
        firstDiv.addEventListener('click', reaction); // передаём имя события и функцию

        secondDiv.addEventListener('dblclick', function() { // нередко функцию делают анонимной
            alert('обнаружен двойной щелчок по диву!');
        });
    </script>
</body>
```

removeEventListener

Чтобы удалить обработчик, назначенный через **addEventListener**, используется метод **element.removeEventListener(event, handler[, phase])** с полностью аналогичным набором аргументов.

```
<div>на меня можно кликнуть</div>
<div>а на меня можно кликнуть дважды</div>
<script>
    let clickables = document.body.getElementsByTagName('div');
    let firstDiv = clickables[0];
    let secondDiv = clickables[1];

    function reaction() {
        alert('привет! ты кликнул по диву!');
    }
    firstDiv.addEventListener('click', reaction); // добавили обработчик
    firstDiv.removeEventListener('click', reaction); // и тут же удалили, клик не отрабатывает

    secondDiv.addEventListener('dblclick', function() {
        alert('обнаружен двойной щелчок по диву!');
    });
    secondDiv.removeEventListener('dblclick', function() {  // так удаление НЕ РАБОТАЕТ!
        alert('обнаружен двойной щелчок по диву!');         // нужно передать ТУ ЖЕ функцию
    });
</script>
```

Объект события

Каждая функция-обработчик при вызове получает один аргумент — **объект события**. Из свойств этого объекта можно получить дополнительную информацию о событии:

- **event.type** — тип события (строка)
- **event.currentTarget** — элемент, на котором происходит обработка события
- **event.target** — элемент, на котором произошло само событие

Для разных видов событий в этом объекте будут содержаться и разные дополнительные данные.

Объект события

```
<body>
    <button class='btn-info'>нажми меня</button>
    <script>
        let btn = document.body.getElementsByTagName('button')[0];

        function pushTheButton(e) { // чаще всего объект события обозначают e или evt
            console.log(e.type);            // click
            console.log(e.currentTarget);   // сам объект btn
            console.log(e.currentTarget.classList.contains('btn-info')); // true
        }

        btn.addEventListener('click', pushTheButton);
    </script>
</body>
```

События мыши

Простые события мыши:

- mousedown — возникает в момент нажатия на клавишу мыши
- mouseup — в момент отпускания клавиши мыши
- mouseover — в момент появления курсора над элементом
- mouseout — в момент ухода курсора с элемента
- mousemove — в момент каждого движения курсора над элементом

Составные события:

- click — возникает после того, когда клавиша мыши нажата и отпущена (т.е. после mousedown и mouseup)
- dblclick — после двух кликов с малым промежутком времени

События мыши

```
<body>
    <button>нажми меня</button>
    <script>
        let btn = document.body.getElementsByTagName('button')[0];

        function pushTheButton(e) {
            console.log(e.type);    // просто выводим тип события
        }

        let mouseEvents = ['mousedown', 'mouseup', 'mouseover', 'mousemove', 'mouseout',
                           'click', 'dblclick'];

        mouseEvents.forEach(function(evtName) { // вешаем один и тот же обработчик на всё сразу
            btn.addEventListener(evtName, pushTheButton);
        });
    </script>
</body>
```

Объект события мыши

В объектах всех событий мыши доступны свойства:

- button — какая клавиша мыши была нажата (если была)
  - 0 — основная (обычно левая)
  - 1 — средняя (или нажат скролл)
  - 2 — правая
  - 3, 4 — дополнительные
- clientX, clientY — координаты курсора мыши в момент возникновения события. Считаются в пикселях относительно окна просмотра, верхний левый угол — (0; 0).

Объект события мыши

