6 занятие. События

Теоретическая часть

Практика

Статьи:

Введение в события
Справочник по событиям
Браузер: документ, события, интерфейсы

Замыкания в JavaScript

Браузерные события в Javascript — urvanov.ru
Выразительный JavaScript: Обработка событий / Хабр

EventTarget
Метод EventTarget.addEventListener()
Метод EventTarget.removeEventListener()
Создание и вызов событий
KeyboardEvent

Примеры стандартных компонентов для самостоятельного изучения:
Components Bootstrap v4.5
Скрипты для сайтов

События

События — это действие, которое пользователь делает с какими-то элементами страницы (например, кликает по кнопке или нажимает клавишу).

События:

1. пользовательские действия (события ввода)
2. загрузка (события браузера)
3. кастомные события (созданные приложением)

События на DOM-элементах:

- общие для всех элементов события: клик, наведение курсора, убирание курсора
- специфические события для элементов определенного назначения: фокус, увод фокуса, ввод неправильного значения, окончание загрузки, ошибка при загрузке, ввод с клавиатуры

“*Добавить обработчик*” значит что какой-то код будет выполнен в тот момент, когда происходит какое-то событие на странице

Событие	Описание
onclick	возникает при щелчке левой кнопкой мыши на элементе
onload	возникает при загрузке объекта
onunload	срабатывает в том случае, когда страница не загрузилась по каким-либо причинам, либо при закрытии окна (вкладки)
onchange	возникает при изменении значения элемента формы, вроде текстового поля или списка(применяется к тегам <input>, <keygen>, <select>, <textarea>)
onmouseover	срабатывает, когда курсор мыши наводится на элемент, или на один из его потомков
onmouseout	срабатывает, когда курсор мыши выходит за пределы элемента
onmousedown	срабатывает в момент нажатия на кнопку мыши над элементом
onmouseup	срабатывает когда пользователь отпускает кнопку мыши над элементом
onblur	возникает при потере элемента фокуса
onfocus	возникает при получении элементом фокуса

События **onload** и **onunload** вызываются, когда пользователь входит или выходит со страницы. Это может быть полезным для выполнения действий после загрузки страницы.

```
<body onload="doSomething()">
```

Событие **window.onload** может использоваться для запуска кода после полной загрузки страницы.

```
window.onload = function() {
  //some code
}
```

Событие при клике на кнопку

1 вариант: События могут быть добавлены в элементы HTML в качестве атрибутов.

```
<button onclick="openModal()">some text</button>
```

Пример

Применяется на сегодняшний день крайне редко.

---

2 вариант: Обработчики событий могут быть присвоены к элементам.

```
let button = document.querySelector('button');
button.onclick = function() {
  console.log('Клик!');
};
```

function() - функции, которые создаются в момент передачи и не имеют названия, называются **анонимными функциями**.

Мы передаем функцию в обработчики, а не вызываем её.

JavaScript выполняет программу последовательно, команду за командой. Но к событиям это не относится. Инструкции обработчика выполняются не сразу, а только тогда, когда произойдёт событие. События **асинхронны**, то есть происходят не по расписанию в какое-то конкретное время, друг за другом, а тогда, когда понадобится.

Пример: изменение классов элементов при клике

Задача: Модальное окно с контактами компании

Запрограммируй всплывающее окно с контактами компании. Тебе нужен элемент с классом **modal**.

При клике по кнопке с классом **button-open** попапу должен добавляться класс **modal--show**, так окно появится на странице.

По клику по кнопке с классом **button-close** у окна должен удаляться класс **modal--show**, и попап должен закрыться.

Пользовательские атрибуты data-

HTML — гибкий язык, и в нём можно создавать свои собственные атрибуты. Имена таких атрибутов начинаются с префикса **data-**, после которого идёт любое выбранное разработчиком слово.

Получить значение такого атрибута можно при помощи свойства **dataset**, после которого указывают имя атрибута без префикса data-:

```
элемент.dataset.имяАтрибутаБезПрефикса
```

```
<p data-user-name="Piter">
let element = document.querySelector('p');
console.log(element.dataset.userName);
```

Пример: появление тултипа-подсказки

---

3 вариант: Слушатель событий

```
element.addEventListener(event, function, useCapture);
```

```
button.addEventListener('click', function () {
  // Инструкции
});
```

Первый параметр - название события, которое мы хотим поймать. Название записывается строкой, поэтому обязательно указывается в кавычках. Для всех событий есть специальные зафиксированные названия
Справочник по событиям

Второй параметр - функция. Это и есть обработчик события. Можно передать уже написанную ранее функцию, либо описать поведение новой безымянной функции. Функция, переданная в обработчик, не выполняется сразу.

