



Лекция 7

Введение в браузерные события

Что мы научимся делать?

- вешать события на элементы;
- познакомимся с механизмом всплытия;
- увидим в действии объект события;
- узнаем о делегировании.

Событие (Event) --

это сигнал от браузера о том, что что-то произошло. Существует много видов событий, которые происходят при использовании пользователем клавиатуры, мыши, загрузке страницы.

Некоторые виды событий

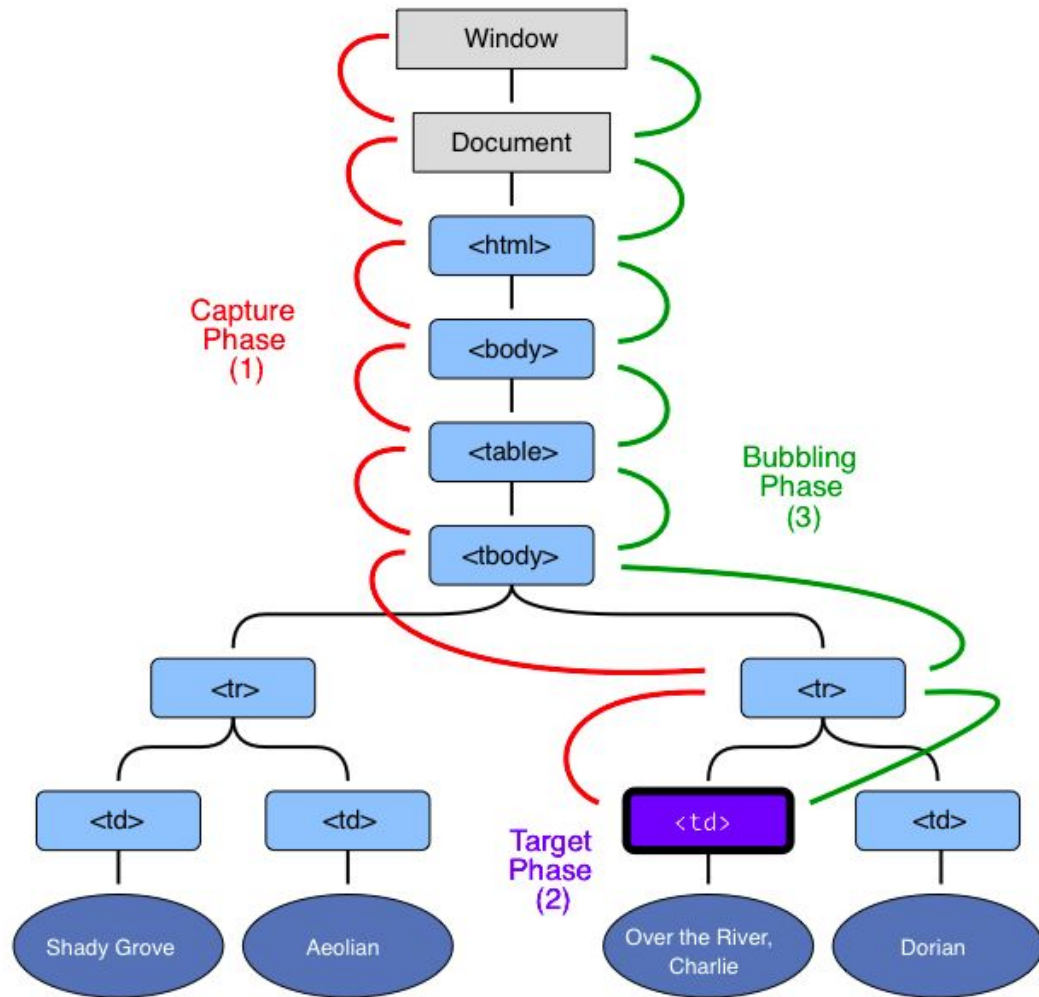
- click, dblclick, mousedown, mouseup
- mouseover, mouseout, mouseenter, mousemove, mouseleave
- contextmenu, wheel
- keydown, keypress, keyup
- focus, blur, change
- DOMContentLoaded, load

Фазы события

Захват (Capturing) - когда происходит событие, информация о нем спускается от корневого элемента DOM-дерева вниз, до того элемента, на котором произошло событие.

Достижение цели (Target) - событие достигает исходного элемента;

Всплытие (Bubbling) - информация о событии поднимается обратно от элемента, на котором произошло событие до корневого элемента DOM-дерева.





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

Как повесить обработчик на событие?

1 способ -- HTML атрибуты

Самый простой способ назначить обработчик -- использование HTML атрибутов, которые начинаются с "on-". В кавычках указывается функция, которая выполнится тогда, когда произойдет событие.

Не рекомендуется, поскольку
смешивается HTML и JS



```
<div class="buttons">  
  <button onclick="buttonClick();">Button 1</button>  
  <button onmouseover="buttonMouseOver();">Button 2</button>  
  <button onfocus="alert(this.innerHTML);">Button 3</button>  
</div>
```

2 способ -- обращение к свойствам элемента



```
const clickButton = document.querySelector('.js-click');
const mouseOverButton = document.querySelector('.js-mouseover');

clickButton.onclick = function() {
  alert('Clicked');
  console.log(this); // element
}

mouseOverButton.onmouseover = () => {
  alert('Mouse over');
  console.log(this); // Window
}
```

Таким образом на элемент
можно повесить только один
обработчик

3 способ -- eventListener



```
const clickButton = document.querySelector('.js-click');
const mouseOverButton = document.querySelector('.js-mouseover');

clickButton.addEventListener('click', function() {
  console.log(this); // element
  alert('Clicked');
})

mouseOverButton.addEventListener('mouseover', () => {
  console.log(this); // Window
  alert('Mouse over');
})
```

Как удалить обработчик?



```
const tasksList = document.querySelector('.tasks');
```

```
const deleteTask = function(e) {  
    if (e.target.nodeName === 'BUTTON') {  
        e.originalTarget.parentNode.remove();  
    }  
}
```

```
tasksList.addEventListener('click', deleteTask);
```


```
// Если нам нужно удалить событие, то мы используем метод  
// removeEventListener, в который нужно передать ссылку на  
// функцию
```

```
tasksList.removeEventListener('click', deleteTask);
```

Объект Event

Чтобы хорошо обработать событие, недостаточно знать о том, что это – «клик» или «нажатие клавиши». Могут понадобиться детали: координаты курсора, введённый символ и другие, в зависимости от события.

Детали произошедшего браузер записывает в «объект события», который передается первым аргументом в обработчик.



```
clickButton.addEventListener('click', function(event) {  
  console.log(this); // <button class="js-click">  
  console.log(event.type); // click  
  console.log(event.originalTarget.parentNode); // <div class="buttons">  
  alert('Clicked');  
});
```


Отмена действия по умолчанию



```
/*
```

Многие события автоматически влекут за собой действие браузера. Если мы обрабатываем событие в JavaScript, то зачастую такое действие браузера нам не нужно. Его можно отменить

```
*/
```

```
const form = document.getElementById('form');
```

```
form.addEventListener('submit', function(e) {  
    e.preventDefault();  
    alert('Форма отправлена');  
});
```

Делегирование событий

Делегирование

Суть делегирования заключается 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.