Лекция № 4. Использование объектной модели документа

УЧЕБНЫЕ ВОПРСЫ

1. Структура и обход документа в JavaScript
2. Методы JavaScript для поиска элементов на странице

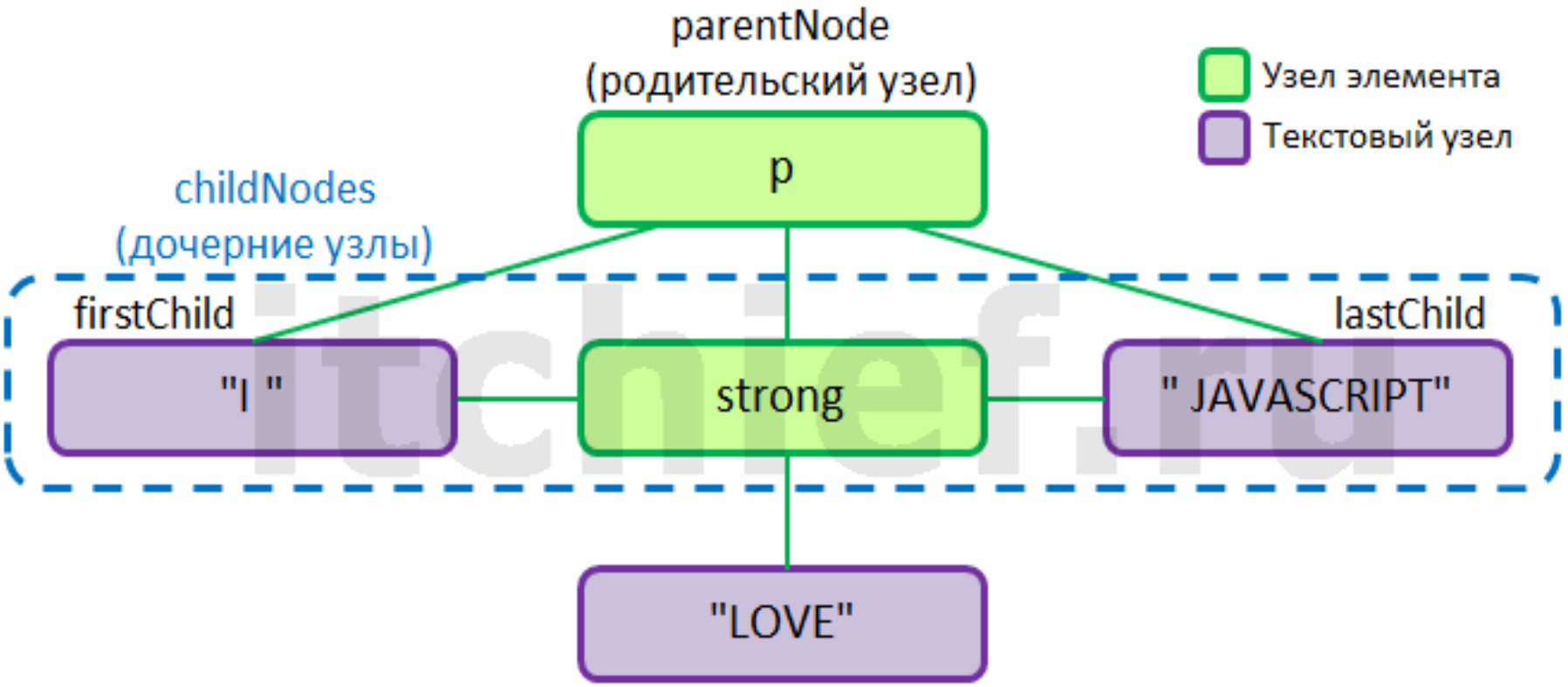
Вопрос №1. Структура и обход документа в JavaScript

Связи (отношения) между узлами

Для того чтобы **изменить** свойства узла, **добавить** к нему новый дочерний узел (ребёнка) или **удалить** существующий у него дочерний узел, необходимо данный узел сначала **получить**, т.е. необходимо до него как-то добраться.

