JS в браузере

Из чего состоит JS на web-страничке?

- Имя глобального объекта JS в браузере - `window`
- (1) Глобальные объекты ядра JS: `Number`, `parseInt`, 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` - геолокоция
  - etc, etc, etc

Как добавить JS-код на web-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, содержащий спецификатор псевдопротокола `javascript:`

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

Какой из способов добавить JS лучше всех?

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

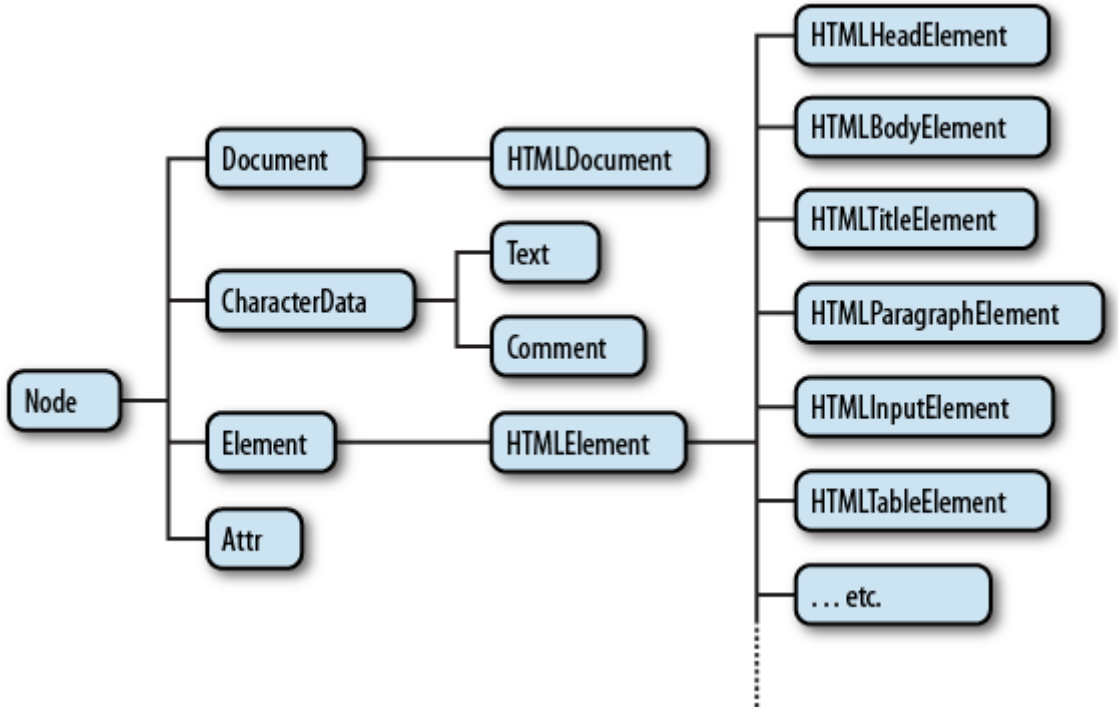
- Разделение кода и представления
- Повторное использование JS кода
- Кэширование JS-кода
- Возможность запустить код не только в браузере
- Возможность указать URL с другого сайта

Очередность загрузки и выполнения программы

- (1 этап) Загрузка и выполнение кода в `<script></script>`
  - Все файлы скачиваются параллельно и выполняются в порядке следования
  - Код в каждом скрипте выполняется последовательно
  - Рендеринг страницы при этом блокируется
  - `<script src="..." async>` - скрипт выполнится как только загрузится
  - `<script src="..." defer>` - скрипты выполятся в порядке следования после окончания рендера страницы
- (2 этап): Асинхронный и управляемый событиями
  - Браузер вызывает ранее объявленные обработчики в ответ на возникающие события

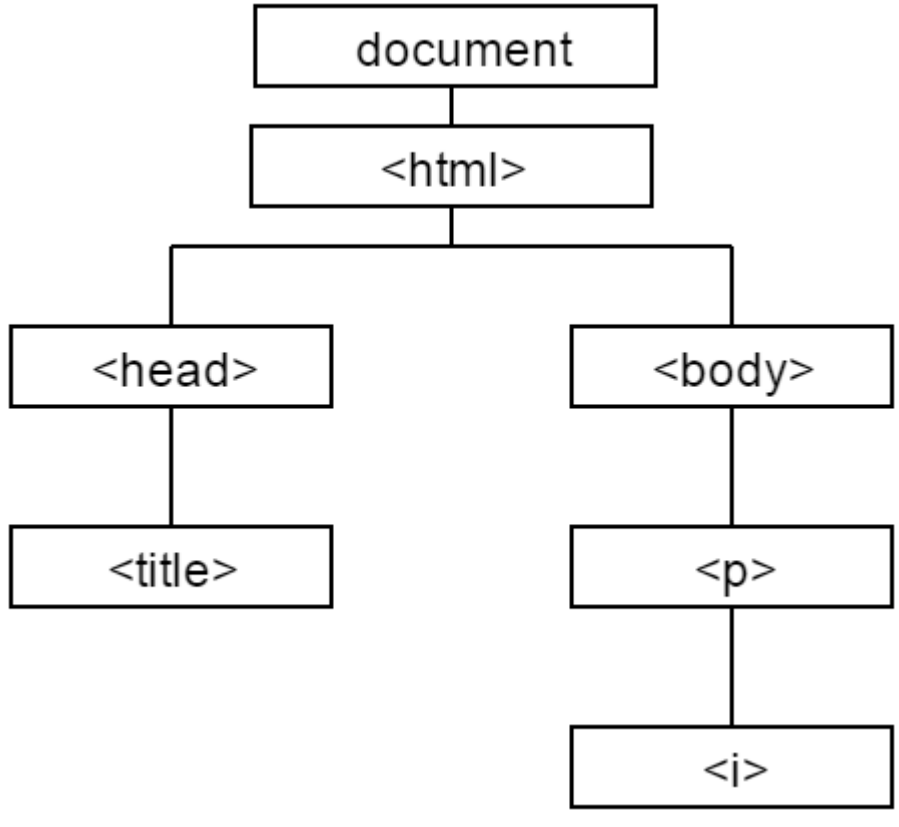
Document Object Model (DOM)

- DOM - представление страницы в виде дерева узлов (DOM Node Tree)
- каждый тег, текст, комментарий, doctype и сам документ - это узел DOM
- любой узел DOM - объект определенного класса с методами и свойствами
- манипулируя объектами DOM из JS, можно изменять документ в браузере

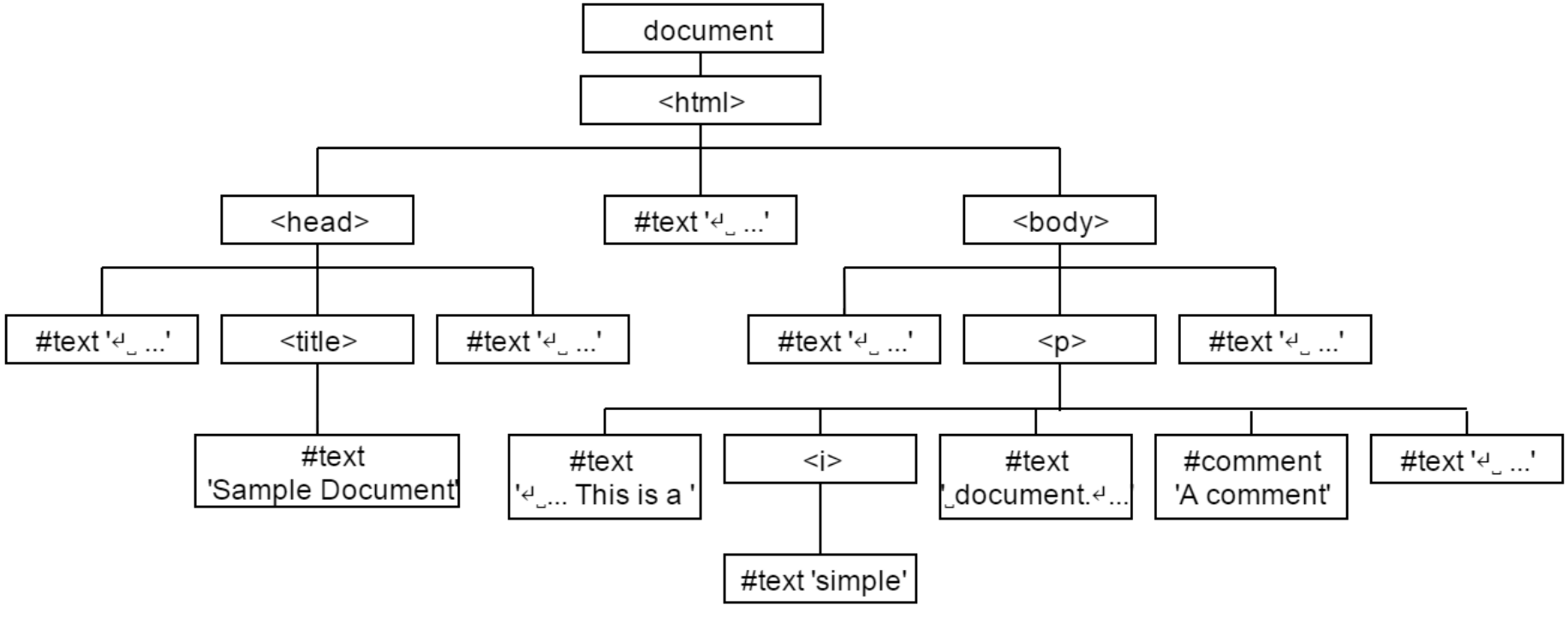


Структура DOM-дерева

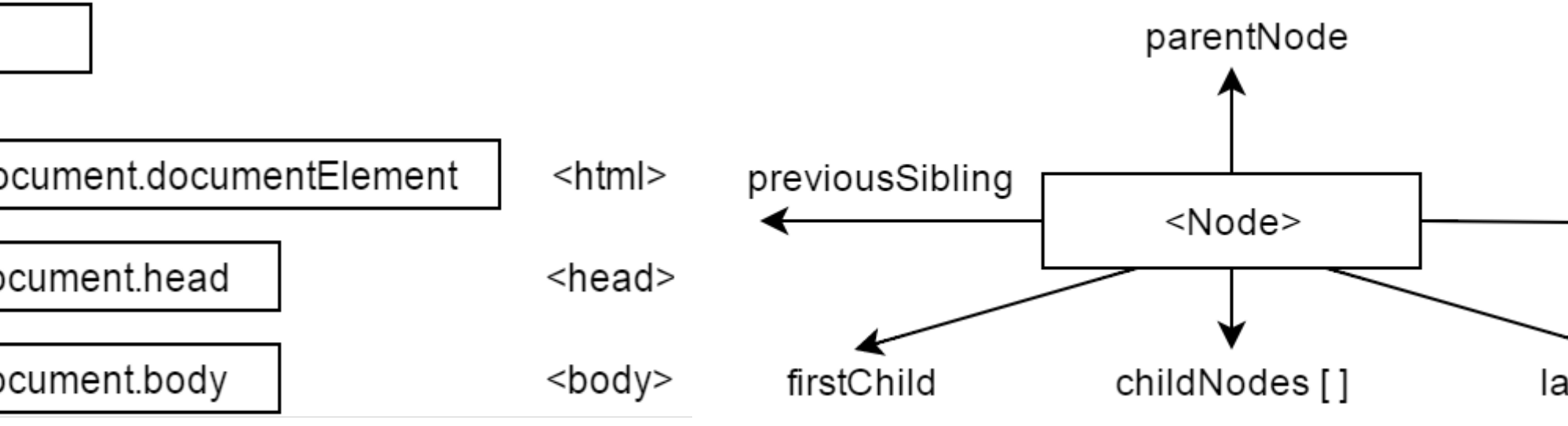
```
<html>
    <head>
        <title>Sample Document</title>
    </head>
    <body>
        <p>
            This is a <i>simple</i> document.
            <!-- This is a comment -->
        </p>
    </body>
</html>
```



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` - array-like свойство, коллекция узлов (класс NodeList)
- В случае отсутствия узла ссылка указывает на null

Навигация по DOM-дереву

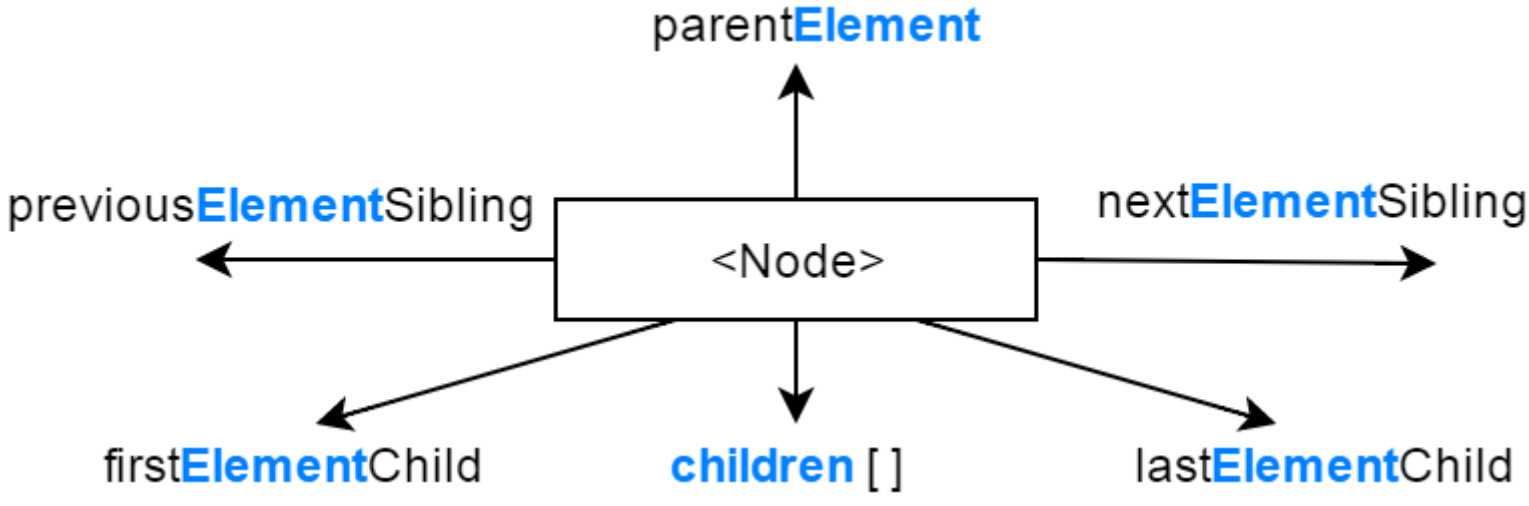
```
<html>
    <head>
        <title>Sample Document</title>
    </head>
    <body>
        <p>
            This is a <i>simple</i> document.
            <!-- This is a comment -->
        </p>
    </body>
</html>
```

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

p.previousSibling;              // Text
p.previousSibling.nodeName;     // '#text'
p.previousSibling.nodeType;     // 3
p.previousSibling.textContent; // '↵␣␣␣␣'

p.nextSibling;     // Text ('↵↵')
p.firstChild;      // Text ('↵␣␣␣␣␣␣␣␣This is a␣')
p.lastChild;       // Text ('↵␣␣␣␣')
p.childNodes;      // NodeList [Text, HTMLElement (<i>), Text, Comment, Text]
```

Навигация по элементам DOM-дерева (без текстовых узлов)



- `parentNode/parentElement` отличаются только для самых верхних элементов (`<html>` и `doctype`):
- `document.documentElement.parentNode` => `document`
- `document.documentElement.parentElement` => `null`
- `children` - array-like свойство (класс `HTMLCollection`)

Навигация по DOM-дереву без текстовых узлов

```html
<!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.nextElementSibling;     // 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?	Возвр
`getElementById`	id attr	-	Elemer
`getElementsByName`	name attr	-	live col
`getElementsByTagName`	имя тега	✔	live col
`getElementsByClassName`	class attr	✔	live col

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

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

  <li class="list">
```

```
document.getElementById('my-id');      // HTMLLIElement
document.getElementsByName('my-p');  // NodeList [1]
document.getElementsByName('no-name'); // NodeList [0]
document.getElementsByTagName('p');    // HTMLCollection [3
let li = document.getElementById('my-id');
li.getElementsByTagName('p');          // HTMLCollection [1

let clt = document.getElementsByClassName("list");
for (let el of clt) console.log(el.tagName); // 'UL' 'LI'
```

```
    <p>3</p>
  </li>
</ul>
```

```
document.body.children[0].children[2].remove();
clt.length; // 1 (!)
```

Поиск по CSS селектору

Метод	Ищет по..	Контекст?	Возвраща
`querySelector`	CSS селектор	✔	Element/nu
`querySelectorAll`	CSS селектор	✔	static collection
`closest`	CSS селектор	✔	Element/nu

```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>
```

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

for (el of document.querySelectorAll('li p'))
    console.log(el.textContent); // '1', '2', '3'

let li = document.querySelector('li.list');
let p = li.querySelector('ul p'); // HTMLParagraphElement (!
p.textContent;  // '3'

p.closest('.list');  // HTMLLIElement (li.list)
```

```
</ul>
<p>finish</p>
```

Поиск в DOM-дереве: особенности

- На практике нет разницы между `NodeList` и `HTMLCollection`:
  - Оба типа имеют свойство `length` (перебор через `for (...)`)
  - Оба типа имеют default итератор (перебор значений через `for ... of`)
- "Live" коллекции изменяются автоматически при изменении DOM-дерева
- "Static" коллекции обладают более привычным поведением
- `querySelector` и `querySelectorAll` ищут во всем документе, а затем отбрасывают все элементы, находящиеся вне контекста

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
- `data` - содержимое узлов - не-элементов
- `textContent` - конкатенация текста во внутренних узлах элемента
- `value`, `id`, `href` etc-etc-etc - зависит от типа узла

Свойства узлов DOM

```
<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

```
let ul = document.querySelector('ul'),
    li = ul.children[1],
    txt = li.childNodes[0];
```

```
ul.nodeType;     // 1
ul.nodeName;     // 'UL'
ul.tagName;      // 'UL'
ul.data;         // undefined
ul.textContent;
// '↵␣␣first↵␣␣second↵␣␣third↵'
ul.innerHTML;
// '↵␣␣<li>first ... third</li>↵
ul.outerHTML;
// '<ul>↵␣␣ ... ↵</ul>'
```

```
li.nodeType;     // 1
li.nodeName;     // 'LI'
li.tagName;      // 'LI'
li.data;         // undefined
li.textContent; // 'second'

li.innerHTML; //'second'

li.outerHTML; //'<li>second</li>
```

```
txt.nodeType;    // 3
txt.nodeName;    // '#text'
txt.tagName;     // undefined
txt.data;        // 'second'
txt.textContent; // 'second'

txt.innerHTML;   // undefined

txt.outerHTML;   // undefined
```

Атрибуты и свойства DOM-объектов

- Атрибуты HTML и свойства DOM-объектов часто, (но не всегда!) отображаются друг в друга
- Свойства DOM-объектов - это обычные свойства JS
- Атрибуты HTML - то что записано атрибутом элемента в тексте HTML

	Properties	**Attributes**
Тип данных	Любой	Строка
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

- Большинство стандартных атрибутов HTML становятся свойствами DOM
- Нестандартные атрибуты не становятся свойствами
- `id`: свойство <=> атрибут (меняет атрибут и наоборот)
- `href`: свойство <=> атрибут, но свойство должно быть полной ссылкой
- `checked`: свойство может быть `true/false`, в атрибуте важно только его наличие; атрибут устанавливает только начальное значение свойства
- `class` <=> свойство `className` (`for <=> htmlFor`)
- `class` <=> объект `classList` (`add/remove/toggle/contains`)

Props & attrs: исключений больше чем правил

```
<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.myAttribute;                  // undefined
input.getAttribute('myAttribute'); // 'myValue'

input.checked = false;          // галочка снята
input.hasAttribute('checked'); // true, связь между атрибутом и свойством разорвана

[...input.classList];                    // ['my-class']
input.classList.add('my-class-2');
input.className;                         // 'my-class my-class-2'
input.classList.toggle('my-class');
[...input.classList];                    // ['my-class-2']
input.classList.contains('my-class-2'); // true
```

data-атрибуты

- Атрибуты вида data-* зарезервированы в HTML5 для нужд пользователя
- 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"
```

Изменение структуры DOM-дерева

- `document.createElement(tagName)`- создать новый элемент
- `document.createTextNode(text)`- создать новый текстовый узел
- `root.appendChild(node)`- добавить узел в конец потомков `root`
- `root.replaceChild(newNode, oldNode)`- заместить узел другим
- `root.removeChild(node)`- удалить узел
- `node.remove()` - удалить без ссылки на родительский элемент
- `node.cloneNode(deep)`- клонировать узел (deep/shallow)
- `root.insertBefore(node, nextNode)`- добавить узел в список потомков `root` перед `nextNode`

Изменение структуры DOM-дерева

```
<ul id="list">
    <li>0</li>
    <li>1</li>
    <li>2</li>
</ul>
```

```
let ul = document.getElementById('list'),
    li, txt;

li = document.createElement('li');
txt = document.createTextNode('-1');
li.appendChild(txt);                  // создали li с текстом -1
ul.insertBefore(li, ul.children[0]); // добавили его в начало списка

li = li.cloneNode(false);
li.innerHTML = '4';
ul.appendChild(li); // вставили li с текстом 4 в конец
ul.replaceChild(li, ul.children[1]); // удалили второй li и вставили на ее место последний
```

Оптимизация работы с DOM

```
let ul = document.createElement('ul');
document.body.appendChild(ul);
for (...) ul.appendChild(li);
```

```
let ul = document.createElement('ul');
for(...) ul.appendChild(li);
document.body.appendChild(ul);
```

- поочередная вставка элементов в DOM - медленное действие
- лучше собирать элементы в памяти, и вставлять их в DOM один раз
- `innerHTML=...` обновляет HTML элемента целиком
- `elem.insertAdjacentHTML(where, html)`
- `documentFragment`- специальный вид узла
  - `document.createDocumentFragment()` - создать новый фрагмент
  - `documentFragment` он может только хранить другие элементы
  - при вставке в DOM вместо фрагмента вставляется его содержимое

Мультивставка

- `elem.append(...nodes)` – вставить список узлов в конец elem
- `elem.prepend(...nodes)` – вставить список узлов в начало elem
- `elem.after(...nodes)` – вставить список узлов после узла elem
- `elem.before(...nodes)` – вставить список узлов перед узлом elem
- `elem.replaceWith(...nodes)` – вставить список узлов вместо elem

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.insertAdjacentHTML('beforeEnd', html);
```

LoDash templates

```
<!-- Любой код в теге script с нестандартным mime-type будет невидим,
     и не будет запущен как JS -->
<script type="text/template" id="menu-template">
    <div class="menu">
      <span class="title"><%-title%></span>
      <ul>
        <% for (item of items) { %>
        <li><%-item%></li>
        <% } %>
      </ul>
    </div>
</script>

let tmplString = document.getElementById('menu-template').innerHTML,
    tmpl = _.template(tmplString);
    // tmpl теперь "скомпилированный шаблон": функция, генерирующая HTML

let html = tmpl({
        title: 'myMenu',
        items: ['item1', 'item2', 'item3']
    });

document.body.innerHTML = html;
```