Инженерный вестник Дона, №3 (2022)
ivdon.ru/ru/magazine/archive/n3y2022/7519

Концепция Virtual Dom в библиотеке React.js

К.Ю. Бетеев, Г.В. Муратова

Южный федеральный университет, Ростов-на-Дону

Аннотация: В статье рассматривается концепция работы технологии Virtual DOM в библиотеке для создания пользовательских веб-интерфейсов React.js. Производится сравнительный анализ подходов к построению современных пользовательских интерфейсов. Рассматривается также технологическая реализация внутренних механизмов и алгоритмов работы концепции Virtual DOM в React.js. Описываются современные тенденции в области разработки веб-приложений.
Ключевые слова: обработка данных, визуализация данных, веб-приложения, пользовательские интерфейсы, алгоритмы, разработка программного обеспечения, обновление данных, клиент-серверные приложения, тенденции веб-разработки.

Темпы развития технологий создания современных веб-приложений значительно выросли с того момента, как появилась потребность в использовании динамических интерфейсов. Такой рост был ознаменован появлением AJAX (от англ. Asynchronous Javascript and XML — «асинхронный JavaScript и XML») - подходом к построению интерактивных пользовательских интерфейсов веб-приложений, заключающимся в «фоновом» обмене данными браузера с веб-сервером [1]. Веб-приложения, использующие такой подход, позволяли взаимодействовать с сервисом без перезагрузки страницы браузера. Помимо улучшения опыта использования веб-приложений, данный подход позволил в некоторой мере уменьшить нагрузку на сервера приложений за счет постепенного сокращения времени на генерацию HTML документа, отображаемом в браузере пользователей.

Появление AJAX также и сказалось и на общей реорганизации проектирования архитектуры веб-сервисов. Если раньше в обязанности одних и тех же специалистов входили такие задачи, как проектирование баз данных, разработка бизнес-логики приложений, проектирование и генерация HTML и CSS документов, то теперь общий фронт работ по разработке сервиса можно поделить на 2 условные части: frontend и backend.

Соответственно, в обязанности backend программистов входили задачи по разработке общей логики, без привязки в среде отображения к конечным пользователям, тогда как frontend программисты брали на себя обязанность в осуществлении доступного и корректного отображения результатов взаимодействия с основной логикой сервиса в браузере.

Вопрос корректного и эффективного отображения пользовательских интерфейсов в браузере стоит и по сей день. Различные фреймворки и библиотеки предлагают свои инструменты для работы с отображением страниц, среди которых есть достаточно интересные для рассмотрения решения. В частности, стоит отметить фреймворк jQuery [2], пользовавшийся большой популярностью для разработки интерфейсов до выхода новой версии JavaScript (ES6/ES2015) в 2015 году, который предоставлял множество недостающих возможностей и инструментов в языке до ES6.

DOM в браузере

Для взаимодействия с содержимым веб-страницы в браузере разработчиками предусмотрен механизм, который называется DOM API.

DOM (от англ. Document Object Model — «объектная модель документа») — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов [3]. DOM обеспечивает представление документа в виде структурированной группы узлов и объектов, которые имеют свойства и методы.

Соответственно, DOM API предоставляет регламент для взаимодействия внешних скриптов с содержимым веб-страницы в браузере [4, 5]. DOM API предоставляет широкий спектр функциональности, среди которых имеются как методы для управления содержимым страницы, так и методы для асинхронной обработки данных.

Следует отметить, что большое количество приложений и в настоящий момент успешно функционируют с помощью прямого обращения к узлам DOM. Язык JavaScript предоставляет множество встроенных методов для таких манипуляций [6], и в целом в данном подходе нет никаких проблем, пока веб-приложение удается поддерживать на приемлемом уровне без потерь производительности.

```
/**
 * Получение ссылки к DOM элементу по его идентификатору и её присвоение переменной.
 * */
const completeBtn = document.getElementById( elementId: 'complete-button');

/**
 * Объявление функции, которую необходимо выполнить при инициации события.
 * */
function completeAction() {
    // ⟶ Логика выполнения операции.
}

/**
 * Установка слушателя на клик по объявленному элементу из DOM.
 * */
if (completeBtn) {
    completeBtn.addEventListener( type: 'click', completeAction);
} else {
    throw new Error('completeBtn is not found on page');
}
```

Рис. 1. – Пример прямого обращения к DOM из внешнего скрипта

Однако если такой подход попытаться масштабировать для крупных веб-приложений, то это может привести к следующим проблемам:

1. Сложность в масштабировании императивной кодовой базы. Подход, при котором разработчику необходимо напрямую обращаться к DOM API, является императивным, так как ему требуется каждый раз описывать, как именно браузеру отображать тот или иной узел. При масштабировании проекта,

такой подход приведет к увеличению сложности понимания кодовой базы.

