События

Браузерные события

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).

События мыши:

click – происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании).
contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши.
mouseover / mouseout – когда мышь наводится на / покидает элемент.
mousedown / mouseup – когда нажали / отжали кнопку мыши на элементе.
mousemove – при движении мыши.

События на элементах управления:

submit – пользователь отправил форму <form>.
focus – пользователь фокусируется на элементе, например нажимает на <input>.

Клавиатурные события:

keydown и keyup – когда пользователь нажимает / отпускает клавишу.

События документа:

DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

CSS events:

transitionend – когда CSS-анимация завершена.

Обработчики событий
Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.

Есть несколько способов назначить событию обработчик.

Использование атрибута HTML

```
<input value="Нажми меня" onclick="alert('Клик!')" type="button">
```

Использование свойства DOM-объекта

```
<input id="elem" type="button" value="Нажми меня!">
<script>
  elem.onclick = function() {
    alert('Спасибо');
  };
</script>
```

```
function sayThanks() {
  alert('Спасибо!');
}

elem.onclick = sayThanks;
```

обработчиком можно назначить и уже существующую функцию:

Доступ к элементу через this

Внутри обработчика события this ссылается на текущий элемент, то есть на тот, на котором, как говорят, «висит» (т.е. назначен) обработчик.

```
<button onclick="alert(this.innerHTML)">Нажми меня</button>
```

addEventListener

```
element.addEventListener(event, handler, [options]);
```

event Имя события, например "click".
handler Ссылка на функцию-обработчик.
options Дополнительный объект со свойствами:

Свойства:

1. once: если true, тогда обработчик будет автоматически удалён после выполнения.
2. capture: фаза, на которой должен сработать обработчик (подробнее в теме Всплытие и погружение)
3. passive: если true, то указывает, что обработчик никогда не вызовет preventDefault() (подробнее Действия браузера по умолчанию)

Для удаления обработчика следует использовать **removeEventListener**:

```
element.removeEventListener(event, handler, [options]);
```

Для удаления нужно передать именно ту функцию-обработчик которая была назначена.

```
function handler() {
  alert( 'Спасибо!' );
}

input.addEventListener("click", handler);
// ....
input.removeEventListener("click", handler);
```

Метод addEventListener позволяет добавлять несколько обработчиков на одно событие одного элемента, например:

```
<!doctype html>
<body>
<input id="elem" type="button" value="Нажми меня"/>

<script>
  function handler1() {
    alert('Спасибо!');
  };

  function handler2() {
    alert('Спасибо ещё раз!');
  }

  elem.onclick = () => alert("Привет");
  elem.addEventListener("click", handler1); // Спасибо!
  elem.addEventListener("click", handler2); // Спасибо ещё раз!
</script>
</body>
```

Объект события

Чтобы хорошо обработать событие, могут понадобиться детали того, что произошло. Не просто «клик» или «нажатие клавиши», а также – какие координаты указателя мыши, какая клавиша нажата и так далее.

Когда происходит событие, браузер создаёт объект события, записывает в него детали и передаёт его в качестве аргумента функции-обработчику.

Пример ниже демонстрирует получение координат мыши из объекта события:

```
<input type="button" value="Нажми меня" id="elem">

<script>
  elem.onclick = function(event) {
    // вывести тип события, элемент и координаты клика
    alert(event.type + " на " + event.currentTarget);
    alert("Координаты: " + event.clientX + ":" + event.clientY);
  };
</script>
```

Некоторые свойства объекта event:

event.type Тип события, в данном случае "click".
event.currentTarget Элемент, на котором сработал обработчик. Значение – обычно такое же, как и у this, но если обработчик является функцией-стрелкой или при помощи bind привязан другой объект в качестве this, то мы можем получить элемент из event.currentTarget.
event.clientX / event.clientY Координаты курсора в момент клика относительно окна, для событий мыши.

Объект-обработчик: handleEvent

Мы можем назначить обработчиком не только функцию, но и объект при помощи addEventListener. В этом случае, когда происходит событие, вызывается метод объекта handleEvent.

