

Конспект по теме 1.3 «Объект события»

Содержание конспекта:

1. События. Определение, виды, примеры
2. Обработчики события
3. Контекст обработчика `this`
4. Событие `onclick`
5. Объект события
6. События клавиатуры

1. События. Определение, виды, примеры

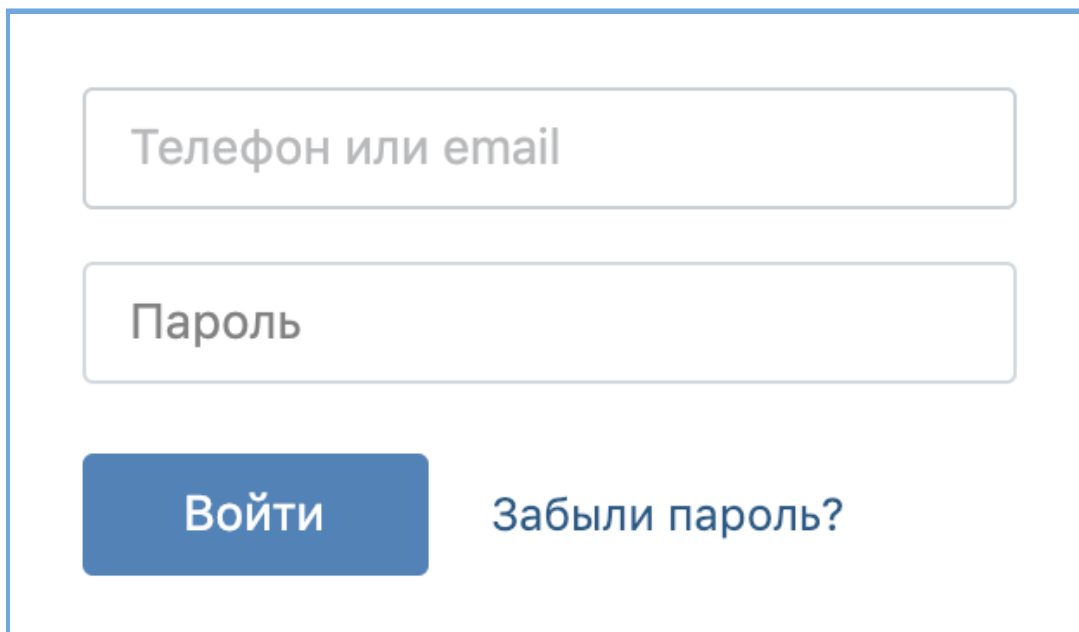
JavaScript — это язык управления сценариями на веб-странице, позволяющий реагировать на действия пользователя. Изначально JS создавался как короткий язык сценариев веб-страницы. Он должен был добавить активности безжизненным страницам и дать возможность разработчикам описывать обратную связь действиям пользователя.

Каждый раз, взаимодействуя со страницей в браузере, мы можем совершать какие-то действия: 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 формирования реакции на действия можно коротко назвать событием.

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

В ответ на совершённые действия мы хотим заставить веб-страницу не просто отображать статическую информацию, заранее запрограммированную в HTML-разметке, но и, используя введенные данные, открыть частную страницу или обработать наши данные для выдачи новой порции информации. Другими словами, совершая какое-либо действие на веб-странице, мы ждем обработку.

Пример: **форма авторизации** — заполняем логин и пароль и ожидаем перехода на личную страницу:

Скриншот формы авторизации, состоящей из двух текстовых полей для ввода, кнопки входа и ссылки для восстановления пароля. Поля имеют серый текст-подсказку. Кнопка и ссылка имеют синий цвет. Вся форма окружена синей рамкой.

Телефон или email

Пароль

Войти

[Забыли пароль?](#)

Всем знакомая процедура авторизации также задействует события. Во время авторизации происходят события, связанные с получением и потерей фокуса, вводом с клавиатуры, отправкой формы и загрузка ответа от сервера. Только обработав все вышеперечисленные события, можно предоставить пользователю качественную обратную связь при взаимодействии с сайтом.

Самые распространённые события в JavaScript:

1. События мыши:

- **click** — клик на левую кнопку мыши,
- **contextmenu** — клик на правую кнопку мыши,
- **mouseover** — когда мышь наводится на элемент,
- **ondblclick** — двойной клик мыши.

2. События клавиатуры:

- **keydown** — когда нажимаем на кнопку,
- **keyup** — когда отпускаем кнопку.

3. События форм и элементов:

- **focus** — когда элемент в фокусе,
- **blur** — когда элемент теряет фокус,
- **submit** — когда отправляем форму.

Приведем ещё 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 задаётся в обработчике событий.

