



# ПРИНЦИП ОБРАБОТКИ СОБЫТИЙ



АЛЕКСЕЙ СУДНИЧНИКОВ



# АЛЕКСЕЙ СУДНИЧНИКОВ

Руководитель группы разработки  
«Портал ПФДО»



[@avsudnichnikov](https://www.instagram.com/avsudnichnikov)



# ПЛАН ЗАНЯТИЯ

1. [Перехват и всплытие событий](#)
2. [Делегирование событий](#)
3. [Удаление обработчиков событий](#)
4. [Полезные ссылки](#)



# **ПЕРЕХВАТ И ВСПЛЫТИЕ (CAPTURING & BUBBLING)**

---

# КАТАЛОГ ТОВАРОВ

У нас есть интернет-магазин и есть список продуктов на странице. Каждый продукт представляет из себя карточку с изображением продукта, наименованием, ценой и кнопкой «Добавить в корзину».

1. При клике по карточке продукта открывается окно предпросмотра продукта.
2. При клике на кнопке «Добавить в корзину» продукт добавляется в корзину.

# РАЗМЕТКА КАРТОЧКИ ТОВАРА

Предположим, что у карточки товара следующая несложная разметка:

```
1 <div id="product">
2   
3   <h3 id="h3">Телепорт бытовой VZНІН-101</h3>
4   <div id="price">170 000 руб.</div>
5   <button id="addToCart">Добавить в корзину</button>
6 </div>
```

То есть, у нас есть карточка продукта, в которой есть изображение продукта, его наименование, цена и кнопка «Добавить в корзину».

## «ПРЕДПРОСМОТР» ТОВАРА

Для упрощения мы не будем разрабатывать функционал предпросмотра товара, а просто выведем в консоль информацию о том, что данный функционал был вызван по клику на карточку продукта:

```
1  const product = document.querySelector('#product');
2  product.addEventListener('click', function() {
3      console.log('Предпросмотр продукта');
4  });
```

## ДОБАВЛЯЕМ В КОРЗИНУ

Также при клике на кнопке «Добавить в корзину» продукт должен добавляться в корзину – для этого нам нужно добавить событие на кнопку.

Аналогично с предпросмотром, мы не будем реализовывать функционал добавления в корзину, а также выведем эту информацию в консоль.

```
1  const addToCart = document.querySelector('#addToCart');
2  addToCart.addEventListener('click', function() {
3      console.log('Продукт был добавлен в корзину');
4  });
```



# ПРОВЕРЯЕМ РАБОТУ СОБЫТИЙ

Первое, что мы хотим попробовать — добавить товар в корзину. Кликаем по кнопке «Добавить в корзину» и мы видим, что при клике на кнопку у нас в консоль вывелось:

Продукт был добавлен в корзину

Предпросмотр продукта

## СРАБАТЫВАЮТ ОБА СОБЫТИЯ

Из-за того, что кнопка вложена в карточку товара, у нас произошло оба события – клик по кнопке и клик по карточке.

Следует обратить внимание на порядок: сначала произошел клик на кнопке, а потом на карточке, что означает, что события вызвались по очереди — от вложенного элемента к родительскому. В JavaScript это называется **всплытием событий**.

# stopPropagation

Какая проблема возникает в данном случае? Вызвались оба события, хотя нужно было вызвать только одно событие — добавление продукта в корзину.

Скорее всего, такое поведение не устроит. Ведь нужно вызвать только то событие, которое нужно. И нужно как-то отменить всплытие, так как в данном случае оно нам только мешает и вызывает ненужные действия.

Для отмены всплытия у события в JavaScript существует метод `stopPropagation` — нам его нужно вызвать в тот момент, когда мы хотим прекратить дальнейшее всплытие.

## ОТМЕНЯЕМ ВСПЛЫТИЕ

В нашем случае это клик по кнопке «Добавить в корзину»:

```
1  const addToCart = document.querySelector('#addToCart')
2  addToCart.addEventListener('click', function(event) {
3      console.log('Продукт был добавлен в корзину')
4      event.stopPropagation();
5  });
```

Важно обратить внимание на то, что в первом примере мы не передавали аргумент `event`, так как он нам не был нужен. Теперь, когда он нам потребовался (метод `stopPropagation` существует у объекта события), мы его добавили в обработчик клика.

