НЕТОЛОГИЯ

ОБЪЕКТ СОБЫТИЯ

АЛЕКСЕЙ СУДНИЧНИКОВ

АЛЕКСЕЙ СУДНИЧНИКОВ

Руководитель группы разработки
«Портал ПФДО»

@avsudnichnikov

ПЛАН ЗАНЯТИЯ

1. Понятие события. Определение, виды, примеры
2. Обработчики события. Определение, разница, способы задачи
3. Разбор на примере onclick
4. Контекст обработчика this
5. Объект события (event)
6. События клавиатуры
7. Другие обработчики событий
8. Выводы, краткий обзор выученного

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: какие способы получения HTML-элемента вы знаете?

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: какие способы получения HTML-элемента вы знаете?

Ответ: для получения HTML элементов со страницы существуют методы:

- getElementsByTagName
- getElementsByClassName
- getElementsByName
- getElementById

```
let images = document.getElementsByTagName("img");
let colors = document.getElementsByClassName("blue");
let elements = document.getElementsByName("age");
let text = document.getElementById("text");
```

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: как получить массив HTML элементов?

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: как получить массив HTML элементов?

Ответ: для получения массива HTML элементов следует воспользоваться функцией `Array.from()`

СОБЫТИЯ

СОБЫТИЯ

Javascript — это язык управления сценариями на веб-странице. Каждый раз, взаимодействуя со страницей в браузере, мы можем совершать какие-то действия, будь то клик мыши, ввод логина и пароля, заполнение анкеты или выбор правильного ответа из предложенного списка.

Совершая простой клик мыши, мы ожидаем ответную реакцию от страницы. Этот процесс можно коротко назвать **событием**.

События — действия, которые происходят в момент взаимодействия с каким-либо элементом на web–странице в браузере. Существует много видов событий.

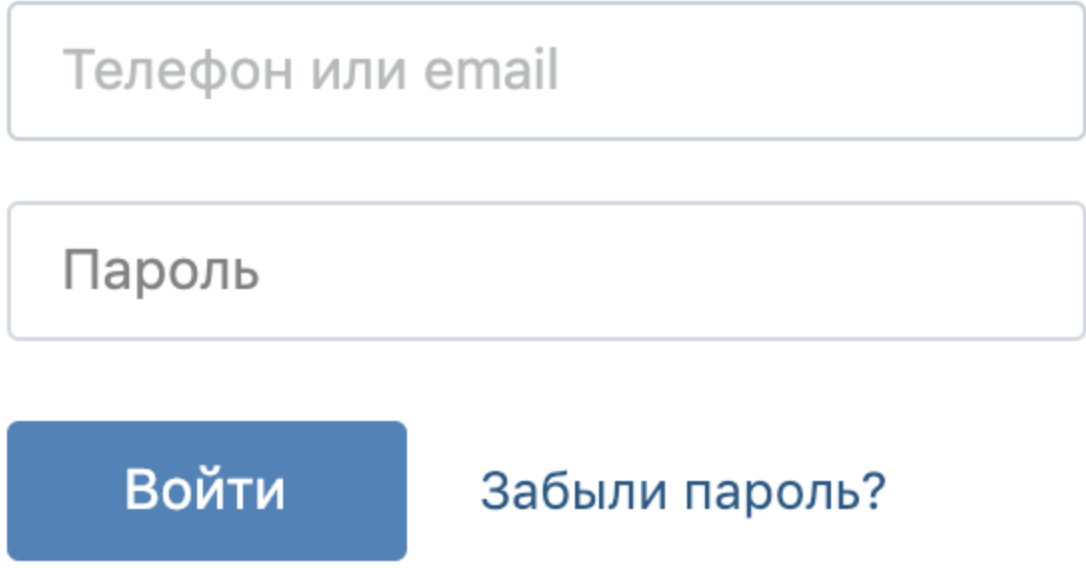
ЧТО МЫ ЖДЕМ В ОТВЕТ НА СОВЕРШЕННОЕ ДЕЙСТВИЕ?

