Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 25**

**Инструкционно-технологическая карта**

Тема: Всплытие и погружение события в DOM. Event.target. Прекращение всплытия. Погружение.

Цель: Научиться использовать всплытия и погружения события в DOM, Event.target, прекращение всплытия, погружение.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Теоретические сведения

## Классы DOM-узлов

У разных DOM-узлов могут быть разные свойства. Например, у узла, соответствующего тегу **<a>**, есть свойства, связанные со ссылками, а у соответствующего тегу **<input>** – свойства, связанные с полем ввода и т.д. Текстовые узлы отличаются от узлов-элементов. Но у них есть общие свойства и методы, потому что все классы DOM-узлов образуют единую иерархию.

Каждый DOM-узел принадлежит соответствующему встроенному классу.

Корнем иерархии является **EventTarget**, от него наследует **Node** и остальные DOM-узлы.

Существуют следующие классы:

* **EventTarget** – это корневой «абстрактный» класс. Объекты этого класса никогда не создаются. Он служит основой, благодаря которой все DOM-узлы поддерживают так называемые «события».
* **Node** – также является «абстрактным» классом, и служит основой для DOM-узлов. Он обеспечивает базовую функциональность: **parentNode, nextSibling, childNodes** и т.д. (это геттеры). Объекты класса **Node** никогда не создаются. Но есть определённые классы узлов, которые наследуют от него: **Text** – для текстовых узлов, **Element** – для узлов-элементов и более экзотический **Comment** – для узлов-комментариев.
* **Element** – это базовый класс для DOM-элементов. Он обеспечивает навигацию на уровне элементов: *nextElementSibling*, *children* и методы поиска: *getElementsByTagName, querySelector*. Браузер поддерживает не только HTML, но также XML и SVG. Класс **Element** служит базой для следующих классов: **SVGElement, XMLElement** и **HTMLElemen**t.
* HTMLElement – является базовым классом для всех остальных HTML-элементов. От него наследуют конкретные элементы:
* **HTMLInputElement** – класс для тега **<input>,**
* **HTMLBodyElement** – класс для тега **<body>,**
* **HTMLAnchorElement** – класс для тега **<a>,**
* …и т.д, каждому тегу соответствует свой класс, который предоставляет определённые свойства и методы.

Таким образом, полный набор свойств и методов данного узла собирается в результате наследования.

Рассмотрим DOM-объект для тега **<input>**. Он принадлежит классу HTMLInputElement.

Он получает свойства и методы из (в порядке наследования):

* **HTMLInputElement** – этот класс предоставляет специфичные для элементов формы свойства,
* **HTMLElement** – предоставляет общие для HTML-элементов методы (и геттеры/сеттеры),
* **Element** – предоставляет типовые методы элемента,
* **Node** – предоставляет общие свойства DOM-узлов,
* **EventTarget** – обеспечивает поддержку событий,
* …и, наконец, он наследует от **Object,** поэтому доступны также методы «обычного объекта», такие как **hasOwnProperty**.

Для того, чтобы узнать имя класса DOM-узла, вспомним, что обычно у объекта есть свойство **constructor.** Оно ссылается на конструктор класса, и в свойстве **constructor.name** содержится его имя:

alert( document.body.constructor.name ); // HTMLBodyElement

…Или можно просто привести его к строке:

alert( document.body ); // [object HTMLBodyElement]

Проверить наследование можно также при помощи **instanceof**:

alert(document.body instanceof HTMLBodyElement); // true

alert(document.body instanceof HTMLElement); // true

alert(document.body instanceof Element); // true

alert(document.body instanceof Node); // true

alert(document.body instanceof EventTarget); // true

DOM-узлы – это обычные JavaScript объекты. Для наследования они используют классы, основанные на прототипах.

В этом легко убедиться, если вывести в консоли браузера любой элемент через **console.dir(elem)**. Или даже напрямую обратиться к методам, которые хранятся в **HTMLElement.prototype, Element.prototype** и т.д.

## Свойство «nodeType»

Свойство **nodeType** предоставляет ещё один, «старомодный» способ узнать «тип» DOM-узла.

Его значением является цифра:

* **elem.nodeType** == 1 для узлов-элементов,
* **elem.nodeType** == 3 для текстовых узлов,
* **elem.nodeType** == 9 для объектов документа,
* В спецификации можно посмотреть остальные значения.

