

# Лекция № 3. Объектная модель документа

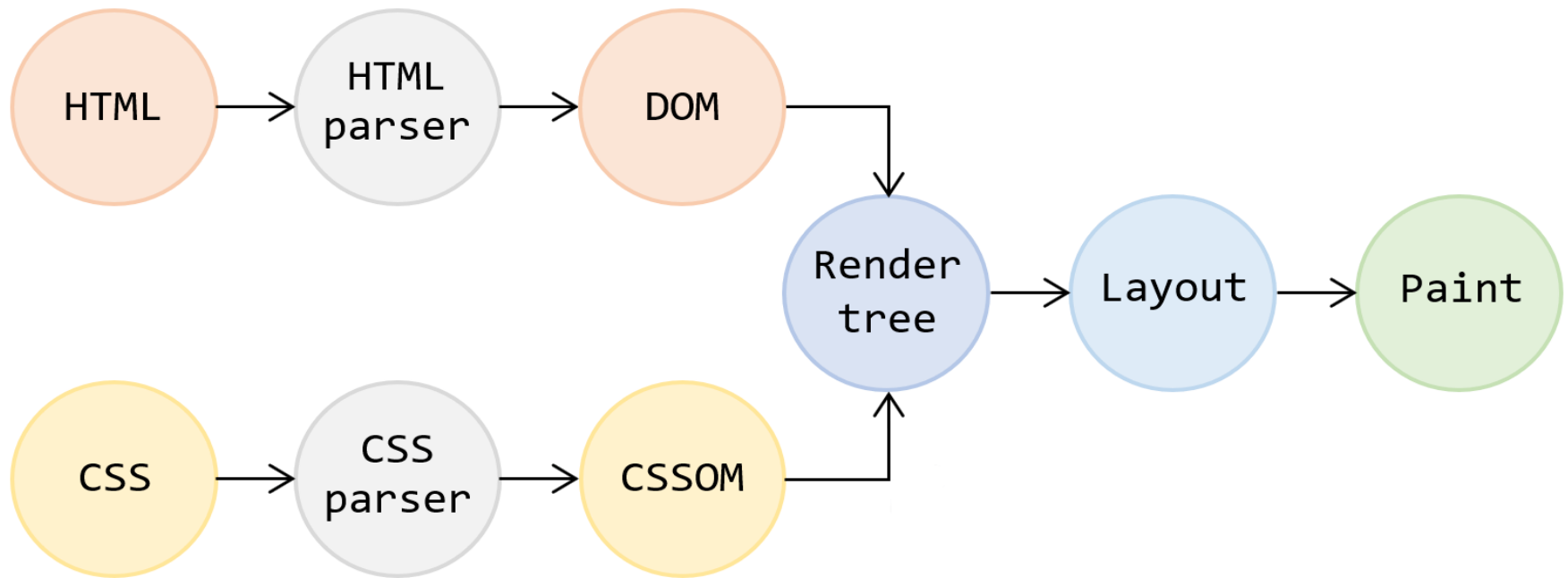
## **УЧЕБНЫЕ ВОПРОСЫ**

1. Последовательность создания веб-страницы.
2. Понятие DOM
3. Построение DOM
4. Типы и имена DOM-узлов
5. Исследование DOM

Вопрос №1. Последовательность создания веб-страницы

Когда браузер загружает HTML-код страницы, он строит на основании него **объектную модель документа** (на английском Document Object Model или сокращённо **DOM**).

Процесс преобразования исходного кода HTML-документа в отображение стилизованной и интерактивной картинки на экране называется **Critical Rendering Path (CRP)**



Хотя этот процесс состоит из большого количества шагов, их грубо можно представить в виде двух:

1. **Анализирует** HTML-документ, чтобы определить то, что в конечном итоге нужно отобразить на странице;
2. **Выполняет отрисовку** того что нужно отобразить.

Результатом первого этапа является формирование **дерева рендеринга** (render tree). Данное дерево содержит **видимые элементы** и **текст**, которые нужно отобразить на странице, и также связанные с ними **стили**.

В render tree каждый элемент **содержит** соответствующий ему **объект DOM** и рассчитанные для него **стили**.

Таким образом, render tree описывает **визуальное представление DOM**.

Чтобы построить дерево рендеринга, браузеру нужны две вещи:

- **DOM**, который он формирует из полученного HTML-кода;
- **CSSOM** (CSS Object Model), который он строит из загруженных и распознанных стилей.

На втором этапе браузер выполняет **отрисовку** render tree.

Для этого он:

- **рассчитывает** положение и размеры каждого элемента в render tree, этот шаг называется Layout;
- **выполняет** рисование, этот шаг называется Paint.

После **Paint** все нарисованные элементы находятся на **одном слое**. Для повышения производительности страницы браузер выполняет ещё один шаг, который **называется Composite**.

В нем он группирует элементы по **композиционным слоям**. Именно благодаря этому этапу мы можем создать на странице **плавную анимацию** элементов при использовании таких свойств как *transform*, *opacity*. Так как изменение этих свойств вызовет только одну задачу Composite.



**Layout** и **Paint** – это ресурсоемкие процессы, поэтому для хорошей отзывчивости вашей страницы или веб-приложения, необходимо свести к **минимуму** операции которые их вызывают.

Список свойств, изменение которых **вызывают Paint**:

- color;
- background;
- visibility;
- border-style...

Список свойств, изменение которых **вызывает Layout**:

- width и height;
- padding и margin;
- display;
- border;
- top, left, right и bottom;
- position;
- font-size и другие.

Кроме этого, Layout срабатывает не только при изменении CSS-свойств, но также, например когда мы хотим **получить смещение элемента**

Вопрос №2. Понятие DOM

DOM – это объектное представление исходного HTML-кода документа. Процесс формирования DOM происходит так: браузер **получает** HTML-код, **парсит** его и **строит** DOM.

Затем, как мы уже отмечали выше браузер использует DOM (а не исходный HTML) для строительства дерева рендеринга, потом выполняет layout и так далее.

Почему не использовать в этом случае просто HTML? Потому что **HTML – это текст**, и с ним невозможно работать так как есть. Для этого нужно его разобрать и создать на его основе объект, что и делает браузер. И этим объектом является **DOM**.