Мы хотим заставить веб-страницу не просто отображать статическую информацию, заранее запрограммированную в HTML-разметке, но и, использовав данные, что мы вводим, открыть частную страницу или, как пример, обработать наши данные для выдачи новой порции информации.

Другими словами, совершая какое-либо действие на веб-странице, мы ждем обработку.

ЧТО МЫ ЖДЕМ В ОТВЕТ НА СОВЕРШЕННОЕ ДЕЙСТВИЕ?

Пример: форма. Заполняем логин и пароль и ожидаем перехода на личную страницу.



СОБЫТИЯ

Самые распространенные события в JavaScript:

- События мыши:
  - *click* – клик на левую кнопку мыши;
  - *contextmenu* – клик на праву кнопку мыши;
  - *mouseover* – когда мышь наводится на элемент;
  - *ondblclick* – двойной клик мыши.
- События клавиатуры:
  - *onkeypress* – когда нажимаем на кнопку;
  - *onkeyup* – когда отпускаем кнопку.
- События форм и элементов:
  - *onfocus* – когда элемент в фокусе;
  - *onblur* – когда элемент теряет фокус;
  - *onsubmit* – когда отправляем форму.

СОБЫТИЯ

Приведем еще один пример события, которое происходит каждый раз при загрузке контента на любой веб странице – *onload*.

```
window.onload = function() {
    alert(“Страница загружена”)
}
```

Реакция на действие пользователя задается в обработчике событий.

ОБРАБОТЧИКИ СОБЫТИЙ. ОПРЕДЕЛЕНИЕ, РАЗНИЦА, СПОСОБЫ ЗАДАЧИ

ОБРАБОТЧИКИ СОБЫТИЙ. ОПРЕДЕЛЕНИЕ, РАЗНИЦА, СПОСОБЫ ЗАДАЧИ

Обработчик — это функция, в которой описана реакция на событие.

Существует несколько способов назначить событию обработчик. Сейчас мы их рассмотрим, начиная с самого простого.

СПОСОБ №1 – ЧЕРЕЗ АТРИБУТ ТЭГА В HTML

Обработчик события можно указать в виде inline-записи, прямо в атрибуте `on` события. Например, для обработки события *contextmenu* на кнопке `<button>`, можно назначить обработчик `oncontextmenu` вот так:

```
<div>Правый клик на этой кнопке выведет "Клик".</div>
<button oncontextmenu="alert('Клик!');">Правый клик сюда</button>
```

Это событие выведет alert-окно при нажатии правой кнопкой мыши.

Напомним, имена атрибутов HTML-тегов нечувствительны к регистру, поэтому атрибут `oNcOnTeXtmEnU` сработает также, как `onContextmenu` или oncontextmenu.

СПОСОБ №1 – ЧЕРЕЗ АТРИБУТ ТЭГА В HTML

Вопрос: когда использовать такой способ задачи обработчика?

СПОСОБ №1 – ЧЕРЕЗ АТРИБУТ ТЭГА В HTML

Вопрос: когда использовать такой способ задачи обработчика?

Ответ: такой способ установки обработчиков очень удобен - он нагляден и прост, поэтому часто используется в решении простых задач.

У этого способа установки обработчика есть и **минусы**. Как только обработчик начинает занимать больше одной строки, читабельность резко падает.

Впрочем, сколько-нибудь сложные обработчики в HTML никто не пишет. Вместо этого лучше устанавливать обработчики из JavaScript-способами, которые будут представлены дальше.

