НЕТОЛОГИЯ

ИЗМЕНЕНИЕ СТРУКТУРЫ HTML-ДОКУМЕНТА

ВЛАДИМИР ЧЕБУКИН

ВЛАДИМИР ЧЕБУКИН

Frontend-разработчик

vovachebr@mail.ru fb.me/vovachebr @User123423

ПЛАН ЗАНЯТИЯ

1. Содержимое HTML-элемента
2. Создание HTML-элементов
3. Удаление и добавление HTML-элементов
4. Замена и вставка HTML-элементов
5. Клонирование HTML-элементов

ВОПРОСЫ С ПРЕДЫДУЩЕГО ЗАНЯТИЯ

1. Какие существуют основные типы полей формы и какие их наиболее важные события?
2. С помощью какого события форму отправляют на сервер?
3. Какой существует стандартный механизм валидации форм?

ОТВЕТЫ НА ВОПРОСЫ С ПРЕДЫДУЩЕГО ЗАНЯТИЯ

1.

Типы полей	**input text, textarea**	**select, radio, checkbox**
События	input, change, focus, blur	change, (focus, blur)

2. Submit
3. JavaScript Validation API, метод checkValidity

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: как получить значения элементов ввода?

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Ответ: у каждого элемента ввода есть свойство value, которое содержит текущее значение элемента ввода.

```
let nameInput = document.getElementsByTagName("input").namedItem("personName");
console.log(nameInput.value);
console.log(nameInput['value']);
```

ПОЛУЧЕНИЕ ТЕКСТА ВНУТРИ ЭЛЕМЕНТА

Иногда нужно получить текст, находящийся внутри элемента.
Для этого у каждого элемента есть свойство textContent, которое позволяет получить текстовый контент указанного узла и всех его потомков.

> Данное значение можно представить как сложение всех текстовых узлов, которые являются потомками узла, для которого вызывается данное свойство. Если элемент, для которого вызывается свойство textContent, содержит один дочерний текстовый узел, то данное свойство вернёт значение этого HTML-элемента.

СВОЙСТВО INNERTEXT

В отличие от свойства textContent, свойство innerText «как бы копирует» текст в веб-браузере, который отображается HTML-кодом, расположенным между открывающим и закрывающим тегом того элемента, для которого вызывается данное свойство. Т.е. свойство innerText учитывает стили элементов, а именно — отображается элемент или нет, а, следовательно, и его содержимое в браузере.

> Данное свойство также позволяет установить элементу заданный текстовый контент, т.е. заменить содержимое элемента, расположенное между его открывающим и закрывающим тегом, на указанное.

СВОЙСТВО OUTERTEXT

Кроме свойства innerText также существует свойство outerText, которое возвращает текст аналогично свойству innerText. А вот при установлении значения свойству outerText для элемента, данное свойство заменяет не только содержимое, расположенное между открывающим и закрывающим тегом элемента, но и сам элемент.

РАЗЛИЧИЯ INNERTEXT И OUTERTEXT

Допустим есть разметка:

```
<div>
  <p>Old text</p>
</div>
```

При выполнении `p.innerText = "New Text!"` получим:

```
<div>
  <p>New Text!</p><!--Изменился внутренний текст-->
</div>
```

При выполнении `p.outerText = "New Text!"` получим:

```
<div>
  New Text!<!--Изменился весь элемент-->
</div>
```

INNERHTML И OUTERHTML

Иногда нужно получить или заменить не текст внутри элемента,
а его содержимое.

Это пригодится в ДЗ

Для таких случаев существуют свойства innerHTML и outerHTML

Эти свойства схожи с innerText и outerText, только возвращают HTML-разметку.

ПРИМЕР

```
<b><i>Super</i>Text!</b>
```

```
let b = document.getElementsByTagName("b")[0];
console.log(b.innerHTML);//<i>Super</i>Text!
console.log(b.outerHTML);//<b><i>Super</i>Text!</b>
```

РАЗЛИЧИЯ МЕЖДУ INNERTEXT И INNERHTML

Допустим, у нас есть разметка:

```
<div>
  <p>Old text</p>
</div>
```

Вопрос: что будет если мы применим следующую операцию?