К примеру:

```
<button id="elem">Нажми меня</button>

<script>
  elem.addEventListener('click', {
    handleEvent(event) {
      alert(event.type + " на " + event.currentTarget);
    }
  });
</script>
```

если addEventListener получает объект в качестве обработчика, он вызывает object.handleEvent(event), когда происходит событие.

```
<button id="elem">Нажми меня</button>

<script>
  class Menu {
    handleEvent(event) {
      switch(event.type) {
        case 'mousedown':
          elem.innerHTML = "Нажата кнопка мыши";
          break;
        case 'mouseup':
          elem.innerHTML += "...и отжата.";
          break;
      }
    }
  }

  let menu = new Menu();
  elem.addEventListener('mousedown', menu);
  elem.addEventListener('mouseup', menu);
</script>
```

Здесь один и тот же объект обрабатывает оба события. Обратите внимание, нужно явно назначить оба обработчика через addEventListener. Тогда объект menu будет получать события mousedown и mouseup,

```html
<!doctype html>
<body>
<button id="elem">Нажми меня</button>

<script>
  class Menu {
    handleEvent(event) {
      // mousedown -> onMousedown
      let method = 'on' + event.type[0].toUpperCase() + event.type.slice(1);
      this[method](event);
    }

    onMousedown() {
      elem.innerHTML = "Кнопка мыши нажата";
    }

    onMouseup() {
      elem.innerHTML += "...и отжата.";
    }
  }

  let menu = new Menu();
  elem.addEventListener('mousedown', menu);
  elem.addEventListener('mouseup', menu);
</script>
</body>
```

Метод handleEvent не обязательно должен выполнять всю работу сам. Он может вызывать другие методы, которые заточены под обработку конкретных типов событий

Всплытие и погружение

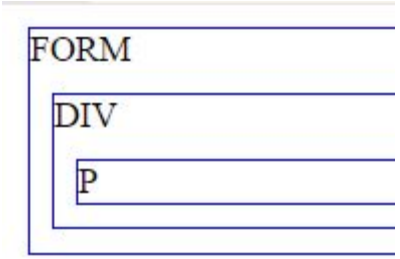
Всплытие

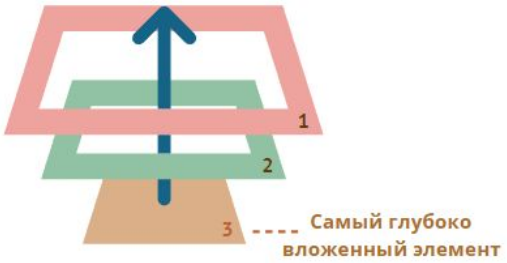
Принцип всплытия очень простой.

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

```
<style>
  body * {
    margin: 10px;
    border: 1px solid blue;
  }
</style>

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





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.

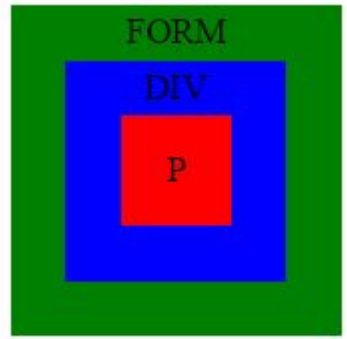
event.target

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.

Возможна и ситуация, когда event.target и this – один и тот же элемент, например, если клик был непосредственно на самом элементе <form>, а не на его подэлементе.



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().

Например, здесь при клике на кнопку <button> обработчик body.onclick не сработает:

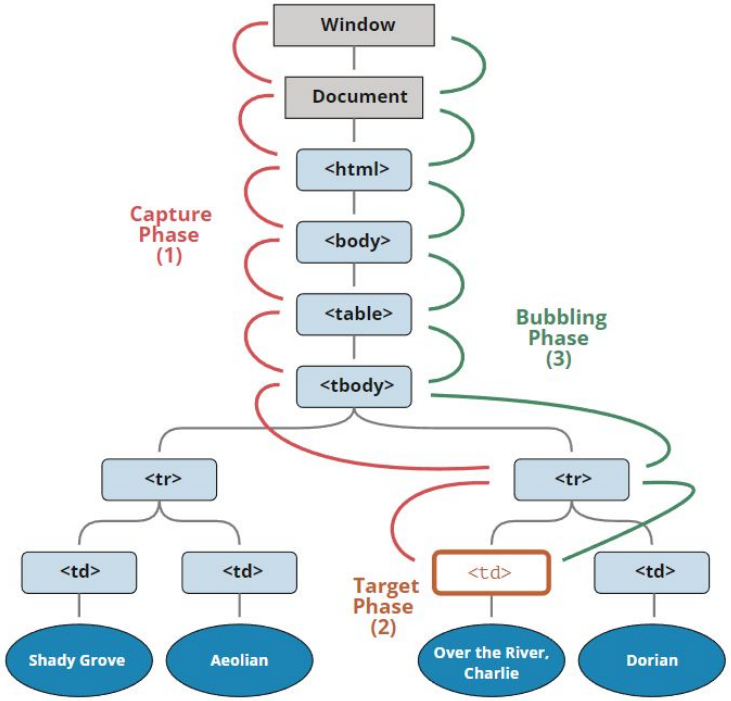
```
<body onclick="alert(`сюда всплытие не дойдёт`)">
  <button onclick="event.stopPropagation()">Кликни меня</button>
</body>
```

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.
Картинка из спецификации демонстрирует, как это работает при клике по ячейке <td>, расположенной внутри таблицы.



То есть при клике на <td> событие путешествует по цепочке родителей сначала вниз к элементу (погружается), затем оно достигает целевой элемент (фаза цели), а потом идёт наверх (всплытие), вызывая по пути обработчики.

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:

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

