# Практическая работа 11. Работа с деревом DOM: изменение элементов html-страницы. Исследование объектной модели документа в браузере.

Цель работы: изучить браузерную модель документа и освоить способы доступа к элементам веб-страницы и их изменения средствами языка JavaScript.

## Описание работы

### КАК ПОЛУЧИТЬ ИЛИ ОБНОВИТЬ СОДЕРЖИМОЕ ЭЛЕМЕНТА

В прошлой работе вы научились находить элементы в дереве DOM. В этой работе вы научитесь получать доступ к **содержимому элемента и обновлять его**. Конкретные приемы работы зависят от конкретного контента.

|  |  |
| --- | --- |
|  | Этот элемент **li** имеет:  • один элемент-потомок, содержащий единственное слово «инжир», которое вы видите в списке;  • узел атрибута, содержащий атрибут id. |
|  |
|  | В этом элементе **li:**  • Добавляется элемент **em**. Он становится **первым** потомком.  • У элемента **em** есть свой **дочерний** текстовый узел, где содержится слово «свежий».  • Текстовый узел со словом «инжир» теперь становится смежным тому узлу, который представляет элемент **em**. |
|  |
|  | Когда текст добавляется перед элементом **em**:  • первый дочерний узел элемента **li** — это текстовый узел, содержащий слово «самый»;  • у него есть смежный элемент — узел элемента em; в свою очередь, у **em** есть дочерний текстовый узел, содержащий слово «свежий»;  • наконец, есть еще текстовый узел со словом «инжир», являющийся смежным как для текстового узла со словом «самый», так и для узла элемента **em**. |
|  |

Давайте рассмотрим три примера элементов li в этой таблице. Каждый из них добавляет еще немного разметки, потому фрагменты дерева DOM каждого из этих элементов значительно отличаются.

• Первый элемент списка содержит обычный текст.

• Второй и третий элементы содержат и текст, и теги em.

Как видите, даже при добавлении очень простого элемента em структура дерева DOM существенно изменяется. Соответственно, подобные изменения могут повлиять на то, как вы станете работать с этим элементом списка. Когда элемент содержит не только текст, но и другие элементы, вы, скорее всего, будете работать с элементом-предком, а не с каждым отдельным его узлом-потомком.

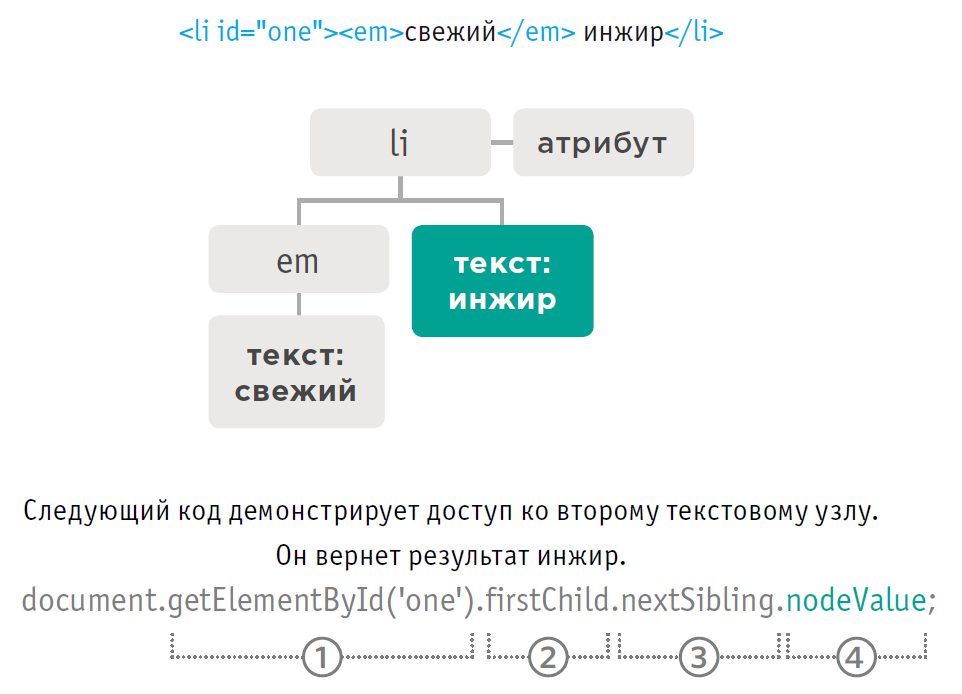
### 2. Работа с объемлющим элементом

Когда вы работаете с узлом элемента (а не с его текстовым узлом), этот элемент может содержать разметку. Приходится выбирать, что вы хотите сделать: извлечь (получить) разметку или установить (обновить) ее, а также и текст. При использовании свойств, перечисленных ниже, новый контент затирает все прежнее содержимое элемента (как текст, так и разметку). Например, если вы примените любое из этих свойств для обновления содержимого элемента body, вы тем самым обновите всю веб-страницу.

|  |  |
| --- | --- |
| СВОЙСТВО | ОПИСАНИЕ |
| innerHTML | Устанавливает/получает текст и разметку |
| textContent | Устанавливает/получает только текст |
| innerText | Устанавливает/получает только текст |

### 3. ДОСТУП К ТЕКСТОВОМУ УЗЛУ И ОБНОВЛЕНИЕ ЕГО СОДЕРЖИМОГО ПРИ ПОМОЩИ СВОЙСТВА NODEVALUE

Можно извлечь или изменить содержимое выделенного текстового узла при помощи свойства nodeValue.



① Узел элемента li выделяется при помощи метода getElementById().

