Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Формализация алгоритма обновления гипертекстового документа Fiber и анализ эффективности его использования в современных web-приложениях»

Студент группы **ИУ7-76Б** _____ **П. А. Калашков**

(Подпись, дата) (И.О. Фамилия)

Руководитель _____ **Д. Е. Бекасов**

(Подпись, дата) (И.О. Фамилия)

***2023** г.*

РЕФЕРАТ

Расчётно-пояснительная записка содержит 28 с., 4 рис., 0 табл., TODO ист.

ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА, ВИРТУАЛЬНАЯ ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА, ОБНОВЛЕНИЕ ГИПЕРТЕКСТОВОГО ДОКУМЕНТА, DOM, VDOM, АЛГОРИТМ СОГЛАСОВАНИЯ

Целью работы: анализ существующих методов обновления гипертекстовых документов, формализация алгоритма обновления гипертекстового документа Fiber, его анализ и сравнение его эффективности по сравнению с другими существующими алгоритмами.

Результаты: TODO

СОДЕРЖАНИЕ

НОРМАТИВНЫЕ ССЫЛКИ

В настоящем отчете о НИР использованы ссылки на следующие стандарты:

1) HTML Living standart [2];
2) DOM Living standart [1];
3) DOM Level 3 Core Specification [3].

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

HTML — HyperText Markup Language — язык гипертекстовой разметки.

CSS — Cascading Style Sheets — каскадные таблицы стилей.

HTTP — HyperText Transfer Protocol — протол передачи гипертекстовых документов.

DOM — Document Object Model — объектная модель документа.

API — Application Programming Interface — программный интерфейс приложения.

UI — User Interface — пользовательский интерфейс.

SSR — Server Side Rendering — рендеринг на стороне сервера.

ВВЕДЕНИЕ

Работа с гипертекстовыми документами является неотъемлемой частью жизни каждого человека, пользующегося Всемирной сетью и часто появляется потребность в просмотре различных гипертекстовых документов, а также в выполнении операций, приводящих к их изменению [4]. Возникает вопрос: каким образом стоит отображать документ и производить операции его обновления и построения?

Для взаимодействия с гипертекстовыми документами, входящими в сеть Интернет, существуют программы-браузеры. Преимущественнная часть браузеров использует стандарт [1], обеспечивающий использование объектной модели документа [5].

Целью данной работы является анализ алгоритмов построения, обновления и отображения гипертекстового документа при помощи объектной модели и виртуальной объектной модели. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) изучить принципы работы объектной модели документа и виртуальной объектной модели документа;
2) сравнить и проанализировать трудоёмкости алгоритмов обновления документа с использованием объектной модели и виртуальной объектной модели на основе теоретических расчётов;
3) сделать выводы об эффективности использования изученных алгоритмов.

1 Аналитическая часть

В данном разделе будут рассмотрены принципы работы с гипертекстовыми документами согласно объектной модели документа и виртуальной объектной модели документа.

1.1 Гипертекстовые документы

Гипертекстовым [6] документом является документ, состоящий из текстовых страниц, имеющих перекрёстные ссылки. В данной работе под гипертекстовыми документами будут подразумеваться документы, написанные при помощи языка гипертекстовой разметки (англ. *HyperText Markup Language* — HTML) [7], а именно документы, соблюдающие стандарт HTML5 [2].

Данный выбор обусловлен тем, что использование HTML5 получило широкое распространение в Всемирной сети [8] благодаря рекомендации [9] к использованию от Консорциума Всемирной паутины (англ. *World Wide Web Consortium* — W3C) [10]. Вследствие данной рекомендации HTML документы поддерживают большинство самых распространённых браузеров в России, такие, как Google Chrome, Яндекс.Браузер и Safari [11].

1.2 Web-приложение

Web-приложение (англ. *Web application*) [12] — web-страница (HTML или её вариант и CSS) или набор web-страниц, доставляемых про протоколу HTTP, которые используют обработку на стороне сервера или клиента (например, JavaScript) для обеспечения прикладного взаимодействия в браузере.

Web-приложения отличаются от простого web-контента наличием интерактивных элементов.

1.3 Существующие алгоритмы обновления гипертекстового документа

Из самых используемых фреймворков для разработки web-приложений на июнь 2023 года [13] можно выделить шесть, предоставляющих готовые решения в области обновления гипертекстовых документов: React, Angular, Vue.js, Flask, Django, Svelte. Алгоритмы этих готовых решений можно разделить на три крупных категории: алгоритмы, использующие объектную модель документа, алгоритмы, использующие виртуальную объектную модель документа и алгоритмы, использующие рендеринг на стороне сервера. Рассмотрим подробнее каждую из этих категорий.

1.3.1 Алгоритмы, использующие объектную модель документа

Объектная модель документа (англ. *Document Object Model* — DOM) [5] — программный интерфейс для HTML, XML и CSV документов. Он обеспечивает структурированное представление документа в виде дерева [14], и определяет способ, по которому структура может быть доступна для программы, для изменения структуры документа, его стиля и содержания. Представление DOM состоит из структурированной группы узлов и объектов, которые имеют свойства и методы.

Стадарт W3C DOM [1] формирует основы DOM, реализованные в большинстве современных браузеров. Для описания структуры DOM потребуются следующие термины: корневой, родительские и дочерние элементы. Корневой элемент находится в основании всей структуры и не имеет родительского элемента. Дочерние элементы не просто находятся внутри родительских, но и наследуют различные свойства от них. Рассмотрим, как выглядит DOM-представление HTML документа, представленного на листинге 1.

Листинг 1 – Пример простого HTML документа

```
<!DOCTYPE HTML>
<html>
<head>
  <link/>
  <meta/>
  <title/>
</head>
<body>
  <header/>
  <section>
    <div/>
    <p/>
    <a/>
  </section>
  <footer/>
</body>
</html>
```

Корневым элементом здесь является `html`, он не имеет родительского элемента и имеет два дочерних — `head` и `body`. По отношению друг к другу элементы `head` и `body` являются сиблингами (братьями и сестрами). В каждый из них можно вложить еще много дочерних элементов. Например, в `head` обычно находятся `link`, `meta`, `script` или `title`.

Данный HTML документ будет иметь следующее DOM-представление, изображённое на рисунке 1.

Все эти теги не являются уникальными, и в одном документе может быть по несколько экземпляров каждого из них.

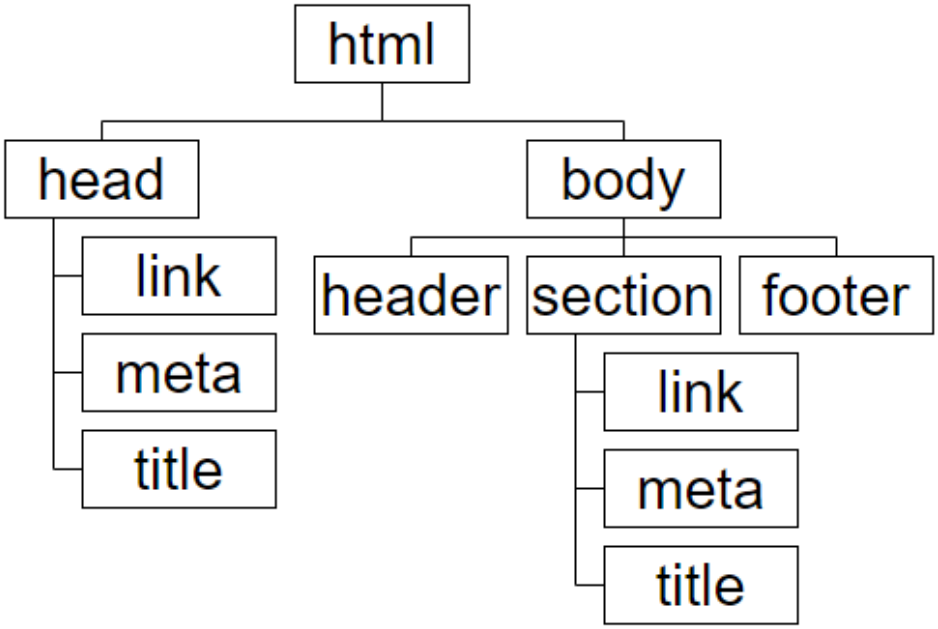

Рисунок 1 – Пример DOM-представления для простого HTML документа

В `body` могут находиться разнообразные элементы. Например, в родительском `body` — дочерний элемент `header`, в элементе `header` — дочерний элемент `section`, в родительском `section` — дочерний `div`, в `div` — элемент `h3`, и, наконец, в `h3` — элемент `span`. В этом случае `span` не имеет дочерних элементов, но их можно добавить в любой момент. Пример того, как можно описать добавление данных элементов, представлен на рисунке 2.

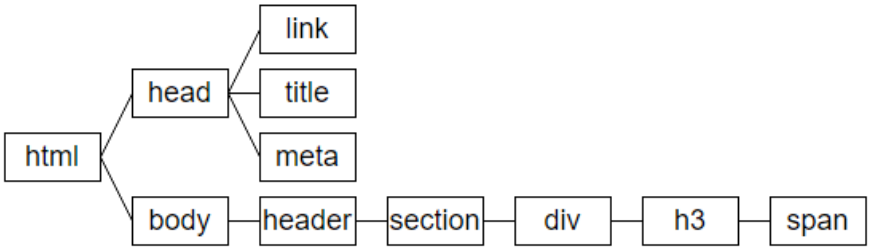

Рисунок 2 – Пример DOM-представления для чуть более сложного HTML документа

Элементы могут наследовать не все, но многие свойства своих родителей — например, цвет, шрифт, видимость, размеры и т. д.

Таким образом, чтобы задать стиль шрифта на всей странице, потребует-

ся не прописывать цвет для каждого элемента, а задать его только для `body`. А чтобы изменить наследуемое свойство у дочернего элемента, нужно прописать только ему новые свойства. Наследование удобно для создания единообразной страницы. DOM-узлы содержат гораздо больше информации, чем просто данные о дочерних узлах [15]. Они также содержат информацию о родительском узле, стилях, обработчиках событий и свойствах элемента, его исходном коде и т. д. В таких браузерах, как Google Chrome, DOM-дерево страницы можно посмотреть, например, при помощи инструментов разработчика [16].

1.3.2 Обновления документа с использованием DOM

Обновление структуры DOM — распространённая и часто используемая операция, производимая, например, в случае, когда документ должен меняться в ответ на действия пользователя (любая активность на странице).

Рекоммендация W3C в таком случае призывает повторно отрисовать обновлённое дерево с нуля — то есть, каждый раз, когда необходимо обновить дерево, будет использоваться алгоритм отображения [17]: DOM-дерево строится заново и производится операция вставки элемента n раз, где n - количество элементов.

Благодаря тому, что современные браузеры реализуют DOM и предоставляют API для взаимодействия с DOM-деревом, фреймворки, такие, как Angular, используют именно этот API в своих готовых решениях [18].

1.4 Алгоритмы, использующие рендеринг на стороне сервера

Рендеринг на стороне сервера (англ. *Server Side Rendering* — SSR) — метод разработки программного обеспечения в web-приложении, которое обрабатывает запросы от пользователя на сервере, который исполняет алгоритмы в соответствии с потребностями бизнеса, отправляя пользователю ответ как ре-

зультат работы почти со всем процессом рендеринга [19].

SSR подразумевает, что при обращении к web-странице пользователь получает HTML страницу, работающую согласно одному из HTML стандартов. В данной работе алгоритмы, использующие SSR, не будут рассмотрены подробно, поскольку использование SSR помогает ускорить загрузку страниц, но не обновление их содержимого [19], алгоритм обновления эквивалентен алгоритму обновления, использующему объектную модель документа, рассмотренную выше.

Таким образом, при обновлении гипертекстовых документов с использованием SSR используется рекомендованный W3C алгоритм обновления с использованием DOM, поэтому имеет смысл рассматривать именно алгоритм обнвления с использованием DOM.

1.5 Алгоритмы, использующие виртуальную объектную модель документа

Основной проблемой DOM является то, что он никогда не был приспособлен для динамического интерфейса [3]. Он предоставляет удобный программный интерфейс, позволяющий взаимодействовать с ним из кода, но это не решает проблем с производительностью. Современные социальные сети, такие, как ВКонтакте, Twitter или Facebook будут использовать тысячи DOM узлов, взаимодействие с которыми может занимать ощутимое для пользователя время, в то время как создатели браузера Google Chrome, к примеру, рекомендуют [20] не использовать в одной странице более 800 узлов.