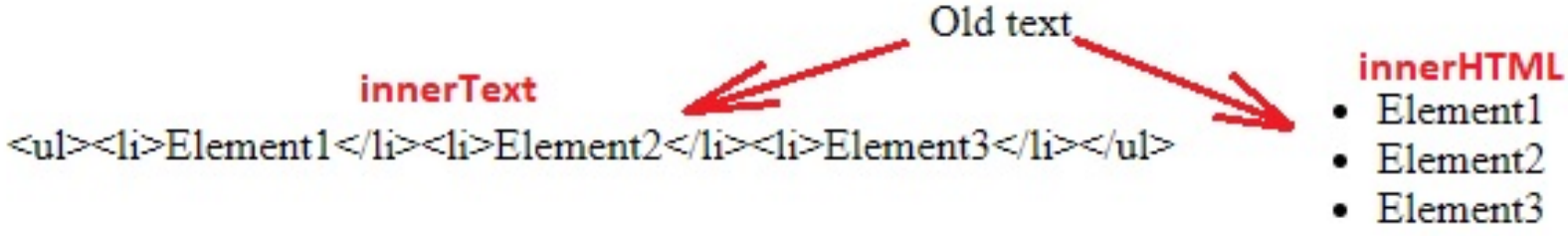
```
p.innerText = "<ul><li>Element1</li><li>Element2</li><li>Element3</li></ul>"
```

а если следующее присвоение?

```
p.innerHTML = "<ul><li>Element1</li><li>Element2</li><li>Element3</li></ul>"
```

ЗАКРЕПЛЯЕМ ЗНАНИЯ

Ответ:



РЕЗЮМИРУЕМ

Итого:

inner* — что находится между тегами элемента.

outer* — содержимое элемента, включая теги.

*Text — текстовые значения.

*HTML — HTML-разметка.

МЕТОД INSERTADJACENTHTML

Иногда нужно не заменить всё, что находится внутри HTML-элемента, а добавить каку-либо разметку. Для таких случаев используется метод insertAdjacentHTML().

> Метод insertAdjacentHTML() более гибкий, чем свойство innerHTML. При присвоении свойства innerHTML у элемента меняется вся внутренняя разметка, а при использовании метода insertAdjacentHTML() разметка добавляется в указанную позицию.

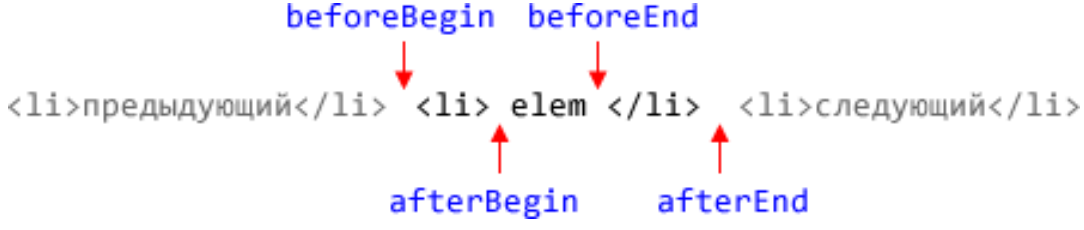
МЕТОД INSERTADJACENTHTML

Синтаксис метода выглядит так:

```
elem.insertAdjacentHTML(where, html);
```

elem — элемент, которому добавляется разметка
where — позиция, куда добавляется разметка
html — вставляемая разметка

Значение where может быть одним из нескольких значений:
beforeBegin — перед elem
afterBegin — внутрь elem, в самое начало
beforeEnd — внутрь elem, в конец
afterEnd — после elem



МЕТОД INSERTADJACENTHTML

(на двух слайдах)

Рассмотрим пример:

```
<ul class="exclusive">
  <li>Element 1</li>
  <li>Element 2</li>
  <li>Element 3</li>
</ul>
```

