События в JS

Кирилл Талецкий

TeachMeSkills
31 августа 2023

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

Часто используемые события в браузере

- События на элементах управления
- События мыши
- Клавиатурные события
- События документа
- События CSS

Обработчик

- Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.
- Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.
- Есть три способа назначения обработчиков событий

Атрибут HTML

DOM свойство у элемента

Метод addEventListener

Атрибут HTML тега

```
<button onclick="console.log('clicked')">Click me</button>
```

- Название атрибута — on*ИмяСобытия*, например, onclick, onmousemove
- HTML-атрибуты используются редко потому, что JavaScript в HTML-теге выглядит не очень опрятно. К тому же, много кода там не напишешь.

Атрибут HTML тега

```
<script>
  function handleClick() {
    console.log('clicked')
    return;
  };
</script>

<button onclick="handleClick()">Click me</button>
```

- Есть опция вынести исполняемый код в отдельную функцию
- Так можно написать чуть более сложные обработки
- Но это решение всё равно плохо годится для сложной логики

Свойство DOM элемента

```
<script>
  const button = document.getElementById(‘button');

  button.onclick = () => {
    console.log('clicked')
    return;
  };
</script>

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

- Хорошая альтернатива — добавить обработчик через свойство DOM элемента
- Всё ещё не позволяет назначить более одного обработчика

Свойство DOM элемента

```
<script>
  const button = document.getElementById('button');

  const handleClick = () => {
    console.log('clicked');
    return;
  };

  button.onclick = handleClick; // используем готовую функцию
  button.onclick = null; // удаляем обработчик
</script>

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

Метод addEventListener

```
<script>
  const button = document.getElementById('button');

  button.addEventListener('click', () => {
    console.log('clicked')
  });
</script>
<button id="button">Click me</button>
```

- Более гибкий способ, так как позволяет
  - Добавлять более одного обработчика событий
  - Выборочно удалять уже установленные обработчики

Метод addEventListener

Имя события: клик, скролл, нажатия клавиатуры и т.д.

Не обязательный объект с опциями

```
target.addEventListener('name', callback, options)
```

Функция, которая будет вызвана при срабатывании события

Метод addEventListener

Имя события: клик, скролл, нажатия клавиатуры и т.д.

Не обязательный объект с опциями

```
target.addEventListener('name', callback, options)
```

Функция, которая будет вызвана при срабатывании события

Поле	Значение
once	Если `true`, то обработчик будет автоматически удалён после выполнения
capture	Фаза, на которой должен сработать обработчик. Если true, то погружение, false — всплытие (подробности далее)
passive	Если `true`, то браузер будет знать, что обработчик никогда не вызовет `preventDefault()` (подробности далее)

Метод removeEventListener

```
target.removeEventListener(‘name', callback, options)
```

- Позволяет удалить обработчик с элемента `target`

Метод removeEventListener

```
target.removeEventListener(‘name', callback, options)
```

- Позволяет удалить обработчик с элемента `target`

Требует на вход **ту же функцию (по ссылке!)**, что была передана в addEventListener()

Порядок обработки

- Если на одном узле назначено несколько обработчиков, то они сработают в порядке назначения

```
element.addEventListener('click', () => {console.log(1)});
element.addEventListener('click', () => {console.log(2)});
element.addEventListener('click', () => {console.log(3)});

// click
// 1
// 2
// 3
```

Объект события

- Содержит детали о произошедшем событии
  - Элемент, на котором событие сработало
  - Координаты мышки
  - Тип события
  - и т.д.

