# DOM дерево HTML. Введение в DOM.

DOM (Document Object Model) — объектная модель документа — набор правил, согласно которым формируется каждая веб-страница с использованием языка разметки HTML. DOM хранит структуру сайта в виде дерева и информацию, заключённую в тегах.

Каждый элемент этого дерева называется
узлом.

Узлы могут иметь неограниченную вложенность. Однако не рекомендуется делать эту вложенность слишком глубокой: чем больше уровней, тем сложнее ориентироваться в коде и тем медленнее работает страница.

Дерево DOM служит нам инструментом для поиска нужного элемента на странице и взаимодействия с ним.

При создании парсеров вам не требуется выполнять какие-то изменения в узлах DOM. Главное — понимать структуру DOM и знать, как получить доступ к определённому узлу и информации в нём.

# DOM дерево HTML. Элементы и их виды.

**Блочные элементы** служат для формирования основной структуры веб-страницы.

| Особенность | Описание |
|---|---|
| Структурная Организация | Блочные элементы часто служат контейнерами для других блочных или строчных элементов. Это делает их хорошими кандидатами для извлечения группы данных. |
| CSS Селекторы | Из-за их структурной роли, блочные элементы часто имеют уникальные классы или идентификаторы, что упрощает их выборку. |
| Контекстуальная Информация | Поскольку блочные элементы часто содержат другие элементы, они могут предоставить контекст для извлекаемых данных. |
| Стабильность | Блочные элементы обычно менее подвержены изменениям на динамически загружаемых веб-страницах, что делает их более надежными для парсинга. |
| Количество | В целом, блочных элементов обычно меньше, чем строчных, что может упростить процесс парсинга. |

**Особенности**:

1. Блоки располагаются вертикально, один под другим.
2. Вставлять блочные элементы внутрь строчных запрещено.
3. Блоки занимают всё доступное пространство по ширине контейнера.
4. Высота блока вычисляется автоматически на основе его содержимого.

**Примеры блочных элементов**:

- `<div>` - Универсальный блочный контейнер
- `<p>` - Абзац
- `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>` - Заголовки
- `<ul>` - Ненумерованный список
- `<ol>` - Нумерованный список
- `<li>` - Элемент списка (обычно используется внутри `<ul>` или `<ol>`)
- `<blockquote>` - Длинная цитата
- `<pre>` - Форматированный текст
- `<hr>` - Горизонтальная линия (разделитель)
- `<table>` - Таблица
- `<thead>` - Заголовок таблицы
- `<tbody>` - Тело таблицы
- `<tfoot>` - Подвал таблицы
- `<tr>` - Строка таблицы
- `<th>` - Заголовочная ячейка таблицы
- `<td>` - Данные таблицы
- `<article>` - Статья
- `<section>` - Раздел
- `<header>` - Шапка
- `<footer>` - Подвал
- `<nav>` - Навигация
- `<aside>` - Боковая панель
- `<main>` - Основное содержимое
- `<figure>` - Иллюстрация или диаграмма с подписью
- `<figcaption>` - Подпись к `<figure>`
- и др.

# DOM дерево HTML. Элементы и их виды.

**Строчные элементы** применяются для форматирования отдельных фрагментов текста.
Они обычно содержат одно или несколько слов.

| Особенность | Описание |
|---|---|
| Специфичность | Строчные элементы часто содержат конкретные фрагменты данных, такие как текст ссылок, выделенный текст и т.д. |
| Динамичность | Элементы, такие как ссылки или кнопки, могут быть динамически загружены или изменены, что может создать проблемы при парсинге. |
| Вложенность | Строчные элементы часто вложены в блочные, и это может быть использовано для извлечения более точных данных. |
| Множественность | Они часто повторяются внутри блочных элементов, что может требовать более сложной логики парсинга для извлечения всех необходимых данных. |
| Ограниченный Контекст | Строчные элементы редко предоставляют контекстуальную информацию, поскольку они обычно фокусируются на конкретных данных. |

