# Всплытие и погружение, делегирование

Когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе, затем выше и так далее, вверх по цепочке предков**.** Это работает почти для всех событий, но есть исключения, н-р, focus не всплывает.

**event.target**

Всегда можно узнать, на каком конкретно элементе произошло событие. Самый глубокий элемент, который вызывает событие, называется целевым элементом, и он доступен через event.target.

Отличия от **this (=event.currentTarget):**

event.target – это «целевой» элемент, на котором произошло событие, в процессе всплытия он неизменен.

this – это «текущий» элемент, до которого дошло всплытие, на нём сейчас выполняется обработчик.

Например, если стоит только один обработчик form.onclick, то он «поймает» все клики внутри формы. Где бы ни был клик внутри – он всплывёт до элемента <form>, на котором сработает обработчик. При этом внутри обработчика form.onclick: this (=event.currentTarget) всегда будет элемент <form>, так как обработчик сработал на ней. event.target будет содержать ссылку на конкретный элемент внутри формы, на котором произошёл клик.

**event.stopPropagation()** – прекращение всплытия (вертикального). Для внутреннего элемента.

**event.stopImmediatePropagation() –** этот метод не только предотвращает всплытие, но и останавливает обработку событий на текущем элементе, если есть несколько обработчиков.

## Погружение

Существует ещё одна фаза из жизненного цикла события – «погружение» (иногда её называют «перехват»).

Стандарт DOM Events описывает 3 фазы прохода события:

Фаза погружения (**capturing phase**) – событие сначала идёт сверху вниз.
Фаза цели (**target phase**) – событие достигло целевого(исходного) элемента.
Фаза всплытия (**bubbling stage**) – событие начинает всплывать.

Чтобы поймать событие на стадии погружения, нужно в **addEventListener(event, handler)** использовать третий аргумент **capture** вот так:

```
elem.addEventListener(..., {capture: true})
// или просто "true", как сокращение для {capture: true}
elem.addEventListener(..., true)
```

Если аргумент **false** (по умолчанию), то событие будет поймано при всплытии.
Если аргумент **true**, то событие будет перехвачено при погружении.

Обратите внимание, что хоть и формально существует 3 фазы, 2-ую фазу («фазу цели»: событие достигло элемента) нельзя обработать отдельно, при её достижении вызываются все обработчики: и на всплытие, и на погружение.

Существует свойство **event.eventPhase**, содержащее номер фазы, на которой событие было поймано.

Чтобы убрать обработчик removeEventListener, нужна та же фаза

```
addEventListener(..., true)
removeEventListener(..., true)
```

## Делегирование событий

Идея в том, что если у нас есть много элементов, события на которых нужно обрабатывать похожим образом, то вместо того, чтобы назначать обработчик каждому, мы ставим один обработчик на их общего предка.

Из него можно получить целевой элемент **event.target**, понять на каком именно потомке произошло событие и обработать его.
Для проверки также можно использовать метод **elem.closest(selector)** возвращает ближайшего предка, соответствующего селектору и **elem.contains(selector)**

Можно применять **data-атрибуты**:

```
<div id="menu">
 <button data-action="save">Сохранить</button>
 <button data-action="load">Загрузить</button>
 <button data-action="search">Поиск</button>
</div>
<script>
 class Menu {
  constructor(elem) {
   this._elem = elem;
   elem.onclick = this.onClick.bind(this); // (*)
  }
  save() {
   alert('сохраняю');
  }
  load() {
   alert('загружаю');
  }
  search() {
   alert('ищу');
  }
  onClick(event) {
   let action = event.target.dataset.action;
   if (action) {
    this[action]();
   }
  };
 }
 new Menu(menu);
</script>
```

Делегирование событий можно использовать для добавления элементам «поведения» **(behavior),** декларативно задавая хитрые обработчики установкой специальных HTML-атрибутов и классов.
1. **Приём проектирования «поведение»** состоит из двух частей:
   - Элементу ставится пользовательский атрибут, описывающий его поведение.
   - При помощи делегирования ставится обработчик на документ, который ловит все клики (или другие события) и, если элемент имеет нужный атрибут, производит соответствующее действие.

```
Счётчик: <input type="button" value="1" data-counter>
Ещё счётчик: <input type="button" value="2" data-counter>
<script>
 document.addEventListener('click', function(event) {
  if (event.target.dataset.counter != undefined) { // если есть атрибут...
   event.target.value++;
```

```
    }
  });
</script>
```

2. **Поведение: «Переключатель» (Toggler)** Ещё один пример поведения. Сделаем так, что при клике на элемент с атрибутом data-toggle-id будет скрываться/показываться элемент с заданным id:

```
<button data-toggle-id="subscribe-mail">
  Показать форму подписки
</button>

<form id="subscribe-mail" hidden>
  Ваша почта: <input type="email">
</form>

<script>
  document.addEventListener('click', function(event) {
    let id = event.target.dataset.toggleId;
    if (!id) return;
    let elem = document.getElementById(id);
    elem.hidden = !elem.hidden;
  });
</script>
```