Объект события

- Пример использования

```
window.addEventListener('click', (event) => {
  console.log(`clicked on ${event.currentTarget}`);
  console.log(`coordinates: [${event.clientX}, ${event.clientY}]`);
});
```

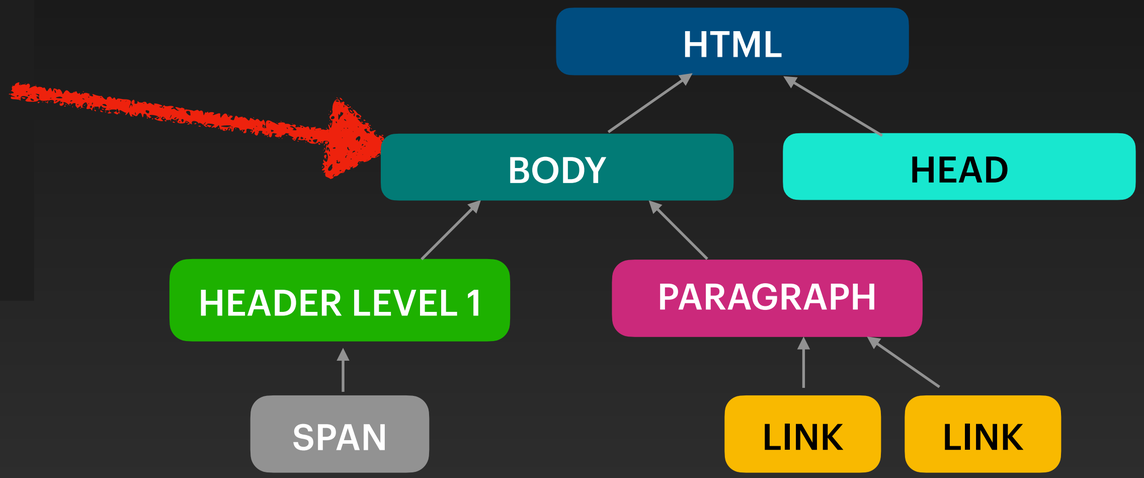
Всплытие и погружение

Всплытие: пример

```
document.body.addEventListener('click', (event) => {
  console.log(`clicked on ${event.currentTarget}`);
});
```

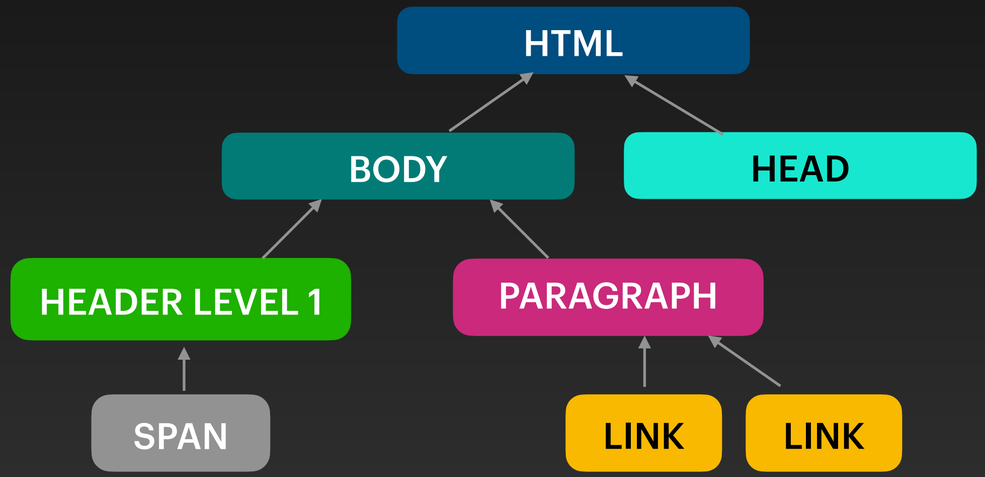
Добавляем обработчик

```
document.body.addEventListener('click', (event) => {
  console.log(`
    clicked on ${event.currentTarget}
  `);
});
```