DOM – это **объектная модель документа**, которую браузер создаёт в памяти компьютера на основании HTML-кода.

По-простому, HTML-код – это **текст страницы**, а DOM – это **объект**, созданный браузером при парсинге этого текста.

Но, браузер использует DOM **не только** для выполнения процесса CRP, но также **предоставляет программный доступ** к нему. Следовательно, с помощью JavaScript мы можем **изменять DOM**.

Все объекты и методы, которые предоставляет браузер описаны в **спецификации** HTML DOM API, поддерживаемой W3C. С помощью них мы можем читать и изменять документ в памяти браузера.

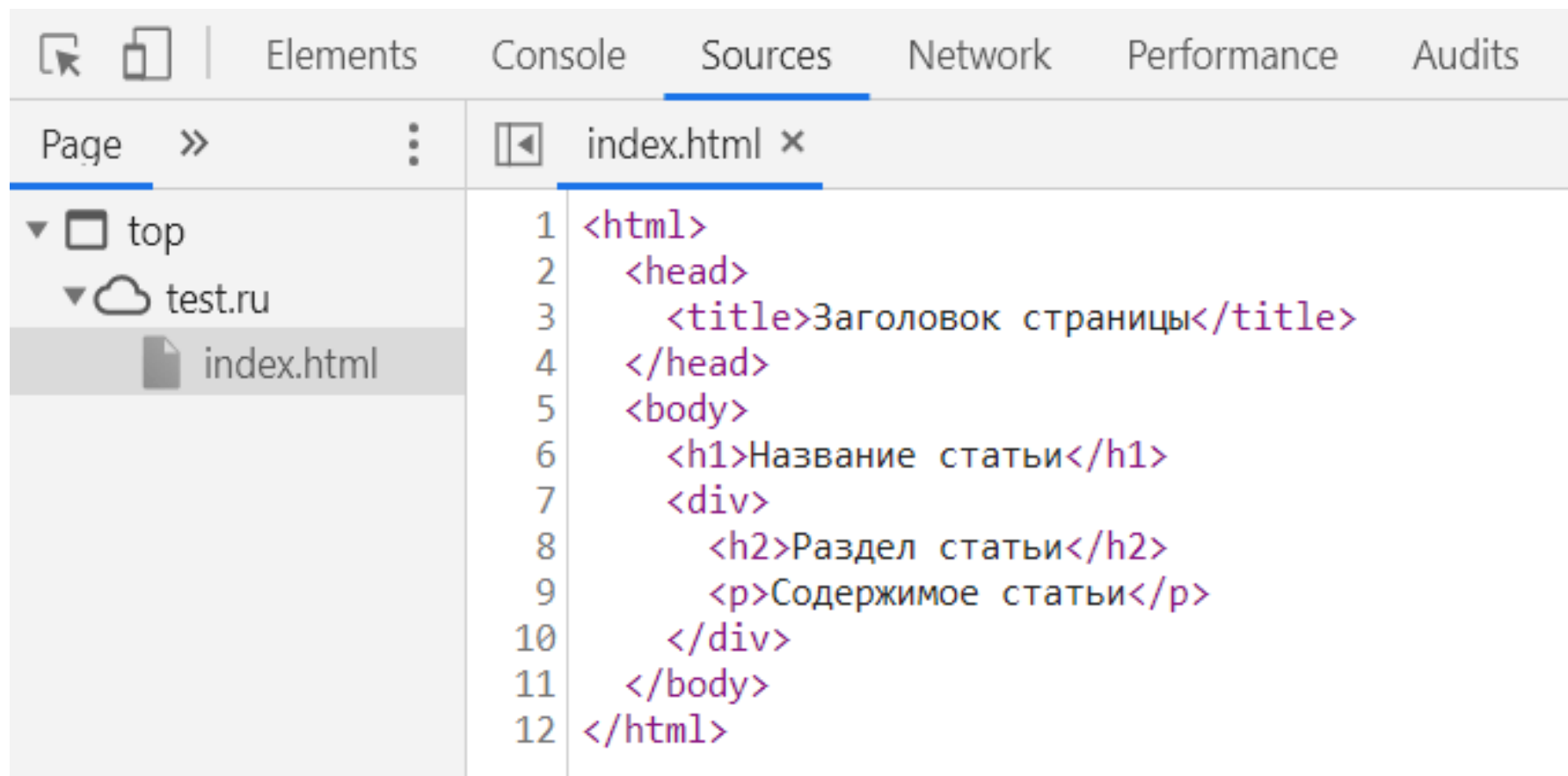
Например, с помощью JavaScript мы можем:

- добавлять, изменять и удалять любые HTML-элементы на странице, в том числе их атрибуты и стили;
- получать доступ к данным формы и управлять ими;
- реагировать на все существующие HTML-события на странице и создавать новые;
- рисовать графику на HTML-элементе `<canvas>` и многое другое.

При *изменении* DOM браузер **проходит** по шагам CRP и почти мгновенно **обновляет** изображение страницы. В результате у нас всегда отрисовка страницы соответствует DOM.

Благодаря тому, что JavaScript позволяет изменять DOM, мы можем создавать **динамические** и **интерактивные** веб-приложения и сайты. С помощью JavaScript мы можем менять всё что есть на странице. Сейчас в вебе практически нет сайтов, в которых не используется работа с DOM.

В браузере Chrome исходный HTML-код страницы, можно посмотреть во вкладке «Source» на панели «Инструменты веб-разработчика»:

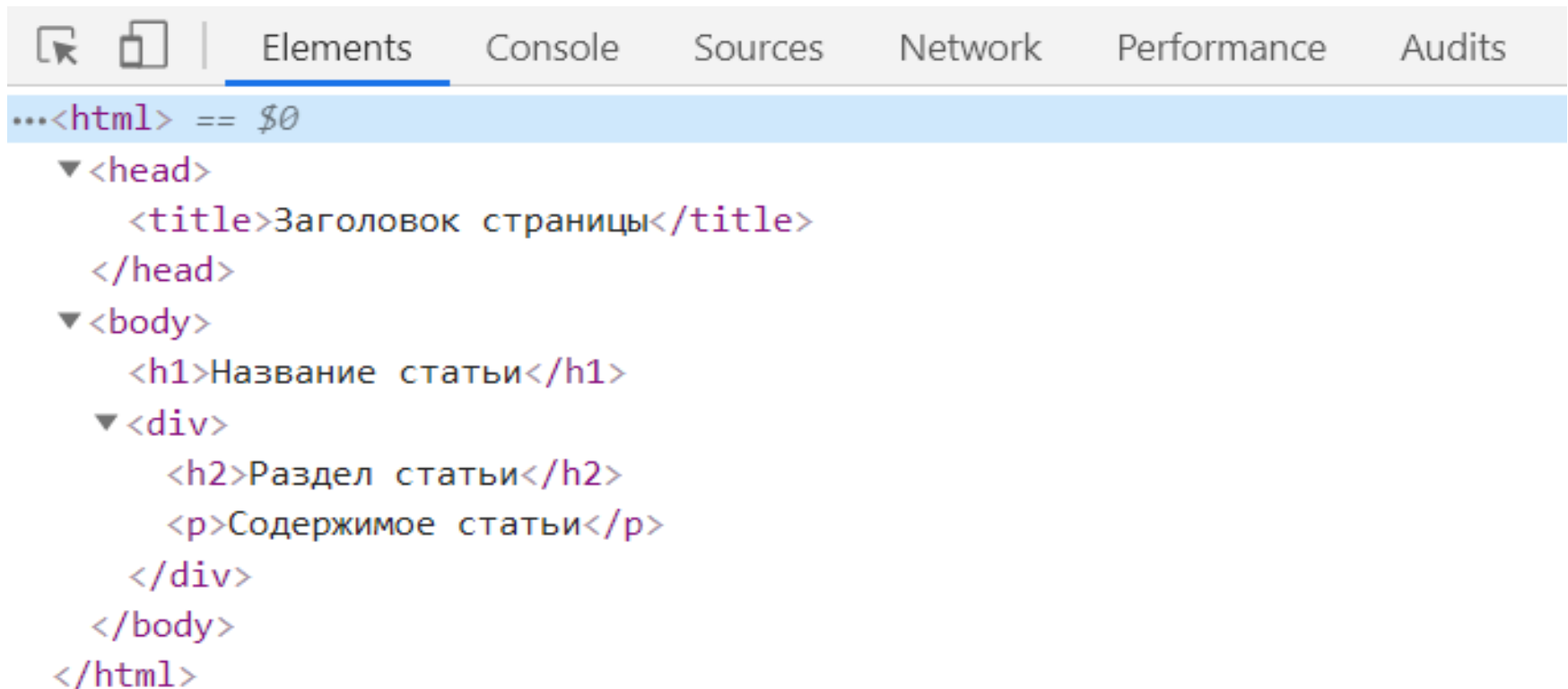


The image shows the Chrome DevTools interface with the 'Sources' tab selected. The left sidebar shows a file tree with 'index.html' selected. The main area displays the source code of the HTML document, which includes a title, a main heading, a sub-heading, and a paragraph of text.

```
1 <html>
2   <head>
3     <title>Заголовок страницы</title>
4   </head>
5   <body>
6     <h1>Название статьи</h1>
7     <div>
8       <h2>Раздел статьи</h2>
9       <p>Содержимое статьи</p>
10    </div>
11  </body>
12 </html>
```



На вкладке Elements мы видим что-то очень похожее на DOM:



The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The DOM tree is expanded to show the root <html> element. The structure is as follows:

```
...<html> == $0
  ▼ <head>
    <title>Заголовок страницы</title>
  </head>
  ▼ <body>
    <h1>Название статьи</h1>
    ▼ <div>
      <h2>Раздел статьи</h2>
      <p>Содержимое статьи</p>
    </div>
  </body>
</html>
```

Вопрос №3. Построение DOM

```
<!doctype html>  
<html lang="ru">  
<head>  
  <title>Моя страница</title>  
</head>  
<body>  
  <h1>Мобильные ОС</h1>  
  <ul>  
    <li>Android</li>  
    <li>iOS</li>  
  </ul>  
</body>  
</html>
```

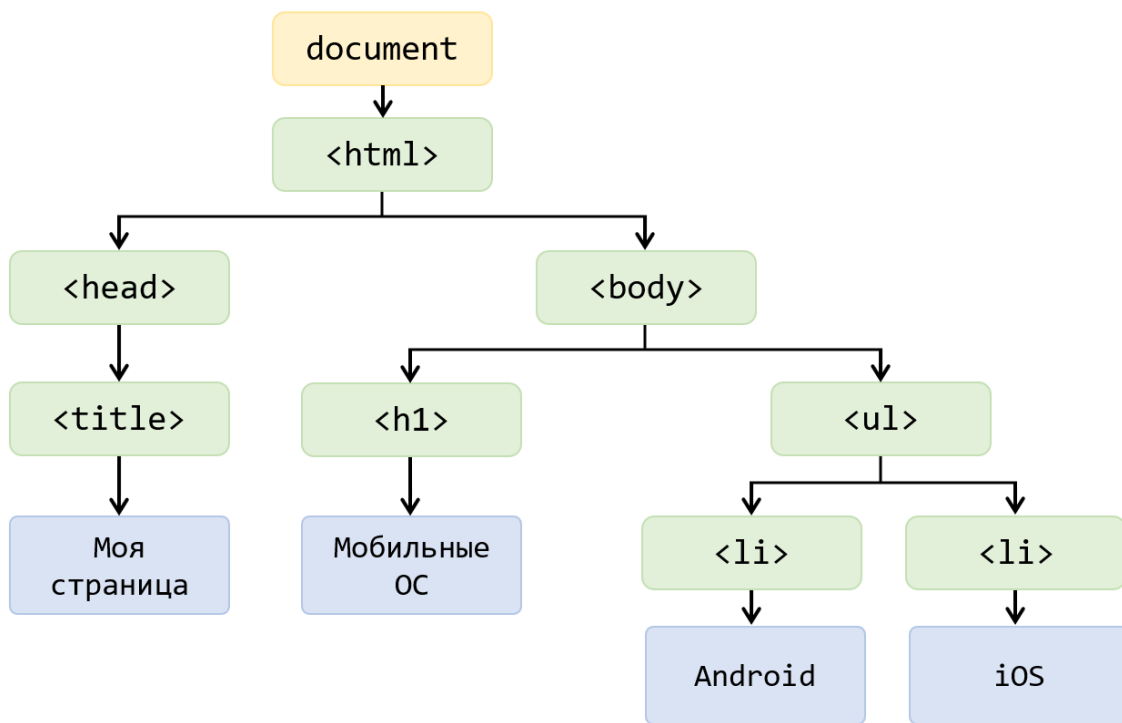
Теперь рассмотрим, как браузер на основании HTML-кода строит DOM.

