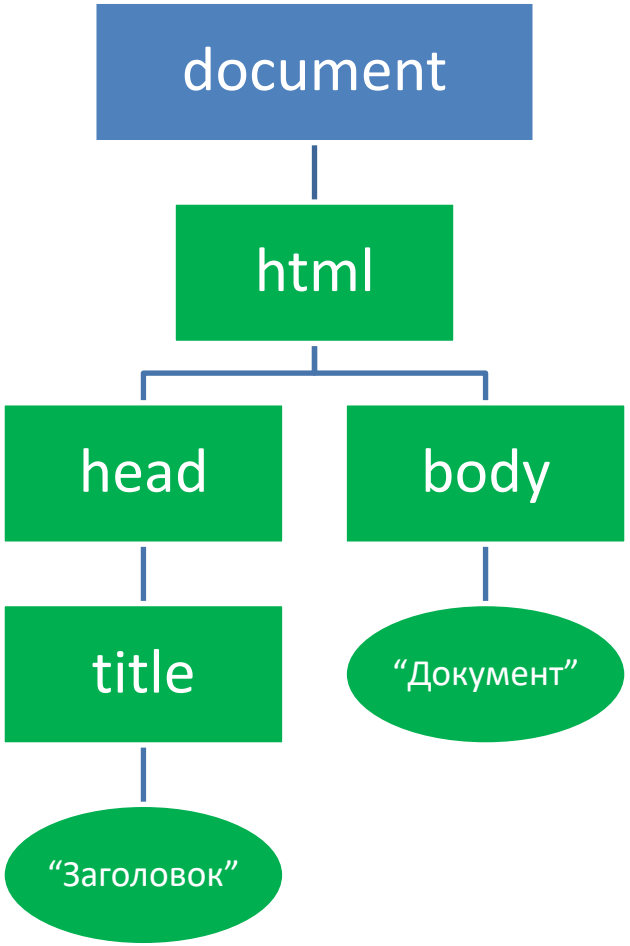
Объектная модель документа

DOM – Document Object Model

DOM

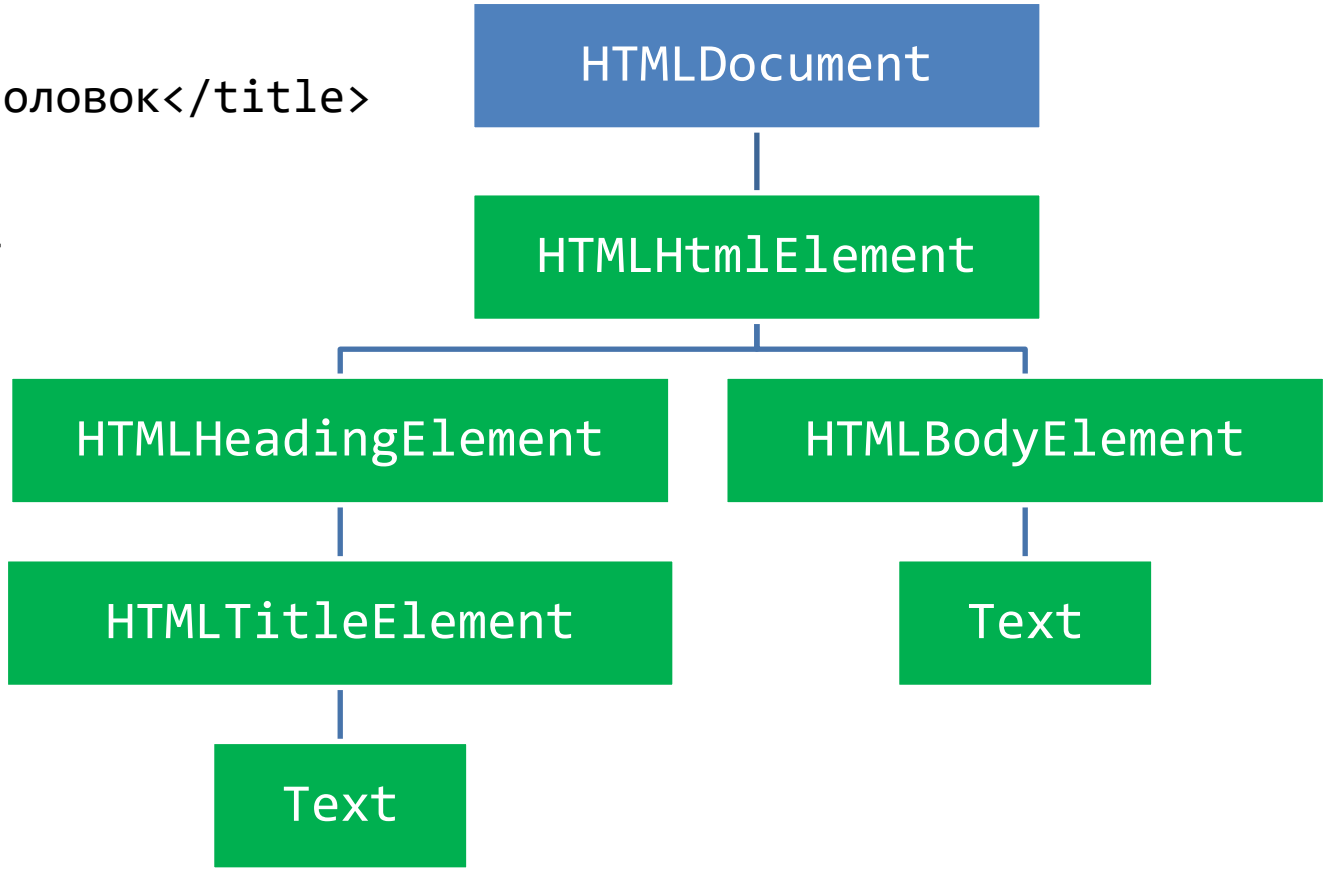
```
<html>
  <head>
     <title>Заголовок</title>
  </head>
  <body>
       Документ
  </body>
</html>
```