Кликаем по части заголовка

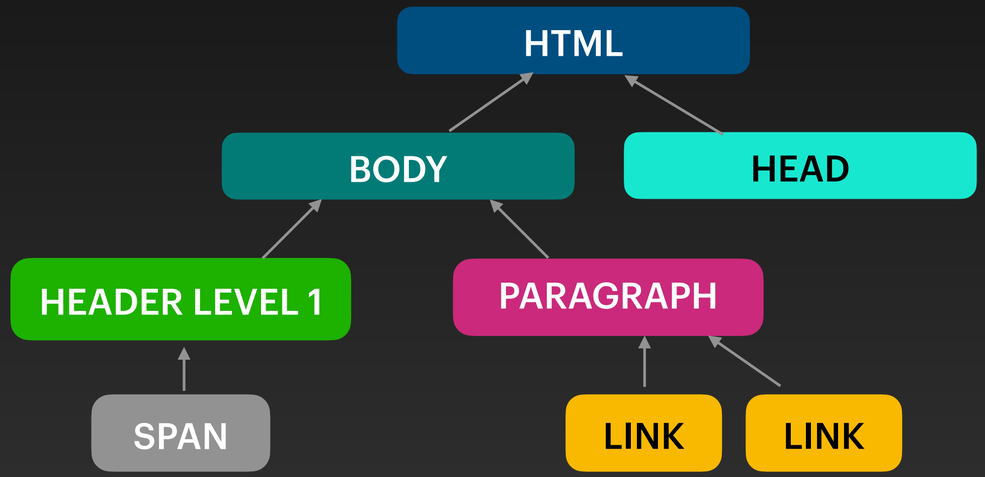
```
document.body.addEventListener('click', (event) => {
  console.log(`
    clicked on ${event.currentTarget}
  `);
});
```



Click!

Событие на целевом элементе

```
document.body.addEventListener('click', (event) => {
  console.log(`
    clicked on ${event.currentTarget}
  `);
});
```