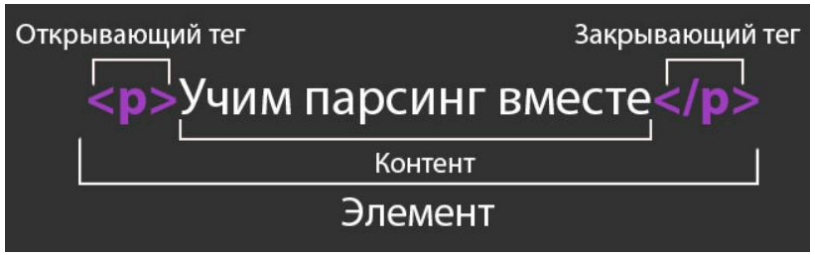
**Примеры строчных элементов**:

- `<span>` - Универсальный строчный контейнер
- `<a>` - Ссылка
- `<strong>` - Важный текст
- `<em>` - Выделенный текст
- `<b>` - Полужирный текст (без семантического ударения)
- `<i>` - Курсивный текст (без семантического ударения)
- `<u>` - Подчеркнутый текст
- `<s>` - Зачеркнутый текст
- `<small>` - Мелкий текст
- `<big>` - Увеличенный текст (устаревший тег)
- `<abbr>` - Аббревиатура
- `<acronym>` - Акроним (устаревший тег)
- `<code>` - Код
- `<q>` - Короткая цитата
- `<sub>` - Нижний индекс
- `<sup>` - Верхний индекс
- `<time>` - Время или дата
- и др.

**Особенности**:

1. Строчные элементы, расположенные подряд, отображаются на одной строке. При необходимости, они автоматически переносятся на следующую строку.
2. Внутрь строчных элементов можно вставлять текст или другие строчные элементы. Вставка блочных элементов запрещена.

# DOM дерево HTML. HTML Атрибуты.




**Открывающий тег (Opening tag)**: Этот компонент HTML-структуры состоит из имени элемента, взятого в угловые скобки. Открывающий тег служит индикатором начала действия определенного элемента. В данном контексте он сигнализирует о начале нового абзаца, обозначенного тегом `<p>` .

**Закрывающий тег (Closing tag):** Внешне похож на открывающий тег, но имеет дополнительную косую черту перед именем элемента. Этот тег указывает на место, где действие элемента завершается. В примере выше, он обозначает конец абзаца/параграфа. Отсутствие закрывающего тега — это одна из наиболее распространенных ошибок среди новичков в веб-разработке, и это может привести к непредсказуемым и неожиданным результатам .

**Контент (Content)**: Этот термин описывает собственно содержимое элемента, которое в нашем случае является текстом. Контент располагается между открывающим и закрывающим тегами и представляет собой информацию, которую необходимо отобразить и именно контент мы и будем парсить.

**Элемент(Element)**: Открывающий тег, закрывающий тег и контент в совокупности формируют полноценный HTML-элемент. Элемент представляет собой базовую единицу структуры HTML-документа и служит для оформления или структурирования информации на веб-странице.

# DOM дерево HTML. HTML Атрибуты.

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример незакрытого тега</title>
</head>
<body>

    <p>Это первый абзац.</p>
    <p>Это второй абзац с <strong>важным текстом
    <p>Это третий абзац.</p>