2. Обработчики события

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

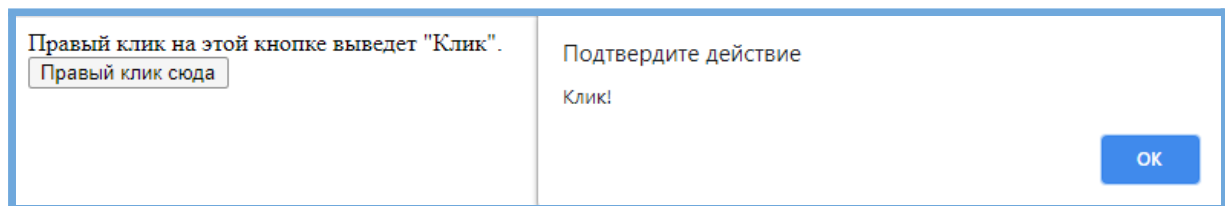
Обработчик события — это просто функция, в которой описана логика реакции на событие. Способов назначить определённому событию обработчик несколько. Рассмотрим, начиная с самого простого.

Способ 1. Использование атрибута HTML

Обработчик события можно указать в виде inline-записи прямо в атрибуте `on`-события. Пример: 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`.

Использовать такой способ задачи обработчика можно в решении простых задач, поскольку такой способ установки обработчиков нагляден и прост и поэтому очень удобен.

Но есть у него и минусы. Как только обработчик начинает занимать больше одной строки, читабельность резко падает.

Этот способ сейчас используют редко, потому что принято разделять отображение (html) и логику (js) в программном коде. Сколько-нибудь сложные обработчики в HTML никто не пишет. Вместо этого лучше устанавливать обработчики из JavaScript способами, которые будут представлены ниже.

Способ 2. Через свойство объекта

Для описания события через свойство объекта необходимо:

1. Найти HTML-элемент на странице.
2. Назначить обработчик **он** и имя обработчика: записать функцию в свойство, начинающиеся на **он** плюс **название события**, например, `onclick`.

Описание элемента имеет вид:

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

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

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

Теперь обратимся к его свойству `onclick`:

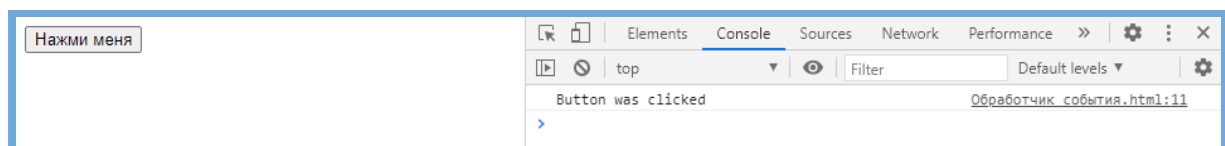
```
document.getElementById('clickMe').onclick;
```

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

Описание обработчика `onclick`:

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

Присваивать можно любую функцию — как обычную (с именем), так и анонимную или стрелочную:



Обратите внимание:

1. Это свойство, а не атрибут, желательно использовать прямое присвоение значения.
2. Свойства чувствительны к регистру.
3. Необходимо проверять наличие элемента на странице.
4. В этом варианте можно задать только один обработчик события.

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

В качестве присвоенного значения свойству может выступить не только анонимная функция:

```
function toClick() {  
    console.log(' Information ');  
}  
document.getElementById('clickMe').onclick = toClick;
```

Обратите внимание, что присваивать функцию мы должны без её вызова.

Так как у функции `toClick` нет вызова `return`, то обработчик не работает, 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"/>
```

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.

Чтобы назначить обработчик, нужно вызвать `addEventListener` с тремя аргументами:

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

1. Аргумент **event** — это имя события: `click`.
2. Аргумент **handler** — ссылка на функцию, которую надо назначить обработчиком.
3. Аргумент **phase** — опциональный аргумент, используется редко, что позволяет нам использовать функцию `addEventListener` только с двумя аргументами `event` и `handler`.

Чтобы удалить обработчик, вызываем `removeEventListener`:

```
elem.addEventListener("click", function() {alert('Спасибо!')});
elem.removeEventListener("click", function() {alert('Спасибо!')});
```

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

```
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!
```

Чтобы добавить несколько обработчиков события, достаточно вызвать **addEventListener** на элементе несколько раз с разными функциями в качестве параметра.

Чтобы удалить, используем метод **removeEventListener**, в который нужно передать именно ту функцию-обработчик, которая была назначена через **addEventListener**.

Обратите внимание, что имя события указывается **без** префикса “on”.

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.

Удаление обработчика осуществляется вызовом метода `removeEventListener`:

```
function handler() {  
    alert( 'Спасибо!' );  
}  
input.addEventListener("click", handler);  
// ....  
input.removeEventListener("click", handler);
```

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

Обратите внимание, что имя события указывается без префикса “on”. Решение World Wide Web Consortium (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.

Итоги

1. Обработчик — это функция.
2. Существует 3 способа задать обработчик:
 - HTML-атрибуты,
 - свойства объектов,
 - специальные методы.

3. Контекст обработчика this

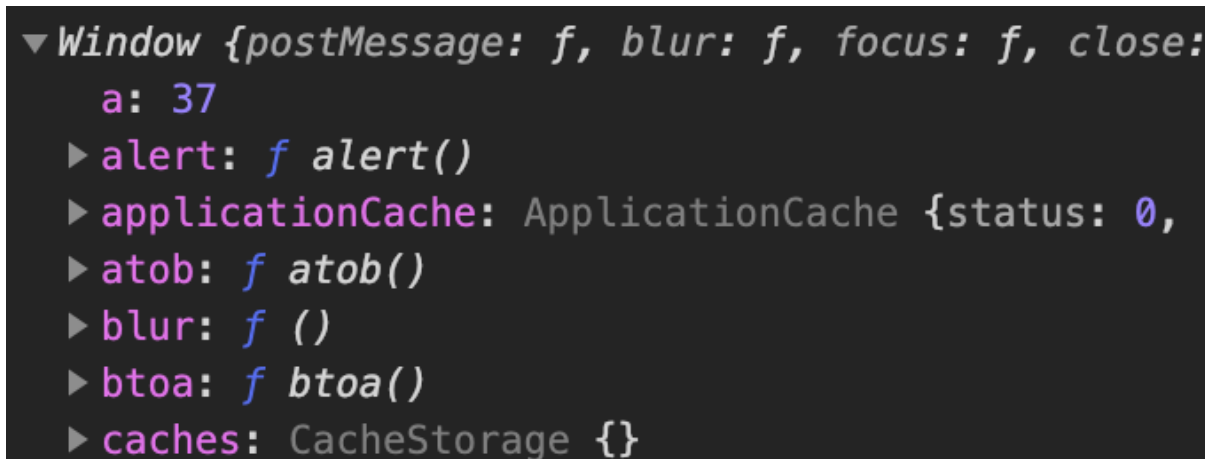
Мы используем `this` так же, как местоимение в языках, подобных английскому или русскому. Мы не всегда называем имя или предмет, а говорим применительно к нему «этот». То же самое происходит при использовании ключевого слова `this` в языках программирования.

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

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
```

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**:



```
▼ Window {postMessage: f, blur: f, focus: f, close:  
  a: 37  
  ► alert: f alert()  
  ► applicationCache: ApplicationCache {status: 0,  
  ► atob: f atob()  
  ► blur: f ()  
  ► btoa: f btoa()  
  ► caches: CacheStorage {}}
```

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`.

При использовании функции в качестве обработчика событий `this` ссылается на сам элемент:

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

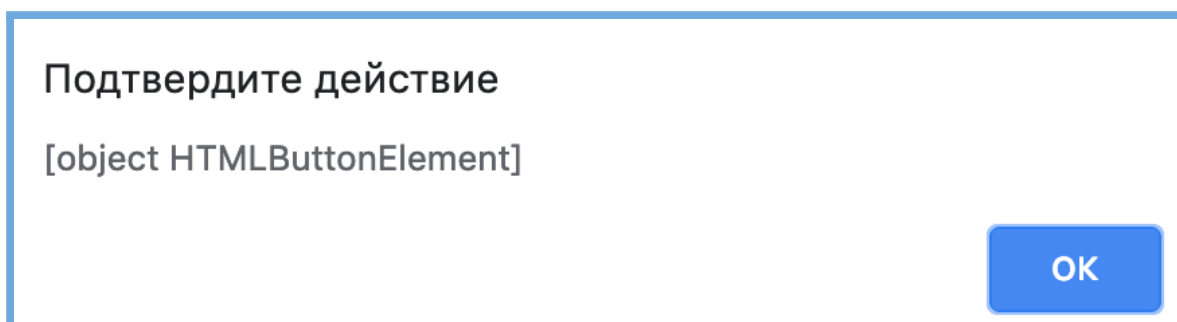
Результатом консоли будет сам элемент, на котором сработал обработчик:


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

При вызове inline-обработчика **this** указывает на **DOM**-элемент, в котором расположен код события:

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

И как результат получаем **alert** окно с именем тега:



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

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.