СПОСОБ №2 – ЧЕРЕЗ СВОЙСТВО ОБЪЕКТА

Нам понадобится выполнить следующие шаги:

1. Найти элемент
2. Назначить обработчик `on` и имя обработчика

Представим, что наша разметка выглядит как `input` с атрибутом `id="clickMe"`:

```
<input id="clickMe" type="button" value="Нажми меня"/>
```

Для того, чтобы найти его с помощью Javascript, воспользуемся методом `getElementById()`:

```
document.getElementById('clickMe');
```

СПОСОБ №2 – ЧЕРЕЗ СВОЙСТВО ОБЪЕКТА

Теперь обратимся к его свойству `onclick`:

```
document.getElementById('clickMe').onclick;
```

Присвоим ему функцию, которая будет выводить информацию в `console.log`:

```
1 document.getElementById('clickMe').onclick = function() {
2     console.log('Button was clicked')
3 };
```

СПОСОБ №2 – ЧЕРЕЗ СВОЙСТВО ОБЪЕКТА

Обратим внимание:

1. Это свойство, а не атрибут, поэтому пользоваться лучше прямым присвоением значения.
2. Свойства чувствительны к регистру.
3. Обработчик — это не просто текст, а самая настоящая функция JavaScript.

Все вызовы типа `getElementById` должны запускаться после описания соответствующего HTML-узла, а лучше — после окончания загрузки страницы. Иначе узел просто не будет найден.

СПОСОБ №2 – ЧЕРЕЗ СВОЙСТВО ОБЪЕКТА

Безусловно, в качестве присвоенного значения свойству может выступать не только анонимная функция:

```
function toClick() {
    console.log(' Information ');
}

document.getElementById(‘clickMe').onclick = toClick;
```

СПОСОБ №2 – ЧЕРЕЗ СВОЙСТВО ОБЪЕКТА

Обратите внимание, что присваивать функцию мы должны без ее вызова. Так как у функции `toClick` нет вызова return, то обработчик не сработает, ведь значение функции будет присвоено `undefined`.

Сравнивая этот способ с инлайновой записью, мы заметим, что для работы непосредственно с атрибутом нужен именно вызов функции:

```
<input onclick=”toClick();” type="button" value="Нажми меня"/>
```

СПОСОБ №2 – ЧЕРЕЗ СВОЙСТВО ОБЪЕКТА

Описанная установка обработчика через свойство — очень популярный и простой способ. У него есть один **недостаток**: на элемент можно повесить только один обработчик нужного события.

```
let input = document.getElementById('clickMe');
input.onclick = function() {
    console.log(1)
}
input.onclick = function() {
    console.log(2)
}
```

Второй вызов заменит первый. Поэтому с этим способом можно ставить только один обработчик на событие.

СПОСОБ №3 – ЧЕРЕЗ МЕТОД ADDEVENTLISTENER

Этот метод является **современным** способом назначить событие, и при этом позволяет использовать сколько угодно любых обработчиков, что является несомненным преимуществом по сравнению с ранее описанными методами.

СПОСОБ №3 – ЧЕРЕЗ МЕТОД ADDEVENTLISTENER

Назначение обработчика осуществляется вызовом `addEventListener` с тремя аргументами:

```
element.addEventListener(event, handler[, phase]);
```

1. Аргумент *event* – это имя события, например click.
2. Аргумент *handler* – ссылка на функцию, которую надо назначить обработчиком.
3. Аргумент *phase* – опциональный аргумент, используется редко, о нем вы узнаете на продвинутом курсе Javascript.

СПОСОБ №3 – ЧЕРЕЗ МЕТОД ADDEVENTLISTENER

Метод `addEventListener` позволяет добавлять несколько обработчиков на одно событие одного элемента, например:

```
function handler1() {
  alert('Спасибо!');
};

function handler2() {
  alert('Спасибо ещё раз!');
}

elem.onclick = function() { alert("Привет"); };
elem.addEventListener("click", handler1); // Спасибо!
elem.addEventListener("click", handler2); // Спасибо ещё раз!
```

СПОСОБ №3 – ЧЕРЕЗ МЕТОД ADDEVENTLISTENER

Удаление обработчика осуществляется вызовом `removeEventListener`:

```
function handler() {
  alert( 'Спасибо!' );
}

input.addEventListener("click", handler);
// ....
input.removeEventListener("click", handler);
```

СПОСОБ №3 – ЧЕРЕЗ МЕТОД ADDEVENTLISTENER

Для удаления нужно передать именно ту функцию-обработчик, которая была назначена.