</body>
</html>
```

Это первый абзац.

Это второй абзац с **важным текстом**

**Это третий абзац.**

# DOM дерево HTML. Поиск элементов на странице.

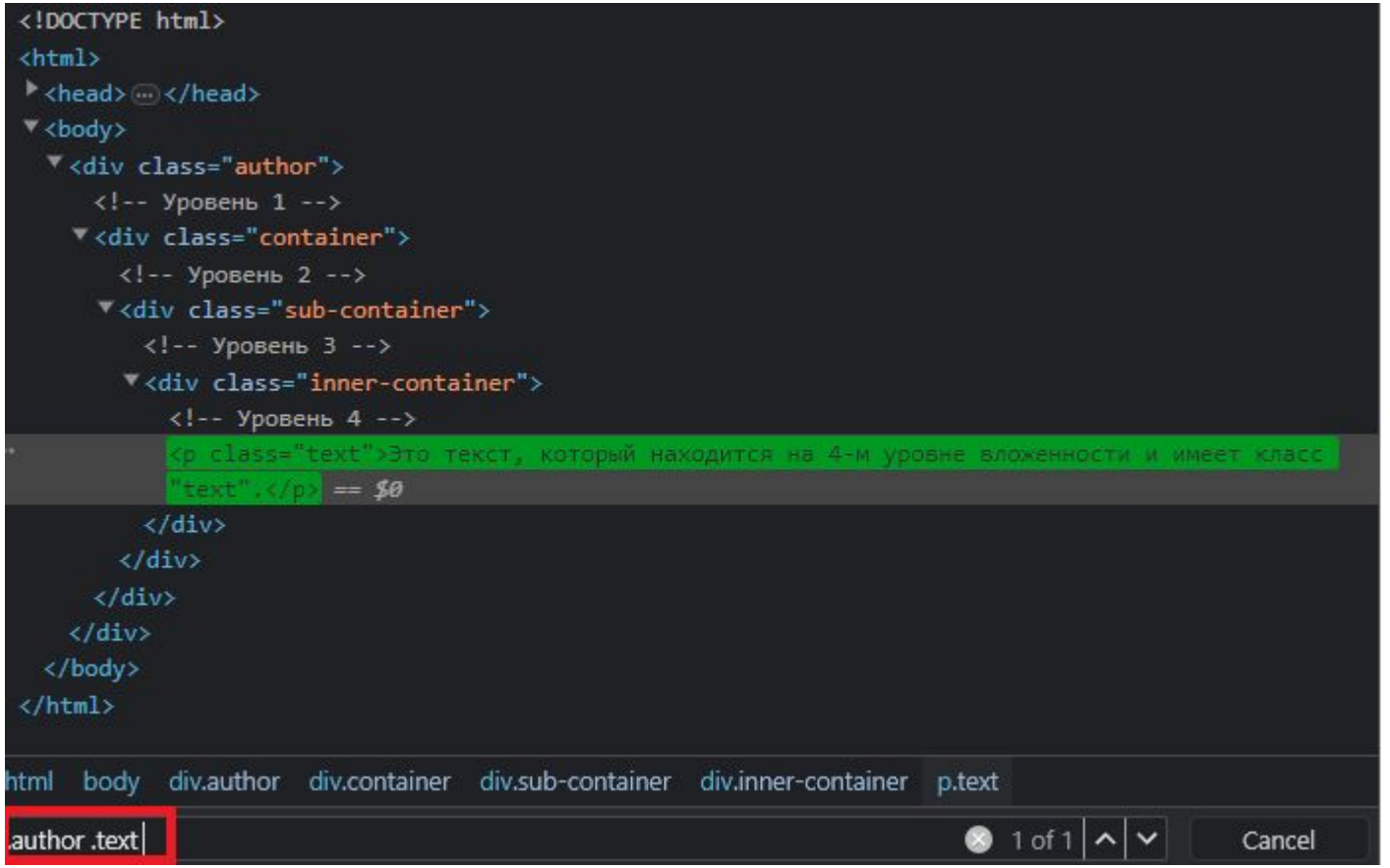
Можно находить элементы на веб-странице различными способами:
используя HTML-теги, CSS-селекторы, атрибуты и их значения

`id="name_id"` : по идентификатору;
`class="name_class"` : по имени класса;
`<div>` : по имени тега;
`href="link"` : по атрибуту;
`name="item"` : по значению атрибута.

