Front-end

JavaScript

JS. BOM, DOM, API

- DOM

JS. BOM, DOM, API

Всплытие и погружение

Всплытие

Принцип всплытия очень простой.

Когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе, затем выше и так далее, вверх по цепочке предков.

Например, есть 3 вложенных элемента FORM > DIV > P с обработчиком на каждом

JS. BOM, DOM, API

Всплытие и погружение

```
<form onclick="alert('form')">FORM
 <div onclick="alert('div')">DIV
  <p onclick="alert('p')">P</p>
 </div>
</form>
```

JS. BOM, DOM, API

Всплытие и погружение

Клик по внутреннему <p> вызовет обработчик onclick:

Сначала на самом <p>.

Потом на внешнем .

Затем на внешнем <form>.

И так далее вверх по цепочке до самого document.

Поэтому если кликнуть на <p>, то мы увидим три оповещения: p → div → form.

Этот процесс называется «всплытием», потому что события «всплывают» от внутреннего элемента вверх через родителей подобно тому, как всплывает пузырёк воздуха в воде.

Почти все события всплывают.

JS. BOM, DOM, API

Всплытие и погружение

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.

JS. BOM, DOM, API

Объект события

Свойства

bubbles — является ли данное событие всплывающим.

cancelable — является ли событие отменяемым.

currentTarget — указывает на элемент, на котором установлен обработчик события.

defaultPrevented — отменено ли поведение события по умолчанию.

eventPhase — указывает на фазу срабатывания события.

isTrusted — указывает на происхождение события, будет в значении true, если событие инициировано действиями пользователя. false — если событие инициировано из кода с помощью dispatchEvent().

target — ссылка на объект, которым было инициировано событие. Например, если событие произошло на поле ввода, мы получим ссылку на этот DOM элемент.

timeStamp — время возникновения события в миллисекундах.

type — тип события.

JS. BOM, DOM, API

Объект события

Методы

composedPath() — вернёт массив элементов, на которых сработает событие.

preventDefault() — предотвращает дефолтное поведение события. Если вызвать этот метод на событии клика по ссылке, то переход по ссылке не произойдёт, но событие продолжит всплытие.

stopPropagation() — предотвращает всплытие события.

stopImmediatePropagation() — делает то же самое, что и stopPropagation, но в том числе предотвращает вызов обработчиков события, которые были установлены на этом же элементе.

JS. BOM, DOM, API

Всплытие и погружение

Прекращение всплытия

Всплытие идёт с «целевого» элемента прямо наверх. По умолчанию событие будет всплывать до элемента <html>, а затем до объекта document, а иногда даже до window, вызывая все обработчики на своём пути.

Но любой промежуточный обработчик может решить, что событие полностью обработано, и остановить всплытие.

Для этого нужно вызвать метод event.stopPropagation().

JS. BOM, DOM, API

Всплытие и погружение

Погружение

Существует ещё одна фаза из жизненного цикла события – «погружение» (иногда её называют «перехват»). Она очень редко используется в реальном коде, однако тоже может быть полезной.

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.

JS. BOM, DOM, API

Всплытие и погружение

Обработчики, добавленные через on<event>-свойство или через HTML-атрибуты, или через addEventListener(event, handler) с двумя аргументами, ничего не знают о фазе погружения, а работают только на 2-ой и 3-ей фазах.

Чтобы поймать событие на стадии погружения, нужно использовать третий аргумент capture вот так:

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

Существуют два варианта значений опции capture:

Если аргумент false (по умолчанию), то событие будет поймано при всплытии.

Если аргумент true, то событие будет перехвачено при погружении.

JS. BOM, DOM, API

Всплытие и погружение

В событии есть два похожих поля: target и currentTarget. Их отличие легко увидеть на примере.

Создадим кнопку, положим в неё текст, обёрнутый в тег , и навесим обработчик событий на кнопку. При клике на кнопку можно заметить, что currentTarget всегда будет кнопкой, на которой обрабатывается событие. При этом target будет меняться в зависимости от того, куда на кнопке мы кликнули — на span внутри кнопки или на неё саму.

```
<button class="button" type="button">
  <span>Моя кнопочка</span>
</button>
```

JS. BOM, DOM, API

Всплытие и погружение

Обработчики, добавленные через on<event>-свойство или через HTML-атрибуты, или через addEventListener(event, handler) с двумя аргументами, ничего не знают о фазе погружения, а работают только на 2-ой и 3-ей фазах.

Чтобы поймать событие на стадии погружения, нужно использовать третий аргумент capture вот так:

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

Существуют два варианта значений опции capture:

Если аргумент false (по умолчанию), то событие будет поймано при всплытии.

Если аргумент true, то событие будет перехвачено при погружении.

JS. BOM, DOM, API

Делегирование событий

Всплытие и перехват событий позволяет реализовать один из самых важных приёмов разработки – делегирование.

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.

JS. BOM, DOM, API

Делегирование событий – это здорово! Пожалуй, это один из самых полезных приёмов для работы с DOM.

Он часто используется, если есть много элементов, обработка которых очень схожа, но не только для этого.

Алгоритм:

Вешаем обработчик на контейнер.

В обработчике проверяем исходный элемент event.target.

Если событие произошло внутри нужного нам элемента, то обрабатываем его.

Зачем использовать:

Упрощает процесс инициализации и экономит память: не нужно вешать много обработчиков.

Меньше кода: при добавлении и удалении элементов не нужно ставить или снимать обработчики.

Удобство изменений DOM: можно массово добавлять или удалять элементы путём изменения innerHTML и ему подобных.

JS. BOM, DOM, API

Делегирование позволяет не добавлять обработчик к каждому элементу, а сделать этого только для одного общего для всех них элемента.

Допустим, есть список:

```
<ul id="list">
  <li>Один</li>
  <li>Два</li>
  <li>Три</li>
</ul>
```

Предположим, что при нажатии на каждый элемент списка должно что-то произойти. Например, выводиться сообщение.

Мы можем добавить отдельный обработчик для каждого <li>. Но что, если у нас элементы <li> динамически добавляются и удаляются из этого списка?

JS. BOM, DOM, API

В этом случае лучшим решением, конечно, будет добавить обработчик к родительскому элементу. Но если вы добавите обработчик к <ul>, то как вы узнаете, какой элемент был нажат?

Всё просто. Для этого предназначено свойство объекта события target. Оно содержит ссылку на элемент, на который фактически нажали.

```
document.querySelector('#list').addEventListener('click', function (e) {
  if (e.target && e.target.nodeName === 'LI') {
    alert(e.target.textContent);
  }
});
```

JS. BOM, DOM, API

Многие события автоматически влекут за собой действие браузера.

Например:

Клик по ссылке инициирует переход на новый URL.

Нажатие на кнопку «отправить» в форме – отсылку её на сервер.

Зажатие кнопки мыши над текстом и её движение в таком состоянии – инициирует его выделение.

Если мы обрабатываем событие в JavaScript, то зачастую такое действие браузера нам не нужно. К счастью, его можно отменить.

JS. BOM, DOM, API

Отмена действия браузера

Есть два способа отменить действие браузера:

Основной способ – это воспользоваться объектом event. Для отмены действия браузера существует стандартный метод event.preventDefault().

Если же обработчик назначен через on<событие> (не через addEventListener), то также можно вернуть false из обработчика.