

Раздел

[Загрузка документа и ресурсов](#)

Навигация по уроку

DOMContentLoaded

window.onload

window.onunload

window.onbeforeunload

readyState

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Браузер: документ, события, интерфейсы](#)  
[→ Загрузка документа и ресурсов](#) 19-го октября 2020

## Страница: DOMContentLoaded, load, beforeunload, unload

У жизненного цикла HTML-страницы есть три важных события:

- **DOMContentLoaded** – браузер полностью загрузил HTML, было построено DOM-дерево, но внешние ресурсы, такие как картинки `<img>` и стили, могут быть ещё не загружены.
- **load** – браузер загрузил HTML и внешние ресурсы (картинки, стили и т.д.).
- **beforeunload/unload** – пользователь покидает страницу.

Каждое из этих событий может быть полезно:

- Событие **DOMContentLoaded** – DOM готов, так что обработчик может искать DOM-узлы и инициализировать интерфейс.
- Событие **load** – внешние ресурсы были загружены, стили применены, размеры картинок известны и т.д.
- Событие **beforeunload** – пользователь покидает страницу. Мы можем проверить, сохранил ли он изменения и спросить, на самом ли деле он хочет уйти.
- **unload** – пользователь почти ушёл, но мы всё ещё можем запустить некоторые операции, например, отправить статистику.

Давайте рассмотрим эти события подробнее.



### DOMContentLoaded



Событие **DOMContentLoaded** срабатывает на объекте `document`.

Мы должны использовать `addEventListener`, чтобы поймать его:

```
1 document.addEventListener("DOMContentLoaded", ready);
2 // не "document.onDOMContentLoaded = ..."
```

Например:

```
1 <script>
2   function ready() {
3       alert('DOM готов');
4
5       // изображение ещё не загружено (если не было закеш
6       alert(`Размер изображения: ${img.offsetWidth}x${img
7   }
8
9   document.addEventListener("DOMContentLoaded", ready);
10 </script>
11
12  {
3     alert("DOM готов!");
4   });
5 </script>
6
7 <script src="https://cdnjs.cloudflare.com/ajax/libs/lod
8
9 <script>
10  alert("Библиотека загружена, встроенный скрипт выполн
11 </script>
```

В примере выше мы сначала увидим «Библиотека загружена...», а затем «DOM готов!» (все скрипты выполнены).



### Скрипты, которые не блокируют DOMContentLoaded

Есть два исключения из этого правила:

1. Скрипты с атрибутом `async`, который мы рассмотрим [немного позже](#), не блокируют `DOMContentLoaded`.
2. Скрипты, сгенерированные динамически при помощи `document.createElement('script')` и затем добавленные на страницу, также не блокируют это событие.

## DOMContentLoaded и стили

Внешние таблицы стилей не затрагивают DOM, поэтому `DOMContentLoaded` их не ждёт.

Но здесь есть подводный камень. Если после стилей у нас есть скрипт, то этот скрипт должен дождаться, пока загрузятся стили:

```
1 <link type="text/css" rel="stylesheet" href="style.css">
2 <script>
3   // скрипт не выполняется, пока не загрузятся стили
4   alert(getComputedStyle(document.body).marginTop);
5 </script>
```

Причина в том, что скрипту может понадобиться получить координаты или другие свойства элементов, зависящих от стилей, как в примере выше. Естественно, он должен дождаться, пока стили загрузятся.

Так как `DOMContentLoaded` дожидается скриптов, то теперь он так же дожидается и стилей перед ними.

## Встроенное в браузер автозаполнение

Firefox, Chrome и Opera автоматически заполняют поля при наступлении `DOMContentLoaded`.

Например, если на странице есть форма логина и пароля и браузер запомнил значения, то при наступлении `DOMContentLoaded` он попытается заполнить их (если получил разрешение от пользователя).

Так что, если `DOMContentLoaded` откладывается из-за долгой загрузки скриптов, в свою очередь – откладывается автозаполнение. Вы наверняка замечали, что на некоторых сайтах (если вы используете автозаполнение в

Раздел

[Загрузка документа и ресурсов](#)

Навигация по уроку

DOMContentLoaded

window.onload

window.onunload

window.onbeforeunload

readyState

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



браузере) поля логина и пароля не заполняются мгновенно, есть некоторая задержка до полной загрузки страницы. Это и есть ожидание события `DOMContentLoaded`.