```
<div class="main author">
    <p class="text">Лев Тослстой</p>
</div>

<div class="main2 name_book">
    <p class="text">Война и мир</p>
</div>
```

`.main.author` - составной селектор, который используется для поиска элемента с двойным классом



# DOM дерево HTML. Поиск элементов при помощи XPath.

Когда возникает ситуация, где нам сложно найти уникальный CSS-селектор для нужного элемента, XPath становится отличной альтернативой.
Этот метод позволяет искать элементы по их пути в дереве HTML-документа.

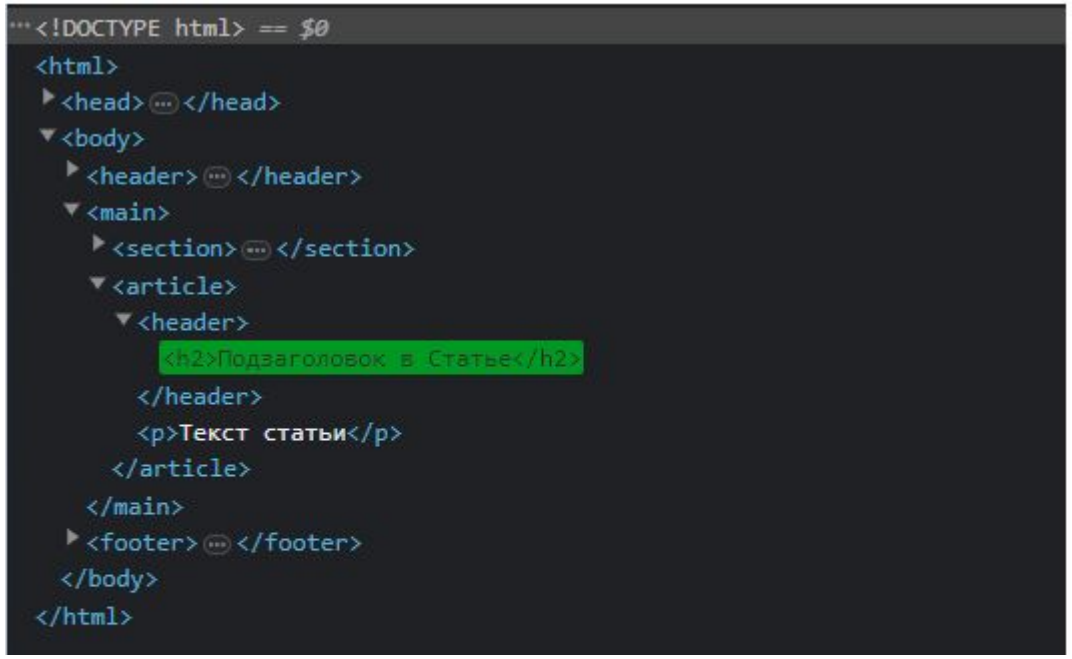
XPath — это мощный и гибкий инструмент, но при этом потенциально рискованный.

1. Во-первых, с **XPath** можно составлять крайне сложные запросы для поиска элементов, что дает большую гибкость.
2. Во-вторых, если разработчик веб-сайта решит изменить структуру страницы, ваш парсер может сломаться, и искомый элемент будет утерян.

# DOM дерево HTML. Поиск элементов при помощи XPath.

1. XPath всегда начинается с символа / или //

- Символ `/` аналогичен символу `>` в CSS-селекторах, а символ `//` — пробелу в CSS.
  - `element1/element2` — выбирает элементы `element2`, которые являются прямыми потомками `element1`.
  - `element1//element2` — выбирает элементы `element2`, которые являются потомками `element1` на любом уровне вложенности.