Объектная модель документа



DOM

```
<html>
  <head>
     <title>Заголовок</title>
  </head>
  <body>
       Документ
  </body>
</html>
```



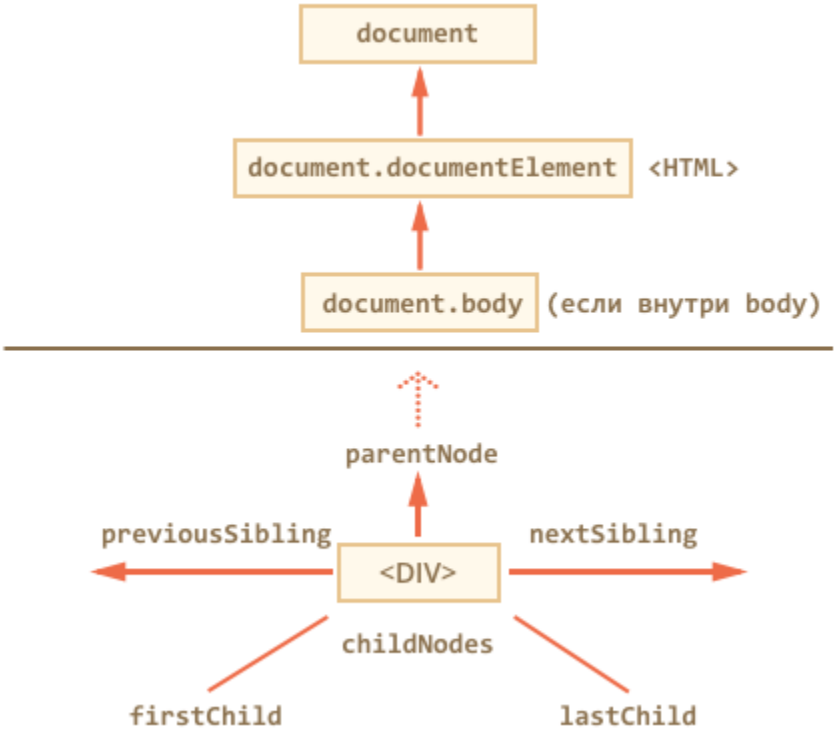
DOM

JS

- Обеспечивает доступ ко всем элементам документа и их атрибутам посредством древовидной структуры объектов
- Позволяет создавать, удалять и изменять элементы документа и их содержимое
- DOM – представление документа, загруженного в окно браузера, в виде дерева тегов

ОБХОД DOM ДЕРЕВА

Доступ к DOM



- document - глобальный объект; является корневым элементом DOM-дерева страницы. Из него можно добраться до любых узлов.