## window.onload

Событие `load` на объекте `window` наступает, когда загрузилась вся страница, включая стили, картинки и другие ресурсы.

В примере ниже правильно показаны размеры картинки, потому что `window.onload` дожидается всех изображений:

```
1 <script>
2   window.onload = function() {
3     alert('Страница загружена');
4
5     // к этому моменту страница загружена
6     alert(`Image size: ${img.offsetWidth}x${img.offsetHeight}`);
7   };
8 </script>
9
10 
```

## window.onunload

Когда посетитель покидает страницу, на объекте `window` генерируется событие `unload`. В этот момент стоит совершать простые действия, не требующие много времени, вроде закрытия связанных всплывающих окон.

Обычно здесь отсылают статистику.

Предположим, мы собрали данные о том, как используется страница: клики, прокрутка, просмотры областей страницы и так далее.

Естественно, событие `unload` – это тот момент, когда пользователь нас покидает и мы хотим сохранить эти данные.

Для этого существует специальный метод `navigator.sendBeacon(url, data)`, описанный в спецификации <https://w3c.github.io/beacon/>.

Он посылает данные в фоне. Переход к другой странице не задерживается: браузер покидает страницу, но всё равно выполняет `sendBeacon`.

Его можно использовать вот так:

```
1 let analyticsData = { /* объект с собранными данными */
2
3   window.addEventListener("unload", function() {
4     navigator.sendBeacon("/analytics", JSON.stringify(analyticsData));
5   });
```

- Отсылается POST-запрос.
- Мы можем послать не только строку, но так же формы и другие форматы, как описано в главе [Fetch](#), но обычно это строковый объект.
- Размер данных ограничен 64 Кб.

К тому моменту, как `sendBeacon` завершится, браузер наверняка уже покинет страницу, так что возможности обработать ответ сервера не будет (для статистики он обычно пустой).

Для таких запросов с закрывающейся страницей есть специальный флаг `keepalive` в методе `fetch` для общих сетевых запросов. Вы можете найти больше информации в главе [Fetch API](#).

Если мы хотим отменить переход на другую страницу, то здесь мы этого сделать не сможем. Но сможем в другом месте – в событии `onbeforeunload`.

## window.onbeforeunload

Раздел

[Загрузка документа и ресурсов](#)

Навигация по уроку

DOMContentLoaded

window.onload

window.onunload

window.onbeforeunload

readyState

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



Если посетитель собирается уйти со страницы или закрыть окно, обработчик `beforeunload` попросит дополнительное подтверждение.

Если мы отменим это событие, то браузер спросит посетителя, уверен ли он.

Вы можете попробовать это, запустив следующий код и затем перезагрузив страницу:

```
1 window.onbeforeunload = function() {
2   return false;
3 };
```



По историческим причинам возврат непустой строки так же считается отменой события. Когда-то браузеры использовали её в качестве сообщения, но, как указывает [современная спецификация](#), они не должны этого делать.

Вот пример:

```
1 window.onbeforeunload = function() {
2   return "Есть несохранённые изменения. Всё равно уходи";
3 };
```



Поведение было изменено, потому что некоторые веб-разработчики злоупотребляли этим обработчиком события, показывая вводящие в заблуждение и надоедливые сообщения. Так что, прямо сейчас старые браузеры всё ещё могут показывать строку как сообщение, но в остальных – нет возможности настроить показ сообщения пользователям.

## readyState

Что произойдёт, если мы установим обработчик `DOMContentLoaded` после того, как документ загрузился?

Естественно, он никогда не запустится.

Есть случаи, когда мы не уверены, готов документ или нет. Мы бы хотели, чтобы наша функция исполнилась, когда DOM загрузился, будь то сейчас или позже.

Свойство `document.readyState` показывает нам текущее состояние загрузки.

Есть три возможных значения:

- `"loading"` – документ загружается.
- `"interactive"` – документ был полностью прочитан.
- `"complete"` – документ был полностью прочитан и все ресурсы (такие как изображения) были тоже загружены.

Так что мы можем проверить `document.readyState` и, либо установить обработчик, либо, если документ готов, выполнить код сразу же.