```
//article/header/h2[text()='Подзаголовок в Статье']
```

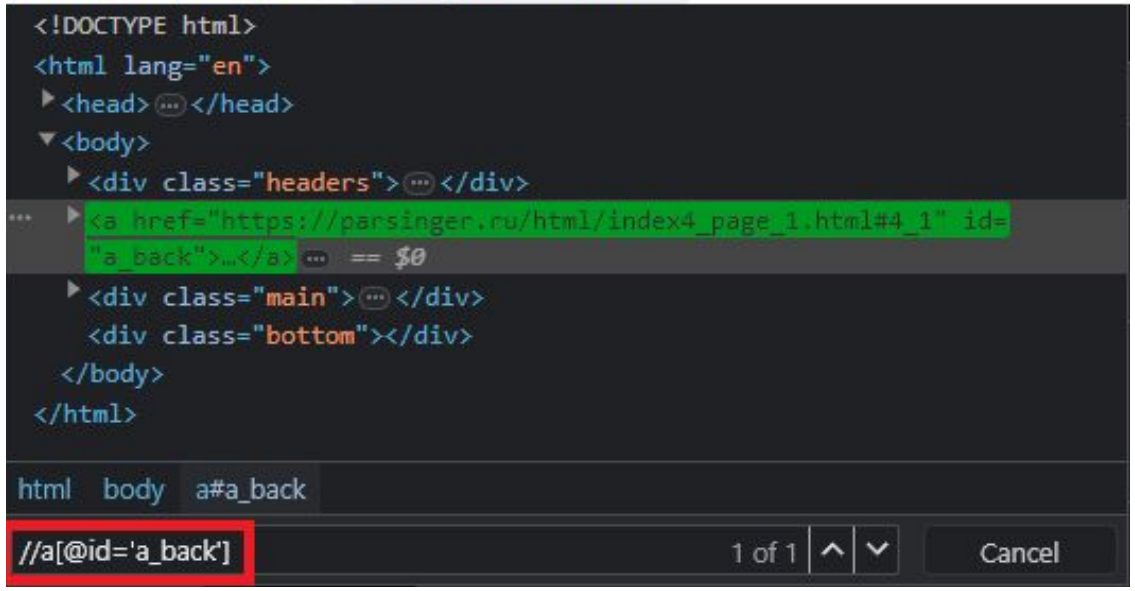
# DOM дерево HTML. Поиск элементов при помощи XPath.

2. `[ ]` - Команда для фильтрации элементов.

Когда по запросу найдено несколько элементов, XPath позволяет произвести фильтрацию согласно правилам, указанным в квадратных скобках `[ ]`.

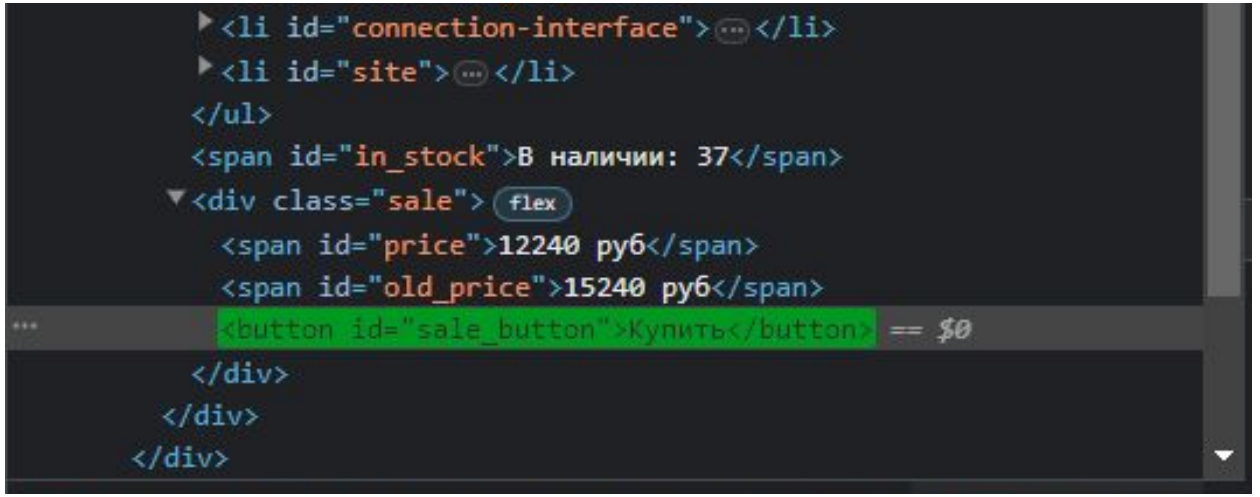
Правил фильтрации очень много:

- **По атрибутам**: Фильтрация может быть произведена по любому атрибуту, такому как `id`, `class`, `title` и так далее. Например, если вы хотите найти ссылку с определенным `id`, запрос будет выглядеть следующим образом: `//a[@id='a_back']`. Здесь `@` указывает на атрибут, а `'a_back'` — это значение атрибута `id`, которое нам нужно.



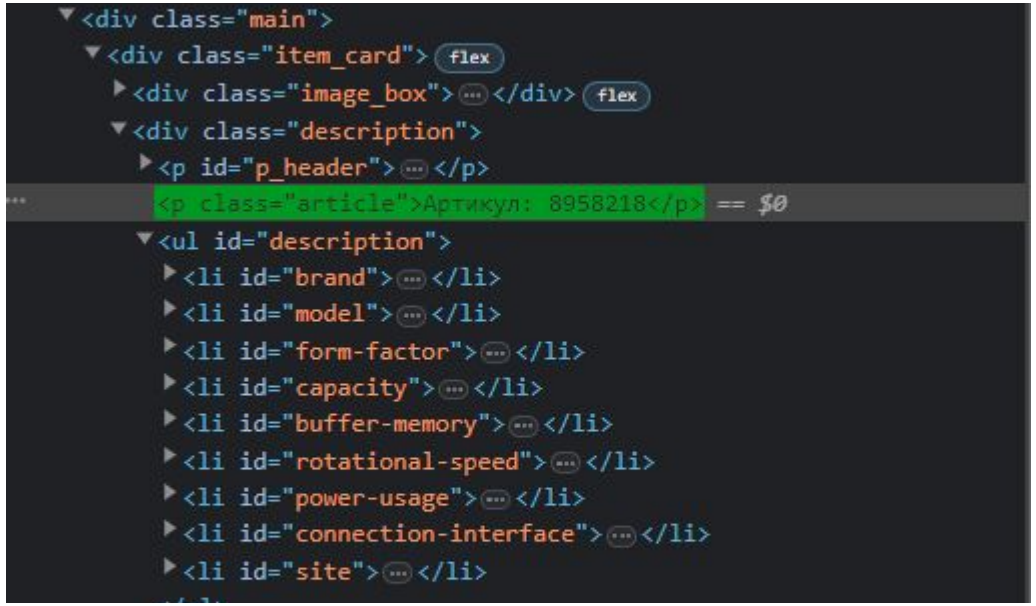
# DOM дерево HTML. Поиск элементов при помощи XPath.

- **По полному совпадению текста**: Если требуется найти элемент по точному текстовому содержанию, **XPath** является вашим единственным выбором. Применение данного метода хорошо иллюстрирует пример с поиском кнопки "Купить" при помощи XPath-селектора `//button[text()="Купить"]`. Стоит отметить, что такой запрос вернет элемент только в случае **полного совпадения** текста. И хотя этот метод может быть весьма эффективным, его использование не всегда является лучшей практикой, особенно если речь идет о мультиязычных сайтах.