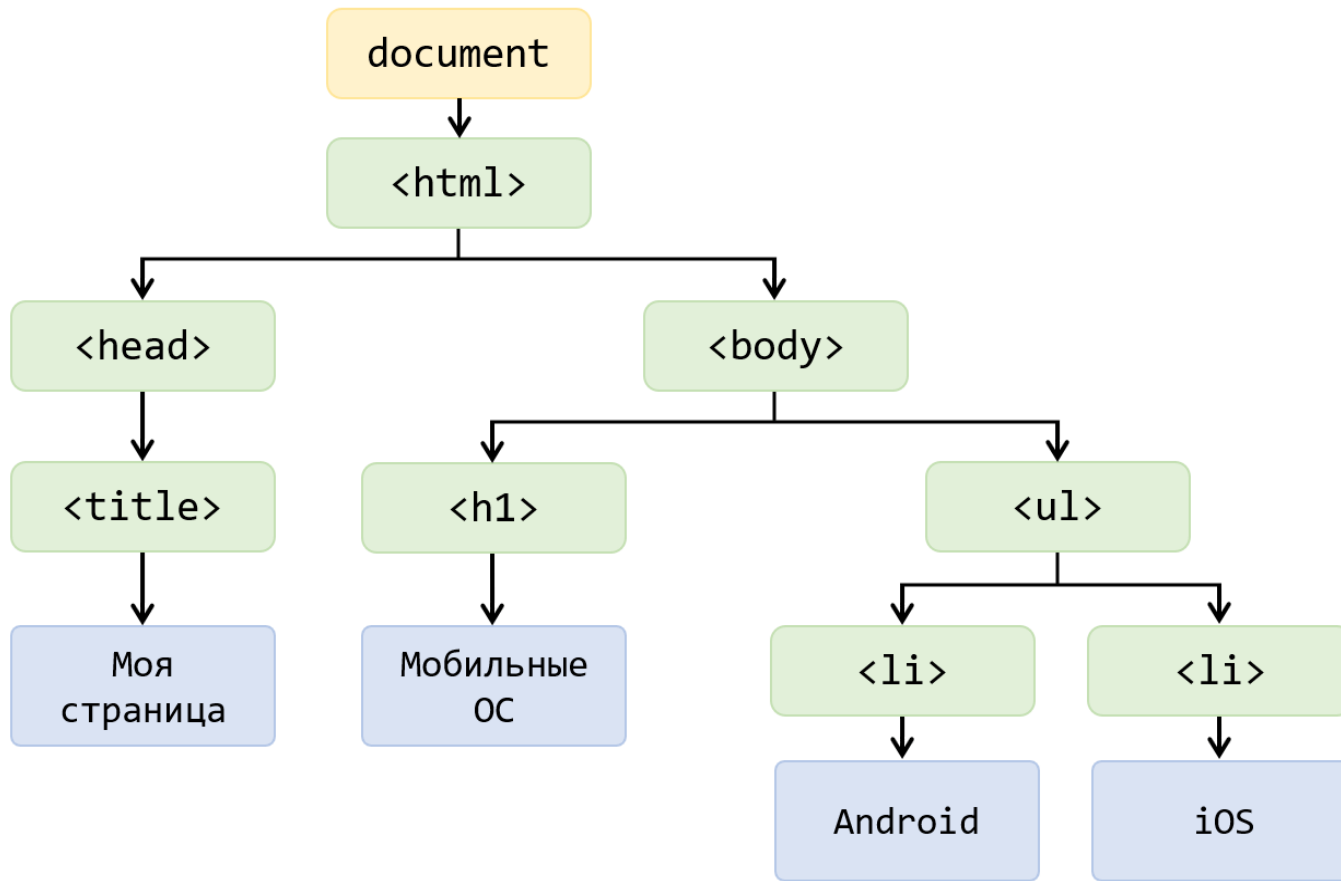
**Объектная структура DOM** представляет собой **дерево узлов** (узел на английском называется *node*). При этом DOM-узлы образуются из всего, что есть в HTML:

тегов,

текстового контента,

комментариев и т.д.





Корневым узлом DOM-дерева является объект `document`, он представляет сам этот документ. Далее в нём расположен узел `<html>`. В `<html>` находятся 2 узла-элемента: `<head>` и `<body>`. В `<title>` находится текстовый узел. Теперь перейдём к `<body>`. В нём находятся 2 элемента `<h1>` и `<ul>`, и так далее.

Узлы в зависимости от того, чем они образованы делятся на **разные типы**. В DOM выделяют:

- **узел**, представляющий собой **весь документ**; этим узлом является объект `document`; он выступает входной точкой в DOM;
- **узлы**, образованные **тегами**, их называют *узлами-элементами* или просто **элементами**;
- **текстовые узлы**, они образуются текстом внутри элементов;
- **узлы-комментарии** и так далее.

Каждый узел в дереве DOM является **объектом**. Но при этом формируют структуру DOM только **узлы-элементы**. Текстовые узлы, например, содержат в себе **только текст**. Они **не могут** содержать внутри себя другие узлы. Поэтому вся работа с DOM в основном связана с узлами-элементами.

Чтобы перемещаться по узлам DOM-дерева нужно знать какие они имеют **отношения**. Зная их можно будет выбирать правильные свойства и методы. Связи между узлами, определяются их **вложенностью**.