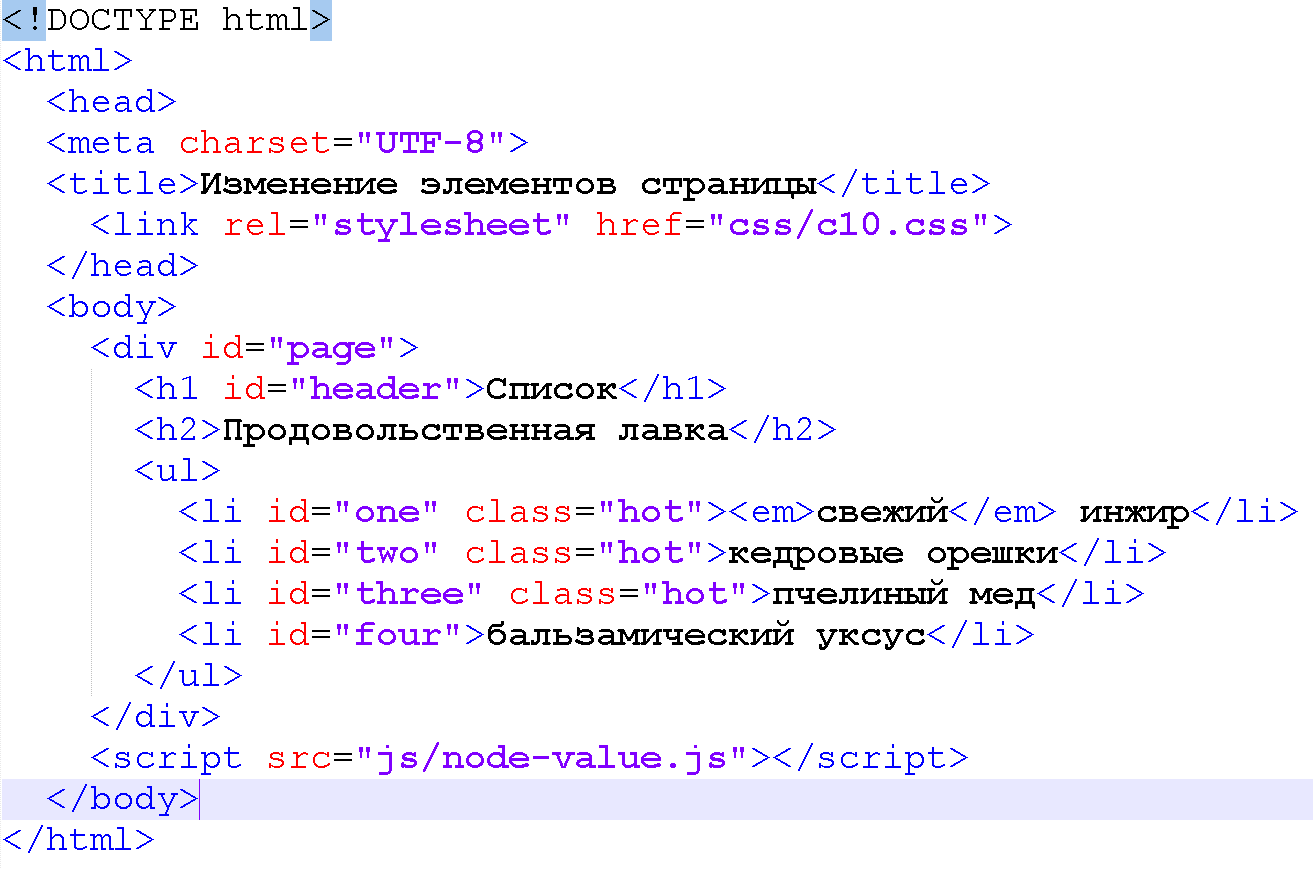
② Первым потомком элемента li является элемент em.

③ Текстовый узел — следующий смежный узел данного элемента em.

④ У вас есть текстовый узел, к его содержимому можно обратиться при помощи свойства nodeValue.

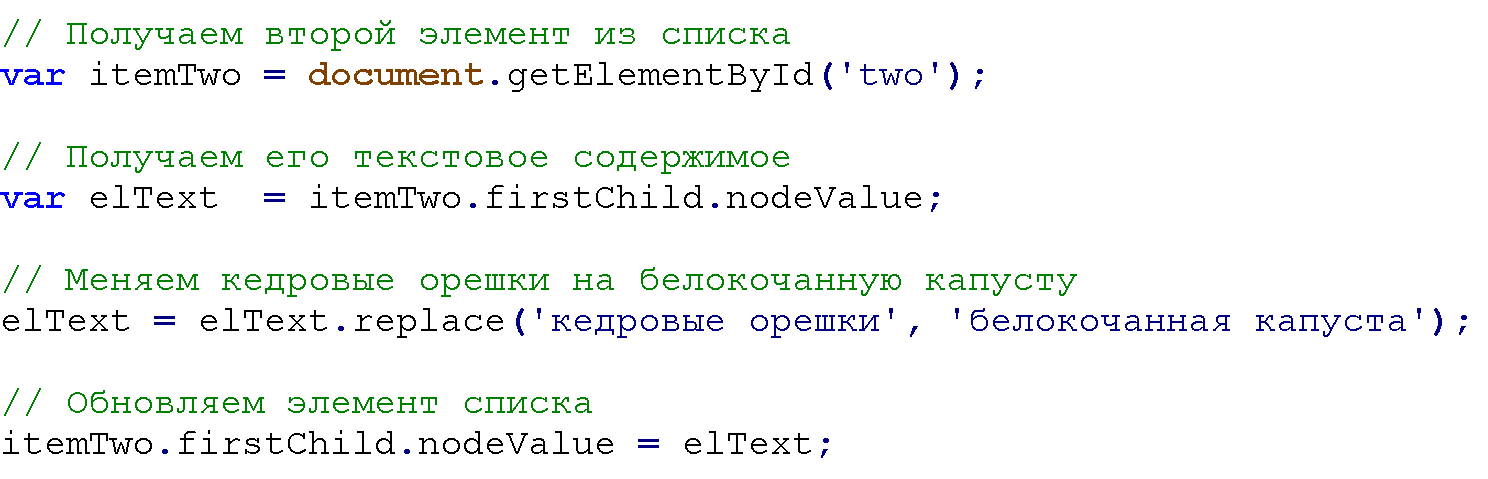
В следующем примере демонстрируется, то, что для работы с текстом в элементе, нужно сначала обратиться к узлу элемента, а затем — к его текстовому узлу. У текстового узла есть свойство nodeValue. Оно возвращает то, что в нем содержится. Кроме того, свойство nodeValue можно использовать для обновления содержимого текстового узла.

1. В папке 11 создайте html документ node-value.html (можно скопировать из прошлой работы, заменить заголовок и строку вызова скрипта»:

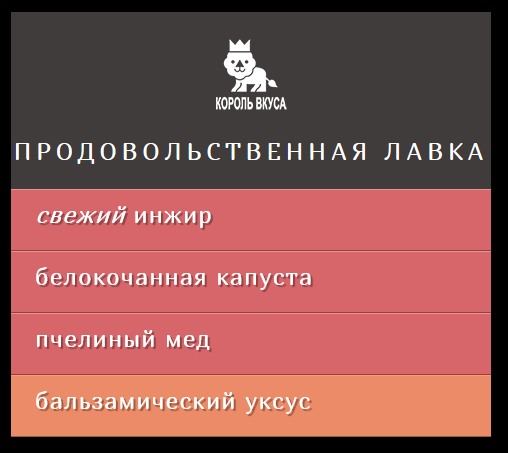


2. Создайте папки 11/css, 11/images и скопируйте в них таблицу стиля и изображения

3. В папке 11/js создайте скрипт node-value.js:



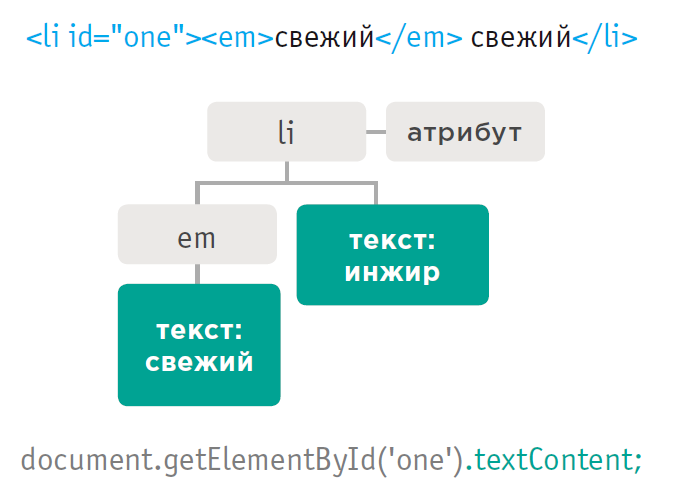
4. Отобразите страницу в браузере:



Свойство nodeValue применимо к текстовому узлу, а не к элементу, содержащему этот текст. Данный пример демонстрирует, что переход от узла элемента к текстовому узлу может оказаться довольно сложен. Если вы не знаете, будут ли между текстовыми узлами находиться узлы элементов, удобнее работать с объемлющим элементом.

