Инженерный вестник Дона, №11 (2023)
ivdon.ru/ru/magazine/archive/n11y2023/8794

Исследование фреймворка React для реализации части Представления клиентских веб-приложений

И.А. Короленко

Yandex Europe B.V. (Amsterdam, Netherlands)

Аннотация: В данной работе рассматривается вопрос необходимости применения фреймворков как таковых, а также React, наиболее часто используемый фреймворк для создания части Представления клиентских веб-приложений. Изучаются его устройство, предлагаемый подход в разработке клиентских приложений, решаемые и создаваемые им проблемы, сильные и слабые стороны, а также особенности и ограничения в применении.
Ключевые слова: view frameworks, клиентские веб-приложения, front end, React, Reconciliation, Fiber, рендеринг.

Введение

Разработка крупного и сложного во внутреннем устройстве проекта с десятками тысяч строк кода, тысячами модулей и сотнями пользовательских сценариев – это крайне сложная задача для любой команды даже самых опытных инженеров. Сложность заключается далеко не только в объемах работы, хотя несомненно это существенная часть, но также в организации работы по задачам, параллелизации разработки, и поддержании читаемости и возможности внесения изменений в код. Эти параметры во многом зависят от консистентности кода, применяемых подходов, и инструментов в разработке. Команда, каждый участник которой использует свои инструменты и мыслит своими категориями и подходами к разработке, крайне маловероятно сможет написать работающий и простой в поддержании продукт, срок жизни которого будет превышать пару лет. Для решения этих фундаментальных проблем, как правило, применяются фреймворки – абстракции, предоставляющие инструменты стандартизации решения определенного набора задач в разработке [1]. Одним из наиболее часто используемых паттернов в веб-фреймворках является Model-View-Controller (MVC). При использовании данного паттерна приложение разделяется на слои Модели, Представления и Контроллера, каждый из которых отвечает за свою часть

логики [2]. Модель отвечает за предоставление данных, Представление за предоставление интерфейса для отображения и пользовательского взаимодействия с данными, а Контроллер обеспечивает связь между Моделью и Контроллером. Таким образом обеспечивается разделение обязанностей и однообразность подходов, что снижает сложность кода и позволяет переиспользовать код без его модификации [3]. Это не единственный возможный паттерн для построения фреймворка, но каждый из них обеспечивает данные существенные преимущества. Целью данной статьи является рассмотрение фреймворка React для построения клиентской части веб-приложений, его внутреннего устройства и особенностей его применения на практике.

Необходимость использования фреймворков

Первый вопрос, который следует адресовать в рамках данной статьи – это вопрос принципиальной необходимости использования фреймворков как таковых для построения клиентского веб-приложения. Достаточно часто в сообществе front end разработчиков можно услышать мнение о том, что фреймворки только усложняют процесс разработки и ограничивают пространство потенциальных решений, поэтому стоит предпочесть им разработку на чистом JavaScript [4]. Однако, данные рассуждения на деле являются крайне непрактичными, так как если вы создадите даже небольшое приложение, используя только чистый JavaScript и идиоматические способы решения задач в этом языке, вы увидите, что логику приложения довольно трудно понять, что вы не можете просто бегло просмотреть код и понять, что он делает и какие части с чем связаны. И если вы попытаетесь масштабировать его, вы создадите некоторые абстракции для решения этих проблем.

Вы неизбежно столкнетесь с теми же проблемами, которые решают фреймворки (модульность, управление состоянием, жизненный цикл

приложения, привязка данных, манипулирование DOM (Document Object Model), читаемость кода и т. д.), и в конечном итоге вы вам придется создать свой собственный небольшой фреймворк, чтобы решить их. Который при этом будет лишен огромного сообщества профессиональных разработчиков, обеспечивающих поддержку и развитие фреймворка, универсальности применения для различных команд, а также существенного объема функционала. Кроме того, вам придется постепенно решать все те же проблемы, которые уже решены командами разработки существующих известных фреймворков. Итак, теперь, когда мы установили необходимость использования фреймворка при разработке крупного приложения, давайте рассмотрим наиболее часто использующиеся решения.

Внутреннее устройство

Данный фреймворк разработан компанией Facebook в 2013 году, ныне называющейся Meta (признана экстремистской организацией в России), и предназначен для построения пользовательского интерфейса в виде набора небольших компонентов, выстроенных в древовидную структуру [5].

Основные особенности:

- использование декларативного и реактивного подхода к построению пользовательского интерфейса (но отсутствие реактивности на уровне данных);
- модульность и композиционность;
- объединение HTML (HyperText Markup Language) с логикой Представления через использование JSX (JavaScript Extensible Markup Language);
- создание древовидной структуры приложения;
- виртуальный DOM;
- отсутствие ограничений в выборе и интеграции инструментов для решения задач, не покрываемых фреймворком.

Приложение, написанное на React, состоит из компонентов. Компонент – это функция (или класс), которая принимает параметры (props) и возвращает разметку, как правило, в формате JSX, хотя React может использоваться и без JSX [6]. Каждый компонент имеет свой жизненный цикл (см. рис. 1), состоящий из монтирования, обновления, то есть, получения новых данных и отображения нового визуального состояния, зависящего от этих данных, и размонтирования.

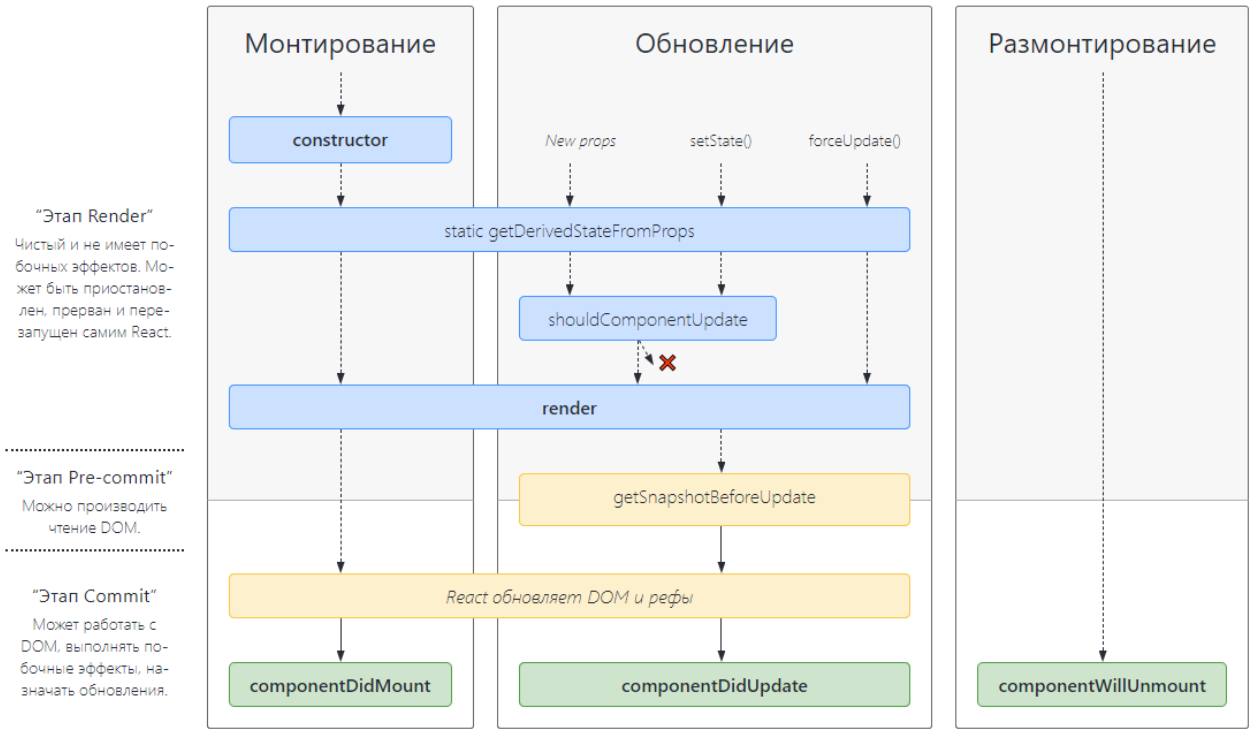

Рис. 1. – Жизненный цикл компонента React

Реактивность компонентов проявляется в том, что разметка меняется тогда, и только тогда, когда изменились входные данные. Это позволяет избежать необходимости ручной синхронизации данных частей приложения, что на практике зачастую оказывается крайне нетривиальной задачей при большом количестве элементов на странице.

Одной из основных особенностей React является использование виртуального DOM – абстракции над DOM, которая позволяет достичь значительно меньшего количества дорогостоящих изменений в DOM без

отсутствия необходимости жертвовать количеством и скоростью обновлений, приводя к ухудшению обратной связи. Виртуальный DOM делает это возможным благодаря проведению манипуляций сначала в JavaScript объектах, после чего разница текущего и предыдущего состояния переносится в DOM внесением в него наиболее оптимальных изменений [7].