---

## КОРРЕКТНАЯ РАБОТА

Теперь, если попробовать поработать с нашим примером, при клике на кнопке «Добавить в корзину» все произойдет так, как нам требуется — продукт будет добавлен в корзину, но предпросмотр продукта, как это было раньше, не отработает.



# НЕ ЗЛОУПОТРЕБЛЯЙТЕ

Другие разработчики в команде могут ожидать всплытия, а вы его отменяете.

## ВСПЛЫВАЕТ БОЛЬШИНСТВО СОБЫТИЙ (НО НЕ ВСЕ)

В примере был рассмотрен вариант с событием «клик», но важно отметить, что «всплывают» не только клики, но и другие события, например, наведение курсора мыши на элемент.

Но есть такие событие, которые не всплывают (например, `focus`, `blur` и т.д.)

Выше была рассмотрена ситуация со «всплытием» событий, но при разработке иногда может так получиться, что «всплытие» от внутреннего элемента к внешнему не устраивает.

# ВСЁ ТОТ ЖЕ ТЕЛЕПОРТ

Добавим новые обработчики событий:

```
1 product.addEventListener(  
2   'mouseover',  
3   () => console.log('Вы навели курсок на блок')  
4 )  
5 img.addEventListener(  
6   'mouseover',  
7   () => console.log('Вы навели курсок на картинку')  
8 )
```



---

# ЛОГИКА РАБОТЫ

Теперь, если навести курсор мыши на изображение, то по правилу всплытия сначала отработает обработчик для самого вложенного элемента (изображение) и выведется сообщение **Вы навели курсор на изображение**, и только потом отработает обработчик для блока продукта и выведется сообщение **Вы навели курсор на блок**.

# ТРИ СТАДИИ

В JavaScript существует три стадии прохода события:

1. Событие идет сверху вниз — стадия перехвата.
2. Событие достигло цели — стадия цели.
3. Событие идет снизу вверх — стадия всплытия.

Для понимания, на какой стадии был вызван обработчик, существует `event.eventPhase` со значениями 1, 2 и 3.

По умолчанию, как было рассказано ранее, события перехватываются на стадии всплытия в порядке от самого вложенного к внешнему.

## ИЗМЕНЯЕМ НАПРАВЛЕНИЕ

Чтобы изменить такое поведение и изменить направление порядка событий, используется третий аргумент в `addEventListener`, который отвечает за то, когда событие будет перехвачено. Если третий параметр равен `true`, то обработчик будет вызван на стадии перехвата события.


# НУЖНЫЙ ПОРЯДОК

Теперь порядок будет таким, какой он нужен — сначала обработчик для блока, затем обработчик для изображения:

```
1 product.addEventListener(  
2   'mouseover',  
3   () => console.log('Вы навели курсок на блок'),  
4   true  
5 )  
6 img.addEventListener(  
7   'mouseover',  
8   () => console.log('Вы навели курсок на картинку'),  
9   true  
10 )
```

# stopPropagation (ЕЩЁ РАЗ)

На самом деле `stopPropagation` отменяет не только всплытие, но и захват, в зависимости от того, на какой стадии был вызван.



# ДЕЛЕГИРОВАНИЕ СОБЫТИЙ

## ПРИМЕР С ТЕЛЕПОРТОМ

Уберём все лишнее, оставим один обработчик события `click` на блоке продукта:

```
1 | product.addEventListener('click', function() {  
2 |     console.log('Предпросмотр продукта')  
3 | })
```

## ОДИН ОБРАБОТЧИК ДЛЯ ВЛОЖЕННЫХ ЭЛЕМЕНТОВ

Если кликнуть по карточке, то данное сообщение выведется в консоль.

А если кликнуть по любому вложенному элементу?

В консоль также выведется сообщение `Предпросмотр продукта`. Это уже известно из «всплытия» событий.

Получается, что обработчик, добавленный для родителя, отработает и на всех вложенных элементах. Из этого следует, что можно добавить один обработчик, который будет отрабатывать и для всех вложенных элементов.

Но такой обработчик скорее всего будет бесполезен, если не будет возможности понять, на каком именно элементе случилось событие.



## event.target

Для отслеживания того, на каком элементе случилось событие, в JavaScript существует элемент `event.target` или **целевой элемент**. Это самый глубокий элемент, на котором случилось событие. С этим элементом можно работать и из информации в данном объекте можно узнать, на каком именно элементе произошло событие.

Не путайте `event.currentTarget` (или знакомый вам `this`) и `event.target`.