Одним из решений данной проблемы служит технология виртуальной объектной модели, которая, хоть и не является стандартом, позволяет по-прежнему взаимодействовать с DOM, но делать это как можно реже.

Виртуальная объектная модель документа (англ. *Virtual Document Object Model* — VDOM) [21] — концепт программирования, в которой идеальное («вир-

туальное») представление пользовательского интерфейса хранится в памяти и синхронизуется с «настоящим» DOM при помощи алгоритмов согласования.

Вместо того, чтобы работать с DOM напрямую, согласно концепции VDOM работа происходит с его легковесной копией, хранящийся непосредственно в памяти компьютера. За счёт этого операции вставки, сравнения, удаления и обхода узлов дерева происходит быстрее благодаря отсутствию необходимости отрисовывать изменения после каждой операции.

Когда в пользовательский интерфейс добавляются новые элементы, создаётся виртуальная модель DOM, представленная в виде дерева. Если состояние любого из элементов изменяется, создаётся новое виртуальное дерево DOM. Затем это дерево сравнивается с предыдущим виртуальным деревом.

Рассмотрим, как для простого HTML документа, представленного на листинге 2, осуществляется представление DOM- (рисунок 3) и VDOM-дерева (листинг 3).

Листинг 2 – Простой HTML документ

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <link/>
</head>
<body>
  <ul class="list">
    <li class="list__item">List item</li>
  </ul>
</body>
</html>
```

Листинг 3 – VDOM-дерево для простого HTML документа

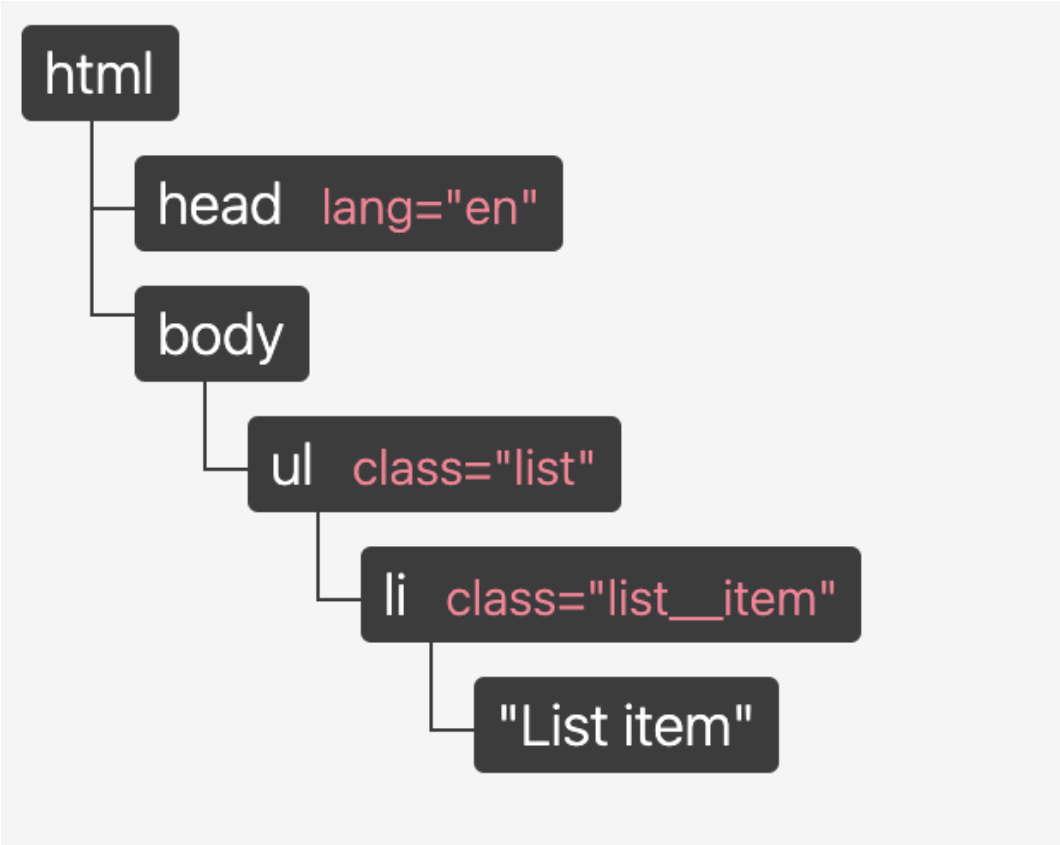

Рисунок 3 – DOM-дерево для простого HTML документа