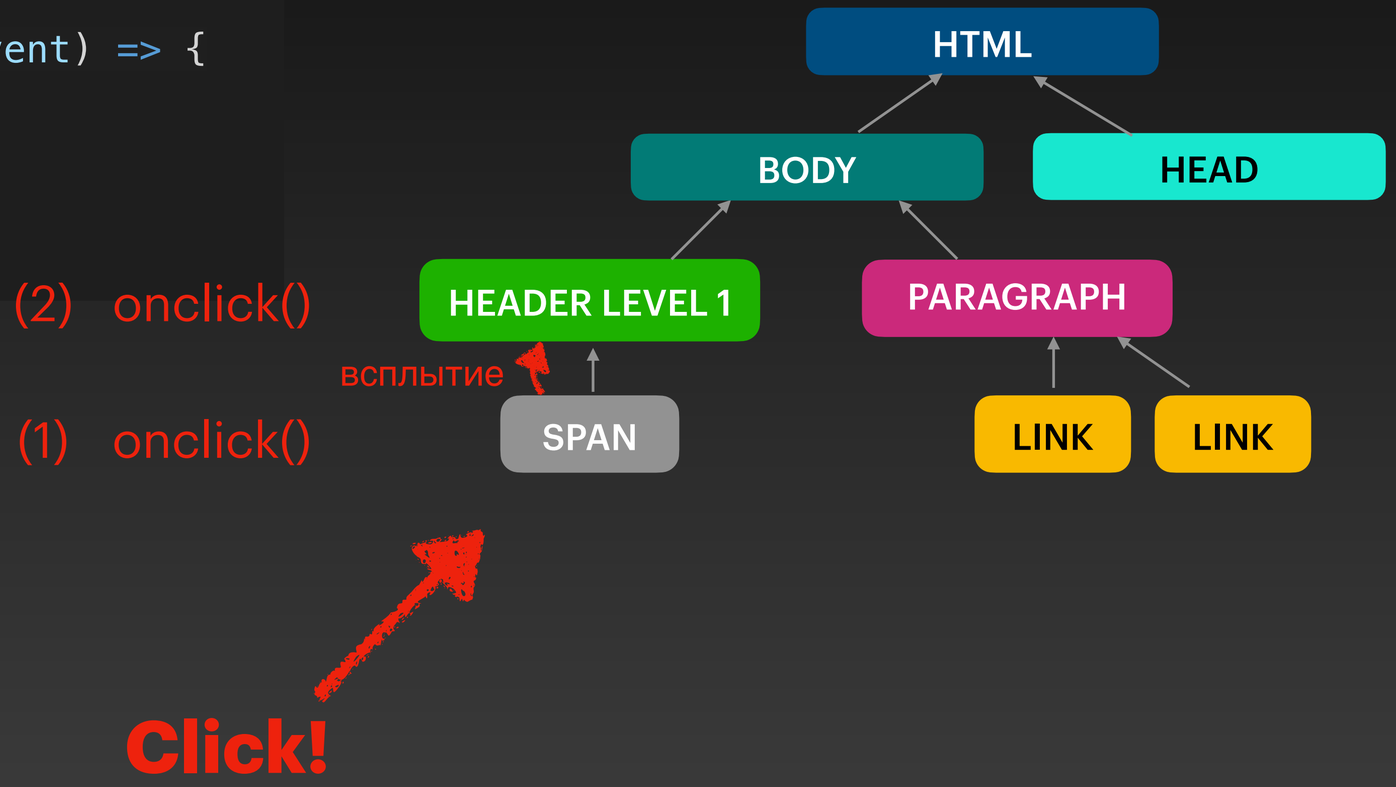


(1) onclick()

Click!

Событие всплыло на уровень выше

```
document.body.addEventListener('click', (event) => {
  console.log(`
    clicked on ${event.currentTarget}
  `);
});
```



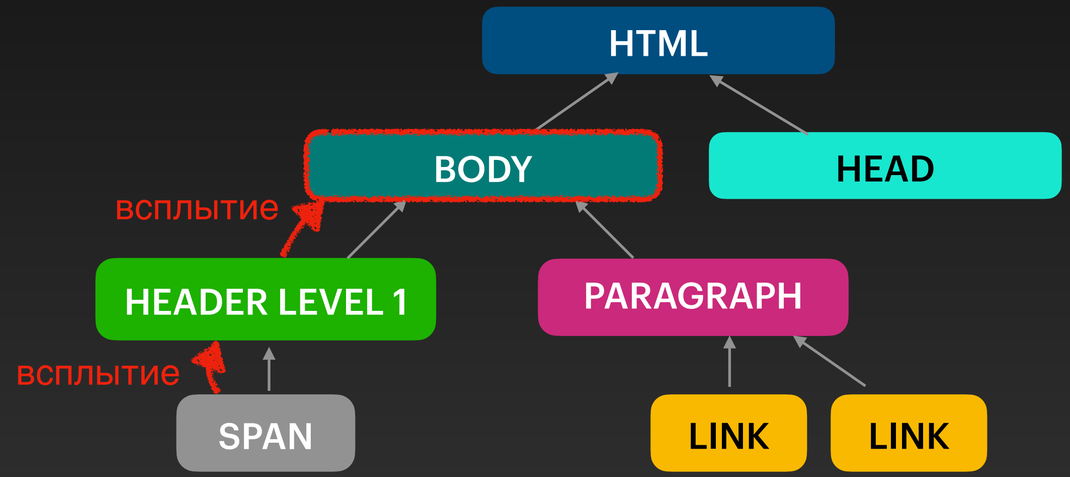
Событие всплыло до body, срабатывает обработчик

```
document.body.addEventListener('click', (event) => {
  console.log(`
    clicked on ${event.currentTarget}
  `);
});
```

(3) onclick()

(2) onclick()