### 4. ДОСТУП К ТЕКСТУ И ЕГО ОБНОВЛЕНИЕ ПРИ ПОМОЩИ СВОЙСТВ textContent и innerText

Свойство **textContent** позволяет собирать и обновлять текст и только текст, содержащийся в объемлющем элементе (и его дочерних элементах).



Для сбора текста из элементов li в нашем примере (игнорируя всю разметку, содержащуюся в элементе) можно использовать свойство textContent элемента li, содержащего текст. В данном примере оно вернет значение «свежий инжир».

При помощи свойства textContent также можно обновить контент элемента. Оно заменяет все содержимое элемента, включая разметку.

Возможно, вам иногда будет встречаться и свойство innerText. Однако им, как правило, не рекомендуется пользоваться по трем основным причинам:

ПОДДЕРЖКА. Хотя большинство основных браузеров поддерживают innerText, Firefox его игнорирует, поскольку это свойство не входит ни в какой стандарт.

ПОДЧИНЕННОСТЬ CSS. Это свойство не отображает текст, скрытый при помощи CSS. Например, при наличии правила CSS, скрывающего элементы em, свойство innerText вернуло бы только слово инжир.

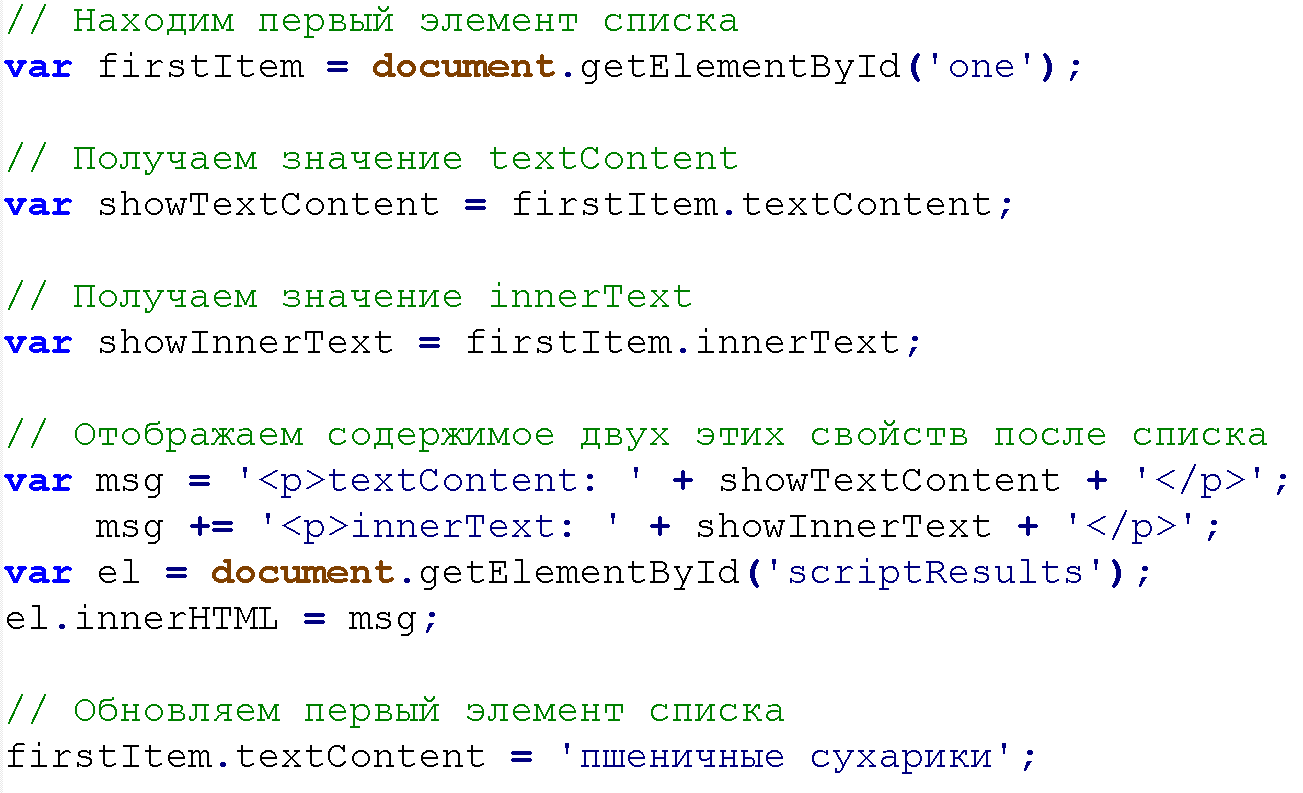
ПРОИЗВОДИТЕЛЬНОСТЬ. Поскольку свойство innerText учитывает правила CSS , указывающие, будет ли виден элемент, текст может извлекаться медленнее, чем при применении свойства textContent.

Чтобы продемонстрировать разницу между свойствами textContent и innerText, рассмотрим пример, в котором действует правило CSS, скрывающее содержимое элемента em.

1. Сделайте копию html-документа, использовавшегося в прошлом примере, назовите копию inner-text-and-text-content.html, добавьте в него стиль для элемента em, измените строку вызова скрипта.

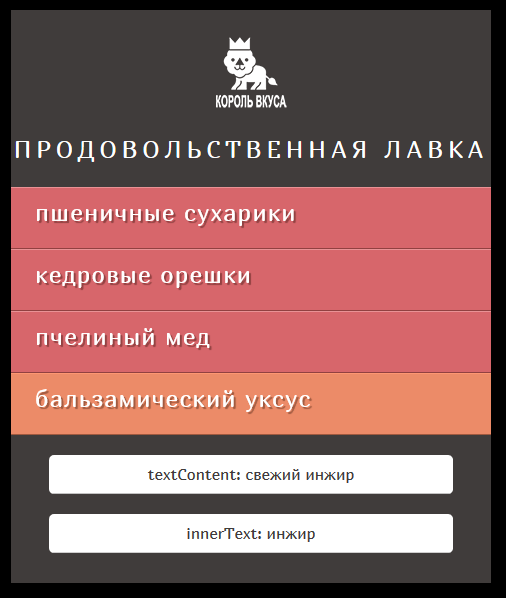


2. В папке 11/js создайте скрипт



Выполнение этого сценария начинается с того, что он получает содержимое первого элемента, используя одновременно два свойства: textContent и innerText. Затем он ставит эти значения после списка. Наконец, значение первого элемента списка обновляется и принимает вид «пшеничные сухарики». Это делается при помощи свойства textContent.

