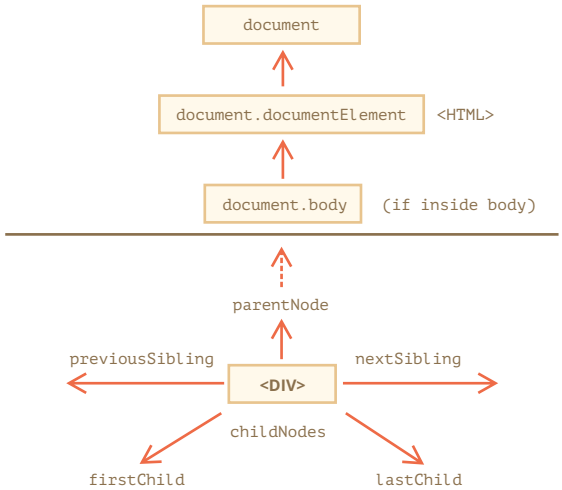
Навигация по DOM-элементам

DOM позволяет нам делать что угодно с элементами и их содержимым, но для начала нужно получить соответствующий DOM-объект.

Все операции с DOM начинаются с объекта `document`. Это главная «точка входа» в DOM. Из него мы можем получить доступ к любому узлу.

Так выглядят основные ссылки, по которым можно переходить между узлами DOM:



Поговорим об этом подробнее.

Сверху: documentElement и body

Самые верхние элементы дерева доступны как свойства объекта `document`:

`<html> = document.documentElement`

Самый верхний узел документа: `document.documentElement`. В DOM он соответствует тегу `<html>`.

`<body> = document.body`

Другой часто используемый DOM-узел – узел тега `<body>`: `document.body`.

`<head> = document.head`

Тег `<head>` доступен как `document.head`.

Есть одна тонкость: `document.body` может быть равен `null`

Нельзя получить доступ к элементу, которого ещё не существует в момент выполнения скрипта.

В частности, если скрипт находится в `<head>`, `document.body` в нём недоступен, потому что браузер его ещё не прочитал.

Поэтому, в примере ниже первый `alert` выведет `null`:

```
<html>

<head>
  <script>
    alert( "Из HEAD: " + document.body ); // null,
  </script>
</head>

<body>

  <script>
    alert( "Из BODY: " + document.body ); // HTMLE
  </script>

</body>
</html>
```

В мире DOM `null` означает «не существует»

В DOM значение `null` значит «не существует» или «нет такого узла».

Дети: childNodes, firstChild, lastChild

Здесь и далее мы будем использовать два принципиально разных термина:

- **Дочерние узлы (или дети)** – элементы, которые являются непосредственными детьми узла. Другими словами, элементы, которые лежат непосредственно внутри данного. Например, `<head>` и `<body>` являются детьми элемента `<html>`.
- **Потомки** – все элементы, которые лежат внутри данного, включая детей, их детей и т.д.

В примере ниже детьми тега `<body>` являются теги `<div>` и `<ul>` (и несколько пустых текстовых узлов):

```
<html>
<body>
  <div>Начало</div>

  <ul>
    <li>
      <b>Информация</b>
    </li>
  </ul>
</body>
</html>
```

...А потомки `<body>` – это и прямые дети `<div>`, `<ul>` и вложенные в них: `<li>` (потомок `<ul>`) и `<b>` (потомок `<li>`) – в общем, все элементы поддерева.

Коллекция `childNodes` содержит список всех детей, включая текстовые узлы.

Пример ниже последовательно выведет детей `document.body`:


```
<html>
<body>
  <div>Начало</div>

  <ul>
    <li>Информация</li>
  </ul>

  <div>Конец</div>

  <script>
    for (let i = 0; i < document.body.childNodes.length
      alert( document.body.childNodes[i] ); // Text, DI
    }
  </script>
  ...какой-то HTML-код...
</body>
</html>
```

Обратим внимание на маленькую деталь. Если запустить пример выше, то последним будет выведен элемент `<script>`. На самом деле, в документе есть ещё «какой-то HTML-код», но на момент выполнения скрипта браузер ещё до него не дошёл, поэтому скрипт не видит его.

Свойства `firstChild` и `lastChild` обеспечивают быстрый доступ к первому и последнему дочернему элементу.