Каждый узел в DOM может иметь следующие виды отношений:

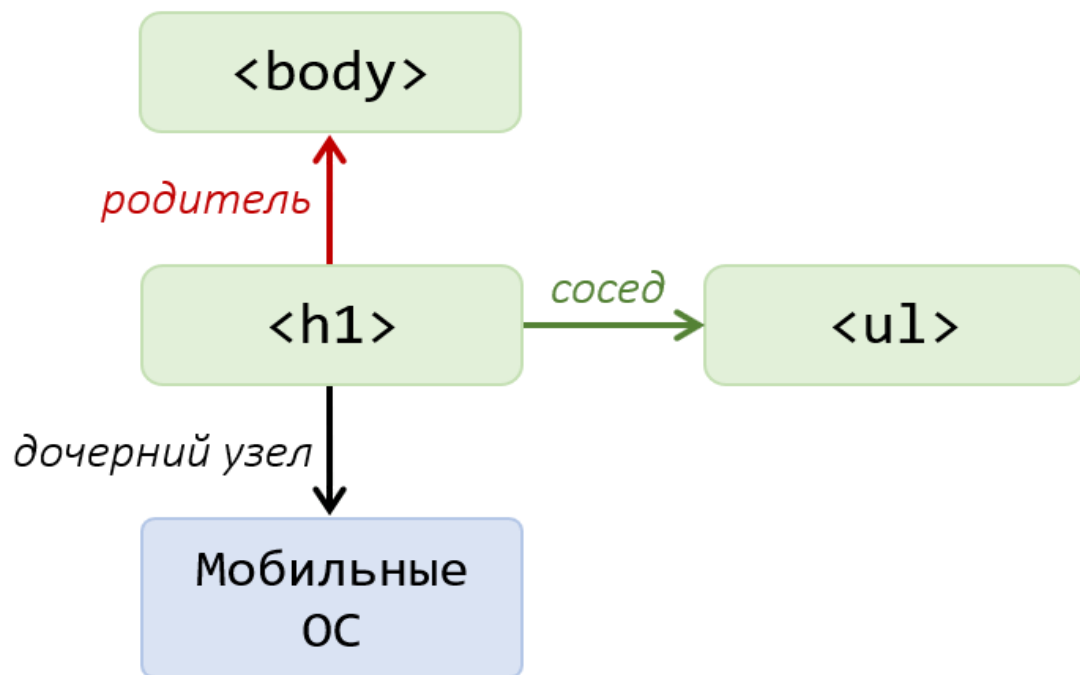
- **родитель** – это узел, в котором он непосредственно расположен; при этом родитель у узла может быть только один; также узел может не иметь родителя, в данном примере им является document;



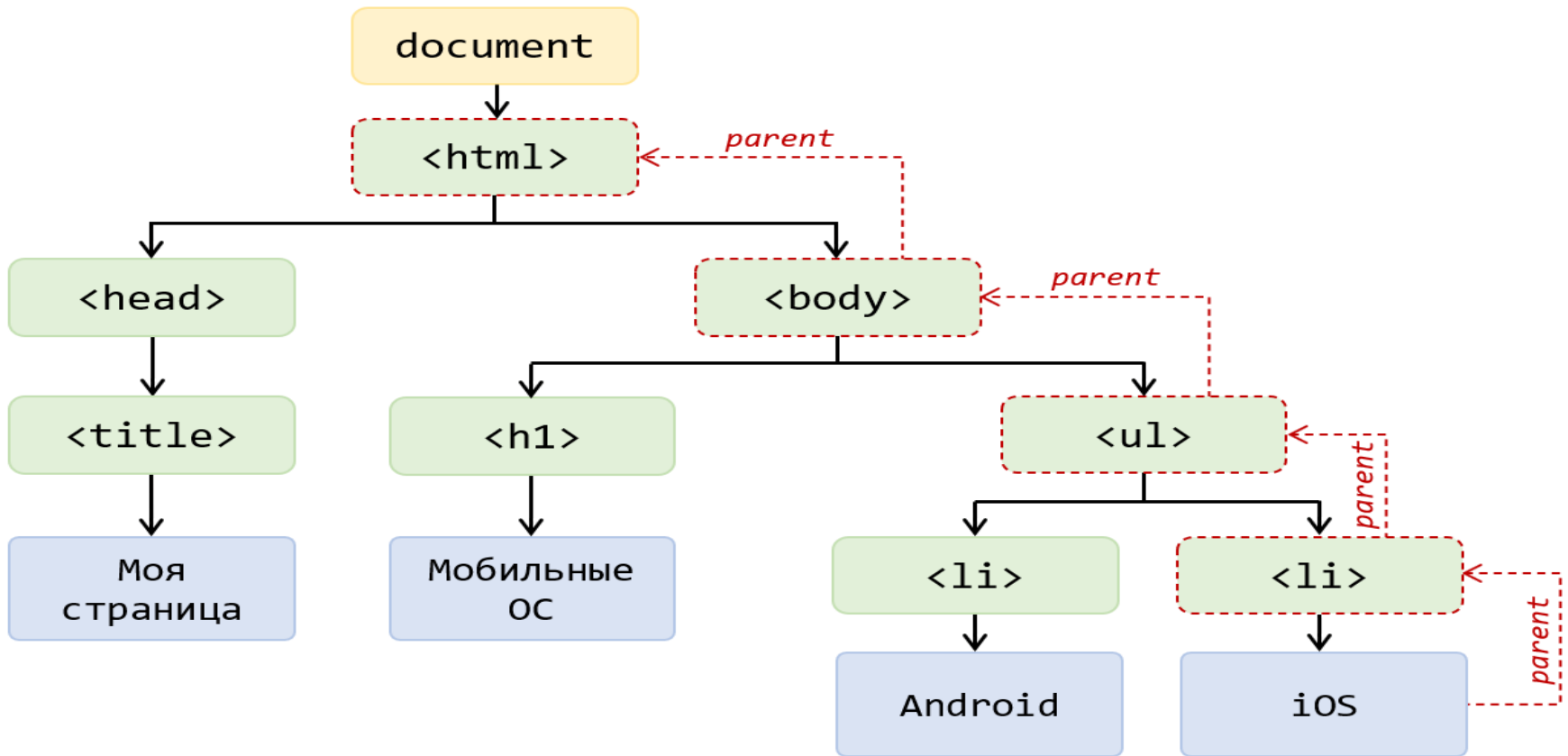
Каждый узел в DOM может иметь следующие виды отношений:

- **дети** или **дочерние узлы** – это все узлы, которые расположены непосредственно в нём; например, узел `<ul>` имеет 2 детей;
- **соседи** или **сиблинги** – это узлы, которые имеют такого же родителя что и этот узел;
- **предки** – это его родитель, родитель его родителя и так далее;
- **потомки** – это все узлы, которые расположены в нём, то есть это его дети, а также дети его детей и так далее.