3. Отобразите страницу в браузере:

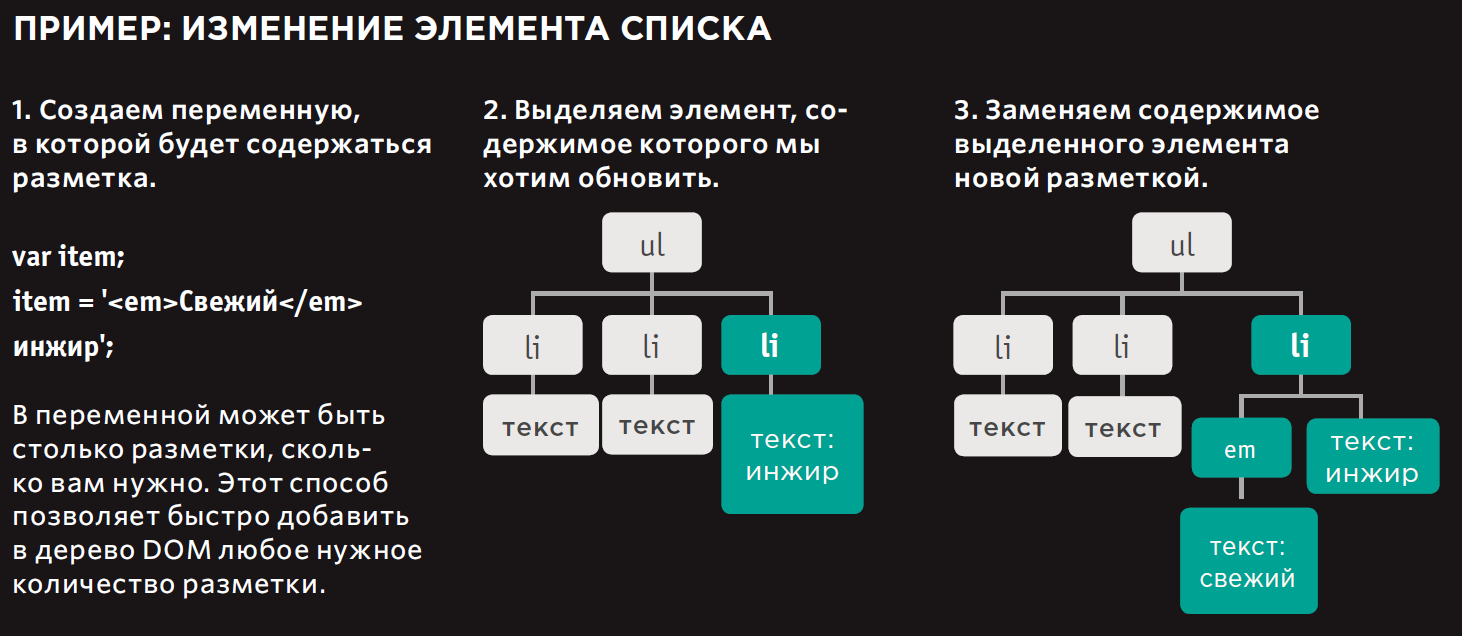


### 5. ДОБАВЛЕНИЕ ИЛИ УДАЛЕНИЕ HTML-КОНТЕНТА с помощью Свойства innerHTML

Существуют два принципиально отличающихся способа добавления контента в дерево DOM и удаления его оттуда. Во-первых, можно воспользоваться свойством innerHTML, во-вторых, выполнить манипуляции с DOM.

Свойство innerHTML можно использовать с любым узлом элемента. Оно применяется как для извлечения, так и для замены контента. Для замены новый материал предоставляется в виде строки, которая может содержать разметку для элементов-потомков

|  |
| --- |
| **ДОБАВЛЕНИЕ КОНТЕНТА:**  1) сохраните новый контент (вместе с разметкой) в переменной как строку;  2) выделите элемент, контент которого вы хотите заменить;  3) установите новую строку в качестве значения свойства innerHTML этого элемента. |



|  |
| --- |
| **УДАЛЕНИЕ КОНТЕНТА:**  Чтобы удалить все содержимое элемента, нужно задать пустую строку в качестве значения его свойства innerHTML. Чтобы удалить один элемент из фрагмента DOM, например, один li из ul, нужно задать в строке  весь фрагмент без удаляемого элемента |

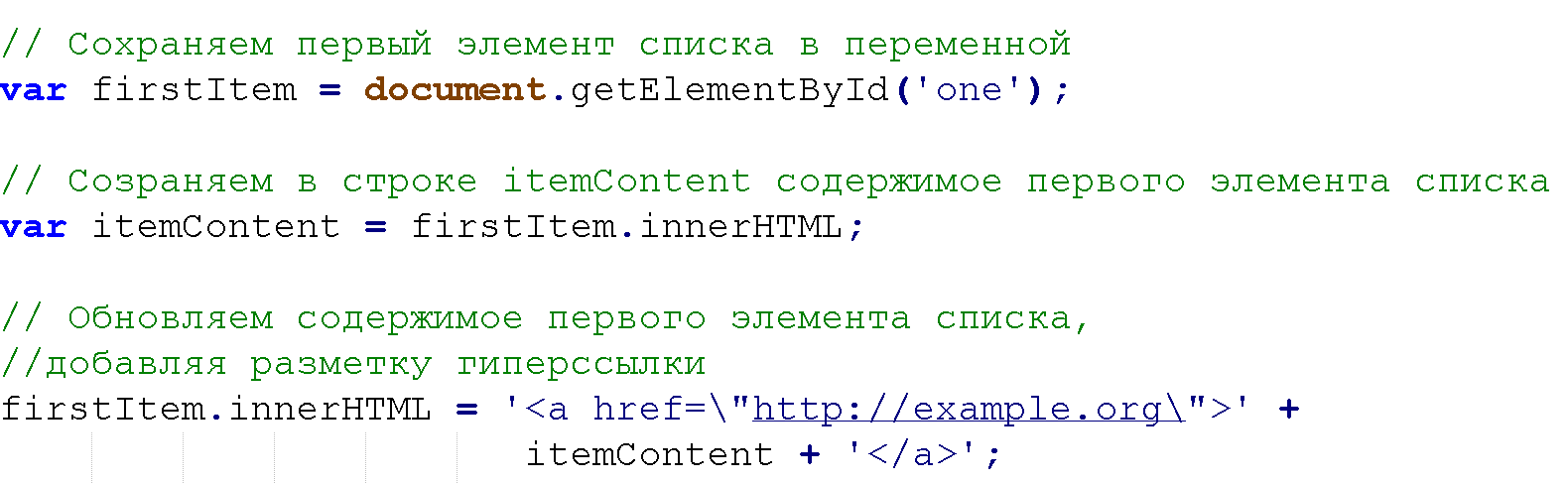
БУДЬТЕ ОСТОРОЖНЫ! Свойство innerHTML возвращает контент элемента как длинную строку, включающую в том числе всю разметку, содержащуюся в элементе. При его использовании для того, чтобы задать элементу новой контент, это свойство берет строку, содержащую разметку, и добавляет все содержащиеся в ней элементы в дерево DOM. При создании нового контента с помощью innerHTML учитывайте, что, пропустив один-единственный закрывающий тег, вы можете испортить компоновку всей страницы. Более того, если свойство innerHTML применяется для добавления на страницу пользовательского контента, в ваш код могут попасть вредоносное содержимое

Рассмотрим пример, в котором первый элемент списка **превращается в гиперссылку**, для которой действуют другие правила CSS.