Они, по сути, являются всего лишь сокращениями. Если у тега есть дочерние узлы, условие ниже всегда верно:

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.la
```

Для проверки наличия дочерних узлов существует также специальная функция `elem.hasChildNodes()`.

DOM-коллекции

Как мы уже видели, `childNodes` похож на массив. На самом деле это не массив, а *коллекция* – особый перебираемый объект-псевдомассив.

И есть два важных следствия из этого:

1. Для перебора *коллекции* мы можем использовать `for..of`:

```
for (let node of document.body.childNodes) {
  alert(node); // покажет все узлы из коллекции
}
```

Это работает, потому что коллекция является перебираемым объектом (есть требуемый для этого метод `Symbol.iterator`).

2. Методы массивов не будут работать, потому что коллекция – это не массив:

```
alert(document.body.childNodes.filter); // undefined (у
```

Первый пункт – это хорошо для нас. Второй – бывает неудобен, но можно пережить. Если нам хочется использовать именно методы массива, то мы можем создать настоящий массив из коллекции, используя `Array.from`:

```
alert( Array.from(document.body.childNodes).filter ); /
```

DOM-коллекции – только для чтения

DOM-коллекции, и даже более – *все* навигационные свойства, перечисленные в этой главе, доступны только для чтения.

Мы не можем заменить один дочерний узел на другой, просто написав `childNodes[i] = ...`.

Для изменения DOM требуются другие методы. Мы увидим их в следующей главе.

DOM-коллекции живые

Почти все DOM-коллекции, за небольшим исключением, *живые*. Другими словами, они отражают текущее состояние DOM.

Если мы сохраним ссылку на `elem.childNodes` и добавим/удалим узлы в DOM, то они появятся в сохранённой коллекции автоматически.

Не используйте цикл `for..in` для перебора коллекций

Коллекции перебираются циклом `for..of`. Некоторые начинающие разработчики пытаются использовать для этого цикл `for..in`.

Не делайте так. Цикл `for..in` перебирает все перечисляемые свойства. А у коллекций есть некоторые «лишние», редко используемые свойства, которые обычно нам не нужны:

```
<body>
<script>
  // выводит 0, 1, length, item, values и другие с
  for (let prop in document.body.childNodes) alert
</script>
</body>
```

Соседи и родитель

Соседи – это узлы, у которых один и тот же родитель.

Например, здесь `<head>` и `<body>` соседи:

```
<html>
  <head>...</head><body>...</body>
</html>
```

- говорят, что `<body>` – «следующий» или «правый» сосед `<head>`
- также можно сказать, что `<head>` «предыдущий» или «левый» сосед `<body>`.

Следующий узел того же родителя (следующий сосед) – в свойстве `nextSibling`, а предыдущий – в `previousSibling`.

Родитель доступен через `parentNode`.

Например:

```
// родителем <body> является <html>
alert( document.body.parentNode === document.documentEl

// после <head> идёт <body>
alert( document.head.nextSibling ); // HTMLBodyElement


```

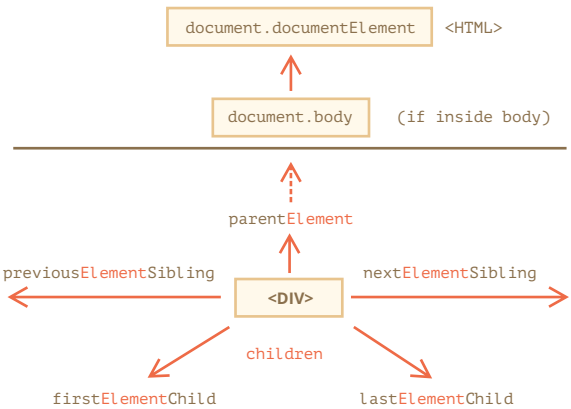
```
8  // перед <body> находится <head>
   alert( document.body.previousSibling ); // HTMLHeadElem
```


Навигация только по элементам

Навигационные свойства, описанные выше, относятся ко *всем* узлам в документе. В частности, в `childNodes` находятся и текстовые узлы и узлы-элементы и узлы-комментарии, если они есть.

Но для большинства задач текстовые узлы и узлы-комментарии нам не нужны. Мы хотим манипулировать узлами-элементами, которые представляют собой теги и формируют структуру страницы.

Поэтому давайте рассмотрим дополнительный набор ссылок, которые учитывают только *узлы-элементы*:



Эти ссылки похожи на те, что раньше, только в ряде мест стоит слово `Element`:

- `children` – коллекция детей, которые являются элементами.
- `firstElementChild`, `lastElementChild` – первый и последний дочерний элемент.
- `previousElementSibling`, `nextElementSibling` – соседи-элементы.
- `parentElement` – родитель-элемент.

Зачем нужен `parentElement`? Разве может родитель быть *не* элементом?

Свойство `parentElement` возвращает родитель-элемент, а `parentNode` возвращает «любого родителя». Обычно эти свойства одинаковы: они оба получают родителя.

За исключением `document.documentElement`:

```
1  alert( document.documentElement.parentNode ); // 
2  alert( document.documentElement.parentElement ); 
```

Причина в том, что родителем корневого узла `document.documentElement` (`<html>`) является `document`. Но `document` – это не узел-элемент, так что `parentNode` вернёт его, а `parentElement` нет.

Эта деталь может быть полезна, если мы хотим пройти вверх по цепочке родителей от произвольного элемента `elem` к `<html>`, но не до `document`:

```
1  while(elem = elem.parentElement) { // идти наверх
2    alert( elem );
3  }
```

Изменим один из примеров выше: заменим `childNodes` на `children`. Теперь цикл выводит только элементы:

```
<html>
<body>
  <div>Начало</div>

  <ul>
    <li>Информация</li>
  </ul>

  <div>Конец</div>

  <script>
    for (let elem of document.body.children) {
      alert(elem); // DIV, UL, DIV, SCRIPT
    }
  </script>
  ...
</body>
</html>
```

Ещё немного ссылок: таблицы

До сих пор мы описывали основные навигационные ссылки.

Некоторые типы DOM-элементов предоставляют для удобства дополнительные свойства, специфичные для их типа.

Таблицы – отличный пример таких элементов.

Элемент `<table>`, в дополнение к свойствам, о которых речь шла выше, поддерживает следующие:

- `table.rows` – коллекция строк `<tr>` таблицы.
- `table.caption/tHead/tFoot` – ссылки на элементы таблицы `<caption>`, `<thead>`, `<tfoot>`.
- `table.tBodies` – коллекция элементов таблицы `<tbody>` (по спецификации их может быть больше одного).

`<thead>`, `<tfoot>`, `<tbody>` предоставляют свойство `rows`:

- `tbody.rows` – коллекция строк `<tr>` секции.

`<tr>`:

- `tr.cells` – коллекция `<td>` и `<th>` ячеек, находящихся внутри строки `<tr>`.
- `tr.sectionRowIndex` – номер строки `<tr>` в текущей секции `<thead>/<tbody>/<tfoot>`.
- `tr.rowIndex` – номер строки `<tr>` в таблице (включая все строки таблицы).

`<td>` and `<th>`:

- `td.cellIndex` – номер ячейки в строке `<tr>`.

Пример использования:

```
<table id="table">
  <tr>
    <td>один</td><td>два</td>
  </tr>
  <tr>
    <td>три</td><td>четыре</td>
  </tr>
</table>

<script>
  // выводит содержимое первой строки, второй ячейки
  alert( table.rows[0].cells[1].innerHTML ) // "два"
</script>
```

Спецификация: tabular data.

Существуют также дополнительные навигационные ссылки для HTML-форм. Мы рассмотрим их позже, когда начнём работать с формами.

Итого

Получив DOM-узел, мы можем перейти к его ближайшим соседям используя навигационные ссылки.

Есть два основных набора ссылок:

- Для всех узлов: `parentNode`, `childNodes`, `firstChild`, `lastChild`, `previousSibling`, `nextSibling`.
- Только для узлов-элементов: `parentElement`, `children`, `firstElementChild`, `lastElementChild`, `previousElementSibling`, `nextElementSibling`.

Некоторые виды DOM-элементов, например таблицы, предоставляют дополнительные ссылки и коллекции для доступа к своему содержимому.

Задачи

Дочерние элементы в DOM

важность: 5

Для страницы:

```
<html>
<body>
  <div>Пользователи:</div>
  <ul>
    <li>Джон</li>
    <li>Пит</li>
  </ul>
</body>
</html>
```

Напишите код, как получить...

- элемент `<div>`?
- `<ul>`?
- второй `<li>` (с именем Пит)?

решение

Вопрос о соседях

важность: 5

Если `elem` – произвольный узел DOM-элемента...

- Правда, что `elem.lastChild.nextSibling` всегда равен `null`?
- Правда, что `elem.children[0].previousSibling` всегда равен `null`?

решение

Выделите ячейки по диагонали

важность: 5

Напишите код, который выделит красным цветом все ячейки в таблице по диагонали.

Вам нужно получить из таблицы `<table>` все диагональные `<td>` и выделить их, используя код:

```
1  //  в переменной td находится DOM-элемент для тега <td>
2  td.style.backgroundColor = 'red';
```

Должно получиться так:

1:1	2:1	3:1	4:1	5:1
1:2	2:2	3:2	4:2	5:2
1:3	2:3	3:3	4:3	5:3
1:4	2:4	3:4	4:4	5:4
1:5	2:5	3:5	4:5	5:5

Открыть песочницу для задачи.

решение

Комментарии

перед тем как писать...