При выполнении следующих команд:

```
let li = document.getElementsByTagName("li");
li[0].insertAdjacentHTML("afterBegin","<a href='http://google.com'>Google</a>");
li[1].insertAdjacentHTML("beforeBegin","<li>Element 1.5</li>");
li[2].insertAdjacentHTML("beforeEnd","<a href='http://google.com'>Google</a>");
```

МЕТОД INSERTADJACENTHTML

(продолжение)

получим следующую разметку

```
<ul class="exclusive">
  <li>
    <a href="http://google.com">Google</a>
    Element 1
  </li>
  <li>Element 1.5</li>
  <li>Element 2</li>
  <li>
    Element 3
    <a href="http://google.com">Google</a>
  </li>
</ul>
```

БРАТЬЯ МЕТОДА INSERTADJACENTHTML

Как и с innerText/innerHTML, для метода insertAdjacentHTML() есть схожие методы:

- insertAdjacentText() — добавляет текст в позицию другого элемента
- insertAdjacentElement() — добавляет элемент в позицию другого элемента

> По поводу создания элементов поговорим чуть позже.

УПРАВЛЕНИЕ HTML-ЭЛЕМЕНТАМИ

Для манипулирования HTML-элементами рассмотрим следующие методы:

1. document.createElement
2. removeChild / remove
3. appendChild
4. insertBefore
5. replaceChild
6. cloneNode

СОЗДАНИЕ ЭЛЕМЕНТОВ

Иногда нужно создать элемент. Для чего это нужно?
Например, для дальнейшей вставки его с помощью метода insertAdjacentElement().

Это пригодится в ДЗ

```
let element = document.createElement('tag_name');
```

Созданным элементом можно пользоваться так же, как и с любым элементом, полученным из DOM-дерева.

> Отличие между созданным элементом и элементом из DOM-дерева в том, что элемент в DOM-дереве уже там находится и скорее всего отображается на странице.

СОЗДАНИЕ ТЕКСТОВЫХ ЭЛЕМЕНТОВ

Для создания текстового элемента существует метод createTextNode(), который похож на метод createElement() и отличается только тем, что создает не HTML-узел, а текстовый узел.

```
let element = document.createTextNode('simple text');
```

УДАЛЕНИЕ ЭЛЕМЕНТОВ ИЗ DOM-ДЕРЕВА

Иногда приходится удалять элементы из DOM-дерева. Для таких случаев существует метод removeChild().

> Метод removeChild() вызывается на элементе, у которого необходимо удалить дочерний элемент. Аргументом функции является дочерний (удаляемый) элемент.

```
let deletableElement = document.getElementsByClassName("deletable")[0];
let childOfDeletable = deletableElement.childNodes[0];// элемент для удаления
deletableElement.removeChild(childOfDeletable);
```

УДАЛЕНИЕ ПОТОМКА ЭЛЕМЕНТА ИЗ DOM ДЕРЕВА

Чтобы полностью удалить элемент, можно вызвать метод remove()

```
let deletableElement = document.getElementsByClassName("deletable")[0];
deletableElement.remove(); // удалить элемент
```

ДОБАВЛЕНИЕ КОНТЕНТА

Если существует удаление элемента, то должно существовать и добавление элемента. Для таких случаев существует метод appendChild().

> Метод node.appendChild(someNode) добавляет узел someNode в качестве последнего дочернего узла у узла node. Если узел someNode уже находился в DOM-дереве, то он будет перемещен из предыдущего места. Аналогично appendChild() существует метод для вставки текста — append().

Это пригодится в ДЗ

```
let b = document.getElementsByTagName("b")[0];
let deletableElement = document.getElementsByClassName("deletable")[0];
b.append("Inserted text");// Добавляем текст
b.appendChild(deletableElement);// Добавляем HTML-элемент
```

ДОБАВЛЕНИЕ ЭЛЕМЕНТА ПЕРЕД НУЖНЫМ ЭЛЕМЕНТОМ

Методы append() и appendChild() добавляют контент после нужного элемента. Для добавления перед нужным элементом существует метод insertBefore().

```
parentElem.insertBefore(elem, nextSibling)
```

Вставляет elem в коллекцию детей parentElem перед элементом nextSibling.