2. Частые манипуляции с DOM негативно влияют на производительность [7, 8].
3. Обновление страницы с большим количеством узлов в DOM является достаточно медленным процессом, так как браузеру необходимо заново провести рендеринг всего содержимого. Данная операция блокирует основной поток исполнения, где длительное обновление может негативно сказаться на пользовательском опыте.

Возникновение вышеприведенных проблем побудило разработчиков к созданию собственных механизмов для эффективной работы с содержимым DOM, среди которых выделяется концепция Virtual DOM.

Концепция Virtual DOM

Virtual DOM (сокращенно VDOM) — это концепция программирования, в которой «идеальное» или «виртуальное» представление пользовательского интерфейса хранится в памяти и синхронизируется с «настоящим» DOM [9].

В библиотеке React для каждого реального объекта в DOM существует виртуальный аналог в Virtual DOM, который представляет собой объект, описывающий определенный узел.

```
<ul class="list">
    <li class="list_item">Первый элемент списка</li>
    <li class="list_item">Второй элемент списка</li>
</ul>
```

Рис. 2. – Пример представления узла в HTML

```
const nodeElement = {
    tagName: 'ul',
    attributes: { class: 'list' },
    children: [
        {
            tagName: 'li',
            attributes: { class: 'list_item' },
            textContent: 'Первый элемент списка',
        },
        {
            tagName: 'li',
            attributes: { class: 'list_item' },
            textContent: 'Второй элемент списка',
        },
    ],
};
```

Рис. 3. – Упрощенный пример представления аналогичного узла в виде объекта в Virtual DOM

Так как данный объект представлен в памяти во время исполнения скрипта, то мы можем манипулировать с ним по любому сценарию с производительностью, не уступающей обычной работе с объектами в JavaScript.

Стоит учесть тот факт, что имеется возможность отложенной отрисовки изменения в Virtual DOM в реальном DOM до определенного момента, например, при работе с большим списком данных, при котором необходимо отсортировать список по различным критериям, постепенно сужая его в несколько итераций. Подобные манипуляции при работе с реальным DOM приводили бы к множественным переотрисовкам, что явно бы сказалось на производительности. Virtual DOM в React имеет возможность отложить лишние перерисовки интерфейса до момента полной сортировки всего списка.

Согласование Virtual DOM с реальным DOM

С момента добавления новых элементов пользовательского интерфейса, в браузере создается его аналог в виде объекта-дерева Virtual DOM, где каждый вложенный элемент является его узлом. При последующем изменении состояния элемента создается новое дерево в Virtual DOM, которое сравнивается с предшествовавшим ему деревом.

После этого вычисляется наиболее эффективный метод внесения изменений в реальный DOM. Цель данных вычислений состоит в минимизации количества операций, совершаемых с реальным DOM, тем самым, уменьшаются накладные расходы, связанные с обновлением DOM.