# DOM дерево HTML. Поиск элементов при помощи XPath.

- **По частичному совпадению текста или атрибута**: Если вам нужно найти элементы, текст или атрибуты которых частично совпадают с заданным выражением, функция `contains` будет крайне полезной. Например, запрос `//p[contains(text(), "Артикул")]` вернет все абзацы, содержащие слово "**Артикул**". Это может быть особенно полезно, если текст динамичен или содержит дополнительные символы и пробелы. Также метод применим для поиска по частичному совпадению атрибутов, что актуально, когда у элемента несколько классов. Если, скажем, в коде сайта есть тег `ul`, а внутри него теги `li` с атрибутами id, содержащими значения `material_frame` и `material_bracer`, то их можно найти селектором `//li[contains(@id, "material")]`.




```
//p[contains(text(), "Артикул")]
```

# DOM дерево HTML. Поиск элементов при помощи XPath.

### 3. Символ * - Команда выбора всех элементов

- В XPath символ `*` используется для выбора всех элементов, соответствующих заданному критерию. Это особенно полезно, если вы не знаете точный тег элемента, который пытаетесь найти. Допустим, вы хотите найти текст цены в определённом заголовке на сайте. В таком случае, запрос `//div/*[@class="price"]` будет искать все элементы внутри тега `div` с классом "**price**", независимо от их типа.