Третий параметр - булево значение, указывающим в каких случаях использовать всплытие событий и перехват событий. Этот параметр является опциональным. Значением по умолчанию является **false**, который означает, что используется тип **всплытия**; если значение установлено на **true**, то событие будет использовать тип **перехват**.

Выполнение событий: всплытие и перехват

Существует два способа выполнения событий в объектной модели HTML: **всплытие и перехват**.

Каждое из выполнений событий позволяет определять порядок элементов при возникновении событий.

```
<div>
  <p>
    <a>text</a>
  </p>
</div>
```

Какой из обработчиков событий будет выполнен первым?

Всплытие: событие лежащего глубже всех элемента обрабатывается в первую очередь, а затем обрабатываются остальные события.
Всплытие идет по объектной модели вверх.

Некоторые события могут не всплывать (focus, blur, invalid)

Перехват: самое внешнее событие элемента будет обработано первым, затем будут обрабатываться лежащие глубже. Захватываются все события.
Перехват идет по объектной модели вниз.

Пример

Объект Event

В объект **event** записывается событие, которое обрабатывается прямо сейчас.

Объект **event** может быть параметром функции-обработчика. Он передаётся браузером в эту функцию в момент наступления события.

Виден глобально, т.е. им можно пользоваться на любом уровне приложения

Чтобы использовать свойства и методы объекта **event**, достаточно указать этот объект параметром функции-обработчика и написать инструкции.

Среди некоторых разработчиков принято называть параметр сокращённо — **evt**

```
link.addEventListener('click', function(evt) {
  // Отменяем действие по умолчанию
  evt.preventDefault();
  // Добавляем инструкции для события клика
  console.log('Произошёл клик');
});
```

Объект Event помогает:

– **keydown/keyup/keypress**: определить какую именно клавишу нажал пользователь

– **mousedown/click**: определить координаты нажатия

– **любое событие**: определить время наступления события

– **событие с действием по умолчанию:** отменить сохранение страницы по (Ctrl+S) и написать свой код

Информация о событии Event

– **x, y, clientX, layerX, offsetX, pageX…**
координаты курсора

– **target**
элемент на котором произошло событие

– **currentTarget**
элемент на котором событие было обработано

– **metaKey, shiftKey, ctrlKey, altKey**
нажатые клавиши в момент наступления события

– **timestamp**
время наступления события

Управление событием Event

1. **preventDefault() -** отмена действия по умолчанию

Некоторые элементы страницы имеют действия по умолчанию - **дефолтные действия**. Например, клик по кнопке отправления формы вызывает отправку данных этой формы на сервер, а при клике по ссылке браузер переходит по этой ссылке.

Дефолтные действия можно отменить при необходимости с помощью **event**:

```
evt.preventDefault();
```

2. **stopPropagation() -** прекращение распространения

Распространение события можно останавливать, с помощью метода Event.stopPropagation(). Иногда это чревато тем, что невозможно будет отловить событие.

Задача: верстальщик сверстал кнопку для попапа ссылкой. Нужно сделать открытие попапа не меняя разметки. В случае, если JavaScript на странице не работает, по кнопке должен происходить переход на страницу контактов.

Обработка событий клавиатуры

У события «нажатие на клавишу» есть специальное название — **keydown**

Слушать это событие можно только на элементах, которые имеют состояние фокуса: поля ввода, кнопки, элементы с атрибутом tabindex, документ. При нажатии фокус должен находиться на соответствующем элементе.

```
document.addEventListener('keydown', function() {
  // Код отсюда выполнится при каждом нажатии любой клавиши
});
```

У объекта **event** события есть свойство, хранящее код клавиши, которую нажал пользователь. Это свойство называется **code**.

Подробнее: KeyboardEvent.code

key='Escape' | code='Escape'
key='Enter' | code='Enter'
key='ArrowUp' | code='ArrowUp'

и так далее...

Задача: дописать код попапа так, чтобы он закрывался по нажатию на клавишу ESC.

Закрытие должно срабатывать **только** по этой клавише, нажатие на другие клавиши не должны влиять на положение всплывающего окна.

Обработчик событий onscroll, прокрутка страницы

Прокрутка
Анимация при прокрутке страницы

```
window.onscroll = function () {
  console.log('Страница прокручена');
}
```

Величина горизонтальной прокрутки хранится в свойстве **pageXOffset**.