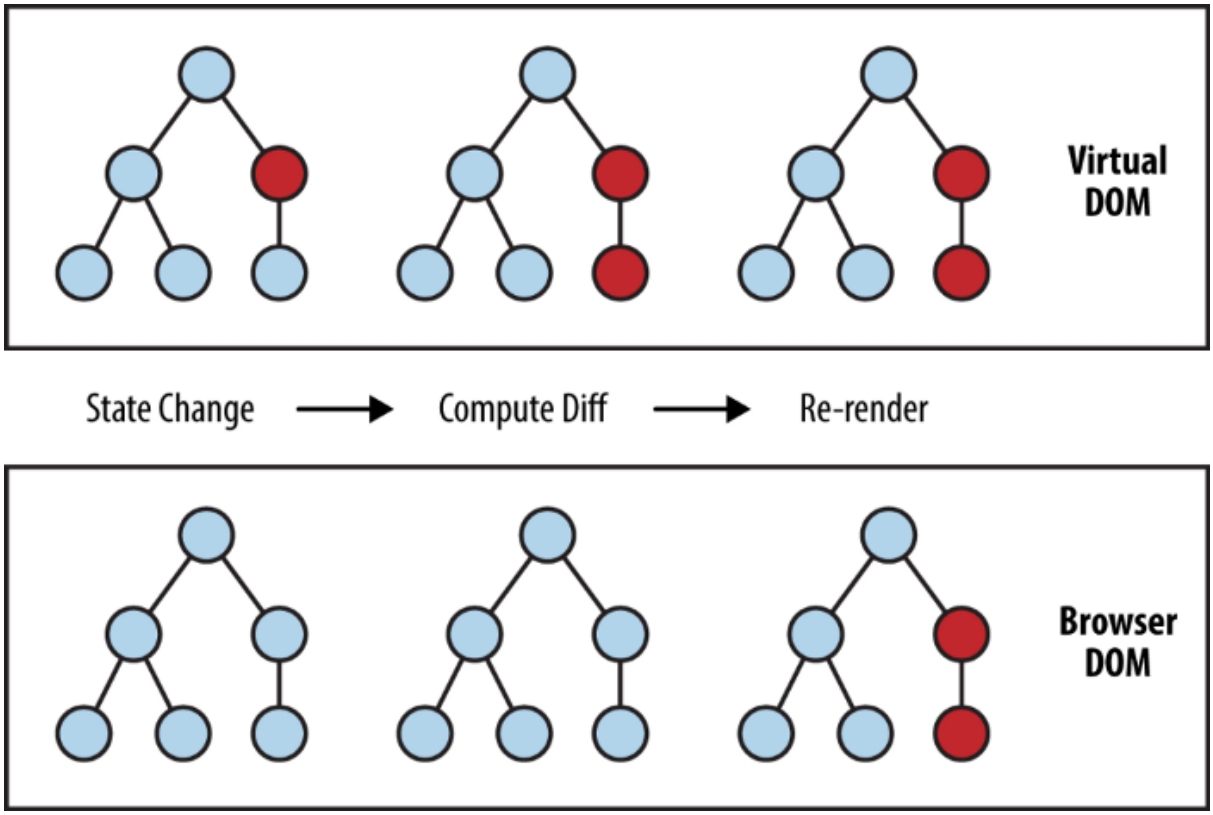

Рис. 4. – Пример согласования Virtual DOM с реальным DOM браузера [10]

На рис. 4 на шаге №1 (State Change) красным цветом отображены узлы в Virtual DOM, состояние которых было изменено в процессе манипуляций с интерфейсом. На шаге №2 (Compute Diff) происходит процесс вычисления оптимального пути для перерисовки дерева в DOM, после чего на шаге №3 (Re-render) осуществляется перерисовка полученного элемента в реальный DOM браузера.

Механизмы React для эффективной перерисовки DOM

Для оптимизации процесса согласования и перерисовки пользовательского интерфейса библиотека React.js использует следующие механизмы [11, 12]:

1. React использует паттерн проектирования «Наблюдатель» (Observer) и реагирует на изменения состояния элемента.

   Каждый элемент, который должен быть отображен в интерфейсе в React, представлен в виде компонента (функционального или классового), который может иметь как собственное состояние, так и состояние, переданное родительским компонентом. React отслеживает изменения состояний компонента, и если его состояние было изменено, запускает цепочку механизмов для сравнения и перерисовки интерфейса в DOM. Стоит отметить, что при изменении состояния определенного компонента другие независимые компоненты обновлены не будут (если они не зависят от состояния изменившегося компонента в Virtual DOM).
2. React использует механизм пакетного (batch) обновления DOM. Как было отмечено ранее, React имеет возможность отслеживать цепочку синхронных изменений, перерисовку которых можно отложить до завершения манипуляций с состоянием. Пакетное обновление в DOM позволяет существенно уменьшить

потенциальную потерю эффективности при множественных обновлениях состояния компонента. Соответственно, даже если состояние компонента было синхронно изменено несколько раз, процесс перерисовки в DOM будет осуществлен только единожды.

3. React использует эффективный алгоритм поиска различий с линейной сложностью O(n), позволяющий сравнивать деревья в Virtual DOM в процессе согласования перед отрисовкой в браузере.

Стоит отметить, что эффективная работа с DOM возможна не только с помощью внедрения технологии Virtual DOM. Многие фреймворки для разработки веб-приложений успешно используют и другие подходы.

Например, фреймворк Svelte не использует концепцию Virtual DOM, что позволяет разрабатывать веб-приложения со значительно меньшим размером исходного файла, за счёт чего достигается оптимизация загрузки компонентов веб-приложения. Вместо того, чтобы эмулировать работу DOM, Svelte компилирует исходный код в чистый JavaScript, который поддерживает точечное обновление необходимых узлов в DOM. Стоит заметить, что Svelte, как и React, предоставляет все возможности декларативного описания пользовательских интерфейсов.

