JS в браузере

Из чего состоит JS на web-страничке?

— `window` - имя глобального объекта в браузере

— (1) Глобальные объекты ядра JS: `Number`, `JSON`, etc

— (2) `document` - модель HTML-страницы: Document Object Model (DOM)

— (3) Другие "подсистемы" браузера - Browser Object Model (BOM)

`navigator` - сведения о браузере

`XMLHttpRequest/fetch` - асинхронные HTTP-запросы (AJAX)

`location`, `history` - адресная строка и навигация

`localStorage/sessionStorage` - хранение данных

`Worker()` - выполнение работы в фоновом режиме

`Notification()` - нотификации

`navigator.geolocation` - геолокация

etc, etc, etc

Как добавить JS-код на страницу?

— Отдельный файл с кодом:

```
<script type="javascript" src="path/to/file.js"></script>
```

— JS-код внутри тегов script

```
<script type="javascript">[JS-код]</script>
```

— JS-код внутри HTML-атрибутов для обработчиков событий

```
<input type="text" onclick="[JS-код]">
```

— URL-адрес с псевдопротоколом `javascript:`

```
<a href="javascript:[JS-код]">link title</a>
```

Очередность загрузки и выполнения JS в браузере

— (1 этап) Загрузка и выполнение кода в `<script></script>`

  Все файлы скачиваются параллельно и выполняются по порядку

  Код в каждом скрипте выполняется последовательно

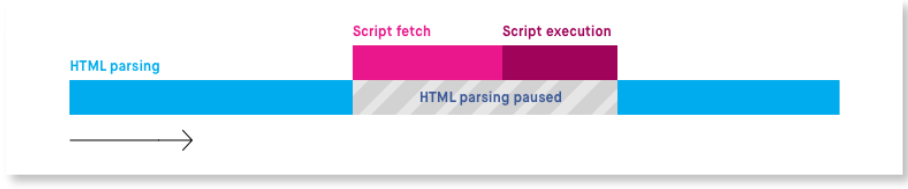
  Рендеринг страницы при этом блокируется

— (2 этап): Асинхронный и управляемый событиями

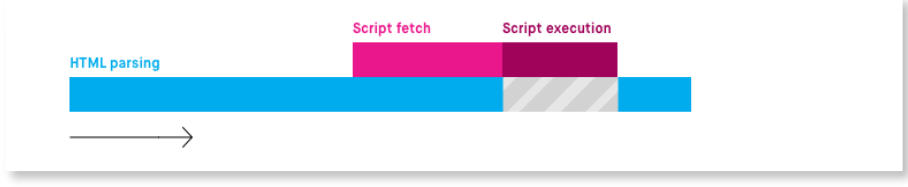
  Браузер вызывает ранее объявленные обработчики в ответ на возникающие события

Script async, defer

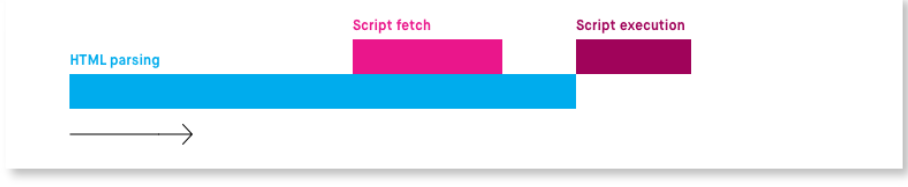
```
<script type="javascript" src="..."></script>
```



```
<script type="javascript" src="..." async></script>
```
! Очередность выполнения не гарантирована !



```
<script type="javascript" src="..." defer></script>
```

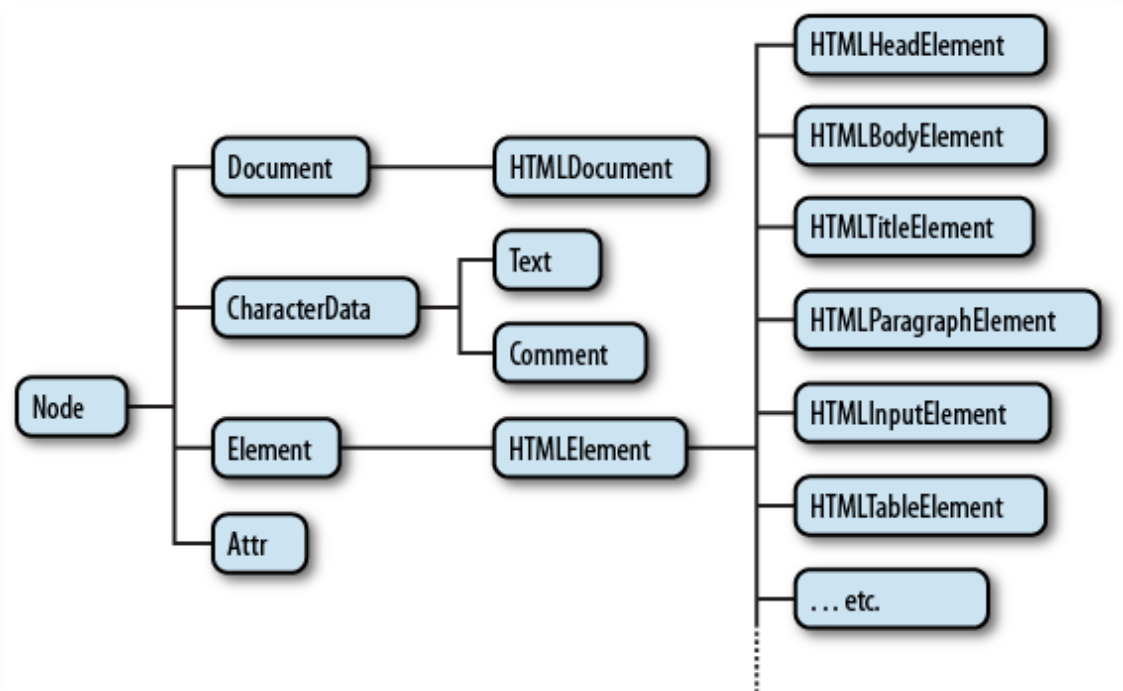


Document Object Model (DOM)

— DOM - представление страницы в виде дерева узлов (DOM Node Tree)

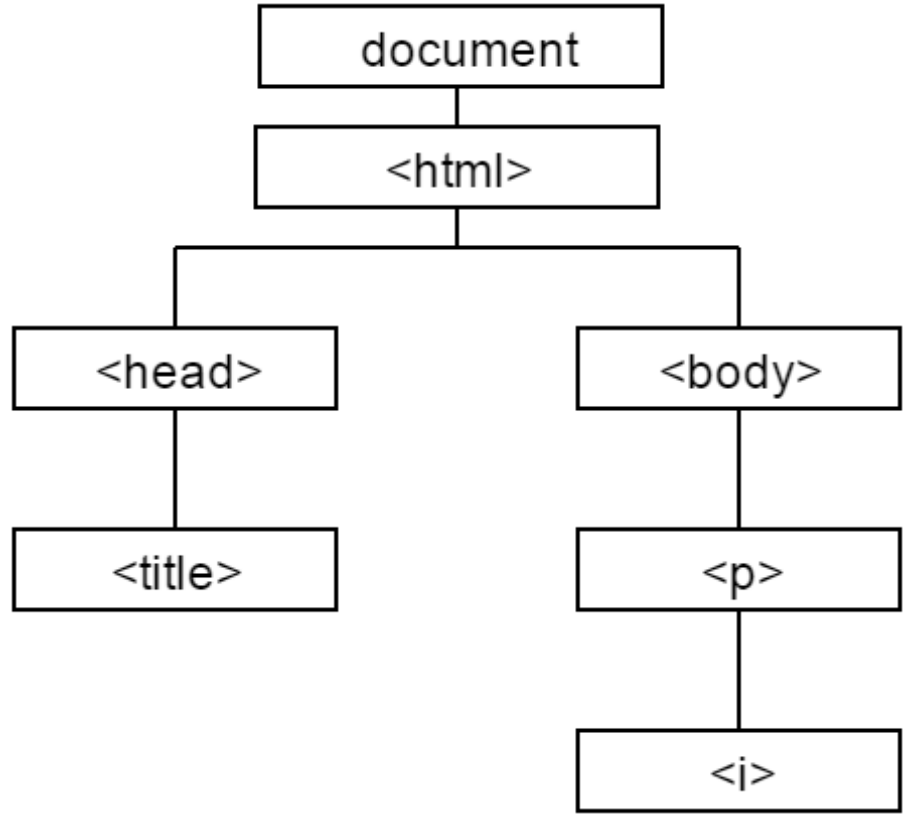
— Теги, текст, комментарии, doctype и сам документ - узлы DOM

— Можно менять страницу, манипулируя объектами DOM из JS

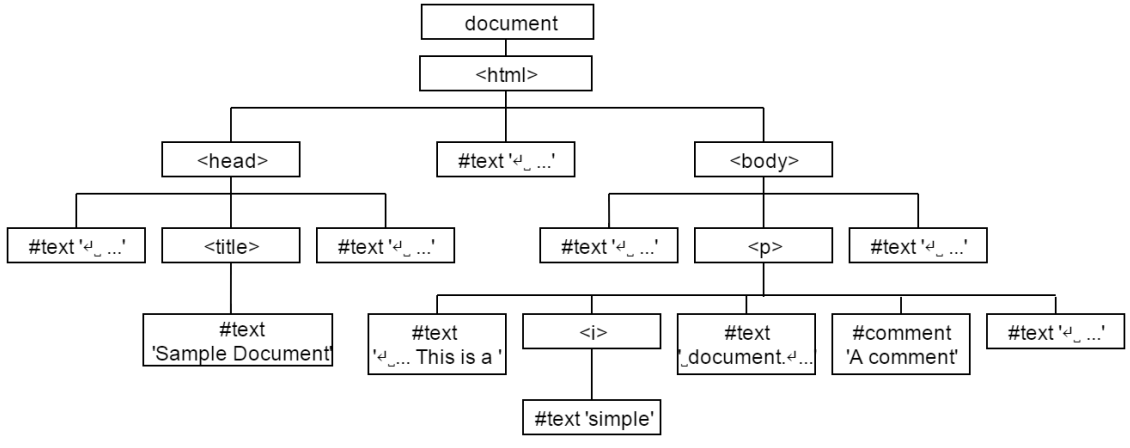


Структура DOM-дерева

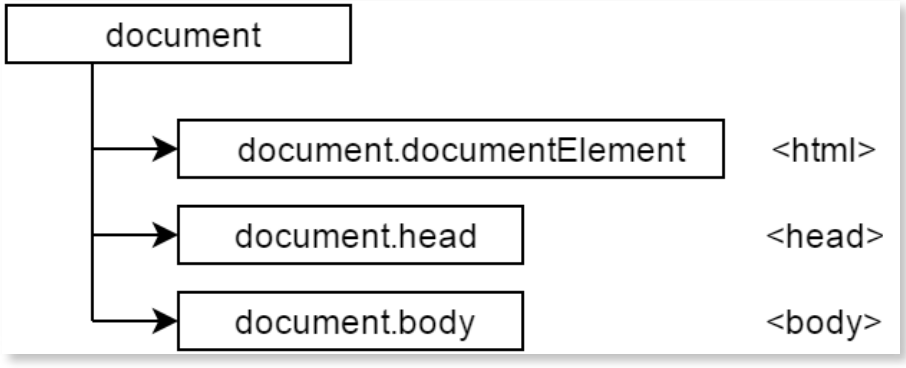
```
<html>
  <head>
    <title>Sample Document</title>
  </head>
  <body>
    <p>
      This is a <i>simple</i> document.
      <!-- This is a comment -->
    </p>
  </body>
</html>
```

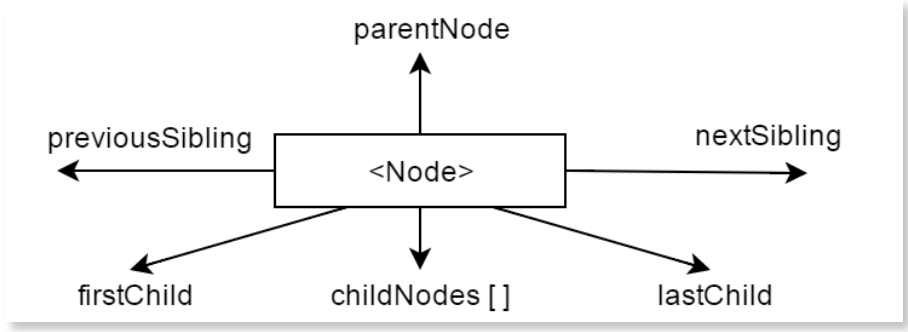


DOM-дерево с текстовыми узлами и комментариями



Навигация по DOM-дереву





— Descendants - узлы, лежащие внутри данного

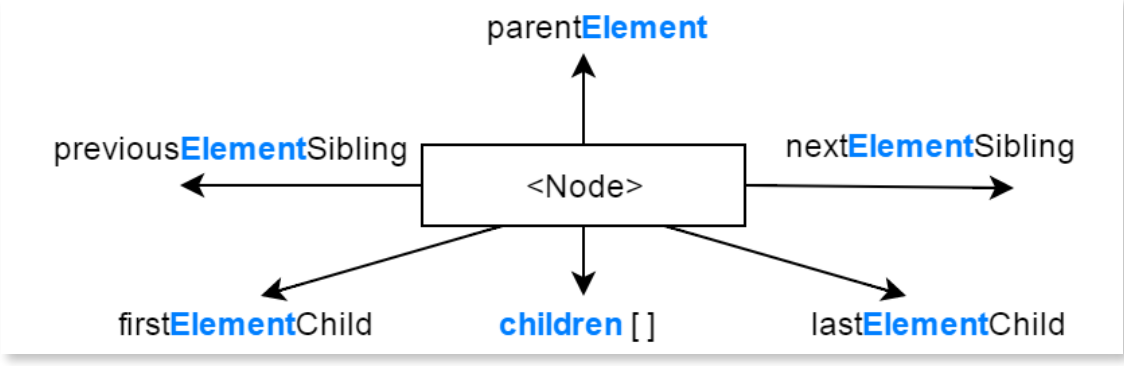
— Children - узлы-непосредственные потомки данного

— Siblings - узлы, лежащие на одном уровне с данным

— `childNodes` - коллекция дочерних узлов (array-like)

— В случае отсутствия узла - `null`

Навигация по тегам DOM-дерева (без текста/комментариев)



— `children` - коллекция дочерних тегов (array-like)

Навигация по DOM-дереву без текстовых узлов

```
<!doctype html>
<html>
    <head>
        <title>Sample Document</title>
    </head>
    <body>
        <p>
            This is a <i>simple</i> document.
            <!-- This is a comment -->
        </p>
    </body>
</html>
```

```
let p = window.document.querySelector("p"); // HTMLParagraphElement (<p> tag)
p.parentElement;                            // HTMLBodyElement (<body> tag)

p.previousElementSibling; // null
p.firstElementChild;      // HTMLElement (<i> tag)
p.lastElementChild;       // HTMLElement (<i> tag)
p.children;               // HTMLCollection [ i ]

p.parentElement.previousElementSibling; // HTMLHeadElement (<head> tag)

document.childNodes;         // NodeList [DocumentType, HTMLHtmlElement]
document.children;           // HTMLCollection [HTMLHtmlElement]
```

Поиск в DOM-дереве

Метод	Ищет по..	Контекст?	Возвращает
`getElementById`	id attr	-	Element/null
`querySelector`	CSS селектор	✔	Element/null
`querySelectorAll`	CSS селектор	✔	collection
`closest`	CSS селектор	✔	Element/null

```html
<ul class="list">
  <li id="my-id">
    <p>1</p>
  </li>

  <li>
    <p name="my-p">2</p>
  </li>

  <li class="list">
    <p>3</p>
  </li>
</ul>
```

```js
document.getElementById("my-id");      // HTMLLIElement

document.querySelector("ul li#my-id p").textContent;  // "1"
document.querySelector('p[name="my-p"]').textContent; // "2"
document.querySelector("li.list p").textContent;      // "3"

// Коллекция - не массив, но у нее есть итераторы!
for (el of document.querySelectorAll("li p"))
    console.log(el.textContent); // "1", "2", "3"

let p = document.querySelector("li.list p")
p.closest(".list");  // HTMLLIElement (li.list)
```

Свойства узлов DOM

— `nodeType` - тип узла (`ELEMENT_NODE` = 1, `TEXT_NODE` = 3 etc)

— `nodeName` - имя тега для Element, описание узла для других типов (`"#text"`, `"#comment"` etc)

— `tagName` - имя тега для Element, и отсутствует для остальных

— `innerHTML` - HTML-содержимое элемента в виде строки

— `outerHTML` - HTML-содержимое, включая обрамляющие теги

— `innerText` - текстовое содержимое элемента

— `value`, `id`, `href` etc-etc-etc - HTML атрибуты

Атрибуты и свойства DOM-объектов

— Атрибуты HTML и свойства DOM часто (но не всегда!) отображаются друг в друга

— Свойства DOM-объектов - обычные свойства JS

— Атрибуты HTML - записаны как атрибуты тега в HTML тексте

	Properties	**Attributes**
Тип данных	Любой	String
Case	sensitive	insensitive
innerHTML	не видны	видны

Как работать	`prop in elem;`
	`elem.prop;`
	`elem.prop = value;`
	`delete elem.prop;`
		`elem.hasAttribute(attr);`
		`elem.getAttribute(attr);`
		`elem.setAttribute(attr, value);`
		`elem.removeAttribute(attr);`

Props & attrs: исключений больше чем правил

— Стандартные атрибуты HTML становятся свойствами DOM

— Нестандартные атрибуты не становятся свойствами

— `id`: свойство <=> атрибут (меняет атрибут и наоборот)

— `checked`: свойство `true/false`, в атрибуте важно его наличие

— `class` <=> свойство `className`

— `class` <=> объект `classList` (`add/remove/toggle/contains`)

```
    <input type="checkbox" class="my-class" myAttribute="myValue" checked />
```

```
let input = document.querySelector("input");
input.className;  // "my-class"
input.type;       // "checkbox"
input.checked;    // true

[...input.classList];                    // ["my-class"]
input.classList.add("my-class-2");
input.className;                         // "my-class my-class-2"
```

data-атрибуты

— Атрибуты вида `data-*` зарезервированы в HTML5 для нужд разработчиков

— Ими можно управлять из JS с помощью свойства `dataset`

— Преобразование имен: `data-attr-name` <=> `el.dataset.attrName`

— Можно хранить только строки

```
<div id="elem" data-order-id="123" data-order-name="pizza"></div>
```

```
let div = document.getElementById("elem");
div.dataset.orderId;          // -> "123"
div.dataset.orderId = "999"; // изменили значение свойства/атрибута
div.dataset.orderName;        // -> "pizza"
```

Шаблонизация

— Для создания больших объемов HTML-разметки JS API слишком громоздкий и низкоуровневый

— Для таких задач используют шаблоны и шаблонизаторы

— Самый простой шаблон - строка в backtics

```
function createOrder(orderId, orderValue) {
    return `<div data-id="${orderId}">${orderValue}</div>`;
}

let html = createOrder(1, "pizza") + createOrder(2, "soup");
document.body.innerHTML = html;
```

События

— Событие - это "нечто", происходящее с DOM элементами

— Каждое событие имеет имя и элемент-источник (target)

— Для реакции на событие к элементу добавляют обработчики событий

— Список событий которые могут возникнуть, зависят от источника

```
<div onclick="alert("hi!");"></div>
```

```
let div = document.getElementById("#myDiv");
div.onclick = () => alert("hi!");
```

```
let div = document.getElementById("myDiv"),
    clickHandler = () => alert("hi!");

div.addEventListener("click", clickHandler);
div.removeEventListener("click", clickHandler);
```

Порядок обработки событий

— При возникновении события вызовутся все прикрепленные обработчики

— Обработчику передается объект события (класс `Event` и производные)

— `this` в обработчике события указывает на элемент, к которому прикреплен обработчик

```
<div id="example" onclick="console.log(event.type); console.log('html handler')">
    example div
</div>
```

```
let div = document.getElementById("example");
div.addEventListener("click", function(e) { console.log("handler 1"); });
div.addEventListener("click", function(e) { console.log(this.innerText); });
div.addEventListener("change", function(e) { console.log("change event"); });

// => "click"
// => "html handler"
// => "handler 1"
// => "example div"
```

Всплытие событий

— Всплытие события - вызов всех обработчиков события, начиная с элемента-инициатора вверх по дереву DOM - до `document` и `window`

— `e.stopPropagation()` - отмена всплытия события после выполнения всех обработчиков на текущем элементе

— `e.stopImmediatePropagation()` - отмена всплытия события, без вызова остальных обработчиков

— `e.preventDefault()` - отмена действия по умолчанию

```
<div onclick="console.log("div 1")">1
  <div onclick="console.log("div 2"); event.stopPropagation();">2
    <div onclick="console.log("div 3")">3
        <div onclick="console.log("div 4")">4</div>
    </div>
  </div>
</div>
// -> "div 4"
// -> "div 3"
// -> "div 2"
```

Делегирование обработки событий

— Объявление обработчиков требует времени и памяти

— Если ваша разметка при этом генерируется динамически - это дополнительный уровень сложности

```
<ul id="items">
    <li data-item-id="1">Товар 1</li>
    <li data-item-id="2">Товар 2</li>
    <li data-item-id="3">Товар 3</li>
</ul>
```

```
let ul = document.getElementById("items");

for (let li of ul.children) {
    // Для каждого li - свой обработчик
    li.addEventListener("click", function(e) {
        addToBasket(li.dataset.itemId);
    });
}
```

Делегирование обработки событий

— Внутри обработчика `event.currentTarget === this`

— `e.target` указывает на элемент-инициатор события

```
<ul id="items-list">
    <li data-item-id="1">Товар 1</li>
    <li data-item-id="2">Товар 2</li>
    <li data-item-id="3">Товар 3</li>
</ul>
```

```
let ul = document.getElementById("items");

// Один обработчик для всех li
let basketHandler = function(e) {
    addToBasket(this.dataset.itemId);
};

for (let li of ul.children) {
    li.addEventListener(
        "click", basketHandler);
}
```

```
let ul = document.getElementById("items");

// Один обработчик для всех li
let basketHandler = function(e) {
    addToBasket(e.target.dataset.itemId);
};

// Обработчик на родительском элементе
ul.addEventListener(
    "click", basketHandler);
```

Загрузка документа: `load` и `DOMContentLoaded`

— `DOMContentLoaded` - документ загружен и готов к работе

— `load` - документ загружен вместе с остальными файлами (изображения, css, iframes etc.)

```
<html>
<head>
    <script type="text/javascript">
        // ничего не найдет: документ еще не загружен
        document.getElementById("div1");

        window.addEventListener("DOMContentLoaded", function() {
            // сработает: документ полностью загружен и готов к работе
            document.getElementById("div1");
        });
    </script>
</head>
<body>
    <div id="div1"></div>
    <script type="text/javascript">
        // сработает: документ еще не загружен полностью, но #div1 уже доступен
        document.getElementById("div1");
    </script>
</body>
</html>
```