Чтобы мы могли **перемещаться** по дереву, необходимо знать, как узлы **взаимосвязаны между собой**, т.е. какие между ними бывают отношения (связи).

I LOVE JAVASCRIPT



Отношение "родитель-ребёнок"

Для перемещения по дереву DOM используются свойства узла:

сверху вниз, т.е. *от родительского узла к дочернему.*

- **childNodes**,
- **firstChild**,
- **lastChild,**

снизу вверх, т.е. *от дочернего узла к родительскому*.

- **parentNode**

Отношение "родитель-ребёнок"

В нашем примере узел **p** является родительским узлом для:

- Узлов, образованных на основе текста: " I" и " JAVASCRIPT".
- Узла, образованным элементом strong.

Следовательно, у узла **p** в массиве **childNodes** будет 3 узла.

```
//nodeP - переменная, хранящая ссылку на узел p
nodeP.childNodes[0]; //1 дочерний узел (ребёнок) #text "I "
nodeP.childNodes[1]; //2 дочерний узел (ребёнок) strong
nodeP.childNodes[2]; //3 дочерний узел( ребёнок) #text " JAVASCRIPT«
```

Каждый узел в дереве имеет массив childNodes, даже если у него **нет** дочерних узлов. В этом случае этот массив просто **пустой**.

Отношение "родитель-ребёнок"

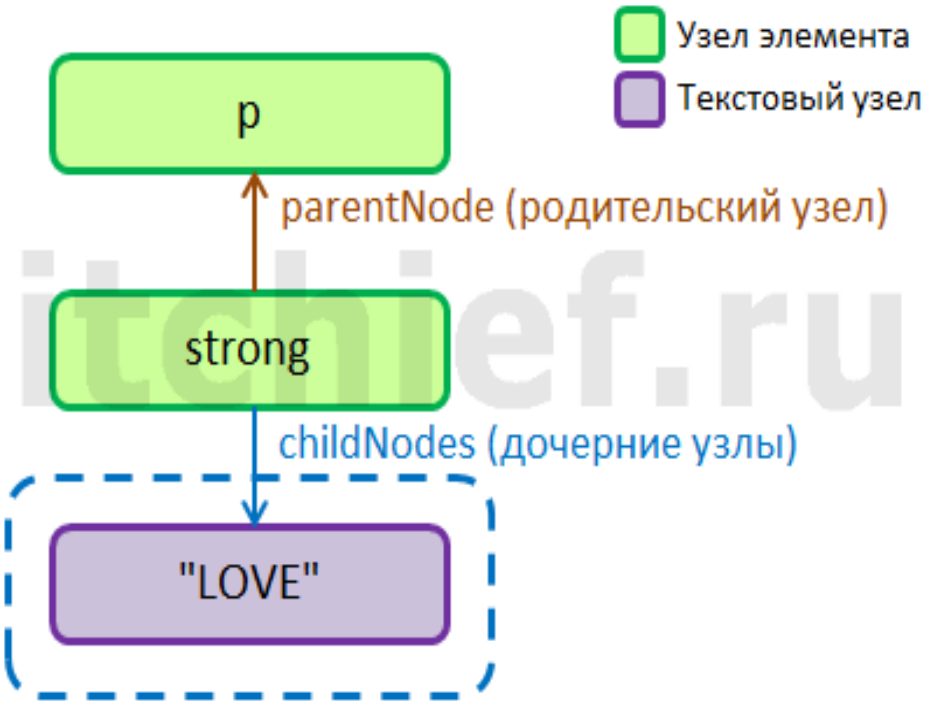
Для того чтобы обратиться к **первому** или **последнему** элементу массива childNodes в JavaScript доступны следующие свойства:

firstChild (первый элемент в массиве childNodes) и

lastChild (последний элемент в массиве childNodes).

Каждый узел в дереве имеет **родительский узел**, обратиться к которому можно с помощью свойства **parentNode**.

Например, рассмотрим узел, образованный с помощью элемента **strong**. Для него **родительским** узлом является узел p, который можно **получить** через свойство **parentNode** узла strong. Кроме родительского узла, у узла strong есть один **дочерний** узел, который является текстовым и имеет значение "LOVE". **Получить** дочерние узлы у узла p можно в виде массива **childNodes** и с помощью свойств **firstChild** и **lastChild**.