Например:

<body>

    <script>

        let elem = document.body;

        alert(elem.nodeType); // 1 => элемент

        // и первый потомок...

        alert(elem.firstChild.nodeType); // 3 => текст

        // для объекта document значение типа -- 9

        alert(document.nodeType); // 9

    </script>

</body>

В современных скриптах, чтобы узнать тип узла, можно использовать метод **instanceof** и другие способы проверить класс, но иногда **nodeType** проще использовать. Значение **nodeType** невозможно изменить, можно только прочитать его.

## Тег: nodeName и tagName

Получив DOM-узел, можно узнать имя его тега из свойств **nodeName** и **tagName**:

Например:

alert(document.body.nodeName); // BODY

alert(document.body.tagName); // BODY

Есть ли какая-то разница между **tagName** и **nodeName**? Да, она отражена в названиях свойств, но не очевидна.

* Свойство **tagName** есть только у элементов **Element**.
* Свойство **nodeName** определено для любых узлов **Node**:
* для элементов оно равно **tagName**.
* для остальных типов узлов (текст, комментарий и т.д.) оно содержит строку с типом узла.

Другими словами, свойство **tagName** есть только у узлов-элементов (поскольку они происходят от класса **Element**), а **nodeName** может что-то сказать о других типах узлов.

Например, сравним **tagName** и **nodeName** на примере объекта **document** и узла-комментария:

<body>

    <!-- комментарий -->

    <script>

        // для комментария

        alert(document.body.firstChild.tagName); // undefined (не элемент)

        alert(document.body.firstChild.nodeName); // #comment

        // for document

        alert(document.tagName); // undefined (не элемент)

        alert(document.nodeName); // #document

    </script>

</body>

## innerHTML: содержимое элемента

Свойство **innerHTML** позволяет получить HTML-содержимое элемента в виде строки. Оно может изменяться. Это один из самых мощных способов менять содержимое на странице.

Пример ниже показывает содержимое **document**.**body**, а затем полностью заменяет его:

<body>

    <p>Параграф</p>

    <div>DIV</div>

    <script>

        alert(document.body.innerHTML); // считываем текущее содержимое

        document.body.innerHTML = 'Новый BODY!'; // заменяем содержимое

    </script>

</body>

При попытке вставить некорректный HTML, браузер исправит ошибки:

<body>

    <script>

        document.body.innerHTML = '<b>тест'; // забыли закрыть тег

        alert(document.body.innerHTML); // <b>тест</b> (исправлено)

    </script>

</body>

**Скрипты не выполнятся**: Если **innerHTML** вставляет в документ тег <**script**> – он становится частью HTML, но не запускается.

#### Необходимо быть внимательным: «innerHTML+=» осуществляет перезапись

Можно добавить HTML к элементу, используя **elem.innerHTML**+="ещё html":

chatDiv.innerHTML += "<div>Привет<img src='smile.gif'/> !</div>";

chatDiv.innerHTML += "Как дела?";

На практике этим следует пользоваться с большой осторожностью, так как фактически происходит не добавление, а перезапись.

Технически эти две строки делают одно и то же:

elem.innerHTML += "...";

// это более короткая запись для:

elem.innerHTML = elem.innerHTML + "..."

Другими словами, **innerHTML+=** делает следующее:

1. Старое содержимое удаляется.
2. На его место становится новое значение innerHTML (с добавленной строкой).

**Так как содержимое «обнуляется» и переписывается заново, все изображения и другие ресурсы будут перезагружены.**

В примере **chatDiv** выше строка **chatDiv.innerHTML+="Как дела?"** заново создаёт содержимое HTML и перезагружает smile.gif. Если в **chatDiv** много текста и изображений, то эта перезагрузка будет очень заметна.

Есть и другие побочные эффекты. Например, если существующий текст выделен мышкой, то при переписывании **innerHTML** большинство браузеров снимут выделение. А если это поле ввода <**input**> с текстом, введённым пользователем, то текст будет удалён. И т.д.

## outerHTML: HTML элемента целиком

Свойство **outerHTML** содержит HTML элемента целиком. Это как **innerHTML** плюс сам элемент.