```
<head>
    <title>JS: events</title>
    <meta charset="utf-8">
    <style>
        .square { border: 1px solid #000; width: 500px; height: 500px; }
    </style>
</head>
<body>
    <div class='square'></div>
    <script>
        let area = document.querySelector('.square');

        area.addEventListener('mousedown', function(evt) {
            switch (evt.button) {
                case 0:
                    console.log('left button pressed'); break;
                case 1:
                    console.log('middle button pressed'); break;
                case 2:
                    console.log('right button pressed'); break;
                default:
                    console.log('unknown button pressed'); break;
            }
        });
        area.addEventListener('mousemove', function(evt) {
            console.log(`coordinates: ${evt.clientX}; ${evt.clientY}`);
        });
    </script>
</body>
```

События клавиатуры

- keydown — возникает в момент нажатия на клавишу
- keyup — в момент отпускания клавиши

Полезные свойства объекта клавиатурного события:

- key — введённый символ; для цифр — цифра, для букв — буква (с учётом регистра и языка раскладки), для специальных клавиш — их название («Enter», «Shift», «F9»)
- code — код клавиши; для букв — ‘KeyA’, ‘KeyB’, для цифр — ‘Digit0’, ‘Digit1’, для специальных клавиш — название и расположение (‘F12’, ‘Tab’, но ‘ShiftLeft’, ‘ControlRight’). Значение свойства code не зависит от выбранного языка раскладки

Кроссбраузерное получение символа

Раньше получение нажатой клавиши было куда более сложным из-за отсутствия единого стандарта:

```
// event.type должен быть keypress
function getChar(event) {
  if (event.which == null) { // IE
    if (event.keyCode < 32) return null; // спец. символ
    return String.fromCharCode(event.keyCode)
  }

  if (event.which != 0 && event.charCode != 0) { // все кроме IE
    if (event.which < 32) return null; // спец. символ
    return String.fromCharCode(event.which); // остальные
  }

  return null; // спец. символ
}
```

События тачскрина

- touchstart — возникает в момент нового касания тачскрина
- touchend — в момент поднятия пальца/стилуса
- touchcancel — отмена касания
- touchmove — движение пальца по поверхности (свайп)

Обычные события нажатия на клавиши мыши — click, mousedown — срабатывают и при тапе по экрану.

Разумеется, события mouseout, mouseover, mousemove на тачскринах не срабатывают.

Событие DOMContentLoaded

DOMContentLoaded — событие, которое возникает при отображении документа в браузере в момент, когда дерево DOM полностью выстроено и готово к использованию.

Очень часто этот момент выбирают для **запуска любого кода**, работающего с DOM. Таким образом мы избегаем проблем с «недостроенной» структурой страницы.

Это событие возникает на глобальном объекте window.

```
<script>
    addEventListener('DOMContentLoaded', function() {
        /* весь код работы с DOM здесь */
    });
</script>
```

Практика

1. Сделайте так, чтобы по нажатию на кнопку выводился alert
2. Создайте кнопку с произвольным текстом, сделайте так, чтобы по нажатию текст изменялся на button.
3. Создайте форму с полем для ввода и двумя кнопками «enable» и «disable». Сделайте так, чтобы по нажатию на кнопку «disable» поле становилось неактивным, на «enable» — активным.
4. Создайте объект произвольной формы и цвета. Сделайте так, чтобы объект можно было двигать, а его координаты на странице выводились в HTML-элементе output.

Всплытие событий

Когда на элементе происходит событие, обработчики срабатывают сначала **на самом элементе**, потом **на его родителе**, потом поочерёдно **на всех предках**, вплоть до document.
Всплывает большинство событий DOM.

```
<div> <p> <span>нажми меня</span> </p> </div>
<script>
    let span = document.querySelector('span');
    let p = document.querySelector('p');
    let div = document.querySelector('div');

    function reaction(e) {
        console.log(e.currentTarget.tagName); // выводим имя тега нажатого элемента
    }
    span.addEventListener('click', reaction);
    p.addEventListener('click', reaction);
    div.addEventListener('click', reaction);
    // если кликнуть по span, выведется три строки!
</script>
```

event.target

Чтобы отследить, на каком элементе событие **возникло**, в объекте события есть свойство target (см. слайд 9):

```
<div> <p> <span>нажми меня</span> </p> </div>
<script>
    let span = document.querySelector('span');
    let p = document.querySelector('p');
    let div = document.querySelector('div');

    function reaction(e) {
        console.log(e.currentTarget.tagName);    // тут при клике по span будет три разных тега
        console.log(e.target.tagName);           // а тут один и тот же — SPAN
    }
    span.addEventListener('click', reaction);
    p.addEventListener('click', reaction);
    div.addEventListener('click', reaction);
</script>
```

this в обработчиках события

Объект this в обработчике события равен event.currentTarget.