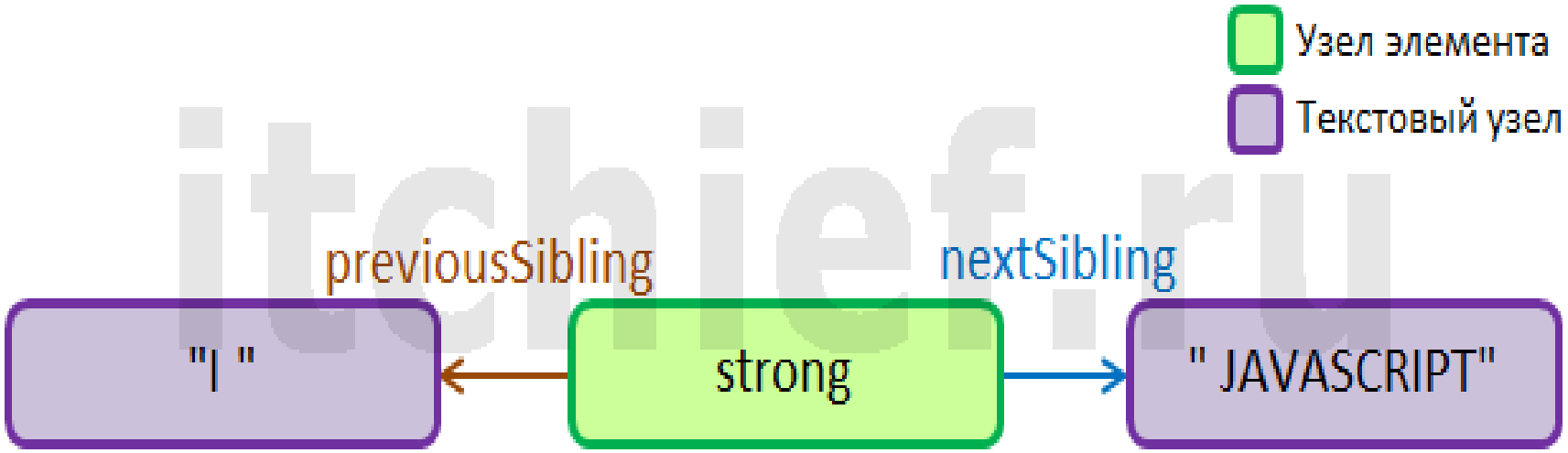
Соседние узлы (сиблинги, брат, сестра)

Для перемещения между соседними узлами в JavaScript нам доступны следующие **свойства**:

- **nextSibling** - для перемещения *слева направо*, т.е. к следующему соседу (сиблингу). Если соседа справа нет, то данное свойство возвращает значение null.
- **previousSibling** - для перемещения **справа налево**, т.е. к предыдущему соседу (сиблингу). Если соседа слева нет, то данное свойство возвращает значение null.

Соседние узлы (сиблинги, брат, сестра)

В качестве примера рассмотрим узел, образованный элементом strong. Данный узел имеет соседа слева - текстовый узел "I " (previousSibling) и соседа справа - текстовый узел " JAVASCRIPT" (nextSibling).



Дополнительные свойства узлов

Кроме рассмотренных выше свойств в JavaScript есть **дополнительные свойства**, с помощью которых можно перемещаться по узлам дерева, образованными **элементами**:

- children (возвращает коллекцию дочерних элементов (детей));
- firstElementChild (возвращает первый дочерний узел-элемент);
- lastElementChild (возвращает последний дочерний узел-элемент);
- parentElement (родительский узел-элемент);
- nextElementSibling (следующий соседний узел-элемент);
- previousElementSibling (предыдущий соседний узел-элемент).

Свойства, позволяющие войти в дерево

Но перед тем как начать перемещаться по узлам дерева нам в него необходимо как-то попасть. Войти в DOM можно с помощью document.childNodes[0], document.firstChild, document.lastChild, document.documentElement и document.body.