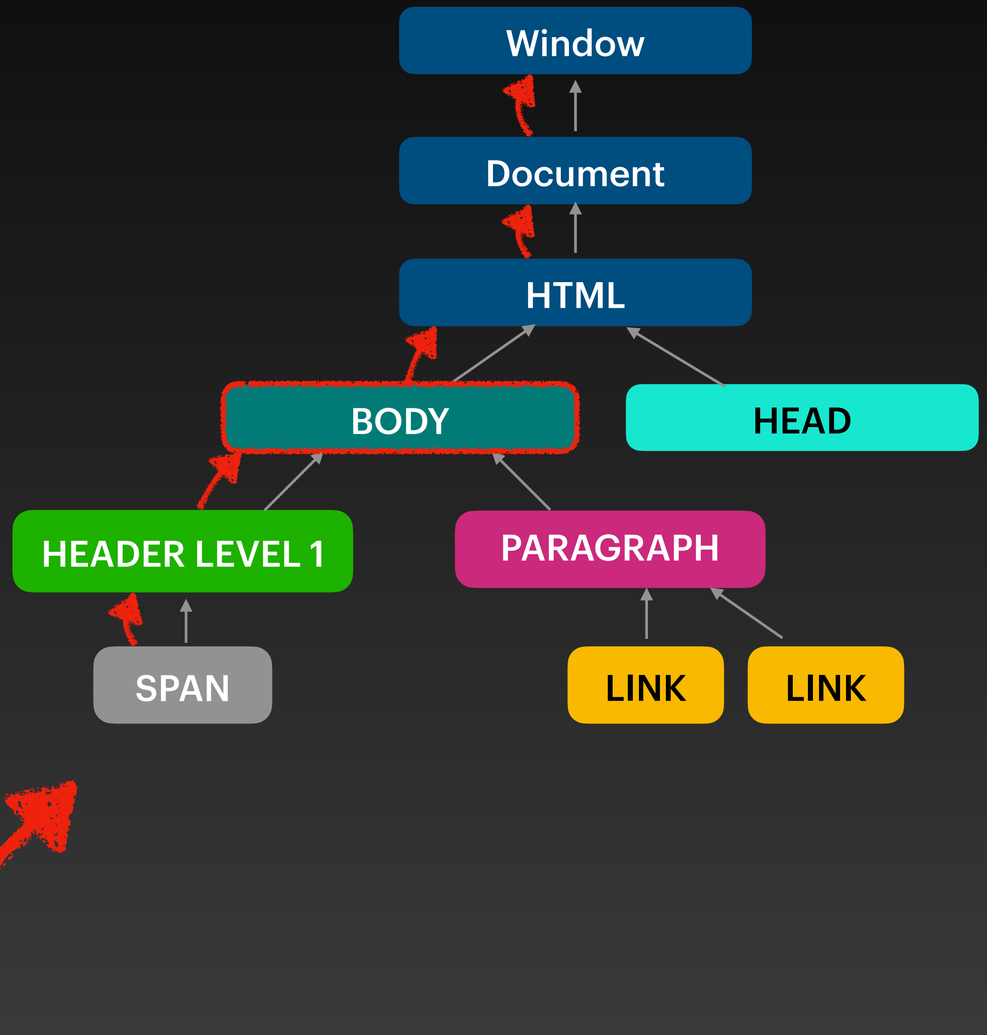
(1) onclick()



Click!

Дальнейшее всплытие

```
document.body.addEventListener('click', (event) => {
  console.log(`
    clicked on ${event.currentTarget}
  `);
});
```



Всплытие

- *Почти* все события всплывают
  - Исключения — события специфичные для одного элемента (scroll, focus и некоторые другие)

- В первом приближении, можно считать что всплывают все события

Event Target

- event.currentTarget — указывает на элемент, на котором сработал обработчик
- Этот элемент из-за всплытия может не совпадать с элементом, на котором изначально произошло событие