Если вторым аргументом указать null, то insertBefore сработает как appendChild

ПРИМЕР ДОБАВЛЕНИЯ ЭЛЕМЕНТА ПЕРЕД НУЖНЫМ ЭЛЕМЕНТОМ

(на двух слайдах)

Допустим есть разметка:

```
<b>
  <i>Super</i>
  Text!Inserted text
  <div class="deletable">deleted</div>
</b>
```

При добавлении

```
b.insertBefore(document.createTextNode('Вставляемый_текст'),b.children[0]);
```

ПРИМЕР ДОБАВЛЕНИЯ ЭЛЕМЕНТА ПЕРЕД НУЖНЫМ ЭЛЕМЕНТОМ

(продолжение)

Получим

```
<b>
  Вставляемый_текст
  <i>Super</i>
  Text!Inserted text
  <div class="deletable">deleted</div>
</b>
```

ЗАМЕНА ЭЛЕМЕНТА

В случае необходимости замены HTML-элемента следует воспользоваться методом replaceChild().

```
replacedNode = parentNode.replaceChild(newChild, oldChild);
```

parentNode — элемент, у которого будет заменяться дочерний элемент
oldChild — элемент, который будет убран
newChild — элемент, который будет добавлен на место прошлого
replacedNode — замененный элемент (тоже самое, что и oldChild)

ПРИМЕР ЗАМЕНЫ ЭЛЕМЕНТА

Допустим, есть разметка

```
<b>
  <i>Super</i>
  Text!Inserted text
  <div class="deletable">deleted</div>
</b>
```

тогда при изменении:

```
let div = document.createElement('div');
b.replaceChild(div, b.children[0]);
```

получим:

```
<b>
  <div></div>
  Text!Inserted text
  <div class="deletable">deleted</div>
</b>
```

ЗАМЕНА САМОГО ЭЛЕМЕНТА

Если нужно заменить сам элемент, то можно использовать метод replaceWith(), который используется на заменяемом элементе и в качестве аргумента принимает элемент, на который будет заменён.

ПРИМЕР ЗАМЕНЫ ЭЛЕМЕНТА

Допустим, есть разметка

```
<b>
  <div></div>
  Text!Inserted text
  <div class="deletable">deleted</div>
</b>
```

тогда при изменении:

```
let div = document.createElement('div');
b.replaceWith(div);
```

получим:

```
<div></div>
```

ИЗМЕНЕНИЕ DOM-ДЕРЕВА

Вопрос: Допустим, есть список, в который нужно добавить несколько элементов. Можно сделать цикл, который на каждой итерации будет добавлять элемент списка в список или сгенерировать коллекцию элементов списка, а затем их все разом добавить. Есть ли разница в этих подходах, если да, то какая? Какой подход предпочтительней и почему?

ИЗМЕНЕНИЕ DOM-ДЕРЕВА

Каждое изменение DOM-дерева приводит к перерисовке всей страницы. А это одна из самых **продолжительных** операций.

Поэтому при большом количестве изменений DOM-дерева могут возникнуть проблемы с производительностью!

КЛОНИРОВАНИЕ ЭЛЕМЕНТА

Иногда приходится клонировать некоторый элемент. Для таких случаев существует метод cloneNode()

> let clonedNode = node.cloneNode(deep);
> Данный метод делает копию элемента node. Если аргумент deep присутствует и равен true, то копируются и дочерние элементы (если false, то копируется только сам элемент).

```
let ul = document.getElementsByTagName("ul")[0];
//Копирование с дочерними элементами
console.log(ul.cloneNode(true));// <ul class="exclusive">...</ul>
//Копирование списка без элементов
console.log(ul.cloneNode(false));// <ul class="exclusive"></ul>
```

РЕЗЮМИРУЕМ

- Метод document.createElement служит для создания элементов.
- Методы removeChild() и remove() используются для удаления элемента или его потомков.
- Метод appendChild() и append() Служат для добавления различного содержимого к элементам.
- Метод insertBefore() используется для добавления внутрь элемента перед другими.
- Метод replaceChild() и replace() служат для изменения или замены элемента.
- Метод cloneNode() служит для копирования элемента.
- Каждое изменение DOM-дерева ведет к перерисовке всей веб-страницы.