4. Событие onclick

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

Мы имеем такую разметку документа:

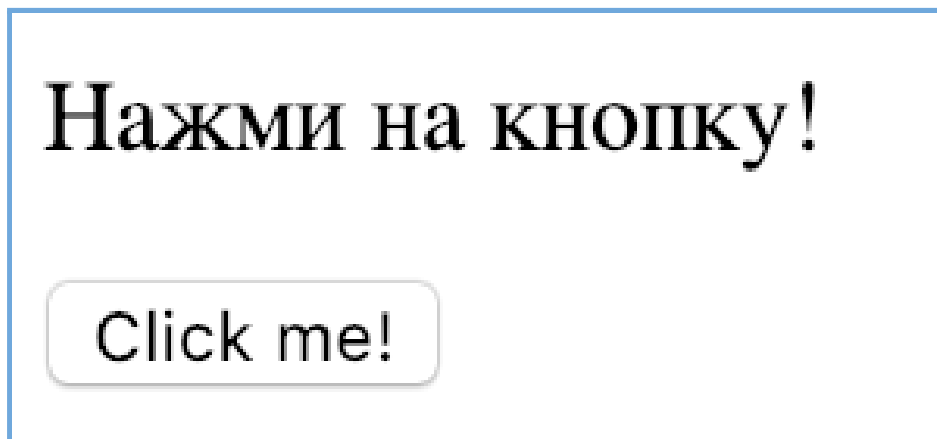
```
<p id="text">Нажми на кнопку!</p>
<button id="clickMe">Click me!</button>
<script src="script.js"></script>
```

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";
});
```

Теперь при каждом клике на кнопку наш текст будет меняться:



5. Объект события

При обработке событий не всегда достаточно одного только факта, что событие произошло. Иногда нужно знать, куда на экране нажали мышкой или какую букву напечатали на клавиатуре.

Для передачи этой дополнительной информации и нужен объект события. Он передаётся в качестве первого аргумента в функцию-обработчик.

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).

В соответствии с типами событий объекты имеют свою классификацию. Базовый тип всех событий Event содержит в себе набор общих свойств.

```
let input = document.getElementById("element");
input.onclick = function(event) {
  console.log(event.type);
}
```

Чаще всего нужно узнать, на каком элементе сработало событие.

Пример: 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 (World Wide Web Consortium), для этого используется `event.target`.

Объект события event может обладать следующими свойствами:

1. `event.type` — тип события.
2. `event.target` — ссылка на объект, который был инициатором события.
3. `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`.
4. `event.clientX/Y` — координаты курсора в момент клика (относительно окна).

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”.

Универсальное кроссбраузерное решение, чтобы получить объект события:

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

6. События клавиатуры

Для обработки нажатий на клавиатуру используется два типа событий:

- **keydown** — нажатие клавиши,
- **keyup** — поднятие клавиши.

Объекты **KeyboardEvent** описывают работу пользователя с клавиатурой.

Свойства KeyboardEvent:

- **code** — содержит информацию о том, какая именно клавиша нажата на клавиатуре;
- **key** — содержит напечатанный символ с учётом регистра и выбранной раскладки;
- **altKey** — сохранит значение **true**, если клавиша **Alt** была активна;
- **shiftKey** — сохранит значение **true**, если клавиша **Shift** была активна;
- **ctrlKey** — сохранит значение **true**, если клавиша **Control** была активна.

При нажатии клавиши события срабатывают 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.

Обработчик события можно поставить на **document**:

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

Как понять, какая клавиша была нажата?

Для этого нам опять нужно обратиться к объекту события. На клавиатурные события создаётся объект **KeyboardEvent**:

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

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

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

Свойство **key** содержит символ, который набран на клавиатуре, в зависимости от языка и регистра. Это может быть **s**, **S**, **Ы**, **ы** и другие символы, если мы будем нажимать на клавишу **S** на клавиатуре:

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

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**.

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.

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.

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** (колёсико мыши), его могут генерировать только прокручиваемые элементы или окно window. Но зато оно генерируется всегда, при любой прокрутке, не обязательно мышью:

- показ дополнительных элементов навигации при прокрутке;

- загрузка и инициализация элементов интерфейса, ставших видимыми после прокрутки, например анимация.

Способы повесить обработчики на событие:

1. `<element onscroll="myScript">` — инлайновый способ через атрибут тега 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: клик по ссылке, 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.

Итоги по теме

- Событие — это сигнал от браузера о том, что что-то произошло.
- Обработчик — это функция, которая сработает, как только событие произошло.
- Способы, как устанавливать обработчик на разные события:
 - через атрибут тега в HTML;
 - через свойство объекта;
 - через метод `addEventListener`.
- Доступ к элементу через `this`: внутри обработчика события `this` ссылается на текущий элемент — на тот, на котором он сработал.
- Методы `addEventListener` и `removeEventListener` являются современным способом назначить или удалить обработчик.
- `Event` представляет собой любое событие, которое происходит в DOM.
- Как предотвратить действие браузера по умолчанию (`event.preventDefault();`).
- Как с этим соотносится `return false`.