Методы получения элементов

JS

- **querySelector** (except IE<8)
- querySelectorAll
- getElementsByClassName (except IE<9)
- getElementById
- getElementsByTagName
- getElementsByName

querySelector

JS

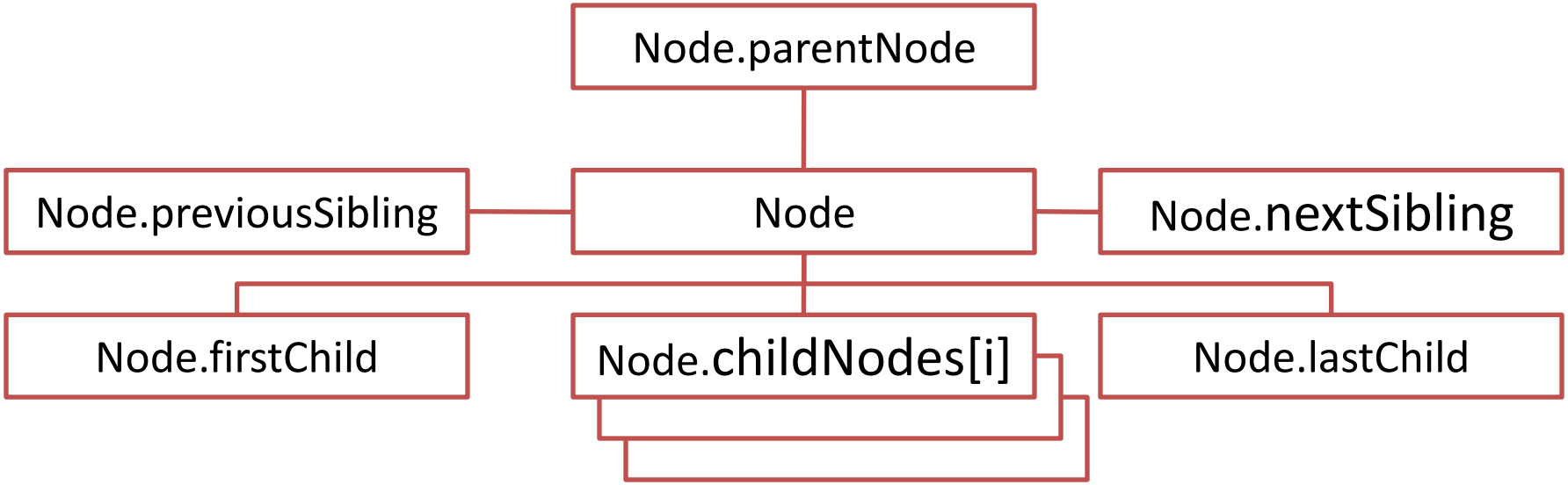
Возвращает элемент из DOM-дерева, соответствующий CSS-селектору

```
document.querySelector('CSS селектор');
document.querySelector('.posts');
document.querySelector('[name="search"]');
```

Навигация по дереву

Свойства для чтения (Node – текущий узел)

- parentNode
- childNodes[i]
- firstChild
- lastChild
- previousSibling
- nextSibling



Проблемы навигации

JS

- Некоторые браузеры трактуют символы пробелов и переходы на новую строку как текстовые узлы
- Это вызывает проблемы при использовании свойств: firstChild, lastChild, nextSibling, previousSibling

Свойства узлов

```
<div>
   Уважаемые
    <span>коллеги</span>
</div>
```

- nodeName – имя тега
  "DIV"
- innerTEXT – текст без тегов
  "Уважаемые коллеги "
- innerHTML – html содержимое
  "↲ Уважаемые коллеги↲ "
- outerHTML – содержит текст и HTML-элементы для данного и вложенных тегов (html-узел целиком)
  " ↲ Уважаемые коллеги↲ "

РЕДАКТИРОВАНИЕ DOM ДЕРЕВА

Методы редактирование дерева

JS

- createElement()
  создание нового узла-элемента
- createTextNode()
  создание нового узла-текста
- appendChild()
  добавление узла в конец коллекции childNodes узла, для которого метод был вызван
- insertBefore()
  добавление узла с возможностью указания места, куда вставляется новый узел

Редактирование дерева

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li>Петя</li>
    <li>Вася</li>
</ol>
```

```
listNode = document.querySelector('ol');
// нахождение узла
```

Редактирование дерева

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li>Петя</li>
    <li>Вася</li>
</ol>
```

```
<li></li>
```

```
listNode = document.querySelector('ol');
newItem = document.createElement('li');
// создание отдельного пустого пункта меню
// <li></li>
```

Редактирование дерева

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li>Петя</li>
    <li>Вася</li>
</ol>
```

```
<li></li>
```

```
Саша
```

```
listNode = document.querySelector('ol');
newItem = document.createElement('li');
newFriend = document.createTextNode('Саша');
// создать текст “Саша”
```

Редактирование дерева

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li>Петя</li>
    <li>Вася</li>
</ol>
```

```
<li>Саша</li>
```

```
listNode = document.querySelector('ol');
newItem = document.createElement('li');
newFriend = document.createTextNode('Саша');
newItem.appendChild(newFriend);
// поместить текст “Саша” в пункт меню
// <li>Саша</li>
```