Пример:

<div id="elem">Привет <b>Мир</b></div>

<script>

  alert(elem.outerHTML); // <div id="elem">Привет <b>Мир</b></div>

</script>

Необходимо быть осторожным: в отличие от **innerHTML**, запись в **outerHTML** не изменяет элемент. Вместо этого элемент заменяется целиком во внешнем контексте.

Пример:

<div>Привет, мир!</div>

<script>

    let div = document.querySelector('div');

    // замена div.outerHTML на <p>...</p>

    div.outerHTML = '<p>Новый элемент</p>'; // (\*)

    // Содержимое div осталось тем же!

    alert(div.outerHTML); // <div>Привет, мир!</div> (\*\*)

</script>

В строке **(\*) div** был наменён на **<p>Новый элемент</p>**. Во внешнем документе можно видеть новое содержимое вместо <**div**>. Но, как видно в строке (**\*\*),** старая переменная **div** осталась прежней!

Это потому, что использование **outerHTML** не изменяет DOM-элемент, а удаляет его из внешнего контекста и вставляет вместо него новый HTML-код.

То есть, при **div.outerHTML=**... произошло следующее:

1. **div** был удалён из документа.
2. Вместо него был вставлен другой HTML **<p>Новый элемент</p>.**
3. В **div** осталось старое значение. Новый HTML не сохранён ни в какой переменной.

Здесь легко сделать ошибку: заменить **div.outerHTML**, а потом продолжить работать с **div**, как будто там новое содержимое. Но это не так. Подобное верно для **innerHTML**, но не для **outerHTML**.

## nodeValue/data: содержимое текстового узла

Свойство **innerHTML** есть только у узлов-элементов.

У других типов узлов, в частности, у текстовых, есть свои аналоги: свойства **nodeValue** и **data**. Эти свойства очень похожи при использовании, есть лишь небольшие различия в спецификации.

Прочитаем содержимое текстового узла и комментария:

<body>

    Привет

    <!-- Комментарий -->

    <script>

        let text = document.body.firstChild;

        alert(text.data); // Привет

        let comment = text.nextSibling;

        alert(comment.data); // Комментарий

    </script>

</body>

Можно представить, для чего может понадобиться читать или изменять текстовый узел, но комментарии?

Иногда их используют для вставки информации и инструкций шаблонизатора в HTML, как в примере ниже:

<!-- if isAdmin -->

<div>Добро пожаловать, Admin!</div>

<!-- /if -->

Затем JavaScript может прочитать это из свойства **data** и обработать инструкции.

## textContent: просто текст

Свойство **textContent** предоставляет доступ к тексту внутри элемента за вычетом всех <тегов>.

Например:

<div id="news">

    <h1>Срочно в номер!</h1>

    <p>Марсиане атаковали человечество!</p>

</div>

<script>

    // Срочно в номер! Марсиане атаковали человечество!

    alert(news.textContent);

</script>

Как можно видеть, возвращается только текст, как если бы все <**теги**> были вырезаны, но текст в них остался.

На практике редко появляется необходимость читать текст таким образом. Намного полезнее возможность записывать текст в **textContent**, т.к. позволяет писать текст «безопасным способом».

Представим, что есть произвольная строка, введённая пользователем, и необходимо показать её.

* С innerHTML вставка происходит «как HTML», со всеми HTML-тегами.
* С textContent вставка получается «как текст», все символы трактуются буквально.

Сравним два тега div:

<div id="elem1"></div>

<div id="elem2"></div>

<script>

    let name = prompt("Введите ваше имя?", "<b>Винни-пух!</b>");

    elem1.innerHTML = name;

    elem2.textContent = name;

</script>

1. В первый <**div**> имя приходит «как HTML»: все теги стали именно тегами, поэтому можно видеть имя, выделенное жирным шрифтом.
2. Во второй <**div**> имя приходит «как текст», поэтому можно видеть <**b>Винни-пух!</b>.**

## Свойство «hidden»

Атрибут и DOM-свойство «**hidden**» указывает на то, виден элемент или нет.

Можно использовать его в HTML или назначать при помощи JavaScript, как в примере ниже:

<div>Оба тега DIV внизу невидимы</div>

<div hidden>С атрибутом "hidden"</div>

<div id="elem">С назначенным JavaScript свойством "hidden"</div>

<script>

    elem.hidden = true;

</script>

Технически, hidden работает так же, как **style="display:none".** Но его применение проще.