Фреймворк для создания веб-приложений Angular также не использует концепцию Virtual DOM. Вместо этого, Angular реализует собственные механизмы для поиска изменений, требующих обновления в реальном DOM. Несмотря на то, что прямые обращения к DOM являются достаточно ресурсоемкими, Angular, как и Svelte, рассчитывает точечные изменения для его эффективного обновления.

Современные тенденции в области создания веб-приложений, в целом, направлены на развитие PWA (от англ. Progressive Web Applications – «прогрессивные веб-приложения»), в которых классический веб-сайт приобретает все признаки полноценного десктопного или мобильного приложения (например, полное клиент-серверное взаимодействие, возможность работы без перезагрузки страницы, кеширование данных и многое другое). Таким образом, наблюдается увеличение количества новых решений, направленных на оптимизацию работы со всеми элементами, составляющих современное веб-приложение.

Литература

1. Маклафин Б. Изучаем Ajax. СПб: Питер, 2008, 443 с.

2. Глод О.Д., Чекулаев А.А. Разработка веб-приложения с использованием Spring Framework и jQuery // Инженерный вестник Дона, 2018, №4. URL: ivdon.ru/ru/magazine/archive/n4y2018/5453.

3. Руководство по DOM. // MDN Web Docs. URL: developer.mozilla.org/ru/docs/Web/API/Document_Object_Model.

4. Балеев И.А., Земцов А.Н., Астахов Д.А. Методы обработки и визуализации данных в веб-интерфейсе с помощью библиотеки Dimensional Charting // Инженерный вестник Дона, 2021, №6. URL: ivdon.ru/ru/magazine/archive/n6y2021/7029.

5. Спецификация DOM. URL: dom.spec.whatwg.org.

6. Спецификация ECMASript. // ECMA International. URL: ecma-international.org/publications-and-standards/standards/ecma-262.

7. Флэнаган Д. JavaScript, Подробное руководство. СПб: Диалектика-Вильямс, 2021, 720 с.

8. Крокфорд Д. Как устроен JavaScript. СПб: Питер, 2019, 304 с.

9. Виртуальный DOM и детали его реализации в React. // React.js Library. URL: ru.reactjs.org/docs/faq-internals.html

10. Ayush Verma. 2021. The Comprehensive Guide to React's Virtual DOM. URL: javascript.plainenglish.io/react-the-virtual-dom-comprehensive-guide-acd19c5e327a.

11. Томас Марк Т. React в действии. СПб: Питер, 2019, 367 с.

12. Чиннатамби К. Изучаем React. Москва: Бомбора, 2019, 368 с.

References

1. McLuffin B. Izuchaem Ajax [Learning Ajax]. SPb.: Piter, 2008, 443 p.

2. Glod O.D., Chekulaev A.A. Inzhenernyj vestnik Dona, 2018, №4. URL: ivdon.ru/ru/magazine/archive/n4y2018/5453.

3. Rukovodstvo po DOM [DOM guide]. MDN Web Docs. URL: developer.mozilla.org/ru/docs/Web/API/Document_Object_Model.

4. Baleev I.A., Zemczov A.N., Astakhov D.A. Inzhenernyj vestnik Dona, 2021, №6. URL: ivdon.ru/ru/magazine/archive/n6y2021/7029.

5. Specifikaciya DOM [DOM Specification]. URL: dom.spec.whatwg.org.

6. Specifikaciya ECMASript [ECMASript Specification]. Ecma international. URL: ecma-international.org/publications-and-standards/standards/ecma-262.

7. Flanagan D. JavaScript, Podrobnoe rukovodstvo [JavaScript: The Definitive Guide]. SPb.: Dialectika-Williams, 2021, 720 p.

8. Crockford D. Kak ustroen JavaScript [JavaScript: The Good Parts]. SPb.: Piter, 2019, 304 p.

9. Virtual`ny`j DOM i detali ego realizacii v React [Virtual DOM and implementation details in React]. React.js Library. URL: ru.reactjs.org/docs/faq-internals.html.

10. Ayush Verma. 2021. The Comprehensive Guide to React's Virtual DOM. URL: javascript.plainenglish.io/react-the-virtual-dom-comprehensive-guide-acd19c5e327a.

11. Thomas Mark T. React v dejstvii [React in Action]. SPb.: Piter, 2019, 367 p.

12. Chinnathambi K. Izuchaem React [Learning React: A Hands-on Guide to Building web applications using react and redux, 2nd ed]. Moscow: Bombora, 2019, 368 p.