```
<!doctype html>
<body>
<style>
  body * {
    margin: 10px;
    border: 1px solid blue;
  }
</style>

<form>FORM
  <div>DIV
    <p>P</p>
  </div>
</form>

<script>
  for(let elem of document.querySelectorAll('*')) {
    elem.addEventListener("click", e => alert(`Погружение: ${elem.tagName}`),
      true);
    elem.addEventListener("click", e => alert(`Всплытие: ${elem.tagName}`));
  }
</script>
</body>
```

Здесь обработчики навешиваются на каждый элемент в документе, чтобы увидеть в каком порядке они вызываются по мере прохода события.

Если вы кликните по <p>, то последовательность следующая:

HTML → BODY → FORM → DIV (фаза погружения, первый обработчик)
P (фаза цели, срабатывают обработчики, установленные и на погружение и на всплытие, так что выведется два раза)
DIV → FORM → BODY → HTML (фаза всплытия, второй обработчик)

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.

Квадрат*Bagua*: Направление, Элемент, Цвет, Значение		
Северо-Запад Металл Серебро Старейшины	**Север** Вода Синий Перемены	**Северо-Восток** Земля Жёлтый Направление
Запад Металл Золото Молодость	**Центр** Всё Пурпурный Гармония	**Восток** Дерево Синий Будущее
Юго-Запад Земля Коричневый Спокойствие	**Юг** Огонь Оранжевый Слава	**Юго-Восток** Дерево Зелёный Роман

```
<table>
  <tr>
    <th colspan="3">Квадрат <em>Bagua</em>: Направление, Элемент, Цвет, Значение</th>
  </tr>
  <tr>
    <td>...<strong>Северо-Запад</strong>...</td>
    <td>...</td>
    <td>...</td>
  </tr>
  <tr>...ещё 2 строки такого же вида...</tr>
  <tr>...ещё 2 строки такого же вида...</tr>
</table>
```

задача – реализовать подсветку ячейки <td> при клике.

Вместо того, чтобы назначать обработчик onclick для каждой ячейки <td> (их может быть очень много) – можно повесить «единый» обработчик на элемент <table>.

Он будет использовать event.target, чтобы получить элемент, на котором произошло событие, и подсветить его.

Код будет таким:

```
let selectedTd;

table.onclick = function(event) {
  let target = event.target; // где был клик?

  if (target.tagName != 'TD') return; // не на TD? тогда не интересует

  highlight(target); // подсветить TD
};

function highlight(td) {
  if (selectedTd) { // убрать существующую подсветку, если есть
    selectedTd.classList.remove('highlight');
  }
  selectedTd = td;
  selectedTd.classList.add('highlight'); // подсветить новый td
}
```

Передвиньте мяч по полю

Пусть мяч перемещается при клике на поле, туда, куда был клик

Требования:

Центр мяча должен совпадать с местом нажатия мыши (если это возможно без пересечения краёв поля);

CSS-анимация желательна, но не обязательна;

Мяч ни в коем случае не должен пересекать границы поля;

При прокрутке страницы ничего не должно ломаться;

Заметки:

Код должен уметь работать с различными размерами мяча и поля, не привязываться к каким-либо фиксированным значениям.

Используйте свойства event.clientX/event.clientY для определения координат мыши при клике.

https://plnkr.co/edit/gRBZystyjxJU5Tc5?p=preview&preview

Добавить кнопку закрытия
Есть список сообщений.

При помощи JavaScript для каждого сообщения добавьте в верхний правый угол кнопку закрытия.

https://plnkr.co/edit/wwVsnFXjnkZ1We9m?p=preview&preview

Спрячьте сообщения с помощью делегирования
Дан список сообщений с кнопками для удаления [x]. Заставьте кнопки работать.
https://plnkr.co/edit/7n3obk1PsA8VrNOs?p=preview&preview