- event.target — ссылается на целевой элемент, с которого началось всплытие

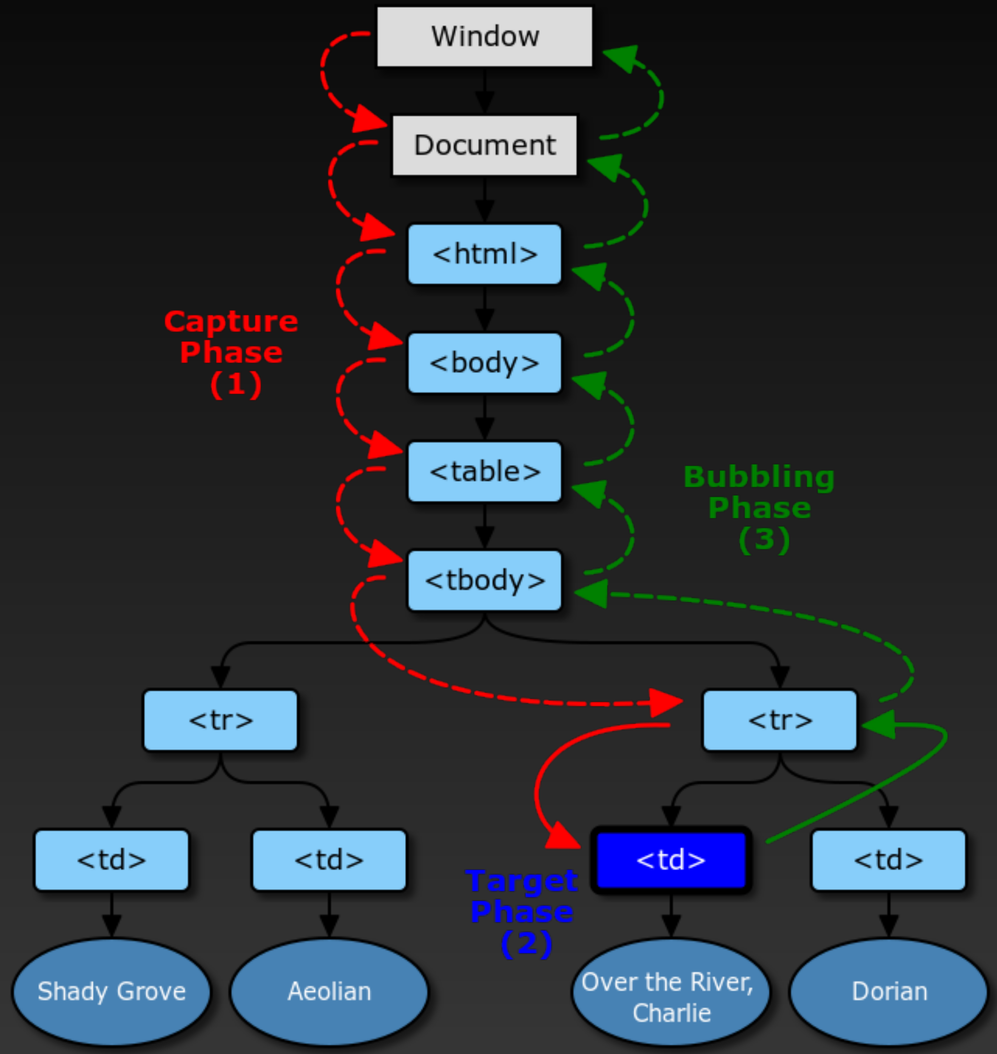
Отмена всплытия

- event.stopPropagation() — останавливает дальнейшее всплытие вверх по документу
- event.stopImmediatePropagation() — делает то же самое, но ещё и не даёт отработать остальным обработчикам на этом же узле

Стоит использовать только в крайнем случае!

Погружение

- В спецификации есть три фазы жизненного цикла события
  - Фаза погружения (capturing)
  - Фаза цели (target)
  - Фаза всплытия (bubbling)



https://www.w3.org/TR/DOM-Level-3-Events/

Обработчик на погружение

```
document.body.addEventListener('click', (event) => {
  console.log(`погружение`);
}, {capture: true});

document.body.addEventListener('click', (event) => {
  console.log(`всплытие`);
});
```

Удаление обработчика

- Чтобы убрать обработчик, нужно указать ту же фазу

```
const handleClick = () => {
  console.log(‘погружение');
}

document.body.addEventListener('click', handleClick, {capture: true});
// ...
document.body.removeEventListener('click', handleClick, {capture: true});
```

Делегирование событий

Делегирование событий

- Если у нас есть много элементов, события на которых нужно обрабатывать похожим образом, то вместо того, чтобы назначать обработчик каждому, мы можем воспользоваться делегированием событий.

Делегирование событий

- Выбираем родительский элемент
- Добавляем на него обработчик событий
- Через event.target определяем, какой элемент был выбран

```
<script>
  const list = document.getElementById('list');

  list.addEventListener('click', (event) => {
      console.log(event.target.dataset.value);
  })
</script>

<ol id="list">
  <li data-value="kek-1">Lol</li>
  <li data-value="kek-2">Kek</li>
  <li data-value="kek-3">Check</li>
</ol>
```