1. Создайте копию html-документа из первого примера, назвав inner-html.html. Замените в нем строку вызова скрипта:



2. Создайте скрипт inner-html.js:



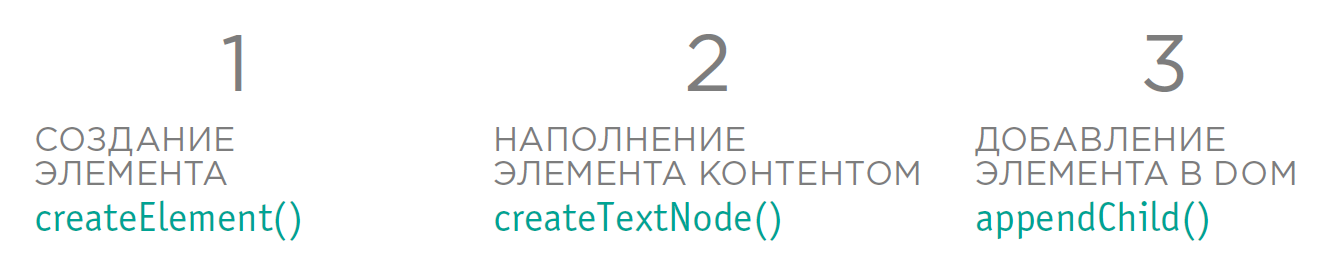
Если вы используете в вашем HTML-коде атрибуты, то удобно **экранировать кавычки** символом обратного слеша \, чтобы показать, что они не входят в состав сценария.

3. Отобразите страницу в браузере, наведите мышку на первый элемент списка. Скрипт превратил его в гиперссылку:



### 6. ДОБАВЛЕНИЕ ЭЛЕМЕНТОВ ПРИ ПОМОЩИ МАНИПУЛЯЦИЙ С DOM

Манипуляция с DOM — это еще один способ добавления нового контента на страницу (альтернативный работе со свойством innerHTML). Он включает три этапа:



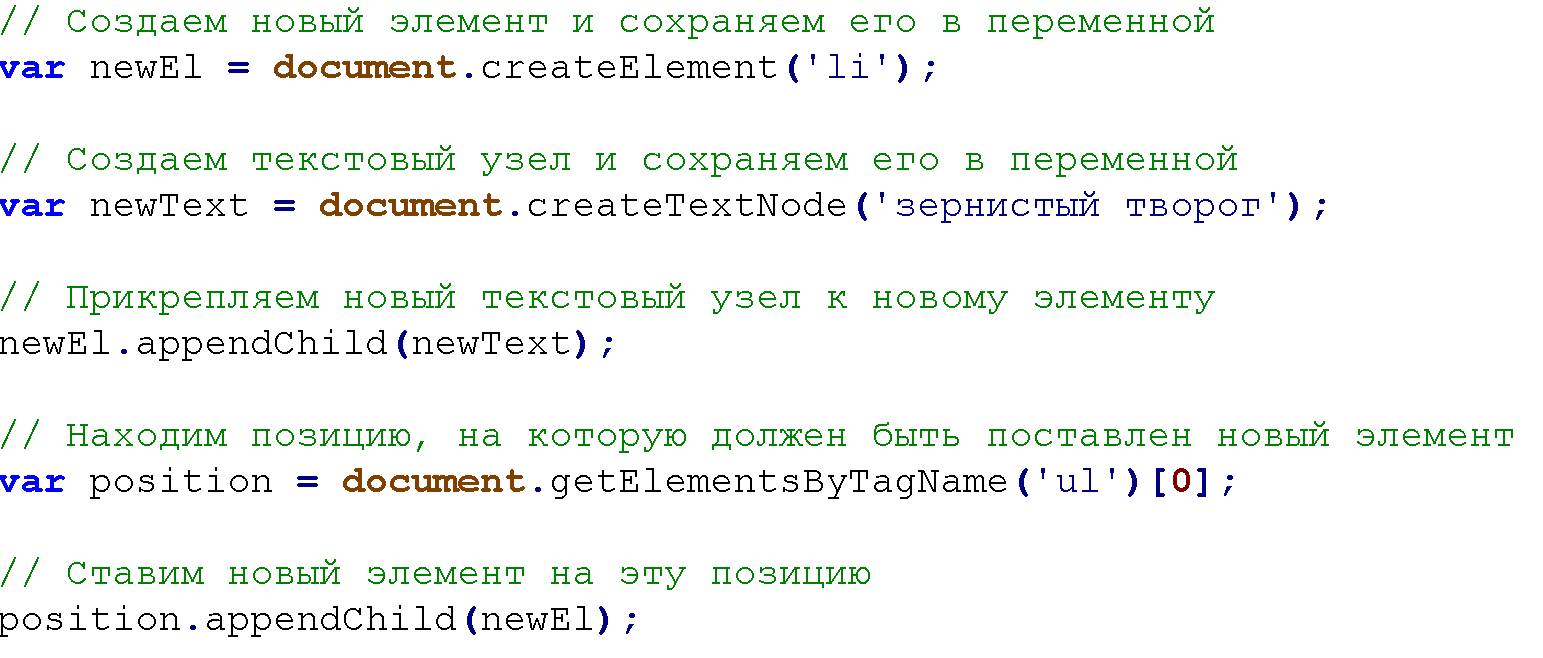


Реализуем рассмотренный выше пример, в котором к списку добавляется новый элемент.