```
const vdom = {
  tagName: "html",
  children: [
    { tagName: "head" },
    {
      tagName: "body",
      children: [
        {
            tagName: "ul",
            attributes: { "class": "list" },
            children: [
                {
                    tagName: "li",
                    attributes: { "class": "list__item" },
                    textContent: "List item"
```

```
                    } // end li
                ]
            } // end ul
        ]
    } // end body
  ]
} // end html
```

Данный пример служит лишь для демонстрации того, как может осуществляться представление VDOM. Реальные деревья содержат намного большее число узлов [15], а элементы DOM имеют гораздо больше полей (классы, идентификаторы, стили, обработчики событий, поля данных, типы и значения, вспомогательные поля и т. п.), и тем не менее содержат гораздо меньше информации, чем узлы DOM за счёт того, что нет необходимости хранить данные, помогающие визуализировать элемент (за это отвечает DOM).

Существует несколько способов использовать VDOM для оптимизации алгоритма отображения гипертекстового документа. с использованием алгоритма согласования и с использованием Fiber алгоритма.

1.5.1 Обновление документа с использованием VDOM и алгоритма согласования

Если состояние любого из элементов необходимо изменить, создаётся новое виртуальное дерево. Затем получившееся дерево сравнивается с предыдущим виртуальным деревом объектной модели документа и происходит вычисления наилучшего из возможных методов внесения изменений в реальной DOM. Это гарантирует минимальное количество операций с реальной DOM, что приводит к снижении стоимости обновления реальной модели.

На рисунке 4 показано виртуальное дерево DOM и процесс сравнения.

Красными кружками обозначены узлы, которые изменились — эти уз-

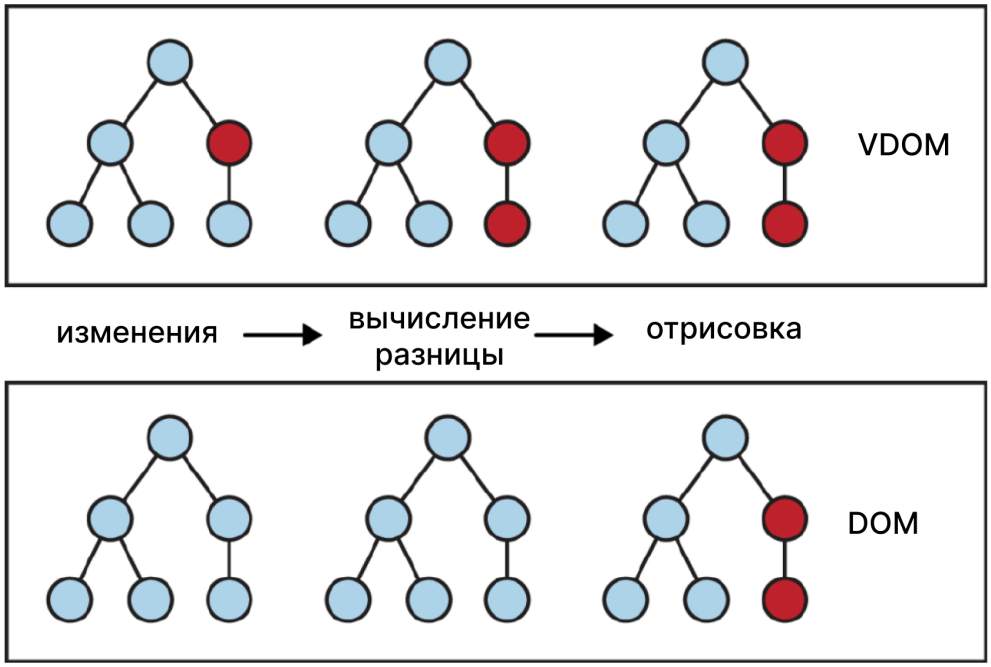

Рисунок 4 – Пример работы алгоритма обновления документа при помощи VDOM

лы представляют собой элементы пользовательского интерфейса, состояние котоых изменилось. Затем вычисляется разница между предыдущей версией VDOM-дерева и текущей (посредством алгоритма согласования), после чего всё родительское поддерево повторно визуализируется для получения обновлённого пользовательского интерфейса.

1.5.1.1 Алгоритм согласования

Алгоритм согласования (англ. *reconciliation*) [22] — эвристический алгоритм решения проблемы трансформации одного дерева в другое за минимальное число операций. Он основан на следующих двух предположениях:

Во-первых, два элемента с разыми типами произведут разные деревья.

Во-вторых, можно указать, какие дочерние элементы могут оставаться стабильными между разными отображениями при помощи специального параметра `key`.

При сравнении двух деревьев первым делом производится сравнение родительских элементов, начиная с корневого. Дальнейшее поведение различает-

ся в зависимости от типов родительских элементов.

Всякий раз, когда родительские элементы имеют различные типы, старое VDOM-дерево уничтожается и с нуля строится новое. Так, переходы от `<div>` к `<a>` или от `<img>` к `<div>` приведут к полному перестроению дерева, старое DOM-дерево также уничтожается.

Любые элементы, лежащие ниже родительского, будут уничтожены, а их состояние уничтожится. На листиге 4 приведён пример изменения узла, при котором меняется его тип и всё поддерево уничтожается.

Листинг 4 – Пример HTML-узлов разного типа, при сравнении которых деревья будут полностью перестроены