Свойство **pageYOffset** окна браузера содержит число пикселей, на которое пользователь прокрутил страницу по вертикали:

```
// Если мы на самом верху страницы
console.log(window.pageYOffset); // Выведет: 0

// Прокрутим страницу на 200px
console.log(window.pageYOffset); // Выведет: 200
```

Как заставить страницу прокрутиться при клике на элемент?

С помощью метода **scrollTo**:

window.scrollTo(координата-X, координата-Y);

Координата X указывает, куда нужно прокрутить страницу по горизонтали, а координата Y — куда нужно прокрутить страницу по вертикали.

Когда браузер выполнит инструкцию, указанная точка окажется в левом верхнем углу окна. Координаты задаются в пикселях, но указывать единицы измерения не нужно:

```
// Прокрутит страницу на 100px вправо и на 50px вниз
window.scrollTo(100, 50);
```

По умолчанию автоматическая прокрутка в браузерах происходит скачком. Чтобы сделать её более плавной, в CSS нужно использовать свойство **scroll-behavior** со значением **smooth**.

Пример с исчезающей кнопкой “Наверх”

Практика

Опять окно

Сделать открытие и закрытие модального окна со спецпредложением при клике на соответствующие кнопки. Также пользователь может закрыть попап при нажатии ESC

Бургеры

Создай адаптивное меню, которое сворачивается в бургер-меню при ширине экрана менее 1024 пикселей. Более этой ширины меню только десктопное.

Интерактивная галерея

Галерея состоит из нескольких миниатюр (элементы с классом **gallery-preview**) и большой фотографии (класс **full-photo**). По клику на миниатюру должно меняться большое изображение. Большая картинка должна соответствовать превью, по которой кликнул пользователь.

Данные для изображений собраны в массив **photos**. Каждый элемент массива — путь до полноразмерной фотографии. Порядок элементов в этом массиве такой же, как и порядок миниатюр в разметке.

Изображения превью должны лежать внутри кнопок с классом gallery-preview, клики именно по этим кнопкам будут менять содержимое большого изображения (класс full-photo).

Например:

```
<section class="gallery">
  <h1>Фотогалерея</h1>
```

```
  <div class="gallery-full">
    <img class="full-photo" src="https://picsum.photos/id/1011/500/300"
         width="550" height="300" alt="Фото большое">
  </div>

  <div class="gallery-previews">

    <button class="gallery-preview" type="button">
      <img src="https://picsum.photos/id/1011/90/90" alt="Превью">
    </button>
    <button class="gallery-preview" type="button">
      <img src="https://picsum.photos/id/1015/90/90" alt="Превью">
    </button>
    <button class="gallery-preview" type="button">
      <img src="https://picsum.photos/id/102/90/90" alt="Превью">
    </button>
    <button class="gallery-preview" type="button">
      <img src="https://picsum.photos/id/1025/90/90" alt="Превью">
    </button>
    <button class="gallery-preview" type="button">
      <img src="https://picsum.photos/id/1043/90/90" alt="Превью">
    </button>

  </div>
</section>
```

Массив данных:

```
let photos = [
  'https://picsum.photos/id/1011/500/300',
  'https://picsum.photos/id/1015/500/300',
  'https://picsum.photos/id/102/500/300',
  'https://picsum.photos/id/1025/500/300',
  'https://picsum.photos/id/1043/500/300'
];
```

Слайдер изображений
Создайте слайдер изображений. При клике “Назад” показывается предыдущая фотография, при клике “Вперед” - следующая. Если фото последняя, то следующей должна быть первая, аналогично в другую сторону.
В разметке есть только 2 кнопки и изображение. При клике на кнопку нужно менять адрес изображения на новый из массива фото. Первое фото соответствует первому элементу массива и находится сразу в разметке.

Разметка:
```
<div class="slider">
  <button class="prev-btn">
    prev </button>
  <img src="https://picsum.photos/id/1011/500/300" alt="фото" width="500" height="300">
  <button class="next-btn"> next </button>
</div>
```

Массив данных:
```
let photos = [
  'https://picsum.photos/id/1011/500/300',
  'https://picsum.photos/id/1015/500/300',
  'https://picsum.photos/id/102/500/300',
  'https://picsum.photos/id/1025/500/300',
  'https://picsum.photos/id/1043/500/300'
];
```

Вкладки (табы)

Пример поведения
Пример поведения

Создать страницу с 3мя вкладками с разным содержимым

В качестве макета можно воспользоваться:
https://www.figma.com/file/XCgvIOgHRTkjr0J4dZ7IHY/Components?node-id=2%3A806

Спойлер / аккордеон

Пример поведения

Создать аккордеон с минимум 3мя частями, каждая из которых показывает/скрывает свое содержимое при клике на заголовок