1. Создайте копию html-документа из первого примера, назвав add-element.html. Замените в нем строку вызова скрипта:



2. В папке 11/js Создайте скрипт add-element.js:



3. Отобразите страницу в браузере:



### 7. УДАЛЕНИЕ ЭЛЕМЕНТОВ ПРИ ПОМОЩИ МАНИПУЛЯЦИЙ С DOM

Удаление элементов из DOM выполняют в три этапа:

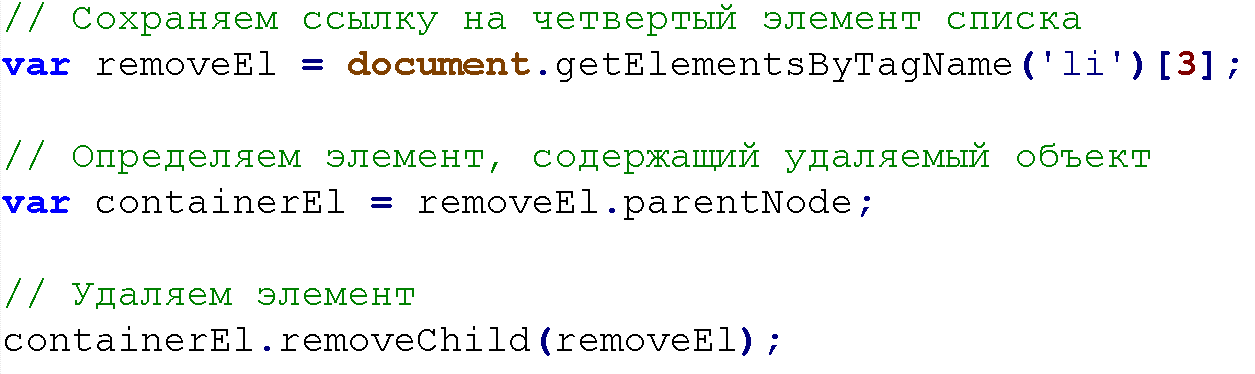
|  |  |  |
| --- | --- | --- |
| 1  ЭЛЕМЕНТ, ПОДГОТАВЛИВАЕМЫЙ К УДАЛЕНИЮ, СОХРАНЯЕТСЯ В ПЕРЕМЕННОЙ | 2  В ПЕРЕМЕННОЙ СОХРАНЯЕТСЯ РОДИТЕЛЬСКИЙ ЭЛЕМЕНТ ТОГО ЭЛЕМЕНТА, КОТОРЫЙ БЫЛ ВЫБРАН НА ПЕРВОМ ЭТАПЕ  **parentNode** | 3  УДАЛЯЕМ ЭЛЕМЕНТ ИЗ ЕГО ОБЪЕМЛЮЩЕГО ЭЛЕМЕНТА  **removeChild()** |

В следующем примере мы используем метод removeChild(), чтобы удалить из списка четвертый элемент (а также все его содержимое).

1. Создайте копию html-документа из первого примера, назвав remove-element.html. Замените в нем строку вызова скрипта:

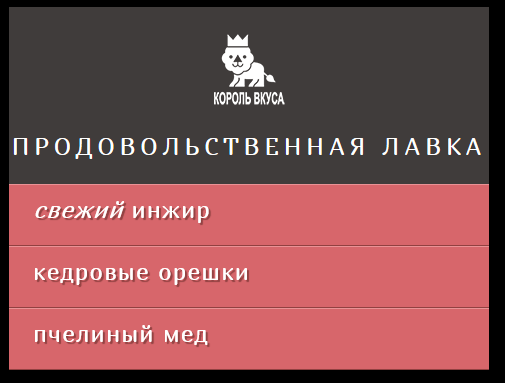


2. Создайте скрипт remove-element.js:



В первой переменной **removeEl** сохраняется сам элемент, который вы хотите удалить со страницы (четвертый элемент списка). Во второй переменной **containerEl** хранится элемент **ul**, содержащий тот элемент, который вы хотите удалить.

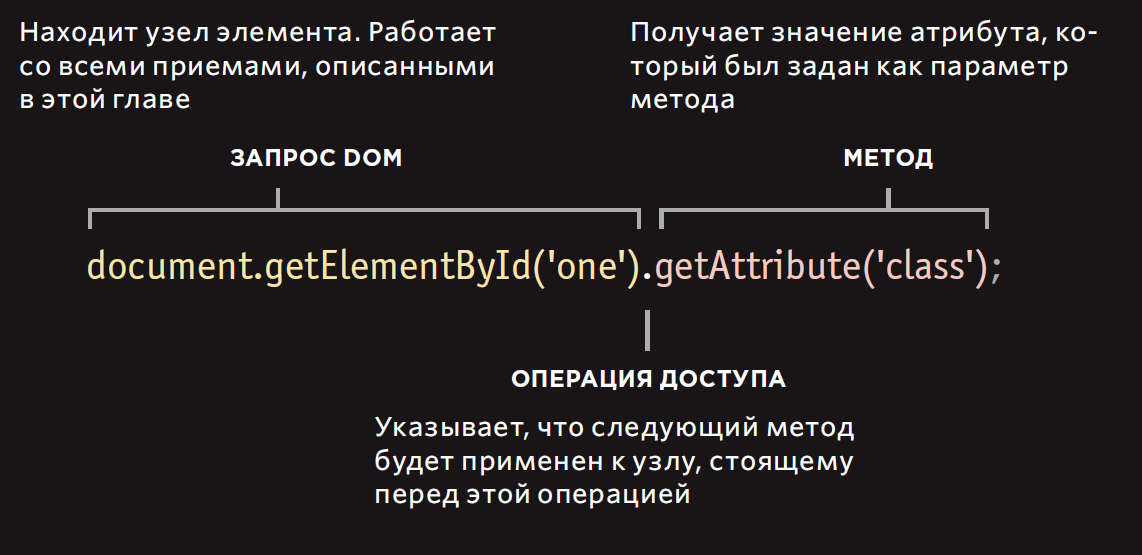
3. Отобразите страницу в браузере:



### 8. ПРОВЕРКА НАЛИЧИЯ АТРИБУТА И ПОЛУЧЕНИЕ ЕГО ЗНАЧЕНИЙ

Доступ к атрибутам и их обновление происходит в два этапа.

|  |  |
| --- | --- |
| 1  выделяем узел элемента, содержащий этот атрибут, а после имени узла ставим точку | 2  применяем какой-нибудь метод или свойство из перечисленных ниже. Они позволяют работать с атрибутами любого элемента  **getAttribute()**  **hasAttribute()**  **setAttribute()**  **removeAttribute()**  **className**  **id** |



Некоторые метода и свойства для работы с узлами атрибутов перечислены ниже в таблице