ЛИСТИНГ КОДА

Весь код, используемый в лекции доступен по ссылке
https://repl.it/@vovachebr/ChangeStructureOfHTMLElements

ЧЕМУ МЫ НАУЧИЛИСЬ?

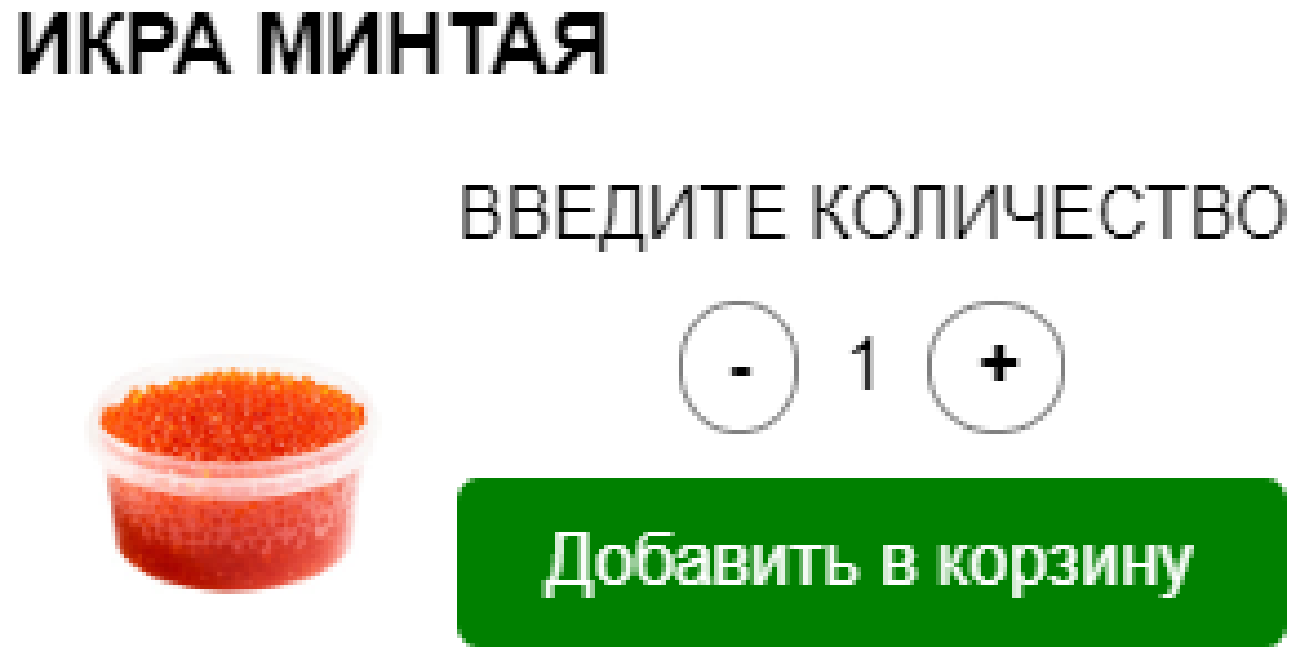
1. Изменять содержимое HTML-элементов
2. Изменять сами HTML-элементы
3. Заменять HTML-элементы
4. Удалять и добавлять новые HTML-элементы
5. Клонировать HTML-элементы

ЧТО МЫ УЗНАЕМ НА СЛЕДУЮЩЕЙ ЛЕКЦИИ

1. Принципы асинхронности
2. HTTP-протокол
3. XMLHttpRequest для асинхронной передачи информации между сервером и клиентом
4. Отправка формы

