

# DOM API и события

# Проблемы с браузером

- Функциональность отличается у разных браузеров
- Мы не выбираем браузер пользователя
- Мы не выбираем версию браузера
- Мы не можем менять настройки пользователя
- Полифил – если реализации нет в браузере, то добавляем эту реализацию сами
- Проверяйте здесь: [caniuse.com](https://caniuse.com)

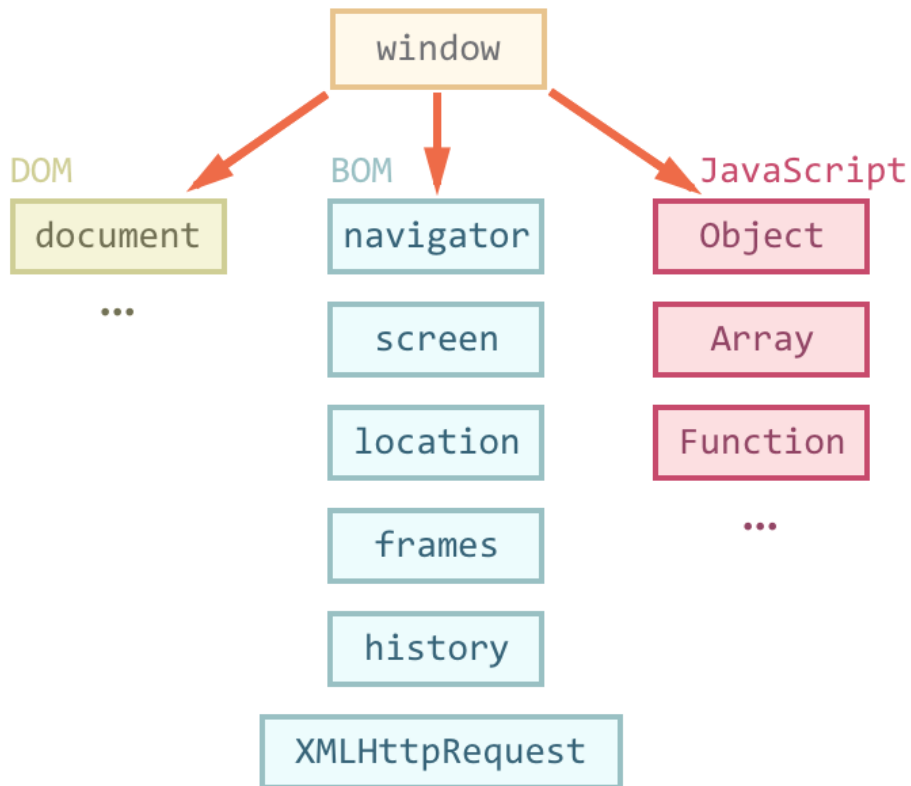
# Полифил

```
if (!Element.prototype.closest) {  
  Element.prototype.closest = function() {  
    // Пишем реализацию полифила  
  };  
}
```

## Полифилы для DOM

# Javascript & Browser

window



# window – глобальный объект

```
window.document === document; // true
```

# Browser Object Model (BOM)

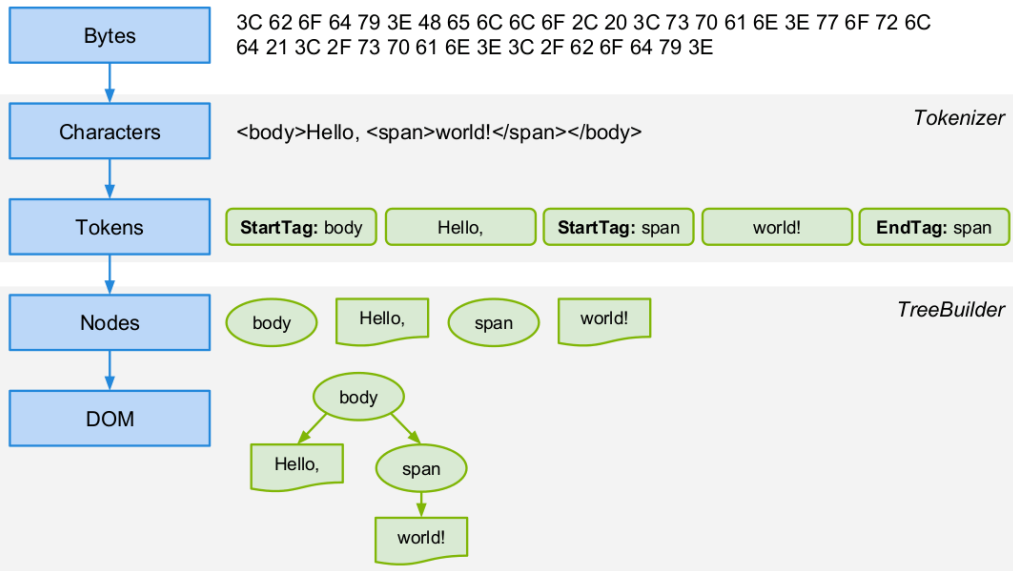
- Объект navigator: информация о самом браузере
- Объект location: адресная строка и переходы по урлу
- Объект history: история переходов и переходы по истории
- Объект screen: информация об экране пользователя
- Функции alert/prompt/confirm



# Document Object Model (DOM)

# HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="description" content="...">
    <title>CERN</title>
  </head>
  <body>
    <p>
      <a href="http://home.web.cern.ch/">CERN</a>
      is a European research organization
    </p>
  </body>
</html>
```



# Типы узлов

- В DOM 12 типов узлов, но мы работаем с 4
- Документ – объект `document`
- Элементы – теги
- Текст
- Комментарии
- Другие

# Поиск элементов

`document.getElementById('id')`

- Ищет элемент с заданным идентификатором
- Возвращает первый найденный элемент
- Возвращает `null`, если элемент с указанным ID не найден в документе

document.getElementById('id')

```
<div id="hello">Hello everybody!</div>
```

```
var elem = document.getElementById('hello');
```

# getElementsByTagName('class')

- Ищет элементы с соответствующим классом
- Есть не только у document, но и у любого элемента
- Возвращает коллекцию



## getElementsByClassName('class')

```
<div class="hello">  
  <div class="greeting">Hello everybody!</div>  
  <div class="greeting">Aloha</div>  
</div>  
<div class="greeting">Привет</div>
```

```
var elems = document.getElementsByClassName('greeting'); // 3 элемента
```

```
var parent = document.getElementsByClassName('hello')[0];
```

```
var elems = parent.getElementsByClassName('greeting'); // 2 элемента
```

# Коллекция – array-like object

- Можно итерировать
- Не стоит использовать `for..in` – будут лишние ключи
- Это не массив: нет привычных методов `map`, `filter`, `reduce` и т.д.

```
// преобразование в массив
var elems = document.getElementsByClassName('hello');
var elemsArray = Array.prototype.slice.call(elems);
var elemsArrayES6 = Array.from(elems); // > ES6
```

# `getElementsByTagName('tag')`

- Ищет элементы соответствующего тега
- Есть не только у `document`, но и у любого элемента
- Возвращает коллекцию

# getElementsByTagName('tag')

```
<div>  
  <span>Hello everybody!</span>  
  <span>Aloha</span>  
</div>  
<span>Good bye</span>
```

```
var elems = document.getElementsByTagName('span'); // 3 элемента
```

```
var parent = document.getElementsByTagName('div')[0];
```

```
var elems = parent.getElementsByTagName('span'); // 2 элемента
```

`querySelectorAll('selector')`

- Возвращает коллекцию по заданному CSS селектору
- Псевдоселекторы тоже работают (:hover, :first-child, и т.д.)
- При невалидном селекторе выбрасывает исключение SYNTAX\_ERR

# querySelectorAll('selector')

```
<div class="container">  
  <div>Hello everybody!</div>  
  <div>Aloha</div>  
</div>  
<div class="container">  
  <div>Goodbye</div>  
  <div>Aloha</div>  
</div>
```

```
document.querySelectorAll('.container div:first-child');  
// NodeList [ <div>Hello everybody!</div>, <div>Goodbye</div> ]
```

## querySelector(selector)

- Возвращает первый найденный элемент по заданному селектору
- Выбрасывает исключение SYNTAX\_ERR в случае передачи невалидного селектора

# querySelector('selector')

```
<div class="greeting">Goodbye</div>
<div class="container">
  <div class="greeting">Hello everybody!</div>
  <div class="greeting">Aloha</div>
</div>
```

```
document.querySelector('.greeting');
// <div class="greeting">Goodbye</div>
```

```
document.querySelector('.container .greeting');
// <div class="greeting">Hello everybody!</div>
```



## closest(selector)

- Возвращает ближайший родительский элемент (или сам элемент), который соответствует заданному CSS-селектору
- Возвращает null, если нет элемента, который соответствует селектору
- При невалидном селекторе выбрасывает исключение SYNTAX\_ERR
- Не доступен в IE и старых версиях браузеров

# closest(selector)

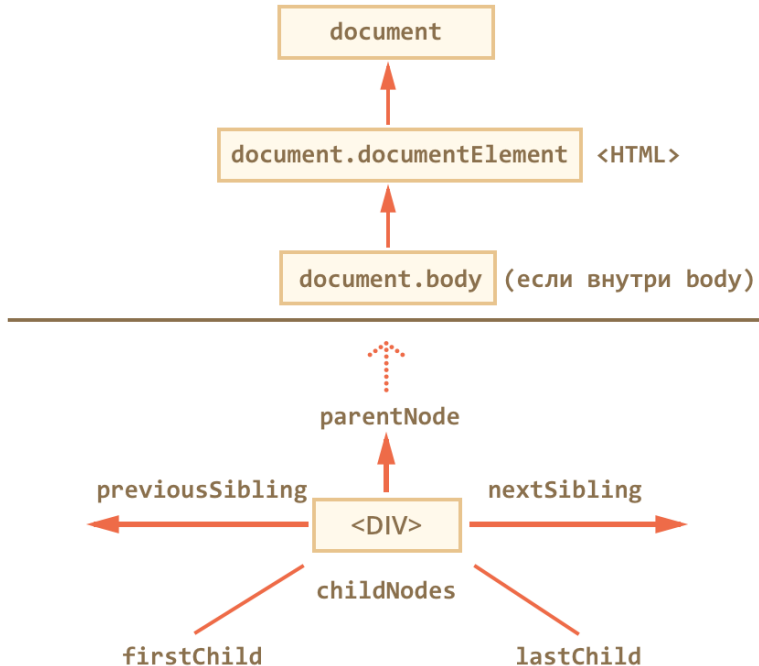
```
<div id="block" title="Я - блок">  
  <a href="#">Я ссылка в никуда</a>  
  
  <a href="http://site.ru">Я ссылка на сайт</a>  
<div>  
  <div id="too"></div>  
</div>  
</div>
```

```
var div = document.querySelector("#too");
```

```
div.closest("#block"); //<div id="block" title="Я - блок">  
div.closest("div"); //Cam <div id="too">  
div.closest("a"); //null
```

# Внутреннее устройство поисковых методов

# Навигация по узлам



# Навигация по узлам

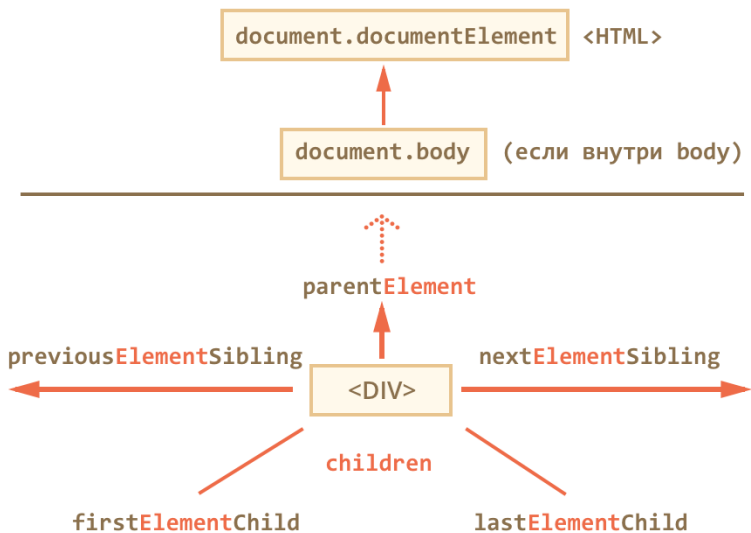
```
<div id="container">  
  <div>Hello</div>  
  Aloha
```

```
  <div>Привет</div>  
</div>
```

```
var container = document.getElementById('container');
```

```
console.log(container.childNodes); // <div>Hello</div> Aloha <div>Привет</div>  
console.log(container.firstChild); // <div>Hello</div>  
console.log(container.lastChild); // <div>Привет</div>  
console.log(container.firstChild.parentNode); // <div id="container">...</div>  
console.log(container.firstChild.nextSibling); // Aloha  
console.log(container.lastChild.previousSibling); // Aloha
```

# Навигация по элементам



# Навигация по элементам

```
<div id="container">  
  hi  
  <div>Hello</div>  
  Aloha  
  <div>Привет</div>  
</div>
```

```
var container = document.getElementById('container');
```

```
console.log(container.children); // <div>Hello</div> <div>Привет</div>  
console.log(container.firstChild); // <div>Hello</div>  
console.log(container.lastElementChild); // <div>Привет</div>  
console.log(container.firstChild.parentNode); // <div id="container">...  
console.log(container.firstChild.nextElementSibling); // <div>Привет</div>  
console.log(container.lastElementChild.previousElementSibling); // <div>Hello</div>
```

# Свойства узлов



## Тип узла – nodeType

- Представлено числом
- Всего типов 12, но используем 2
- 1 – ELEMENT\_NODE
- 3 – TEXT\_NODE
- Остальные

## Тип узла – nodeType

```
<div id="container">  
  <span>Hello everybody!</span>  
  Aloha!  
</div>
```

```
var container = document.getElementById('container');  
console.log(container.firstChild.nodeType); // 1  
console.log(container.lastChild.nodeType); // 3
```

node.tagName

- Возвращает HTML-тег элемента в UPPERCASE
- Только для элемента

node.nodeName

- Для элемента вернет tagName
- Для TEXT\_NODE – строку "#text"

## element.innerHTML

- Содержимое элемента в виде строки
- Доступен на чтение и запись
- Всегда осуществляется перезапись
- Если в innerHTML записывается тег script – он не будет выполнен

## element.outerHTML

- Содержит HTML элемент целиком
- При записи: в DOM исходный элемент замещается на новый элемент
- Изменить outerHTML элемента невозможно

## outerHTML и innerHTML

```
let div = document.querySelector('div');  
console.log(div); // <div>Привет</div>
```

```
div.innerHTML = 'Hi!'  
console.log(div); // <div>Hi!</div>
```

```
div.outerHTML = '<h1>Aloha</h1>'  
console.log(div) // <div>Hi!</div>
```

## data/nodeValue

- Содержимое текстового узла или комментария
- Можно изменять
- Для некоторых типов узлов nodeValue равно null, поэтому для более ожидаемого поведения лучше использовать data



## data/nodeValue

```
<!-- Комментарий -->  
<!DOCTYPE html>
```

```
console.log(document.firstChild.data); // Комментарий  
console.log(document.firstChild.nodeValue); // Комментарий
```

```
console.log(document.lastChild.data); // undefined  
console.log(document.lastChild.nodeValue); // null
```

## Атрибуты

```

```

## Атрибуты

- `element.[get/has/set/remove]Attribute`
- `attributes` – коллекция всех атрибутов элемента

hidden

- Если true – элемент не виден на экране
- Если false – элемент виден

# Классы

- `className` – в виде строки
- `classList` – объект для работы с классами
- `classList.[add/remove]` – добавить/удалить класс
- `classList.toggle` – переключает класс
- `classList.contains` – проверяет есть ли класс

## data-\*

- Позволяют хранить дополнительную информацию в стандартных элементах HTML
- Можно использовать только латинские буквы, дефис (-), двоеточие (:) и подчёркивание (\_)

```
<div data-index-number="123" data-parent="cars"></div>
```

## data-\*

- Все data-\* атрибуты доступны в объекте dataset
- data- отбрасывается, а остаток переводится в camelCase
- Пример: data-user-location будет доступно в element.dataset.userLocation

# Создание узлов

```
//Создание элемента
```

```
var element = document.createElement(tagName);
```

```
//Создание текстового узла
```

```
var textNode = document.createTextNode(text);
```



# Добавление узлов

## appendChild

```
var newLi = document.createElement('li');  
newLi.innerHTML = '3';  
// newLi.appendChild(document.createTextNode('3'))  
  
list.appendChild(newLi);
```

# insertBefore

```
var newLi = document.createElement('li');  
newLi.innerHTML = '3';  
// newLi.appendChild(document.createTextNode('3'))  
  
list.insertBefore(newLi, list.children[1]);
```

## Удаление узлов

- `parentElem.removeChild(elem)` – удаляет `elem` из детей `parentElem`
- `parentElem.replaceChild(newElem, elem)` – удаляет `elem` из детей `parentElem` и вставляет на его место `newElem`

## removeChild

```
var list = document.getElementById('list');  
  
list.removeChild(list.firstChild);
```

## replaceChild

```
var list = document.getElementById('list');  
  
var newLi = document.createElement('li');  
newLi.innerHTML = '3';  
  
list.replaceChild(newLi, list.firstChild);
```

## elem.cloneNode

- Создаёт копию текущего элемента
- Если в качестве аргумента передать true, то создаст глубокую копию элемента, включая атрибуты и подэлементы

# События



## События мыши

- click – на элемент кликнули левой кнопкой мыши
- contextmenu – на элемент кликнули правой кнопкой мыши
- mouseover/mouseout – курсор навели/увели на элемент
- mousedown/mouseup – кнопку мыши нажали (down)/отпустили (up)
- mousemove – при движении курсора мыши над элементом

## События пальцев

- touchstart – элемента коснулись
- touchmove – по элементу провели пальцем
- touchend – касание закончилось и палец убрали
- touchcancel – палец переместился на интерфейс браузера или тач-событие нужно отменить

## Другие события

- `submit` – отправка формы
- `focus` – фокус на элементе
- `keyup/keydown` – печать на клавиатуре
- `resize` – изменение размеров окна
- `transitionend` – завершение css анимации

Ещё больше событий

# Назначение обработчика событий

```
<li onclick="alert('привет');">Щелкни меня</li>
```

```
var li = document.getElementsByTagName('li')[0];  
li.onclick = function () { alert('hello'); }
```

```
li.addEventListener('click', function(event) {  
    alert('aloha');  
});
```

## Удаление обработчика событий

```
var li = document.getElementsByTagName('li')[0];  
li.addEventListener('click', function(event){ alert('привет'); });
```

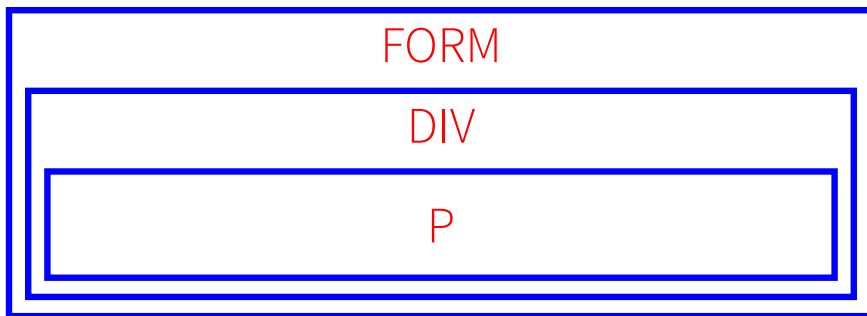
// не работает

```
li.removeEventListener('click', function() { alert('привет'); });
```

```
var li = document.getElementsByTagName('li')[0];  
var onClickHandler = function(event) {  
    alert('привет');  
}  
li.addEventListener('click', onClickHandler);  
li.removeEventListener('click', onClickHandler);
```

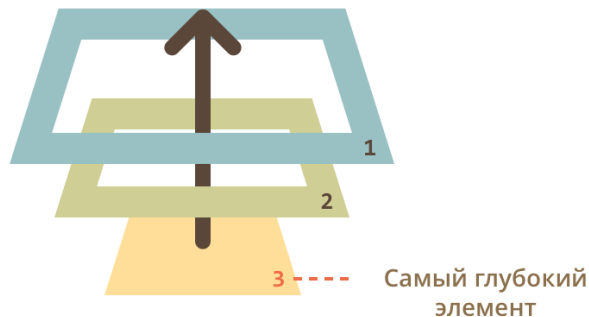
# Всплытие событий

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



# Всплытие событий

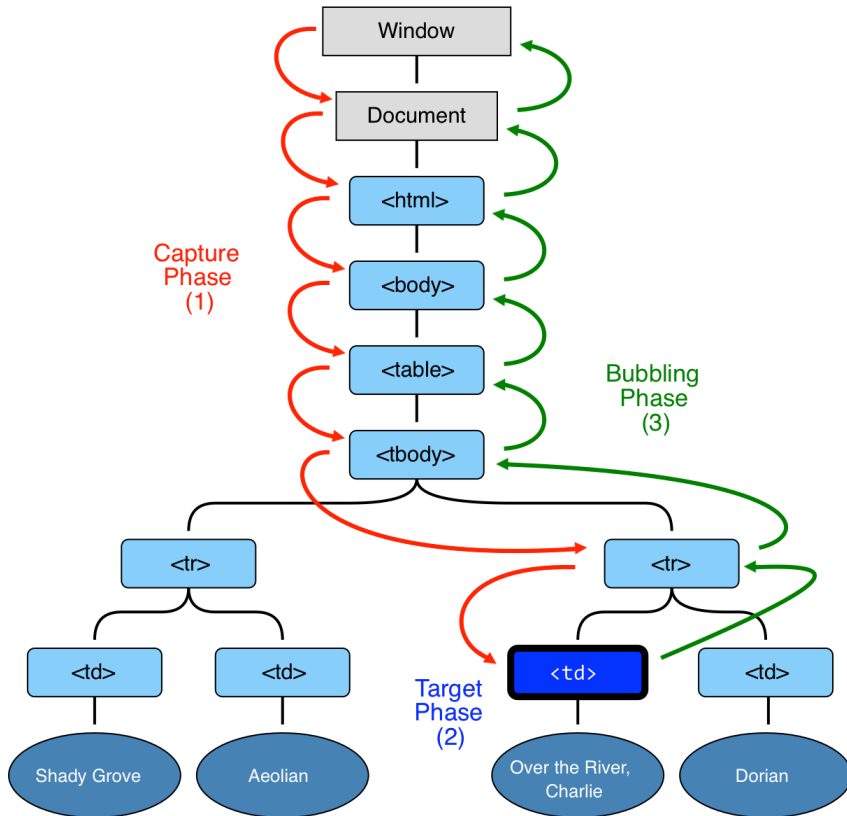
Сначала события срабатывают на **самом вложенном элементе**, затем на его **родителе**, и так далее, вверх до **window**



## Стадии прохода события

- Стадия перехвата (capturing stage) – событие идет сверху вниз
- Стадия цели (target stage) – событие достигло целевого элемента
- Стадия всплытия (bubbling stage) – событие идет снизу вверх





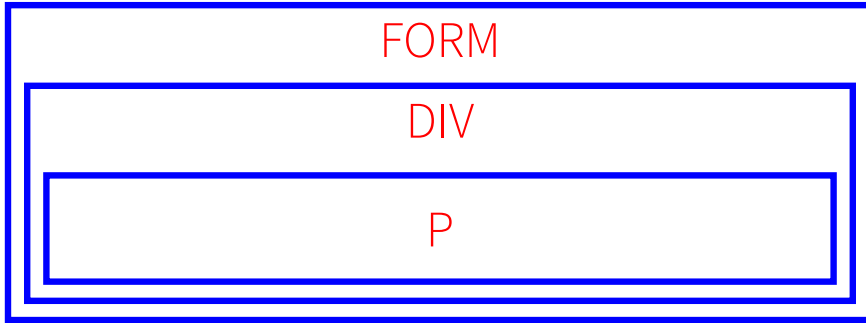
Самый глубокий элемент, который вызывает событие, называется «целевым» или «исходным» элементом и доступен как `event.target`

# event.target

```
<form id="eventform">FORM
  <div>DIV
    <p>P</p>
  </div>
</form>
```

```
eventform.onclick = function(event) {
  alert('target = ', event.target.tagName);
  alert('this = ', this.tagName);
}
```

event.target



## Работа с событием

- `event.preventDefault` – отменяет обработчик по умолчанию
- `event.stopPropagation` – прекращает всплытие
- `event.stopImmediatePropagation` – прекращает всплытие и не выполняет оставшиеся обработчики события

## Перехват события

- on... обработчики не обрабатывают перехват
- `element.addEventListener('event', callback, false)` – обработка на стадии всплытия (дефолтное поведение)
- `element.addEventListener('event', callback, true)` – обработка на стадии перехвата

# Делегирование событий

# Делегирование событий

```
<ul>  
  <li>0</li>  
  <li>1</li>  
  
  <li>2</li>  
</ul>
```

// без делегирования

```
var logger = function(event) {  
  console.log(event.target.innerHTML);  
}
```

```
var liElements = document.getElementsByTagName('li');  
  
for (var i = 0; i < liElements.length; i++) {  
  var li = liElements[i];
```



# Делегирование событий

```
<ul>  
  <li>0</li>  
  <li>1</li>  
  
  <li>2</li>  
</ul>
```

// с делегированием

```
var logger = function(event) {  
  if(event.target.tagName === 'LI') {  
    console.log(event.target.innerHTML);  
  }  
}  
  
var ul = document.getElementsByTagName('ul')[0];  
ul.addEventListener('click', logger);
```

## Преимущества делегирования

- Один обработчик вместо множества
- При добавление новых элементов не нужно добавлять им обработчик

## Полезные ссылки

- [Спецификация DOM](#)
- [caniuse.com](#)
- [learn.javascript.ru](#)
- [MDN](#)

# Вопросы