

События в JavaScript

Урок 8





Кадочников Алексей

Frontend-разработчик

- ☀ Веб-разработчик со стажем более 9 лет
- ☀ Преподаватель GeekBrains с 2015 года
- ☀ Автор курсов по Frontend на портале Geekbrains
- ☀ Работал в таких компаниях, как VK и Wizard-C



План курса

1

Знакомство с JavaScript

2

Основы JavaScript

3

Функции в JavaScript

4

Циклы и массивы

5

Работа с объектами в JavaScript

6

Введение в DOM

7

Работа с DOM

8

Основы событий в JavaScript

9

Работа с событиями в JavaScript

10

Основы шаблонизации, работа с JSON

11






Работа с медиафайлами

12

Основы API, итоги курса



Что будет на уроке сегодня

-  События в DOM
-  Архитектура DOM-событий
-  Сравнение разных целей события
-  Добавление слушателя события
-  Делегирование событий



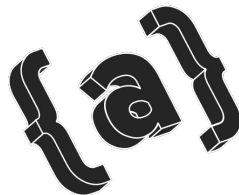
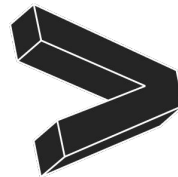
События в DOM





События в DOM

События — это сигналы от браузера или другой среды исполнения JavaScript.





Список возможных событий в DOM очень длинен

- click — нажатие кнопки мыши
- touch — касание
- load — загрузка
- drag — перетаскивание
- change — изменение
- input — ввод
- и так далее

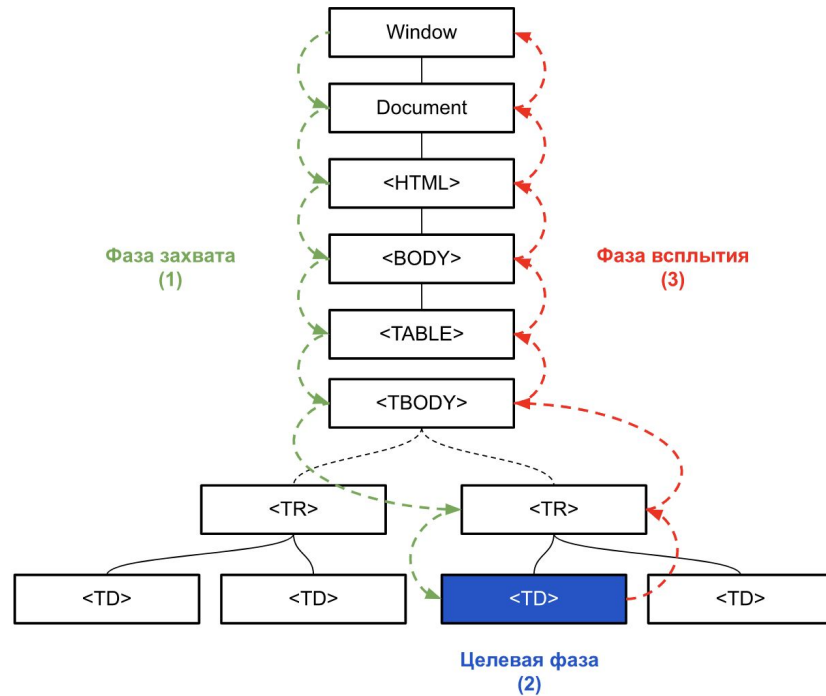


Архитектура DOM-событий





Иерархия интерфейсов DOM





Путь распространения

Это упорядоченный список текущих целей, через которые проходит событие. Путь распространения отражает иерархическую древовидную структуру документа.

Последний элемент списка — это цель события, а предпоследний — родитель цели (parent). Остальные предшествующие элементы списка называются предками цели (ancestors).



Путь распространения

Когда путь распространения определён, объект события проходит через одну или несколько фаз события.

Есть три фазы события: фаза захвата, целевая фаза и фаза всплытия.

Фаза будет пропущена, если она не поддерживается или распространение объекта события остановилось. Например, если для bubbles-атрибута установлено значение false, фаза всплытия пропускается. А если метод `stopPropagation()` был вызван до отправки, пропускаются все фазы.





Фаза захвата (capturing phase)

Этот объект события распространяется через его (события) предков:
от окна к родителю цели.





Целевая фаза (target phase)

Объект события прибывает в цель события. Эта фаза также известна как фаза попадания в цель. Если тип события указывает, что событие не всплывает, объект события остановится после завершения этой фазы.





Фаза всплытия (bubbling phase)

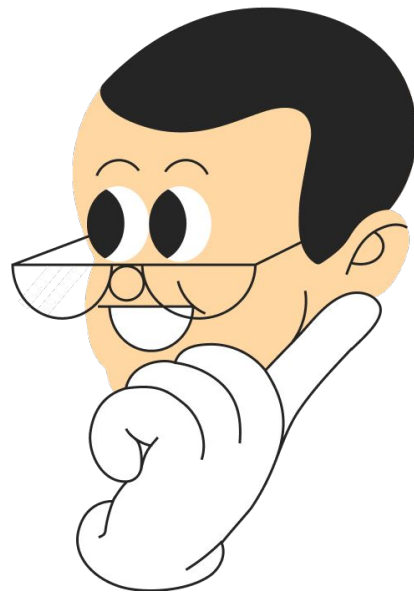
Объект события распространяется через предков цели в обратном порядке, начиная с родителя цели и заканчивая окном.





Важно знать

Не все события в DOM всплывают. Например, события `focus`, `blur`, `load`, `unload`, `change`, `reset`, `scroll`, `mouseenter`, `mouseleave` не всплывают. Чтобы узнать, всплывает ли событие, используется логическое поле `Event.bubbles`.



Есть несколько различных целей для рассмотрения:

| Свойство | Определено в интерфейсе | Описание |
|---------------------|-------------------------|--|
| event.target | DOM Event | Исходный DOM-элемент, на котором произошло событие |
| event.currentTarget | DOM Event | Текущий DOM-элемент, чьи подписчики в настоящее время обрабатываются. По мере того как происходит захват и всплытие событий, это значение изменяется |
| event.relatedTarget | DOM MouseEvent | Определяет вторичную цель события |

Для некоторых пар событий мыши есть дополнительный, связанный с исходным, элемент, на котором находился курсор мыши перед событием.

| Тип события | event.target | event.relatedTarget |
|----------------------------------|-------------------------------------|-------------------------------------|
| mouseover, mouseenter, dragenter | Элемент, в который входит курсор | Элемент, из которого выходит курсор |
| mouseout, mouseleave, dragleave | Элемент, из которого выходит курсор | Элемент, в который входит курсор |



Использование свойства DOM-объекта

Чтобы назначить обработчик, используют свойство DOM-элемента `on<событие>`.

Например, **`button.onclick`**:

```
1 <button>Клики меня!</button>
2
3 <script>
4   const button = document.querySelector('button')
5   button.onclick = (event) => {
6     console.log(event.target === button)
7   }
8 </script>
9
10
```



Использование метода `addEventListener`

Метод **`addEventListener`** — современный способ добавления слушателей событий. Он отличается от описанных выше способов тем, что:

- срабатывает на любом DOM-элементе, а не только на HTML-элементах;
- добавляет несколько обработчиков для одного события;
- предоставляет точный контроль фазы срабатывания (вызова) обработчика (захват или всплытие).

Для удаления обработчика используется метод `removeEventListener` с аналогичным синтаксисом:

```
1 <button>Клики меня!</button>
2
3 <script>
4   const button = document.querySelector('button')
5   const onceListener = () => {
6     console.log('Будет вызван только один раз с включённой опцией
  "once"')
7   }
8   const onceListenerManual = (event) => {
9     console.log('Будет вызван только один раз и удалён вручную через
  вызов removeEventListener')
10    event.target.removeEventListener('click', onceListenerManual)
11  }
12  button.addEventListener('click', onceListener, {once: true})
13  button.addEventListener('click', onceListenerManual)
14 </script>
```



Несколько одинаковых обработчиков события

Если на одном EventTarget зарегистрировано несколько одинаковых EventListener с одинаковыми параметрами, дублирующиеся обработчики игнорируются.





Делегирование событий





Делегирование событий

Часто возникает задача назначить один обработчик события на множество однотипных нод, например, DOM-элементов списка ``. В этом случае вместо получения ссылок на все `` в списке, перебора их в цикле и назначения каждому слушателю, разумно назначить только одного слушателя на родительский элемент ``. Эта техника называется делегированием событий и широко применяется во фронтенд-разработке.



Делегирование событий

При делегировании используется свойство `event.target` для доступа к целевому элементу события. Свойство `event.currentTarget` будет указывать на тот элемент, на который мы делегировали обработчик.

```
1 <ul>
2   <li>Первый</li>
3   <li>Второй</li>
4   <li>Третий</li>
5 </ul>
6
7 <script>
8   const list = document.querySelector('ul')
9   const listener = (e) => {
10     console.log(e.target, e.currentTarget)
11   }
12
13   list.addEventListener('click', listener)
14
15   ['fourth', 'fifth'].forEach((text) => {
16     const listItem = document.createElement('li')
17     listItem.append(text)
18     list.append(listItem)
19   })
20 </script>
21
```









Делегирование событий


Ещё один бонус использования делегирования в том, что динамически добавленные после назначения обработчика элементы списка также будут реагировать на клик, и никаких дополнительных вызовов `addEventListener` для них не потребуется.





Итоги урока

-  События в DOM
-  Архитектура DOM-событий
-  Сравнение разных целей события
-  Добавление слушателя события
-  Делегирование событий

Спасибо 
за внимание