Мигающий элемент:

<div id="elem">Мигающий элемент</div>

<script>

    setInterval(() => elem.hidden = !elem.hidden, 1000);

</script>

## Другие свойства

У DOM-элементов есть дополнительные свойства, в частности, зависящие от класса:

* **value** – значение для **<input>, <select> и <textarea> (HTMLInputElement, HTMLSelectElement…).**
* **href** – адрес ссылки «**href**» для **<a href="..."**> (**HTMLAnchorElement**).
* **id** – значение атрибута «id» для всех элементов (**HTMLElement**).
* …и многие другие…

Например:

<input type="text" id="elem" value="значение">

<script>

    alert(elem.type); // "text"

    alert(elem.id); // "elem"

    alert(elem.value); // значение

</script>

Большинство стандартных HTML-атрибутов имеют соответствующее DOM-свойство, и можно получить к нему доступ.

## Итого

Каждый DOM-узел принадлежит определённому классу. Классы формируют иерархию. Весь набор свойств и методов является результатом наследования.

Главные свойства DOM-узла:

**nodeType –** свойство nodeType позволяет узнать тип DOM-узла. Его значение – числовое: 1 для элементов, 3 для текстовых узлов, и т.д. Только для чтения.

**nodeName/tagName** – для элементов это свойство возвращает название тега (записывается в верхнем регистре, за исключением XML-режима). Для узлов-неэлементов nodeName описывает, что это за узел. Только для чтения.

**innerHTML** – внутреннее HTML-содержимое узла-элемента. Можно изменять.

**outerHTML –** полный HTML узла-элемента. Запись в elem.outerHTML не меняет elem. Вместо этого она заменяет его во внешнем контексте.

**nodeValue/data** – содержимое узла-неэлемента (текст, комментарий). Эти свойства практически одинаковые.

**textContent** – текст внутри элемента: HTML за вычетом всех <тегов>. Запись в него помещает текст в элемент, при этом все специальные символы и теги интерпретируются как текст. Можно использовать для защиты от вставки произвольного HTML кода.

**hidden** – когда значение установлено в true, делает то же самое, что и CSS display:none.

В зависимости от своего класса DOM-узлы имеют и другие свойства. Например у элементов <input> (HTMLInputElement) есть свойства value, type, у элементов <a> (HTMLAnchorElement) есть href и т.д. Большинство стандартных HTML-атрибутов имеют соответствующие свойства DOM.

# Контрольные вопросы

1. Назовите главные свойства DOM-узла?
2. На что указывает Атрибут и DOM-свойство «hidden»?
3. Как работают свойства outerHTML и innerHTML?
4. Какие недостатки у innerHTML?
5. Объектом какого класса является document?
6. Какое место document занимает в DOM-иерархии?
7. От какого класса наследуется document?

# Задания

Действия должны выполняться при помощи скриптов JavaScript.

**Вариант 1**. <https://shop.beltelecom.by/>

1. При помощи скрипта узнайте среднюю стоимость всех товаров.
2. Выведите в качестве текста названия товаров.
3. Задайте изменение фона страницы случайным цветом каждые две секунды.
4. Увеличьте стоимость товаров в два раза.

**Вариант 2.** <https://ru.wikipedia.org/wiki/Интернет>

1. Добавьте в конец страницы все гиперссылки, которые встречались на странице.
2. Скопируйте первую таблицу. Удалите все таблицы кроме скопированной.
3. В скопированной таблице замените цвета местами.
4. Замените изображение карты интернета на любое другое.

**Вариант 3.** <https://pikabu.ru/>

1. Вывести все названия историй.
2. Изменить оформление страницы на светлые цвета.
3. Посчитать среднее значение рейтинга историй на странице и вывести их количество.
4. Удалите или скройте колонку справа (авторизация, комментарий дня, рекомендуемое сообщество и т.д.

**Вариант 4.** <https://yandex.by/>

1. Удалите все элементы кроме поисковой строки.
2. Добавьте новую надпись и логотип над поисковой строкой.
3. Измените фоновый цвет страницы.
4. Скопируйте строку поиска

**Вариант 5.** <https://www.kufar.by/>

1. Посчитать среднюю цену за товары на странице и записать на их место результат.
2. Удалите кнопку подачи объявления.
3. Замените изображения на объявлениях на произвольные.
4. Сделайте все отметки в виде сердечка закрашенными.

# Литература

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич