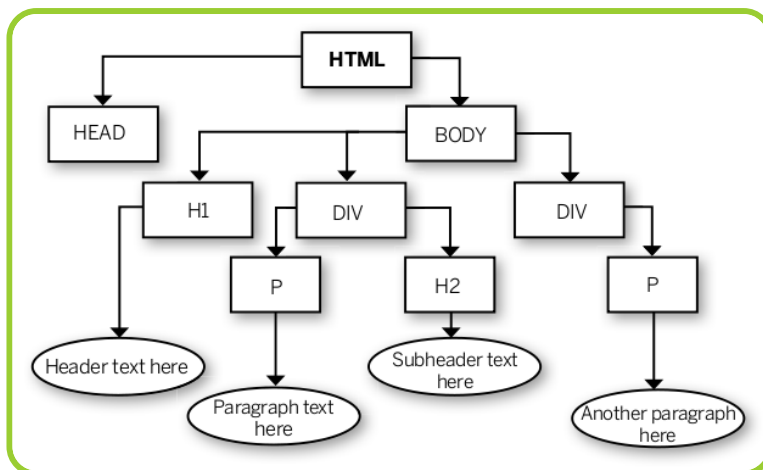


# Объектная модель документа (DOM)

## Настоящая мощь динамических веб страниц

### DOM

- Динамическая модель документа это **API для HTML и XML** документов
  - Представляет **структуру** документа
  - Разработчики могут **модифицировать содержание и пользовательский интерфейс (UI)** на веб странице



- DOM содержит объекты для манипуляции ими на веб странице
  - Все свойства, методы и события **размещенные внутри объектов**
  - Эти объекты доступны через **языки программирования и скрипты**
- Как использовать DOM и HTML страницу?
  - Пишется JavaScript для взаимодействия с DOM
    - JavaScript использует DOM API (встроенное в любой браузер)

### DOM API

- DOM API **содержит объекты и методы для интерактивного взаимодействия на HTML странице**
  - Может добавлять и удалять HTML документы
  - Может динамически применять стили
  - Может добавлять и удалять HTML атрибуты
- DOM представляет объекты и их методы для HTML документов и их свойств
  - `document.documentElement` это `<html>`

- `document.body` тело страницы, в котором содержится контент

## DOM объекты

- Каждый из HTML элементов имеет ассоциированный с ним DOM объект
  - `HTMLLIElement` представляет `<li>`
  - `HTMLAudioElement` представляет `<audio>`
- Каждый из этих объектов имеет присоединенные свойства
  - `HTMLAnchorElement` имеет `href` свойство
    - `HTMLImageElement` имеет `src` свойство
- `document` объект является специальным
  - он является входной точкой в DOM API

## HTML элементы

- HTML элементы имеют свойства которые соответствуют атрибутам
  - `id`, `className`, `draggable`, `style`, `onclick`, и.т.д.
- Различные HTML имеют свои **специфические атрибуты**
  - `HTMLImageElement` имеет свойство `src`
  - `HTMLInputElement` имеет значение `value`
  - `HTMLAnchorElement` имеет свойство `href`
- HTML элементы имеют свойства описывающие контент
  - `innerHTML`
    - Возвращает строку с содержимым элемента без HTML элемента
  - `outerHTML`
    - Возвращает строку с содержимым элемента вместе с элементом
  - `innerText` / `textContent`
    - Возвращает строку с текстом одержимого элемента без тегов

## Выделение DOM элементов

```
<!DOCTYPE html>
▼<html lang="bg" style>
  ►<head>...</head>
  ▼<body>
    ▼<div id="Wrapper">
      ►<header id="MainHeader">...</header>
      ►<nav class="kendo-style-black">...</nav>
      ▼<div id="MainContainer">
        ▼<div id="ImportantMessages">
          ►<div class="importantMessageWarning">...</div>
        </div>
        ►<section id="MainContent">...</section>
      </div>
      ►<footer id="MainFooter">...</footer>
    </div>
  </body>
</html>
```

- HTML могут быть найдены и кешированы в переменные, используя DOM API

- Выделить одиночный элемент

```
var header = document.getElementById('header');  
var nav = document.querySelector('#main-nav');
```

- Выделить коллекцию элементов

```
var inputs = document.getElementsByTagName('li');  
var radiosGroup = document.getElementsByName('genders');  
var header = document.querySelectorAll('#main-nav li');
```

- Использовать предопределенную коллекцию элементов

```
var links = document.links;  
var forms = document.forms;
```

## Используем `getElementsBy` Методы

- DOM API содержит методы для выделения элементов, основанные на их параметрах
  - `getElementById(id)` :
    - Возвращает **одиночный элемент** или `null`

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

- `getElementsByClassName(className)` :
  - Возвращает **коллекцию элементов**

```
var posts = document.getElementsByClassName('post-item');
```

- `getElementsByTagName(tagName)` ;
  - Возвращает **коллекцию элементов**

```
var sidebars = document.getElementsByTagName('sidebar');
```

- `getElementsByName(name)` ;
  - Возвращает **коллекцию элементов**

```
var gendersGroup = document.getElementsByName('genders');
```

## Использование метода `querySelector`

- DOM API имеет методы которые используют CSS-подобные селекторы для поиска и выделения HTML элементов
  - `querySelector(selector)` возвращает первый совпавший с селектором элемент
  - `querySelectorAll(selector)` возвращает **коллекцию всех элементов** которая совпадает с селектором
- Оба метода принимают строковый параметр
  - Который является CSS селектором для элемента

```
//элемент с id="header"
var header = document.querySelector('#header');

//li элемент содержащийся в элементе с id=main-nav
var navItems = document.querySelectorAll('#main-nav li');
```

## Выделение вложенных элементов

- DOM API может быть использована для выделения элементов, которые содержатся внутри других элементов
  - Выделить все `<div>` которые находятся внутри элемента с `id = "wrapper"`

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

// возвращает все div находящиеся внутри элемента с id "wrapper"
var divsInWrapper = wrapper.getElementsByTagName('div');
```

## NodeLists (Лист узлов)

- NodeList это коллекция возвращаемая селекторами DOM дерева:
  - `getElementsByTagName(tagName)`
  - `getElementsByName(name)`
  - `getElementsByClassName(className)`
  - `querySelectorAll(selector)`

```
var divs = document.getElementsByTagName('div'),
    queryDivs = document.querySelectorAll('div');
for(var i = 0, length = divs.length; i < length; i += 1){
    // делаем что либо здесь с divs[i]
}
```

- NodeList очень похож на массив но это не так
  - Это объект со свойствами, подобными массиву
    - Имеет метод определения длины и индексов
  - При переборе с помощью for-in цикла поведение может быть неопределенным:

```
console.log(Array.isArray(divs)); // false

for (var i in divs) {
    console.log('divs[' + i + '] = ' + divs[i]);
}
```

## Статический NodeList and Живой NodeList

- Есть два вида листов узлов NodeLists
  - `getElementsByTagName` метод возвращающий живой лист узлов **LiveNodeList**
  - `querySelectorAll` возвращающий статический лист узлов **StaticNodeList**
- Живые листы отслеживают выделенные элементы
  - Если что то меняется то это происходит и в DOM
  - В то время как статические листы содержат элементы такими, какие они были в момент выделения
- Имейте в виду что `LiveNodeList` медленнее чем обычный массив
  - Нужно кешировать их для лучшей производительности
  - Лучше конвертировать в массив когда итераций много