Обратите внимание, что имя события указывается без префикса “on”. Решение W3C работает во всех современных браузерах, кроме Internet Explorer.

Как и в других случаях, вы должны передать имя обработчика без круглых скобок, иначе функция будет выполнена сразу, а в качестве обработчика будет передан лишь её результат.

ВЫВОД

Обработчик — это функция.

Существует 3 способа задать обработчик:

1. HTML-атрибуты.
2. Свойства объектов.
3. Специальные методы.

РАЗБОР НА ПРИМЕРЕ ONCLICK

РАЗБОР НА ПРИМЕРЕ ONCLICK

Рассмотрим пример работы с содержимым элемента, используя одно из самых распространенных событий — нажатие левой кнопкой мыши по элементу, click.

Мы имеем такую разметку документа:

```
<p id="text">Нажми на кнопку!</p>
<button id="clickMe">Click me!</button>
<script src="script.js"></script>
```

РАЗБОР НА ПРИМЕРЕ ONCLICK

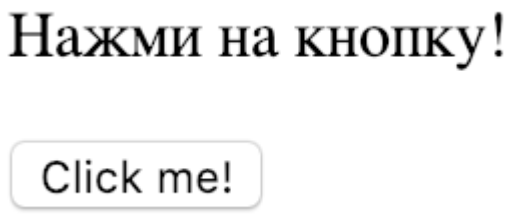
В скрипте мы нашли элементы параграфа и кнопки по идентификаторам text и clickMe и присвоили обработчик на событие `click`:

```
let button = document.getElementById("clickMe");
let text = document.getElementById("text");

button.addEventListener('click', function() {
    text.textContent = "Вы нажали кнопку";
});
```

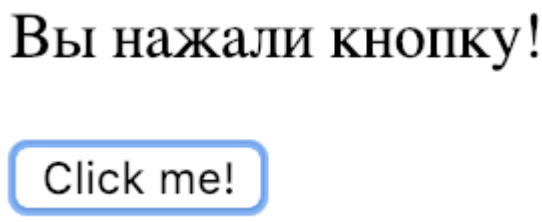
РАЗБОР НА ПРИМЕРЕ ONCLICK

Теперь при каждом клике на кнопку наш текст будет меняться:



РАЗБОР НА ПРИМЕРЕ ONCLICK

Теперь при каждом клике на кнопку наш текст будет меняться:



РАЗБОР НА ПРИМЕРЕ ONCLICK

Вопрос на засыпку: работает ли такой код?

```
window.addEventListener('onClick', (e) => {
    console.log(e.type);
})
```

КОНТЕКСТ ОБРАБОТЧИКА THIS

КОНТЕКСТ ОБРАБОТЧИКА THIS

Мы используем `this` так же, как местоимение в языках, подобных английскому или русскому.

Мы не всегда называем имя или предмет, а говорим применительно к нему «этот». То же самое происходит при использовании ключевого слова `this` в языках программирования.

`This` всегда является ссылкой на свойство объекта, но какой это объект, зависит от контекста.