Например, вот так:

```
1 function work() { /*...*/ }
2
3 if (document.readyState === 'loading') {
4   // ещё загружается, ждём события
5   document.addEventListener('DOMContentLoaded', work);
6 } else {
7   // DOM готов!
8   work();
9 }
```

Также есть событие `readystatechange`, которое генерируется при изменении состояния, так что мы можем вывести все эти состояния таким образом:

Раздел

[Загрузка документа и ресурсов](#)

Навигация по уроку

DOMContentLoaded

window.onload

window.onunload

window.onbeforeunload

readyState

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)

```
1 // текущее состояние
2 console.log(document.readyState);
3
4 // вывести изменения состояния
5 document.addEventListener('readystatechange', () => con
```



Событие `readystatechange` – альтернативный вариант отслеживания состояния загрузки документа, который появился очень давно. На сегодняшний день он используется редко.

Для полноты картины давайте посмотрим на весь поток событий:

Здесь документ с `<iframe>`, `<img>` и обработчиками, которые логируют события:

```
1 <script>
2   log('начальный readyState:' + document.readyState);
3
4   document.addEventListener('readystatechange', () => l
5   document.addEventListener('DOMContentLoaded', () => l
6
7   window.onload = () => log('window onload');
8 </script>
9
10 <iframe src="iframe.html" onload="log('iframe onload')">
11
12 
13 <script>
14   img.onload = () => log('img onload');
15 </script>
```

Рабочий пример есть [в песочнице](#).

Типичный вывод:

```
1. [1] начальный readyState:loading
2. [2] readyState:interactive
3. [2] DOMContentLoaded
4. [3] iframe onload
5. [4] img onload
6. [4] readyState:complete
7. [4] window onload
```

Цифры в квадратных скобках обозначают примерное время события. События, отмеченные одинаковой цифрой, произойдут примерно в одно и то же время ( $\pm$  несколько миллисекунд).

- `document.readyState` станет `interactive` прямо перед `DOMContentLoaded`. Эти две вещи, на самом деле, обозначают одно и то же.
- `document.readyState` станет `complete`, когда все ресурсы (`iframe` и `img`) загрузятся. Здесь мы видим, что это произойдёт примерно в одно время с `img.onload` (`img` последний ресурс) и `window.onload`. Переключение на состояние `complete` означает то же самое, что и `window.onload`. Разница заключается в том, что `window.onload` всегда срабатывает после всех `load` других обработчиков.

## Итого

События загрузки страницы:

- `DOMContentLoaded` генерируется на `document`, когда DOM готов. Мы можем применить JavaScript к элементам на данном этапе.
- Скрипты, вроде `<script>...</script>` или `<script src="..."></script>` блокируют `DOMContentLoaded`, браузер ждёт, пока они выполнятся.
- Изображения и другие ресурсы тоже всё ещё могут продолжать загружаться.

Раздел

[Загрузка документа и ресурсов](#)

Навигация по уроку

DOMContentLoaded

window.onload

window.onunload

window.onbeforeunload

readyState

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



- Событие `load` на `window` генерируется, когда страница и все ресурсы загружены. Мы редко его используем, потому что обычно нет нужды ждать так долго.
- Событие `beforeunload` на `window` генерируется, когда пользователь покидает страницу. Если мы отменим событие, браузер спросит, на самом ли деле пользователь хочет уйти (например, у нас есть несохранённые изменения).
- Событие `unload` на `window` генерируется, когда пользователь окончательно уходит, в обработчике мы можем делать только простые вещи, которые ни о чём не спрашивают пользователя и не заставляют его ждать. Из-за этих ограничений оно редко используется. Мы можем послать сетевой запрос с помощью `navigator.sendBeacon`.
- `document.readyState` – текущее состояние документа, изменения можно отследить с помощью события `readystatechange`:
  - `loading` – документ грузится.
  - `interactive` – документ прочитан, происходит примерно в то же время, что и `DOMContentLoaded`, но до него.
  - `complete` – документ и ресурсы загружены, происходит примерно в то же время, что и `window.onload`, но до него.

Проводим [курсы по JavaScript и фреймворкам](#). ✕

## Комментарии

перед тем как писать...