```
<!-- Старый элемент -->
<div>
  <a href="example.com" />
</div>

<!-- Новый элемент -->
<span>
  <a href="example.com" />
</span>
```

При сравнении DOM-элементов одного типа, алгоритм согласования проверяет их атрибуты, сохраняет лежащий в основе элементов DOM-узел и обновляет только изменённые атрибуты (пример таких элементов приведён на листинге 5).

Листинг 5 – Пример HTML-узлов одного типа, при сравнении которых будет сохранена лишь разница содержимого атрибутов

```
<!-- Старый элемент -->
<div class="before" title="stuff" />
```

```
3
4   <!-- Новый элемент -->
5   <div class="after" title="stuff" />
```

При рекурсивном обходе дочерних элементов DOM-узла алгоритм согласования проходит по спискам потомков одновременно и находит отличие. Например, при добавлении элемента в конец дочерних элементов, преобразование между этими деревьями работает хорошо, определяя минимальный набор операций.

Листинг 6 – Пример HTML-узлов при добавлении элемента в конец дочернего

```
1   <!-- Старый элемент -->
2   <ul>
3     <li>первый</li>
4     <li>второй</li>
5   </ul>
6
7   <!-- Новый элемент -->
8   <ul>
9     <li>первый</li>
10    <li>второй</li>
11    <li>третий</li>
12  </ul>
```

Алгоритм согласования сравнит два дерева `<li>первый</li>`, сравнит два дерева `<li>второй</li>`, а затем вставит дерево `<li>третий</li>`.

При вставке элемента в начало, прямолинейная реализация такого алгоритма будет работать не эффективно. Например, преобразование между этими

деревьями работает плохо, минимальный набор операций для преобразованя одного дерева в другое не будет найден.

Алгоритм согласования будет преобразовывать каждого потомка, вместо того, чтобы оставить элементы `<li>Санкт-Петербург</li>` и `<li>Москва</li>` . Пример таких узлов продемонстрирован на листинге 7.

Листинг 7 – Пример HTML-узлов при добавлении элемента в начало дочернего

```
<!-- Старый элемент -->
<ul>
  <li>Санкт-Петербург</li>
  <li>Москва</li>
</ul>


<!-- Новый элемент -->
<ul>
  <li>Ростов-на-Дону</li>
  <li>Санкт-Петербург</li>
  <li>Москва</li>
</ul>
```

Для решения этой проблемы существуют вспомогательные атрибуты `key` — ключи [23]. Когда у дочерних элементов есть ключи, алгоритм согласования использует их, чтобы сопоставить потомков исходного виртуального дерева с потомками полученного виртуального дерева.

Значение атрибута `key` при этом должно быть уникальным. Так, как значение ключа можно использовать уникальный идентификатор элемента, если он есть, или же можно добавить новое свойство идентификатора или прохешировать данные, чтобы сгенерировать ключ. Достаточно, чтобы ключ элемента

был уникальным среди его соседей, а не глобально.

Например, если добавить `key` к примеру выше, преобразование дерева будет верным.

Листинг 8 – Пример HTML-узлов при добавлении элемента в начало дочернего с использованием ключей