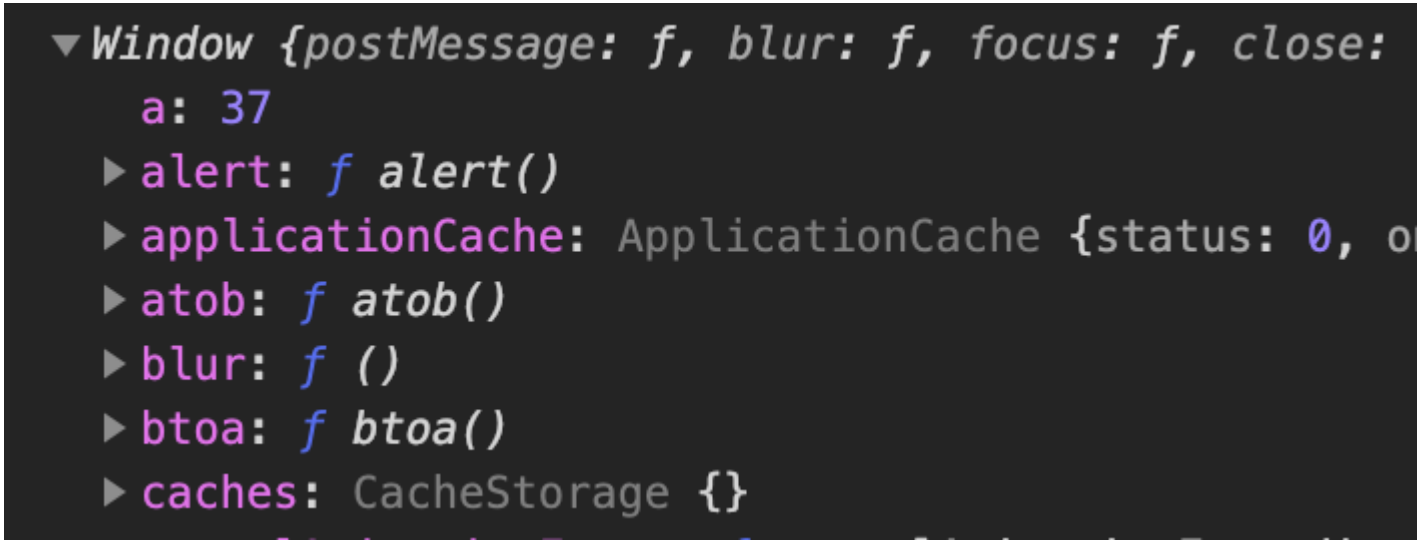
КОНТЕКСТ ОБРАБОТЧИКА THIS

В глобальном контексте выполнения (за пределами каких-либо функций), `this` ссылается на глобальный объект вне зависимости от использования в строгом или нестрогом режиме.

```
this.a = 37;
console.log("this.a = " + window.a);// this.a = 37
```

КОНТЕКСТ ОБРАБОТЧИКА THIS

В пределах функции значение `this` зависит от того, каким образом вызвана функция. В строгом режиме с ссылкой на основу *window* `this` укажет на сам объект *window*:



КОНТЕКСТ ОБРАБОТЧИКА THIS

Когда функция используется как обработчик событий, `this` присваивается элементу, с которого начинается событие (некоторые браузеры не следуют этому соглашению для слушателей, добавленных динамически с помощью всех методов, кроме `addEventListener`).

В примере, рассмотренном ранее, выведем в консоль `this`:

```
let button = document.getElementById("clickMe");
  button.addEventListener('click', function() {
      console.log(this);
  });
```

КОНТЕКСТ ОБРАБОТЧИКА THIS

Результатом консоли будет сам элемент, на котором сработал обработчик:

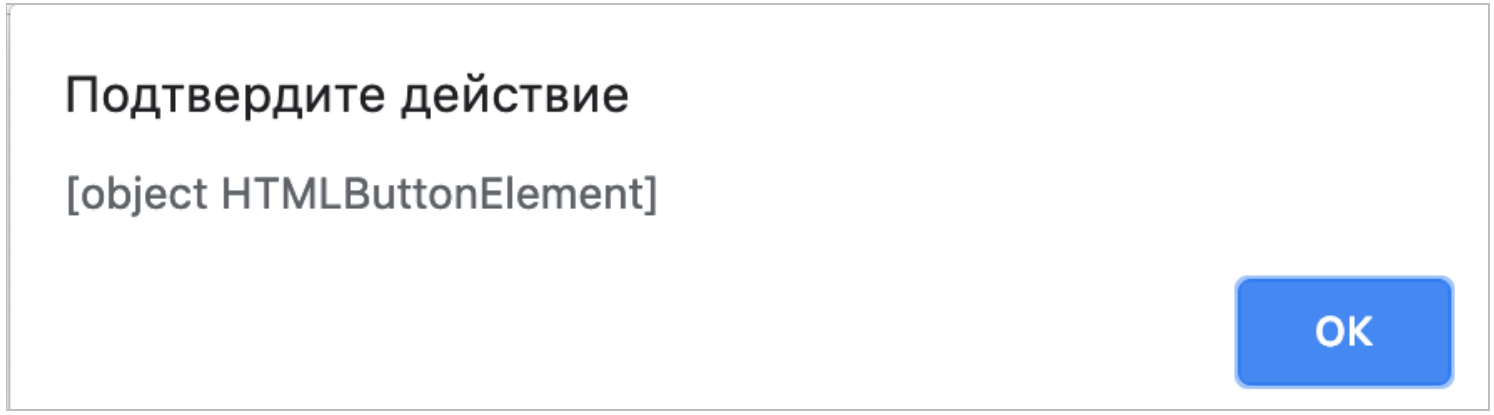
```
<button id="clickMe">Click me!</button>
```