`event.currentTarget` указывает на элемент, на который добавлено событие (в нашем примере это карточка `#product`).

`event.target` — это элемент, на котором событие случилось (в примере это заголовок или изображение).

# Используйте `event.currentTarget`

`this` при использовании внутри стрелочной функции указывает **не** на `event.currentTarget`.

Более универсальный способ: всегда использовать `event.currentTarget`.

## «УЗНАЕМ» ЭЛЕМЕНТ

В нашем примере из `event.target` можно узнать идентификатор элемента, на котором событие произошло:

```
1 product.addEventListener('click', function(event) {  
2   console.log(`Клик на элементе '${ event.target.id }'`)  
3 })
```

## РЕЗУЛЬТАТ РАБОТЫ

Теперь при клике на `<div id="product">` или любом вложенном в него элементе в консоль будет выводиться идентификатор, на котором произошло событие.

При клике на заголовок в консоль будет выведено:

Клик на элементе 'h3'

При клике на картинку:

Клик на элементе 'img'

Можно также узнать имя CSS-класса, значение data-атрибута, идентификатор и так далее.



## ИЗВЛЕКАЕМ ПОЛЬЗУ

Достаточно добавить один обработчик, а внутри него уже разбираться с тем, на каком элементе произошло событие и, в зависимости от этого, вызывать необходимый функционал.

Во-первых, это может быть удобнее.

Во-вторых, нет необходимости добавлять одинаковые обработчики на каждый отдельный элемент, что дает выигрыш в производительности.

Псс, парень, помнишь пример с вкладками?

## ВЫВОДИМ ЭЛЕМЕНТЫ

Немного доработаем пример, чтобы при клике на заголовок в консоль выводилось сообщение о том, что был кликнут заголовок, а при клике на изображении — что клик произошел на изображении:

```
1 product.addEventListener('click', function(event) {  
2     if (event.target.id === 'h3') {  
3         console.log('Был кликнут заголовок');  
4     }  
5     if (event.target.id === 'img') {  
6         console.log('Было кликнуто изображение');  
7     }  
8 });
```

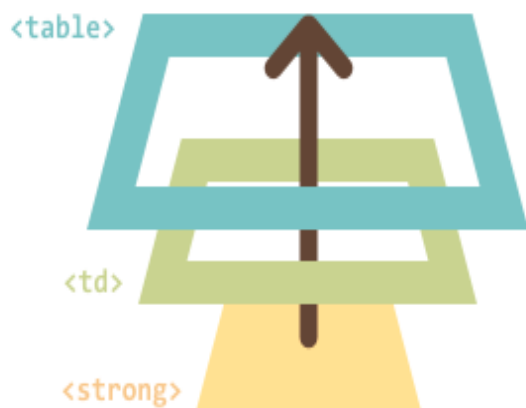
---

## БОЛЕЕ СЛОЖНАЯ ЛОГИКА

Как видно, при клике проверяется значение `event.target.id` и в зависимости от этого отрабатывает разная логика. В приведенном примере логика очень простая (вывод сообщения в консоль), но на реальном примере это может быть гораздо более сложная логика.



Прием, когда вместо нескольких обработчиков ставится один обработчик на родительский элемент, называется **делегированием событий**.





## ДЕЛЕГИРОВАНИЕ ДЛЯ ТАБЛИЦЫ

Другой пример практического применения делегирования — таблица с данными. Чтобы не добавлять обработчик на каждую ячейку, можно добавить один обработчик на всю таблицу и при клике на таблицу вызывать нужное событие для ячейки, на которой произошло событие.

Возьмем следующую разметку для таблицы:

```
1 <table id="table">
2   <tr>
3     <td>Строка 1</td>
4   </tr>
5   <tr>
6     <td>Строка 2</td>
7   </tr>
8 </table>
```

## СОБЫТИЕ НА ТАБЛИЦУ

И добавим события на таблицу:

```
1  const table = document.getElementById('table')
2
3  table.addEventListener('click', function(e) {
4      console.log(`Была кликнута ячейка ${e.target.tagName}`)
5  });
```

В данном случае, сколько бы у нас ни было строк и ячеек, мы не делаем отдельный обработчик на каждую ячейку, а добавляем один на всю таблицу.