- Свойство document.documentElement - возвращает элемент документа, который является корневым. В HTML документах корневым элементом является элемент html. Данное свойство доступно только для чтения.
- Свойство document.body - возвращает узел body текущего документа или null, если такой элемент не существует.

Вопрос 2. Методы JavaScript для поиска элементов на странице

Методы для выбора HTML-элементов

Работа с веб-страницей так или иначе связана с манипулированием HTML-элементами. Но перед тем, как над ними выполнить некоторые действия (например, добавить стили), их сначала **нужно получить**.

Выбор элементов в основном выполняется с помощью этих методов:

- querySelector;
- querySelectorAll.

Они позволяют выполнить поиск HTML-элементов по CSS-селектору. При этом querySelector выбирает **один** элемент, а querySelectorAll – **все**.

Методы для выбора HTML-элементов

Кроме них имеются ещё:

- getElementById;
- getElementsByClassName;
- getElementsByTagName;
- getElementsByName.

Но они сейчас применяются довольно редко. В основном используется либо ***querySelector***, либо ***querySelectorAll***.

Метод querySelectorAll

Метод querySelectorAll применяется для выбора всех HTML-элементов, подходящих под **указанный CSS-селектор**.

Он позволяет искать элементы как **по всей странице**, так и **внутри** определённого элемента.

Метод querySelectorAll принимает в качестве аргумента **CSS-селектор в формате строки**, который соответственно и определяет искомые элементы.

‘#list’ – элемент с id=list

‘.list’ – элемент с class=list

‘.list.btn’ – элемент с классами list и btn

‘footer’ – тег footer

‘#list li’ – элемент с тегом li внутри элемента с id=list

'[name="phone"]‘ – элемент с атрибутом name=“phone”

В качестве результата querySelectorAll возвращает **объект класса NodeList**. Он содержит все найденные элементы.

Метод querySelectorAll

```
// выберем элементы по классу item во всем документе
const items = document.querySelectorAll('.item');
// выберем .btn внутри #slider
const buttons = document.querySelector('#slider').querySelectorAll('.btn');
```

Здесь на первой строчке мы нашли все элементы с классом item. На следующей строчке мы сначала выбрали элемент с id="slider", а затем в нём все HTML-элементы с классом btn.

Метод querySelectorAll

Полученный набор представляет собой **статическую коллекцию** HTML-элементов. Статической она называется потому, что она не изменяется.

Например, вы удалили элемент из HTML-документа, а в ней как был этот элемент, так он и остался. Чтобы обновить набор, querySelectorAll нужно вызвать заново

```
> items = document.querySelectorAll('.item')
◂ ▶ NodeList(3) [div.item, div.item, div.item]

> items[0].remove()
◂ undefined

> items
◂ ▶ NodeList(3) [div.item, div.item, div.item]

> // выберем элементы заново
  document.querySelectorAll('.item')
◂ ▶ NodeList(2) [div.item, div.item]

>
```

Метод querySelectorAll

Узнать количество найденных элементов можно с помощью **свойства length**

```
// выберем элементы с атрибутом type="submit"
const submits = document.querySelectorAll('[type="submit"]');
// получим количество найденных элементов
const countSubmits = submits.length;
```

Метод querySelectorAll

Обращение к определённому HTML-элементу коллекции выполняется также как к элементу массива, то есть по индексу. Индексы начинаются с 0:

```
const submits = document.querySelectorAll('[type="submit"]');
// получим первый элемент
const elFirst = submits[0];
// получим второй элемент
const elSecond = submits[1];
```

Здесь в качестве результата мы получаем **HTML-элемент** или **undefined**, если элемента с таким индексом в наборе NodeList нет.