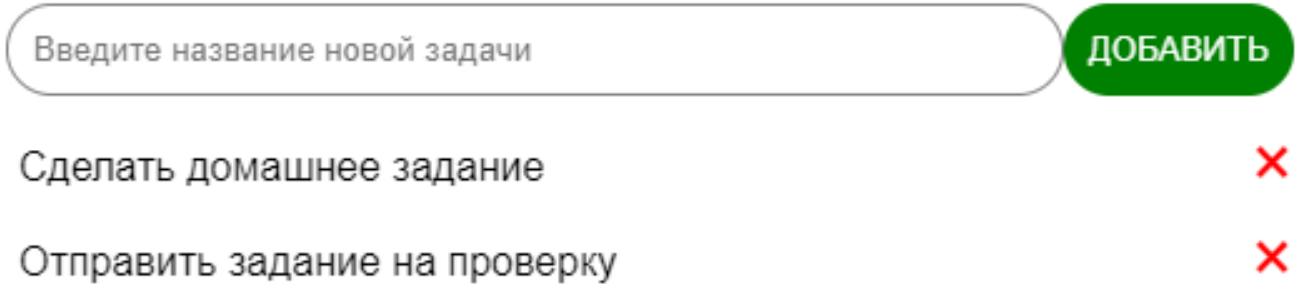
ДОМАШНЕЕ ЗАДАНИЕ

1. Управление количеством позиций, которое нужно добавить в корзину (- количество +). Позиций может быть несколько на странице. После нажатия на кнопку «Добавить в корзину», товар перемещается в корзину с эффектом



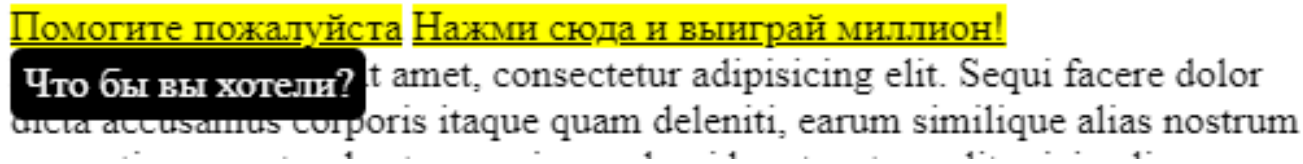
ДОМАШНЕЕ ЗАДАНИЕ

2. Простой список дел с возможностью добавления и удаления значений.



ДОМАШНЕЕ ЗАДАНИЕ

3. Визуальные подсказки



МАТЕРИАЛ ДЛЯ САМОСТОЯТЕЛЬНОГО ОЗНАКОМЛЕНИЯ

Навигация по DOM-элементам

Добавление и удаление узлов

Мультивставка: insertAdjacentHTML и DocumentFragment

Изменение страницы посредством DOM

Изменение элементов с помощью DOM (youtube)

Добавление и удаление элементов с помощью DOM (youtube)

ВОПРОСЫ ДЛЯ СЛЕДУЮЩЕГО ЗАНЯТИЯ

1. Какого типа аргумент функции cloneNode() и для чего он нужен?
2. Можно ли использовать метод insertBefore() как appendChild(). Если можно, то как?
3. Как часто происходит перерисовка веб-страницы?

ОТВЕТЫ НА ВОПРОСЫ ДЛЯ СЛЕДУЮЩЕГО ЗАНЯТИЯ

1. let clonedNode = node.cloneNode(deep);// deep должен быть логического типа. Если deep === true, то копируется элемент вместе с его потомками, в ином случае копируется только сам элемент.
2. Если вторым аргументом указать null, то insertBefore сработает как appendChild.
3. Как минимум при каждом изменении DOM-дерева.

НЕТОЛОГИЯ

Спасибо за внимание!

Время задавать вопросы ☺

ВЛАДИМИР ЧЕБУКИН


vovachebr@mail.ru


fb.me/vovachebr


@User123423