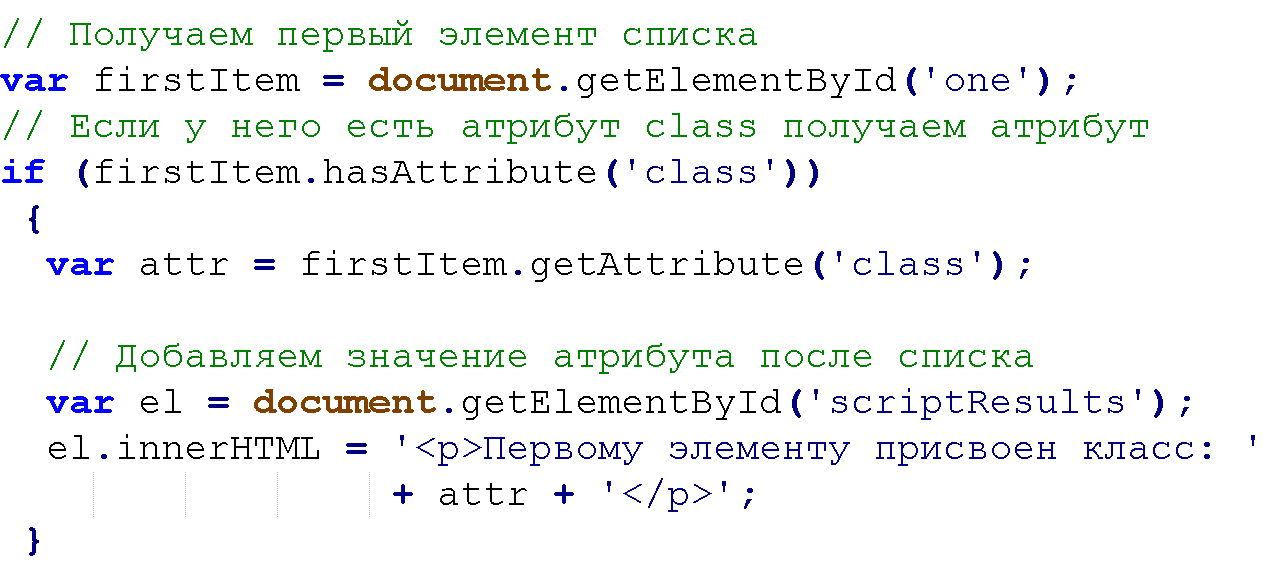
|  |  |
| --- | --- |
| МЕТОД | ОПИСАНИЕ |
| getAttribute() | Получает значение атрибута |
| hasAttribute() | Проверяет, есть ли у узла элемента указанный атрибут |
| setAttribute() | Устанавливает значение для атрибута |
| removeAttribute() | Удаляет атрибут с узла элемента |
| СВОЙСТВО | ОПИСАНИЕ |
| className | Получает или устанавливает значение атрибута class |
| id | Получает или устанавливает значение атрибута id |

Продемонстрируем способы работы с атрибутами на примере скрипта, который проверяет наличие атрибуты и получает его значение.

1. Сделайте копию документа inner-text-and-text-content.htmlназовите копию get-attribute.html. Замените строку вызова скрипта:



2. Создайте скрипт get-attribute.js



3. Отобразите страницу в браузере:



### 9. СОЗДАНИЕ АТРИБУТОВ И ИЗМЕНЕНИЕ ИХ ЗНАЧЕНИЙ

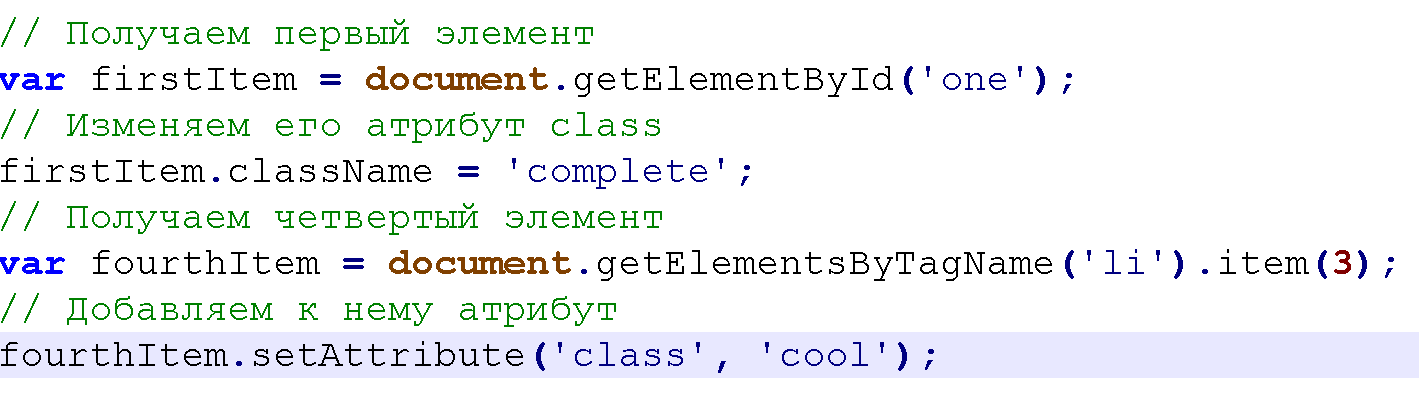
Свойство className позволяет изменять значение атрибута class. Если он еще не существует, то будет создан и получит конкретное значение. В прошлых работах мы часто пользовались данным свойством для обновления состояния элементов списка. Ниже показан иной способ решения этой задачи. Метод setAttribute() позволяет обновить значение любого атрибута. Он принимает два параметра: имя атрибута и его новое значение.

Рассмотрим приме, который добавит атрибут class к четвертому элементу списка

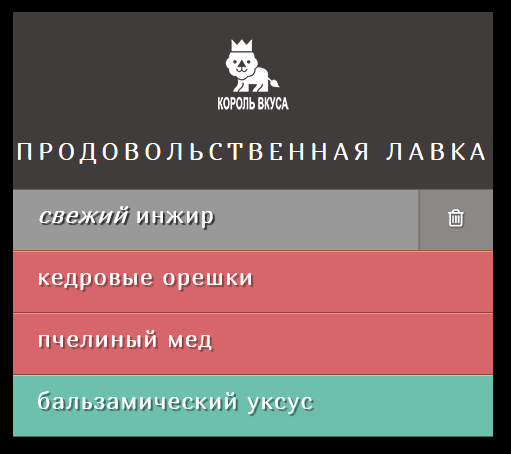
1. Создайте копию html-документа из первого примера, назвав set-attribute.html. Замените в нем строку вызова скрипта:



2.В папке 11/js создайте скрипт set-attribute.js:



3. Отобразите страницу в браузере. Первый элемент списка изменил свой вид за счет изменения имя класса ('complete' вместо ‘hot’), четвертый элемент списка изменил свой вид за счет задания атрибуту class значения ‘cool’.



Примечание. При использовании таких приемов полностью переопределяется значение атрибута class. Они не добавляют новой семантики к уже имеющемуся значению атрибута class. Если требуется добавить новое значение к уже содержащемуся в атрибуте class, нужно сначала считать его контент, а затем добавить новый текст к прежнему.

### 10. УДАЛЕНИЕ АТРИБУТОВ

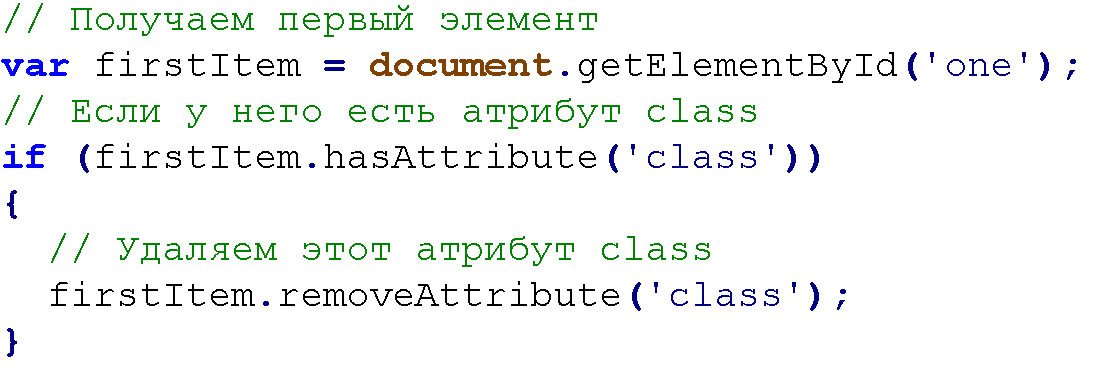
Чтобы удалить атрибут, сначала нужно выделить элемент, затем вызвать метод removeAttribute(). Он имеет один параметр: имя атрибута. При попытке удалить несуществующий атрибут ошибки не произойдет, однако все-таки рекомендуется проверять наличие атрибута, прежде чем пытаться его удалить.