```
<!-- Старый элемент -->
<ul>
  <li key="2015">Санкт-Петербург</li>
  <li key="2016">Москва</li>
</ul>

<!-- Новый элемент -->
<ul>
  <li key="2014">Ростов-на-Дону</li>
  <li key="2015">Санкт-Петербург</li>
  <li key="2016">Москва</li>
</ul>
```

Алгоритм согласования является эвристическим [24] алгоритмом, и если предположения, на которых он основан, не соблюдены, пострадает производительность.

Так, алгоритм не будет пытаться сопоставить поддеревья элементов разных типов, а ключи должны быть стабильными (например, ключ, произведённый случайно, не является стабильным), предсказуемым и уникальным. Нестабильные ключи вызовут необязательное пересоздание многих экземпляров элемента и DOM-узлов, что может вызывать ухудшение производительности и потерю состояния у дочерних элементов.

1.5.2 Обновление документа с использованием Fiber алгоритма

Fiber — основной алгоритм сравнения библиотеки React, начиная с 16 версии конкурирующий с алгоритмом согласования [25]. Для пользовательского интерфейса не важно, чтобы каждое обновление было применено сразу; фактически такое поведение будет лишним, оно будет способствовать падению количества кадров в секунду и ухудшению пользовательского опыта [26].

Волокно (англ. *fiber*) — объект, являющийся абстракцией работы, которую необходимо выполнить [26]. Работой являются любые вычисления, которые должны быть выполнены для того, чтобы определить, какие объекты необходимо обновлять, а какие можно оставить в прежнем состоянии. Такая работа должна иметь возможность быть остановленной и возобновлённой. Рассмотрим следующий пример: пусть имеются HTML-элементы, представленные на листинге 9.

Листинг 9 – Пример HTML-элементов для демонстрации структуры волокна

```
<div class="HostRoot">
    <li class="list">
        <button class="btn">^2</button>
        <div>1</div>
        <div>2</div>
        <div>3</div>
    </li>
</div>
```

Волокно, отвечающее за вычисления, связанные с кнопкой, имеющий класс btn, будет иметь следующий вид, представленный на листинге 10.

Листинг 10 – Fiber объект для элемента кнопки