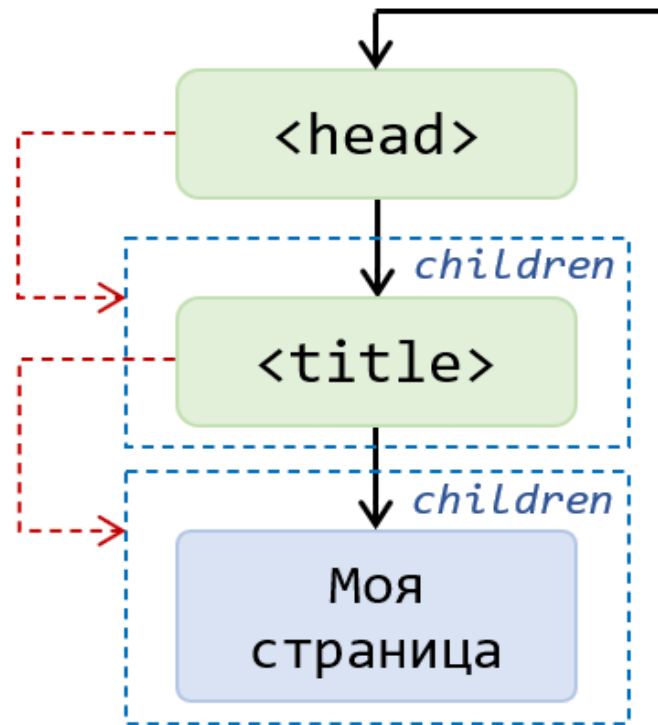
Например, узел-элемент `<h1>` имеет в качестве **родителя** `<body>`. **Ребенок** у него один – это текстовый узел «Мобильные ОС». **Сосед** у него тоже только один – это `<ul>`.



Теперь рассмотрим, каких **предков** имеет текстовый узел «iOS». У него они следующие: `<li>`, `<ul>`, `<body>` и `<html>`.



У элемента `<head>` 2 **потомка**: `<title>` и текстовый узел «Моя страница».



Вопрос №4. Типы и имена DOM-узлов

Основную структуру DOM-дерева составляют именно узлы, образованные HTML-тегами. Их называют **узлами-элементами** или просто **элементами**.

Узнать тип узла в DOM можно с помощью **свойства nodeType**:

```
console.log(document.nodeType); // 9  
console.log(document.body.nodeType); // 1
```

Это свойство возвращает число от 1 до 12, обозначающее тип узла.

## Основные значения:

- 1 – элемент (Node.ELEMENT\_NODE);
- 2 – атрибут (Node.ATTRIBUTE\_NODE);
- 3 – текстовый узел (Node.TEXT\_NODE);
- 8 – комментарий (Node.COMMENT\_NODE);
- 9 – document (Node.DOCUMENT\_NODE);
- 10 – узел, содержащий тип документа (Node.DOCUMENT\_TYPE\_NODE);
- 11 – узел, представляющий фрагмент документа DocumentFragment (Node.DOCUMENT\_FRAGMENT\_NODE).

## свойство nodeName

С его помощью мы можем узнать имя узла или тег, если узел является элементом:

```
console.log(document.body.nodeName); // "BODY"  
console.log(document.doctype.nodeName) // "html"  
console.log(document.nodeName); // "#document"
```

Свойство nodeName для других узлов, не являющимися **элементами** возвращает различные значения:

- для текстовых узлов – "#text";
- для узлов-комментариев – "#comment";
- для document – "#document" и так далее.



Получить имя тега элемента можно не только с помощью `nodeName`, но также посредством **свойства `tagName`**.

`tagName` запрограммирован в браузере как геттер, он содержится в **prototype класса `Element`**.

`nodeName` – это тоже геттер, но находится он в другом месте, в **prototype класса `Node`**.

Поэтому свойство `tagName` доступно только для **узлов-элементов**, и не доступно для других типов узлов.

Вопрос №5. Исследование DOM

В браузерах при разработке веб-приложений и сайтов имеется очень полезный инструмент **DevTools**.

Открыть в браузере Chrome его можно через меню или посредством комбинации клавиш:

- macOS – Cmd + Shift + I;
- Windows – Ctrl + Shift + I или F12;
- Linux – Ctrl + Shift + I.