Когда код вызван из инлайнового обработчика, `this` указывает на DOM элемент, в котором расположен код события:

```
<button id="clickMe" onclick="alert(this);">
  Click me!
</button>
```

КОНТЕКСТ ОБРАБОТЧИКА THIS

И как результат получаем alert окно с именем тэга:



Следует отметить, что this будет указывать на DOM элемент только во внешних (не вложенных) функциях.

ВЫВОДЫ

Итак, если вы попытаетесь обратиться к ключевому слову `this` в глобальной области видимости, оно будет привязано к глобальному контексту, то есть — к объекту window в браузере.

Когда `this` используется внутри объекта, это ключевое слово ссылается на сам объект.

ОБЪЕКТ СОБЫТИЯ (EVENT)

ОБЪЕКТ СОБЫТИЯ (EVENT)

Чтобы хорошо обработать событие, недостаточно знать о том, что это – «клик» или «нажатие клавиши». Могут понадобиться детали: координаты курсора, введённый символ и другие, в зависимости от события.

Детали произошедшего браузер записывает в «объект события», который передаётся первым аргументом в обработчик (event).

```
let input = document.getElementById("element");

input.onclick = function(event) {
    console.log(event.type);
}
```

ОБЪЕКТ СОБЫТИЯ (EVENT)

Чаще всего нужно узнать, на каком элементе сработало событие.

Например, мы поймали событие на внешнем `<div>` и хотим знать, на каком из внутренних элементов оно на самом деле произошло.

В браузерах, работающих по рекомендациям W3C, для этого используется `event.target`.

ОБЪЕКТ СОБЫТИЯ (EVENT)

Рассмотрим некоторые свойства объекта `event`:

1. *event.type* — тип события.
2. *event.currentTarget* — элемент, на котором сработал обработчик. Значение в точности такое же, как и у `this`, но бывают ситуации, когда обработчик является методом объекта и его `this` при помощи `bind` привязан к этому объекту, тогда мы можем использовать `event.currentTarget`.
3. *event.target* — ссылка на объект, который был инициатором события.
4. *event.clientX/Y* — координаты курсора в момент клика (относительно окна).

ОБЪЕКТ СОБЫТИЯ (EVENT)

Внимание: объект события доступен и в HTML.

При назначении обработчика в HTML также можно использовать переменную `event`, это будет работать кроссбраузерно:

Это возможно потому что, когда браузер из атрибута создаёт функцию-обработчик, то она выглядит так:

```
function(event) {
    alert(event.type)
}
```

То есть, её первый аргумент называется “event”.

ОБЪЕКТ СОБЫТИЯ (EVENT)

Универсальное кроссбраузерное решение для получения объекта события:

```
element.onclick = function(e) {
    e = e || window.event;
    // Теперь event - объект события во всех браузерах
};
```

СОБЫТИЯ КЛАВИАТУРЫ

СОБЫТИЯ КЛАВИАТУРЫ

При нажатии клавиши срабатывают события в таком порядке:

1. Первым срабатывает событие *keydown*.
2. Потом срабатывает событие *keypress*, не дожидаясь поднятия клавиши.
3. Событие *keyup* наступает только тогда, когда клавиша отпущена.