```
const fiber = {
    stateNode: HTMLButtonElement,
    child: null,
    return: HTMLLIElement,
    sibling: HTMLDivElement
}
```

Поля объекта, представленного на листинге 10, имеют следующий смысл:

1) stateNode — указатель на объект DOM-дерева, соответствующего элементу волокна (в данном случае кнопке);
2) child — указатель на волокно первого дочернего элемента (в данном случае дочерних элементов нет);
3) return — указатель на волокно родительского элемента (в данном слечае список);
4) sibling — указатель на первый соседний элемент (в данном случае элемент div).

Вывод

В данном разделе были рассмотрены принципы работы с гипертекстовыми документами согласно объектной модели документа (DOM) и витруальной объектной модели документа (VDOM) и проведено их сравнение. Также был рассмотрен алгоритм согласования, находящий минимальный набор операций, необходимых для преобразования одного дерева в другое. В соответствии с поставленной целью, необходимо разработать алгоритмы обновления документа с использованием DOM и VDOM, а также алгоритма согласования.

На вход алгоритма обновления с использованием DOM будут подаваться обрабатываемый элемент, изменяемый элемент и элемент, который нужно

отрисовать вместо него. На вход алгоритма обновления документа с использованием VDOM будут подаваться корни старого VDOM-дерева и нового VDOM-дерева, а на вход алгоритма согласования - обрабатываемые узлы старого и нового VDOM-деревьев, а также указатели на массивы изменений узлов и поддеревьев,

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DOM Living standart [Электронный ресурс]. — Режим доступа: https://dom.spec.whatwg.org/, свободный (дата обращения: 09.11.2022).

2. HTML Living standart [Электронный ресурс]. — Режим доступа: https://html.spec.whatwg.org/multipage/, свободный (дата обращения: 09.11.2022).

3. Document Object Model (DOM) Level 3 Core Specification [Электронный ресурс]. — Режим доступа: https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/, свободный (дата обращения: 25.11.2022).

4. Демин И. С. Проблемы развития гипертекстовых сред — М: Вестник ОГУ, 2004. — 79 с.

5. Document Object Model (DOM) [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model, свободный (дата обращения: 09.11.2022).

6. Гипертекст — определение, Большая российская энциклопедия [Электронный ресурс]. — Режим доступа: https://bigenc.ru/technology_and_technique/text/4426247, свободный (дата обращения: 25.11.2022).

7. HTML [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTML, свободный (дата обращения: 09.11.2022).

8. Интернет — определение, Большая российская энциклопедия [Электронный ресурс]. — Режим доступа: https://bigenc.ru/technology_and_technique/text/2014701, свободный (дата обращения: 25.11.2022).

9. HTML5 is W3C recommendation [Электронный ресурс]. — Режим доступа: https://www.w3.org/blog/news/archives/4167, свободный (дата обращения: 09.11.2022).

10. World Wide Web Consortium (W3C) [Электронный ресурс]. — Режим доступа: https://www.w3.org/, свободный (дата обращения: 25.11.2022).

11. Яндекс.Радар Браузер в России [Электронный ресурс]. — Режим доступа: https://radar.yandex.ru/browsers, свободный (дата обращения: 09.11.2022).

12. W3C Recommendation 14 December 2010 [Электронный ресурс]. — Режим доступа: w3.org/TR/mwabp/#webapp-defined, свободный (дата обращения: 09.11.2022).

13. Most used web frameworks among developers worldwide, as of 2023 [Электронный ресурс]. — Режим доступа: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/, свободный (дата обращения: 09.11.2022).

14. Сбалансированные деревья, М.В. Губко. — М.: Институт проблем управления РАН [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/sbalansirovannye-derevya/viewer, свободный (дата обращения: 09.11.2022).

15. DOM Element, Web API Reference [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Element#specifications, свободный (дата обращения: 25.11.2022).

16. Chrome DevTools [Электронный ресурс]. — Режим доступа: https://developer.chrome.com/docs/devtools/, свободный (дата обращения: 25.11.2022).

17. On Layout & Web Performance [Электронный ресурс]. — Режим доступа: https://kellegous.com/j/2013/01/26/layout-performance/, свободный (дата обращения: 11.10.2023).

18. Angular Core API: Renderer [Электронный ресурс]. — Режим доступа: https://angular.io/api/core/Renderer2, свободный (дата обращения: 11.10.2023).

19. Taufan F. I. Comparison between client-side and server-side rendering in the web development. — IOP Conf. Ser.: Mater. Sci., 2020.

20. Избегайте чрезмерного размера DOM [Электронный ресурс]. — Режим доступа: https://web.dev/i18n/ru/dom-size/, свободный (дата обращения: 25.11.2022).

21. Виртуальный DOM и детали его реализации в React [Электронный ресурс]. — Режим доступа: https://ru.reactjs.org/docs/faq-internals.html#gatsby-focus-wrapper, свободный (дата обращения: 12.11.2022).

22. Согласование [Электронный ресурс]. — Режим доступа: https://ru.reactjs.org/docs/reconciliation.html, свободный (дата обращения: 12.11.2022).

23. Списки и ключи [Электронный ресурс]. — Режим доступа: https://ru.reactjs.org/docs/lists-and-keys.html, свободный (дата обращения: 25.11.2022).

24. Большакова Е. И., Мальковский М. Г., Пильщиков В. Н. Искусственный интеллект. Алгоритмы эвристического поиска (учебное пособие) — М.: Издательский отдел факультета ВМК МГУ (лицензия ИД № 05899 от 24.09.01), 2002. – 83 с.

25. Прокофьев, А. П. FIBER–НОВЫЙ АЛГОРИТМ СРАВНЕНИЯ REACT. — Тольятти, Научный электронный журнал «Инновации. Наука. Образование № 51 (февраль), 2022. — 2149 с.

26. Fedosejev A, Boduch A. React 16 Essentials: A fast-paced, hands-on guide to designing and building scalable and maintainable web apps with React 16. — Birmingham, Packt Publishing Ltd, 2017. — 284 с.

27. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. — М. Наука, Физматлит, 2007. — 376 с.