# **УДАЛЕНИЕ ОБРАБОТЧИКА СОБЫТИЯ**

# КЛИК ПО КНОПКАМ

Например, на все кнопки мы добавляем функцию, которая выводит в консоль информацию о том, что кнопка были кликнута:

```
1  const buttons = document.querySelectorAll('button')
2  function buttonClick() {
3      console.log('Кнопка была кликнута');
4  }
5
6  Array.from(buttons).forEach((button) => {
7      buttons.addEventListener('click', buttonClick);
8  })
```

Теперь при клике на все кнопки в консоль будет выводиться сообщение о том, что кнопка была кликнута.

## ОБРАБОТКА ДОБАВЛЕНИЯ В КОРЗИНУ

Затем мы решили немного переделать функционал и для кнопки добавления в корзину вывести другое сообщение:

```
1 function addToCart() {  
2     console.log('Была кликнута кнопка  
3         "Добавить в корзину"');  
4 }  
5 document.getElementById('addToCart')  
6     .addEventListener('click', addToCart);
```

Теперь при клике на кнопку `<button id="addToCart" />` отработают сразу два обработчика: сначала в консоль выведется `Кнопка была кликнута`, а потом `Была кликнута кнопка "Добавить в корзину"`. События будут вызваны в том порядке, в котором они были назначены.



## ОСТАВЛЯЕМ ОДИН ОБРАБОТЧИК

Но так как в данном случае требуется только один обработчик для клика, то предыдущий, добавленный для всех кнопок, необходимо удалить.

# removeEventListener

Для удаления обработчика событий в JavaScript есть метод `removeEventListener`.

У `removeEventListener`, как и у `addEventListener`, есть три аргумента:

1. `event` — имя удаляемого события, например, `click` или `mouseover`.
2. `handler` — ссылка на ту же функцию, которую мы добавили в качестве обработчика данного события. Если это был анонимный обработчик, то удалить его не получится.
3. `phase` — «фаза», с которой нужно снять обработчик. Должен **совпадать** с аналогичным параметром при вызове `addEventListener`

# ОБЯЗАТЕЛЬНОЕ УКАЗАНИЕ ОБРАБОТЧИКА

Вопрос: зачем при удалении события нужно три атрибута и зачем точно указывать, какой обработчик нужно удалить?

При помощи `addEventListener` можно добавить несколько обработчиков, а удалить нужно какой-то определенный. Поэтому вторым параметром нам обязательно нужно передать ссылку на обработчик, который был навешен на событие.

Это должен быть **тот же** обработчик, а не точно **такой же**.

Ну и не забываем про третий параметр.



## УДАЛЯЕМ ОБЩЕЕ СОБЫТИЕ


Таким образом, ненужное в данном случае событие может быть удалено следующим образом:

```
document.getElementById('addToCart')  
    .removeEventListener('click', buttonClick);
```

Теперь на кнопке `addToCart` только одно событие — `addToCart`. И при клике на кнопку в консоль будет выведено только одно сообщение, которое выводит функция `addToCart`.



# ПОЛЕЗНЫЕ ССЫЛКИ

- 
- [Введение в браузерные события](#)
  - [MDN. EventTarget.addEventListener\(\)](#)
  - [W3C. Description of event flow](#)



# ИТОГИ

## ПЕРЕХВАТ И ВСПЛЫТИЕ

- Если на родителе и на вложенном элементе установлены обработчики, и клик приходится на вложенный элемент, то сработают оба обработчика. Это и есть **всплытие событий**.
- `stopPropagation` отменяет всплытие событий.
- Метод `stopPropagation` существует у объекта события.
- **Всплывает** почти любое событие, не только клик.
- Можно изменить фазу перехвата события, добавив третий аргумент, равный `true`.

# ДЕЛЕГИРОВАНИЕ СОБЫТИЙ

- `event.target` помогает отследить, на каком элементе произошло событие.
- Используйте `event.currentTarget` вместо `this`.
- Один обработчик на родительский элемент, плюс логика проверки элемента, на который кликнули — **делегирование событий**.

# УДАЛЕНИЕ ОБРАБОТЧИКА

- Удалить можно только неанонимную функцию обработчика.
- Удаляем обработчик при помощи `removeEventListener`.
- В аргумент `event` передаем имя удаляемого события.
- В аргумент `handler` — ссылку на функцию-обработчик.
- Не забываем про аргумент `phase`.



Задавайте вопросы и напишите отзыв о лекции!

**АЛЕКСЕЙ СУДНИЧНИКОВ**



[@avsudnichnikov](https://www.instagram.com/avsudnichnikov)