В следующем примере метод getElementById() применяется для извлечения первого элемента из списка. Этому элементу присвоен идентификатор one.

1. Создайте копию html-документа из первого примера, назвав remove-attribute.html. Замените в нем строку вызова скрипта:



2.В папке 11/js создайте скрипт remove-attribute.js:



3. Отобразите страницу в браузере. Первый элемент изменил свой вид, так как теперь у него нет класса ‘hot’



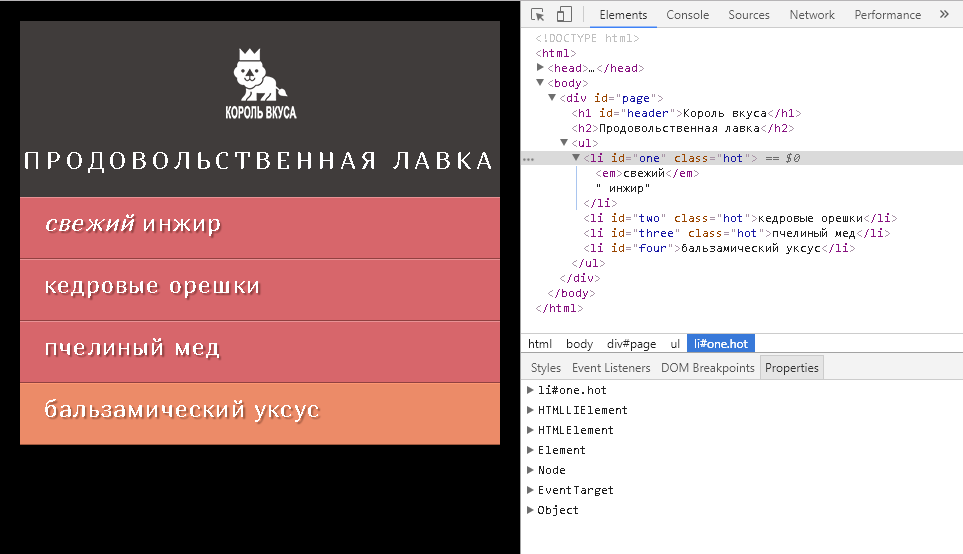
### 11. ИССЛЕДОВАНИЕ ОБЪЕКТНОЙ МОДЕЛИ ДОКУМЕНТА В БРАУЗЕРЕ CHROME

Современные браузеры оснащены инструментами, позволяющими исследовать (инспектировать) страницу, загруженную в браузере, видя при этом структуру ее DOM-дерева.

На компьютере под управлением операционной системы **Windows** в браузере Chrome нажмите кнопку **Инструменты (Tools)** в правом верхнем углу и выберите пункт **Дополнительные инструменты ⇒ Инструменты разработчика (More Tools ⇒ Developer Tools)**. Чтобы открыть инструменты разработчика в браузере Chrome в операционной системе **macOS**, выполните команду **Просмотреть ⇒ Разработчикам ⇒ Инструменты разработчика (View ⇒ Developer ⇒ Developer Tools)**.

Далее можно щелкнуть правой кнопкой мыши по любому элементу и выбрать в контекстном меню команду **Проверить элемент (Inspect Element)**. Если в верхней части открывшейся панели нажать кнопку **Elements (Элементы)**, то вверху отобразится исходный код страницы, а внизу — ряд дополнительных возможностей.

Любой элемент, обладающий элементами-потомками, снабжен стрелкой. Она позволяет раскрывать и сворачивать подробную информацию, скрывая и отображая содержимое элемента.



На данном снимке экрана элемент li подсвечен, а на панели Свойства (Properties) указано, что перед нами:

•элемент li с идентификатором one и классом hot;

•элемент HTMLLIElement;

• элемент HTMLElement;

• элемент;

• узел;

• объект.

Рядом с каждым из этих названий объекта есть стрелка, при помощи которой вы можете раскрыть дополнительную информацию по конкретной характеристике.

Под стрелкой вы увидите, какие свойства доступны для узла такого рода. Характеристики отделены друг от друга, так как некоторые свойства встречаются только у элементов списка, другие — у узлов элементов, третьи — у любых узлов, а четвертые — у любых объектов. Различные свойства перечислены для тех типов узлов, которым они могут соответствовать. Кроме того, такая структура напоминает вам, к каким свойствам открывается доступ через DOM-узел данного элемента.

Панель **Свойства (Properties)**, расположенная сверху, показывает тип объекта, к которому принадлежит выделенный элемент. В некоторых версиях браузера Chrome эта панель оформлена как вкладка. Когда вы выделяете отдельно взятые элементы, расположенные в основном окне слева, справа на панели Свойства (Properties) указывается соответствующая информация.

## Контрольные вопросы и задания для самостоятельной работы

1. Для каких целей можно использовать свойство NodeValue? Запишите пример.
2. Для каких целей используется свойство textContent? Запишите пример.
3. Для каких целей используется свойство innerHTML?
4. Какова последовательность действий при изменении контента с помощью свойства innerHTML? Приведите пример.
5. Какова последовательность действий при удалении контента с помощью свойства innerHTML? Приведите пример.
6. Какова последовательность действий при добавлении контента при помощи манипуляций с DOM? Поясните эту последовательность на примере добавления элемента списка
7. Для чего используется метод removeChild? Приведите пример.
8. Перечислите методы и свойства и их назначение для изменения атрибутов элементов
9. Как в браузере Google Chrome просмотреть информацию о дереве DOM?
10. Создайте скрипт, который определит количество элементов списка li и добавит это число в заголовок (в элемент H1 или H2). Указание: используйте свойство innerHTML или nodeValue.
11. Создайте скрипт, который добавит новый элемент в конец списка. Указания: действуйте по алгоритму:  
    1. Создаем узел элемента.  
    2. Создаем текстовый узел.  
    3. Добавляем текстовый узел к узлу элемента  
    4. Добавляем элемент к дереву DOM с помощью метода **appendChild**():   
    **родительскийЭлемент.appendChild(*новыйЭлемент*);**
12. Создайте скрипт, который добавит новый элемент в начало списка. Указания: действуйте как в задании 11, используйте метод **insertBefore()**:   
    **родительскийЭлемент.insertBefore(*новыйЭлемент, элементПередКоторымВставляемНовый*);**
13. Задать для всех элементов атрибуты class=’cool’. Указание: нужно перебрать в цикле все элементы <li> и изменить значение их атрибутов class на cool.