```
<div> <p> <span>нажми меня</span> </p> </div>
<script>
    let span = document.querySelector('span');
    let p = document.querySelector('p');
    let div = document.querySelector('div');

    function reaction(e) {
        console.log(e.currentTarget == this);    // всегда будет true
    }
    span.addEventListener('click', reaction);
    p.addEventListener('click', reaction);
    div.addEventListener('click', reaction);
</script>
```

Исключением могут стать обработчики, описанные стрелочными функциями (будут рассмотрены позже).

Погружение событий*

На самом деле, ещё до «возникновения» события на конкретном элементе оно проходит путь, обратный всплытию — от document через его потомков к event.target. Этот процесс называется **погружением** события.

Методы addEventListener / removeEventListener могут принимать третий аргумент, в котором указывается конкретная фаза события (погружение, цель, всплытие).
На практике погружение используется нечасто.

Отмена всплытия событий

Мы можем отменить всплытие события на любом этапе, вызвав метод event.stopPropagation():

```
<div> <p> <span>нажми меня</span> </p> </div>
<script>
    let span = document.querySelector('span');
    let p = document.querySelector('p');
    let div = document.querySelector('div');

    function reaction(e) {
        console.log(this.tagName);  // выводим тег текущей цели
        if (this.tagName == 'P') {
            e.stopPropagation(); // останавливаем всплытие на P, до DIV событие не дойдёт
        }
    }
    span.addEventListener('click', reaction);
    p.addEventListener('click', reaction);
    div.addEventListener('click', reaction);
</script>
```

Действия браузера по умолчанию

Многие события автоматически приводят к действиям браузера: например, движение курсора над текстом с зажатой левой клавишей мыши выделяет текст, а ввод текста с клавиатуры на активном input текстового типа приводит к выводу этого текста в тот самый input.

Мы можем предотвратить многие действия по умолчанию, вызвав метод event.preventDefault().

```
<input type='text' id='test'>
<script>
    let test = document.getElementById('test');

    test.addEventListener('keydown', function(e) {
        e.preventDefault(); // символы не выводятся в input!
    });
</script>
```

Делегирование событий

Если у нас на странице есть много схожих элементов, на которых события должны обрабатываться одинаково, мы можем использовать приём **делегирования**:

- назначаем обработчики не на каждый из элементов, а только на их общего предка
- в этом обработчике получаем event.target, чтобы узнать, на каком элементе **возникло** событие
- обрабатываем событие, используя event.target

Делегирование — один из самых полезных приёмов при работе с событиями DOM.

Делегирование событий

```
<head>
    <title>JS: events</title>
    <meta charset="utf-8">
    <style>
        .square { border: 1px solid #000; width: 200px; height: 200px; }
        .colored { background-color: yellow; }
    </style>
</head>
<body>
    <main class='container'>
        <div class='square'></div>
        <div class='square'></div>
        <div class='square'></div>
    </main>
    <script>
        let delegate = document.querySelector('.container');     // находим общего родителя

        delegate.addEventListener('click', function(evt) {       // обрабатываем клик на нём
            if (evt.target.classList.contains('square')) {       // проверяем, откуда всплыл клик
                evt.target.classList.toggle('colored');          // при необходимости переключаем класс
            }
        });
    </script>
</body>
```

Практика

1. Создайте поле ввода, сделайте с помощью JS так, чтобы в него можно было вводить только числовые значения.
2. Создайте блок div, в качестве изображения фона установите ему изображение закрытой папки. Добавьте событие, которое выполняется при двойном клике на блоке и заменяет фон блока на изображение открытой папки.
3. Добавьте в документ 300-400 блоков div квадратной формы с размерами сторон 30px и цветом фона. Добавление элементов выполните с помощью цикла, цвет попытайтесь раздать случайным образом (это условие выполнять не обязательно). Добавьте событие при наведении мыши, которое будет скруглять данные блоки с помощью border-radius до круга. Для замедления эффекта скругления в CSS можно добавить transition.

Внеклассное чтение

https://learn.javascript.ru/events (главы 1-4)

https://learn.javascript.ru/event-details (главы 1, 2, 4)

https://developer.mozilla.org/en-US/docs/Web/Events

https://developer.mozilla.org/en-US/docs/Web/API/Touch_events