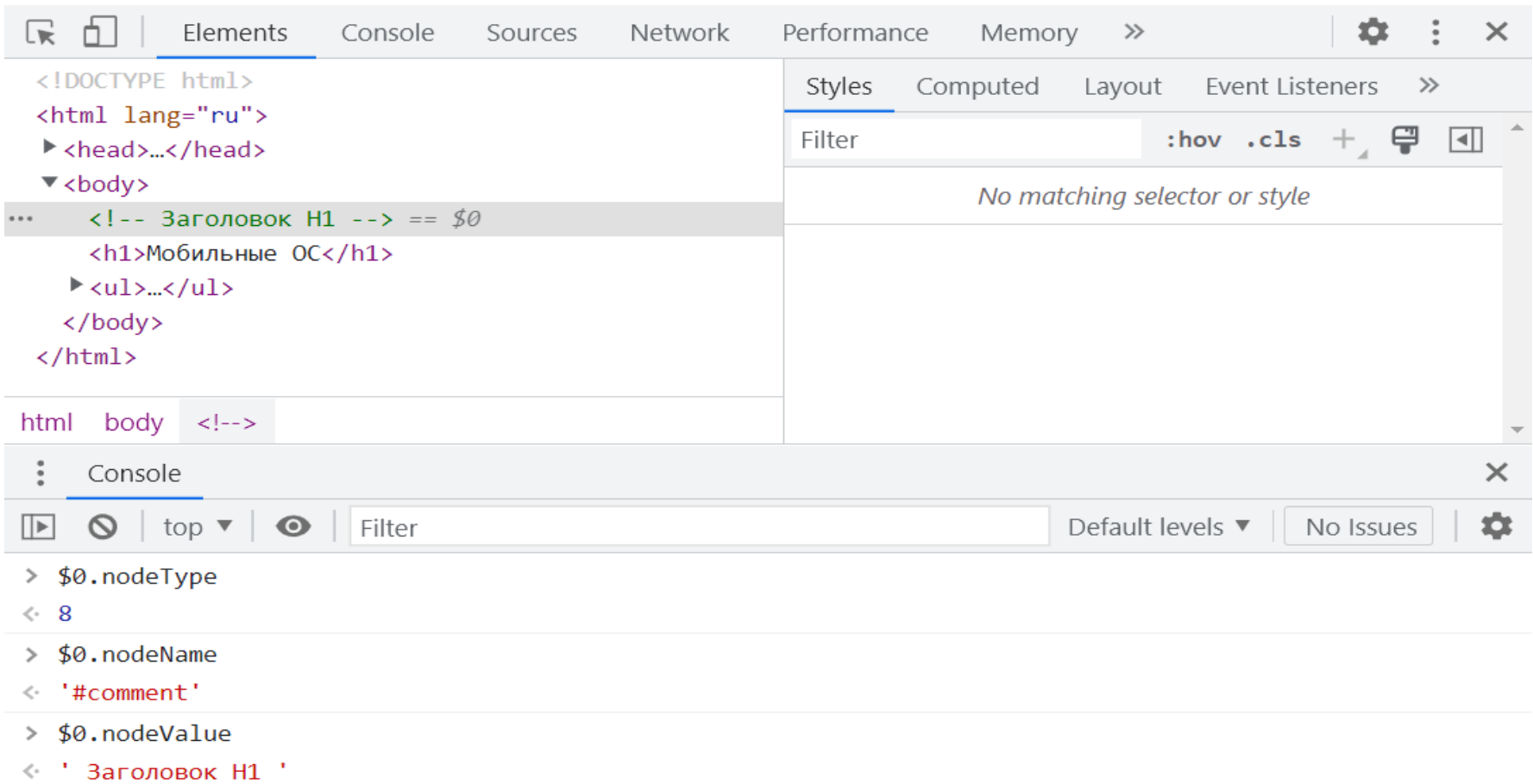
На вкладке **Element** вы можете исследовать DOM и CSS. При необходимости их можно **изменять** прямо здесь, и посмотреть как будут выглядеть эти правки прямо на веб-странице.

**Выбрать нужный элемент** на веб-странице можно разными способами:

- кликнуть по нему правой кнопкой мыши и выбрать в открывшемся меню пункт «Inspect» или «Посмотреть код»;
- найти его в DOM, для поиска элемента дополнительно можно использовать окно поиска, которое можно вызвать с помощью комбинации клавиш Ctrl + F;
- нажать на значок и визуально выбрать нужный элемент.

После выбора узла мы можем обратиться к нему в консоли через **\$0**. При этом предыдущий выбранный узел будет доступен как **\$1** и так далее. Это можно использовать при изучении DOM и отладке сайта.

Например, выберем комментарий



The screenshot displays the Chrome DevTools interface. The top panel shows the 'Elements' tab with the DOM tree. The selected node is a comment: `<!-- Заголовок H1 -->`. The right panel shows the 'Styles' tab with a filter and the message 'No matching selector or style'. The bottom panel shows the 'Console' tab with the following output:

```
> $0.nodeType
< 8

> $0.nodeName
< '#comment'

> $0.nodeValue
< ' Заголовок H1 '
```

**Свойство `nodeValue`** позволяет получить содержимое **текстового узла** или **комментария**. Для остальных узлов оно возвращает в качестве значения **`null`**.

С помощью `nodeValue` мы можем также установить новое значение этому узлу:

```
> $0.nodeValue = 'Новый текст комментария'
```

```
< 'Новый текст комментария'
```

---

```
> $0.nodeValue
```

```
< 'Новый текст комментария'
```

---

Кроме `nodeValue` нам также доступно свойство `data`, с помощью которого мы можем выполнить аналогичные действия:

```
> $0.data
```

```
< 'Android'
```

---

```
> $0.nodeType
```

```
< 3
```

---

```
> $0.data = 'iOS'
```

```
< 'iOS'
```

---

Получить и изменить содержимое элементов можно с помощью других свойств, таких как **textContent** и **innerHTML**. Например, выведем значения которые возвращают эти свойства для элемента **<ul>**:

```
> $0.innerHTML
```

```
< '\n    <li>Android</li>\n    <li>iOS</li>\n  '
```

---

```
> $0.textContent
```

```
< '\n    Android\n    iOS\n  '
```

---



Здесь мы видим `\n` и пробелы. `\n` – это перевод строки.

Так как по факту, например, первый `<li>` **расположен не сразу** после `<ul>`, а перед ним имеется вот такой контент – `\n`.

Он при парсинге страницы будет преобразован браузером в текстовый узел DOM.

Таким образом, первым дочерним узлом `<ul>` будет именно этот текстовый узел, и только потом уже `<li>`.

`\n` образовался из-за того что мы поставили Enter, а четыре пробела – это то количество пробелов, которые мы установили перед тем как написали тег `<li>`.

```
8   <ul>
9   ... <li>Android</li>
10  <li>iOS</li>
11  </ul>
```

В DOM пробелы, переводы строк, знаки табуляции и другие символы расположенные между элементами образуют **текстовые DOM-узлы**.

Например, чтобы их не было в <ul>, его разметка должна быть записана следующим образом:

```
<ul><li>Android</li><li>iOS</li></ul>
```

При выборе DOM-элемента на вкладке **Styles** будет отображаться весь CSS, применённый к этому элементу, в том числе будут отображены и дефолтные стили браузера. Правила можно **редактировать**, **отключать** с помощью чекбоксов и **дописывать** новые. Все изменения применяются сразу.

```
<!DOCTYPE html>
<html lang="ru">
  <head>...</head>
  <body>
    <!-- Заголовок H1 -->
    <h1>Мобильные ОС</h1> == $0
    <ul>...</ul>
  </body>
</html>
```