Редактирование дерева

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li>Петя</li>
    <li>Вася</li>
    <li>Саша</li>
</ol>
```

```
listNode = document.querySelector('ol');
newItem = document.createElement('li');
newFriend = document.createTextNode('Саша');
newItem.appendChild(newFriend);
listNode.appendChild(newItem);
```

Удаление узла дерева

JS

- removeChild()
  удаление потомка узла, для которого метод был вызван
- removeNode()
  удаление узла из документа

Редактирование атрибутов

JS

- createAttribute()
  создает новый атрибут узла
- setAttribute()
  присваивает значение атрибуту
- removeAttribute()
  удаляет атрибут
- getAttribute()
  возвращает текущее значение атрибута

Вставка новой ссылки

```
<script>
function addlink()  {
   link = document.createElement('a')
   link.setAttribute('href','http://math.isu.ru')
   link.setAttribute('name','isu')
   text = document.createTextNode('Сайт ИГУ')
   link.appendChild(text)
   document.getElementById(‘for-link').appendChild(mylink)
}
</script>

<p id='for-link' onclick="addlink()">Добавить ссылку</p>
```

РАБОТА С CSS КЛАССАМИ

Методы classList

JS

- Набор методов для управления классами элемента

classList

JS

- element.classList.add();
  Добавляет класс
- element.classList.remove();
  Удаляет класс.
- element.classList.toggle();
  Переключает класс
- element.classList.contains();
  Сообщает, есть ли класс у элемента.

Добавление CSS класса

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li>Петя</li>
    <li>Вася</li>
    <li>Саша</li>
</ol>
```

```
secondlistNode = document.querySelectorAll('ul')[1];
// нахождение узла
```

Добавление CSS класса

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li>Петя</li>
    <li>Вася</li>
    <li>Саша</li>
</ol>
```

```
<h1>Друзья</h1>
<ol>
    <li>Миша</li>
    <li class='warning'>Петя</li>
    <li>Вася</li>
    <li>Саша</li>
</ol>
```

```
secondlistNode = document.querySelectorAll('ul')[1];
secondlistNode.classList.add('warning');
```

ОБРАБОТКА СОБЫТИЙ

События

JS

- Для реакции на действия посетителя и внутреннего взаимодействия скриптов существуют *события*.
- *Событие* – это сигнал от браузера о том, что что-то произошло.

События

- Мышь:
  - click – происходит, когда кликнули на элемент левой кнопкой мыши
  - contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши
  - mouseover – возникает, когда на элемент наводится мышь
  - mousedown и mouseup – когда кнопку мыши нажали или отжали
  - mousemove – при движении мыши
- Элементах управления:
  - submit – посетитель отправил форму <form>
  - focus – посетитель фокусируется на элементе, например нажимает на <input>
- Клавиатура:
  - keydown – когда посетитель нажимает клавишу
  - keyup – когда посетитель отпускает клавишу
- Документ:
  - DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

Обработка событий

JS

- Событию можно назначить *обработчик*, то есть функцию, которая сработает, как только событие произошло.
- Именно благодаря обработчикам JavaScript-код может реагировать на действия посетителя.

Обработка событий

JS

- Через атрибут HTML
- Через свойства DOM-объекта
- Через метод addEventListener

Через атрибут HTML

JS

```
<input
  value="Нажми меня“
  onclick="alert('Клик!')"
  type="button">
<input
  value="Нажми меня"
  onclick="someFunction()"
  type="button">
```

Через свойства DOM-объекта

```
<input
  id="elem"
  type="button"
  value="Нажми меня" />

<script>
  elem.onclick = function() {
     alert( 'Спасибо' );
  };
</script>
```

Через addEventListener

JS

```
elem.addEventListener(
  "click" ,
  function() {
    alert('Спасибо!')
  }
)
```

addEventListener

JS

```
<input id="elem" type="button" value="Нажми меня"/>
<script>
  function handler1() { alert('Спасибо!'); };
  function handler2() { alert('Спасибо ещё раз!'); }
  elem.onclick = function() { alert("Привет"); };
  elem.addEventListener("click", handler1);
  elem.addEventListener("click", handler2);
</script>

// Привет, Спасибо!, Спасибо ещё раз!
```

addEventListener

JS

```
function addHandler(obj, msg) {
  function say() {
    alert(msg)
  }
  obj.addEventListener('click',say);
}
```