СОБЫТИЯ КЛАВИАТУРЫ

Обработчик события можно поставить на `document`:

```
1  function updatePlayer(event) {
2    console.log(event);
3  }
4  document.addEventListener('keydown', updatePlayer);
```

СОБЫТИЯ КЛАВИАТУРЫ

Как понять, какая клавиша была нажата?

Для этого нам опять нужно обратиться к объекту события. На клавиатурные события создается объект `KeyboardEvent`:

```
function updatePlayer(event) {
  console.log(event instanceof KeyboardEvent);
}
```

СОБЫТИЯ КЛАВИАТУРЫ

У события `KeyboardEvent` есть два свойства, которые помогут решить нашу задачу: `key` и `code`.

```
function updatePlayer(event) {
  console.log(event.key, event.code);
}
```

СОБЫТИЯ КЛАВИАТУРЫ

Свойство `key` содержит символ, который набран на клавиатуре, в зависимости от языка и регистра. Т.е. это может быть s, S, Ыl, ы и другие символы, если мы будем нажимать на клавишу S на клавиатуре.

```
<input onkeydown="alert(event.keyCode)">
// keyCode нажатой клавиши
```

СОБЫТИЯ КЛАВИАТУРЫ

Объекты `KeyboardEvent` описывают работу пользователя с клавиатурой.

`KeyboardEvent` сообщит только о том, что на клавише произошло событие.

Этот интерфейс также наследует методы от своих родителей: `UIEvent` и `Event`.

СОБЫТИЯ КЛАВИАТУРЫ

Свойства KeyboardEvent:

- *KeyboardEvent.altKey* — true, если клавиша Alt была активна;
- *KeyboardEvent.char* — возвращает DOMString, представляющий символьное значение клавиши;
- *KeyboardEvent.charCode* — возвращает Number, представляющий Unicode-номер клавиши;
- *KeyboardEvent.keyCode* — Возвращает Number, представляющий системный код, значение нажатой клавиши.

ОТЛИЧИЯ СОБЫТИЙ КЛАВИАТУРЫ

Событие `keypress` срабатывает только в том случае, если нажатая клавиша печатает какой-то символ. Поэтому в обработчике этого события не поймать нажатия клавиш стрелок и различных функциональных клавиш `Esc`, `Alt` и других. Вот простой пример, нажатие стрелок и функциональных клавиш не генерируют событие `keypress`, но генерируют `keydown`, для остальных символов срабатывают оба события

```
1  function showKey( event ) {
2      console.log( event.type, event.key );
3  }
4  document.addEventListener( 'keypress', showKey );
5  document.addEventListener( 'keydown', showKey );
```

ЗАЖАТАЯ КЛАВИША

Другое важное отличие в повторении. Если мы зажмем клавишу и будем удерживать её нажатой, то события `keydown` и `keypress` будут повторяться с определенным интервалом. А событие `keyup` нет

```
function showKey( event ) {
    console.log( event.type, event.code);
}
document.addEventListener( 'keydown', showKey );
document.addEventListener( 'keyup', showKey );
```

СВОЙСТВО REPEAT

Если мы зажмем и подержим клавишу нажатой, то событие `keydown` сработает несколько раз, хотя мы нажали её только один раз, а событие `keyup` только однажды. Чтобы отличить в `keydown` и `keypress` повторные нажатия, можно проверить свойство `repeat`. Оно будет содержать `true` для событий, сгенерированных автоматически повторно, при удержании клавиши:

```
1  function showKey( event ) {
2      if  (event.repeat) {
3        console.log (` Повторное нажатие ${event.code}`);
4      } else { console.log (` Нажата ${event.code}`);
5      }
6  }
7  document.addEventListener( 'keydown', showKey );
```

ДРУГИЕ ОБРАБОТЧИКИ СОБЫТИЙ

ДРУГИЕ ОБРАБОТЧИКИ СОБЫТИЙ

Когда браузер полностью сформировал DOM-дерево, генерируется событие `DOMContentLoaded`. Потом, когда браузер загрузит все ресурсы (стили, скрипты, изображения), то он также сгенерирует событие `load`:

DOMContentLoaded — означает, что все DOM-элементы разметки уже созданы, можно их искать, вешать обработчики, создавать интерфейс, но при этом, возможно, ещё не догрузились какие-то картинки или стили.

load — страница и все ресурсы загружены, используется редко, обычно нет нужды ждать этого момента.