Styles Computed Layout Event Listeners >>

Filter :hov .cls + [ [ [

```
element.style {
}

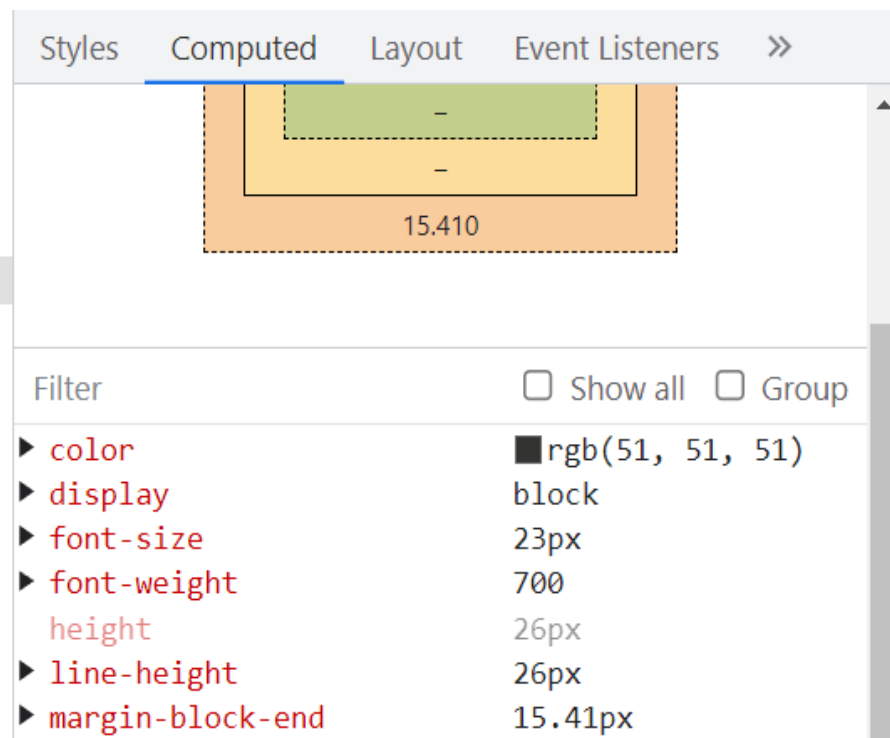
h1 {                                     index.html:7
  font-size: 23px;
  line-height: 26px;
  font-weight: 700;
  color: #333;
}

h1 {                                     user agent stylesheet
  display: block;
  font-size: 2em;
  margin-block-start: 0.67em;
  margin-block-end: 0.67em;
  margin-inline-start: 0px;
  margin-inline-end: 0px;
  font-weight: bold;
}
```

html body h1

На вкладке **Computed** мы можем посмотреть результирующие стили, примененные к элементу.

```
<!DOCTYPE html>
<html lang="ru">
  <head>...</head>
  <body>
    <!-- Заголовок H1 -->
    ... <h1>Мобильные ОС</h1> == $0
    <ul>...</ul>
  </body>
</html>
```



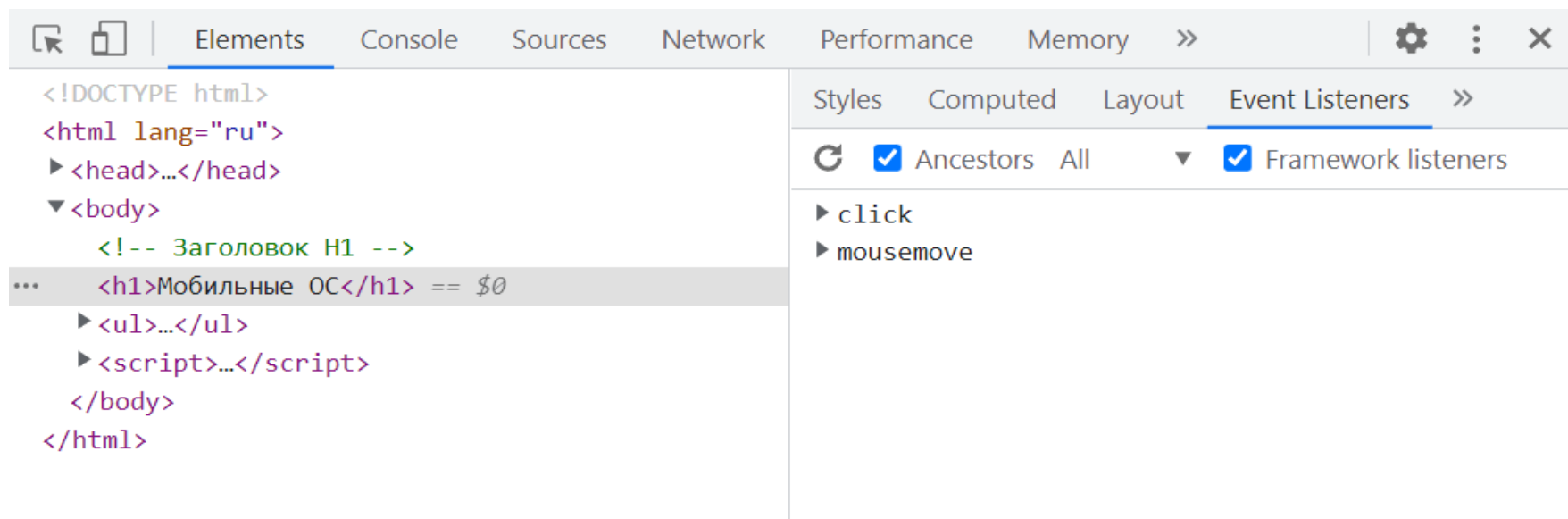
Styles Computed Layout Event Listeners >>

15.410

Filter  Show all  Group

- ▶ color ■ rgb(51, 51, 51)
- ▶ display block
- ▶ font-size 23px
- ▶ font-weight 700
- height 26px
- ▶ line-height 26px
- ▶ margin-block-end 15.41px

На вкладке **Event Listeners** отображаются все обработчики событий, привязанные к данному DOM-элементу.



The screenshot shows the Chrome DevTools interface. The 'Elements' panel on the left displays the DOM tree with the following structure:

```
<!DOCTYPE html>
<html lang="ru">
  <head>...</head>
  <body>
    <!-- Заголовок H1 -->
    <h1>Мобильные ОС</h1> == $0
    <ul>...</ul>
    <script>...</script>
  </body>
</html>
```

The 'Event Listeners' panel on the right is active, showing the following settings and listeners:

- Refresh icon
- Ancestors
- All
- Framework listeners
- ▶ click
- ▶ mousemove