Метод querySelectorAll

Перебор NodeList обычно осуществляется с помощью forEach:

```
// получим все <p> на странице
const elsP = document.querySelectorAll('p');
// переберём выбранные элементы
elsP.forEach((el) => {
  // установим каждому элементу background-color="yellow"
  el.style.backgroundColor = 'yellow';
});
```

Метод querySelectorAll

Также перебрать набор выбранных элементов можно с помощью цикла **for** или **for...of**:

```
// получим все элементы p на странице
const elsP = document.querySelectorAll('p');
// for
for (let i = 0, length = elsP.length; i < length; i++) {
  elsP[i].style.backgroundColor = 'yellow';
}
// for...of
for (let el of elsP) {
  el.style.backgroundColor = 'yellow';
}
```

Метод querySelector

Метод querySelector также как и querySelectorAll выполняет поиск по CSS-селектору. Но в отличие от него, он ищет **только один** HTML-элемент. В качестве результата этот метод **возвращает** найденный **HTML-элемент** или **null**, если он не был найден.

```
// ищем #title во всём документе
const elTitle = document.querySelector('#title');
// ищем footer в <body>
const elFooter = document.body.querySelector('footer');
```

На первой строчке мы выбираем HTML-элемент, имеющий в качестве id значение title. На второй мы ищем в <body> HTML-элемент по **тегу** footer.

Метод querySelector

Обычно перед тем, как выполнить какие-то действия с найденным HTML-элементом необходимо **сначала проверить**, а действительно ли он был найден:

```
const elModal = document.querySelector('.modal');
// если элемент .modal найден, то ...
if (elModal) {
  // переключим у elModal класс show
  elModal.classList.toggle('show');
}
```

Здесь мы сначала проверили существования HTML-элемента, и только потом выполнили над ним некоторые действия.

Метод getElementById

Метод getElementById позволяет найти HTML-элемент на странице по значению id:

```
<div id="comments">...</div>
...
<script>
  // получим HTMLElement и сохраним его в переменную elComments
  const elComments = document.getElementById('comments');
</script>
```

В качестве результата getElementById возвращает объект класса **HTMLElement** или значение **null**, если элемент не был найден. Этот метод имеется только **у объекта document**.

Метод getElementsByClassName

Метод getElementsByClassName позволяет найти все элементы с заданным **классом** или **классами**. Его можно применить для поиска элементов как во **всём документе**, так и **внутри указанного**. В первом случае его нужно будет вызывать как метод **объекта document**, а во втором – как метод соответствующего **HTML-элемента**

```
// найдем элементы с классом control в документе
const elsControl = document.getElementsByClassName('control');
// выберем элементы внутри другого элемента, в данном случае внутри формы с id="myform"
const elsFormControl = document.forms.myform.getElementsByClassName('form-control');
// выберем элементы .btn.btn-danger
const elsBtn = document.getElementsByClassName('btn btn-danger');
```

Метод getElementsByTagName

Метод getElementsByTagName предназначен для получения коллекции элементов по имени тега:

```
// найдем все <a> в документе
const anchors = document.getElementsByTagName('a');
// найдем все <li> внутри #list
const elsLi = document.getElementById('list').getElementsByTagName('li');
```

Метод getElementsByName

В JavaScript getElementsByName можно использовать для выбора элементов, имеющих определенное **значение** атрибута **name**:

```
// получим все элементы с name="phone"
const elsPhone = document.getElementsByName('phone');
```

Метод matches позволяет проверить HTML-элемент на соответствие CSS-селектору. Он возвращает true, если элемент ему соответствует, иначе false.

```
// выберем HTML элемент, имеющий атрибут data-target="slider"
const elSlider = document.querySelector('[data-target="slider"]');
// проверим соответствует ли он CSS селектору 'div'
const result = elSlader.matches('div');
```

Метод closest позволяет найти **ближайшего предка**, подходящего под указанный CSS-селектор. При этом поиск начинается с самого элемента, для которого данный метод вызывается. Если этот элемент будет ему соответствовать, то closest вернёт его.

```
<div class="level-1">
  <div class="level-2">
    <div class="level-3"></div>
  </div>
</div>

<script>
  const el = document.querySelector('.level-3');
  const elAncestor = el.closest('.level-1');
  console.log(elAncestor);
</script>
```

Метод contains позволяет проверить **содержит** ли некоторый узел другой в качестве **потомка**. При этом проверка начинается с самого этого узла, для которого этот метод вызывается.

Если узел соответствует тому для которого мы вызываем данный метод или является его потомком, то contains в качестве результата возвращает логическое значение true. В противном случае false