ДРУГИЕ ОБРАБОТЧИКИ СОБЫТИЙ

Классический пример — установка обработчика на событие “содержимое окна загрузилось”:

```
function init() {
    alert('Документ загружен');

}
window.onload = init();
```

СОБЫТИЕ ПРОКРУТКИ СТРАНИЦ

Событие, которое происходит при прокрутке страницы называется `onscroll`.

В отличие от события `onwheel` (колесико мыши), его могут генерировать только прокручиваемые элементы или окно *window*. Но зато оно генерируется всегда, при любой прокрутке, не обязательно «мышиной».

- Показ дополнительных элементов навигации при прокрутке.
- Подгрузка и инициализация элементов интерфейса, ставших видимыми после прокрутки, например анимация

СОБЫТИЕ ПРОКРУТКИ СТРАНИЦ

Ранее мы разобрали способы повесить обработчики на событие, вы помните их? Давайте вспомним вместе:

1. `<element onscroll="myScript">` – инлайновый способ через атрибут тэга в HTML
2. `object.onscroll = function(){myScript};` – через свойство объекта
3. `object.addEventListener("scroll", callback);` – через метод `addEventListener`

```
window.onscroll = function() {
  var scrolled = document.documentElement.scrollTop;
  console.log(scrolled + 'px');
}
```

ДЕЙСТВИЕ БРАУЗЕРА ПО УМОЛЧАНИЮ

Браузер имеет своё собственное поведение по умолчанию для различных событий.

Например, клик по ссылке или показать контекстное меню и т.п.

Но как быть если нам нужно повесить событие на такой элемент?

```
// <a id="link" href="/">Ссылка</a>
let link = document.getElementById('link');
link.onclick = function(event) {
    event.preventDefault();
    alert('Link is clicked');
```

ВОЗВРАЩЕНИЕ RETURN FALSE ИЗ ОБРАБОТЧИКА

Возвращение *return false* из обработчика события предотвращает действие браузера по умолчанию. В этом смысле следующие два кода эквивалентны:

```
function handler(event) {
  ...
  return false
}
```

ВАЖНО!!! Браузер даже не гарантирует порядок, в котором сработают обработчики на одном элементе. Назначить можно в одном порядке, а сработают в другом.

Поэтому обработчик никак не может влиять на другие того же типа на том же элементе.

ЧЕМУ МЫ НАУЧИЛИСЬ?

ЧЕМУ МЫ НАУЧИЛИСЬ?

- **Разобрались, что такое событие и обработчик событий.**

Событие — это сигнал от браузера о том, что что-то произошло

Обработчик — это функция, которая сработает, как только событие произошло.

- **Изучили способы, как устанавливать обработчик на разные события.**
  - через атрибут тэга в html;
  - через свойство объекта;
  - через метод `addEventListener`.
- **Доступ к элементу через this**

Внутри обработчика события `this` ссылается на текущий элемент, то есть на тот, на котором он сработал.

ЧЕМУ МЫ НАУЧИЛИСЬ?

– Методы `addEventListener` и `removeEventListener` являются современным способом назначить или удалить обработчик.

– `Event` представляет собой любое событие, которое происходит в DOM

– Что такое и как предотвратить действие браузера по умолчанию (`event.preventDefault();`).

– Как с этим соотносится *return false*.

ДОМАШНЕЕ ЗАДАНИЕ

Ваше домашнее задание.

— Для получения зачёта необходимо выполнить все задачи.

— Присылать на проверку можно только все задачи вместе.

— Работа должна соответствовать принятому стилю оформления кода.— Любые вопросы по решению задач задавайте в Slack.

НЕТОЛОГИЯ

АЛЕКСЕЙ СУДНИЧНИКОВ

@avsudnichnikov