Движок согласования, применяемый для манипуляций с виртуальным и реальным DOM, называется React Fiber. Основное правило данного движка: если тип или ключ элемента изменился, нам следует перемонтировать этот компонент и заново создать DOM-узел [8]. Fiber асинхронен, поэтому React может приостанавливать, возобновлять и перезапускать работу по рендерингу компонентов по мере поступления новых обновлений, а также разделять работу на части и расставлять приоритеты задач в зависимости от их важности. Это крайне полезно, потому что в пользовательском интерфейсе нет необходимости немедленно применять каждое обновление, наоборот, это было бы расточительным процессом, приводящим к потере производительности и ухудшению пользовательского опыта. Кроме того, разные типы обновлений в действительности имеют разные приоритеты — обновление анимации должно выполняться быстрее, чем обновление из хранилища данных [9]. Fiber — это повторная реализация стека JS, специализированная для компонентов React. Мы можем воспринимать узел Fiber как фрейм виртуального стека.

React создает дерево Fiber узлов, которые могут изменяться, каждый узел Fiber хранит состояние компонента, параметры (props) и базовый элемент DOM, который он визуализирует. Поскольку Fiber узлы могут изменяться, React не нужно заново создавать каждый узел для обновлений, он может просто клонировать и обновлять их при наличии обновления. В случае Fiber дерева React не выполняет рекурсивный обход, вместо этого он

создает односвязный список и выполняет обход в глубину по принципу «сначала родительский элемент» [10].

Хуки являются одной из самых значимых частей React фреймворка. Они были добавлены в версии 16.8.0 и кардинально изменили возможности функциональных компонентов React. Они, к примеру, позволяют использовать внутреннее состояние компонентов (до введения хуков это было возможно только в классовых компонентах), а также контролируемо использовать сайд-эффекты. Хуки — это по сути JavaScript функции с 2 дополнительными правилами:

- можно вызывать только внутри функциональных компонентов React и других хуков. Нельзя вызывать хуки из обычных JavaScript функций;
- можно вызывать только на верхнем уровне компонента/хука. Запрещается вызывать хуки внутри циклов, условий или вложенных функций.

Преимущества и недостатки

К основным преимуществам фреймворка React следует отнести:

- зрелый по меркам front end сообщества и проверенный временем;
- является наиболее популярным фреймворком для реализации части Представления на клиентской части. Это означает наличие огромного сообщества, которое уже решило все проблемы использования, с которыми команда столкнется во время разработки;
- не ограничивает разработчиков в применяемых инструментах и их интеграции с проектом, использующим React. Существует множество различных инструментов на выбор, если первый выбранный инструмент для решения конкретной задачи не подходит, всегда можно попробовать другой или создать свой. Экосистема React позволяет большую степень гибкости;
- композиционная природа упрощает масштабируемость, поддержку проектов и тестирование;

- декларативный характер ускоряет разработку, уменьшает количество шаблонного кода и вероятность ошибок по сравнению с императивным подходом;

- хуки дают простоту повторного использования кода и удобство разработки, в то же время имея лаконичный API;

- относительно низкий порог вхождения. Можно сказать, что это средний уровень сложности для младших разработчиков, которые не знакомы с концепциями декларативности и функционального программирования, и легкий уровень сложности для опытных разработчиков.

К недостаткам фреймворка React можно отнести следующее:

- не ограничивает разработчиков в применяемых инструментах. Это означает, что выбранные вами инструменты могут устареть, их команды могут внести критические изменения, ломающие ваш проект, и в них могут возникнуть внутренние проблемы, которые вам предстоит устранить. Решение всех этих проблем ложится на вас, как и выбор из множества инструментов, который может приводить к так называемой JavaScript fatigue;

- очень сложно понять глубокие внутренние механизмы React, если вам когда-нибудь понадобится в них погружаться. Например, если вы захотите изменить поведение по умолчанию для конкретного случая или иметь больше свободы во время отладки, чтобы полностью понять, что происходит.

Заключение

При разработке клиентских веб-приложений высокой сложности использование фреймворка необходимо. Выбор фреймворка также крайне важен для создания качественной кодовой базы и ее дальнейшего поддержания. Неправильно выбранный фреймворк может стоит тысяч человеко-часов и впустую затраченных средств. Данная статья рассматривает наиболее популярный фреймворк для создания части Представления клиентских веб-приложений, его внутреннее устройство, сильные и слабые

стороны, а также особенности и ограничения в применении. Из приведенной информации можно сделать вывод о том, что фреймворк React - отличный выбор для разработки большинства клиентских веб-приложений значительной сложности.

Литература (References)

1. Riehle D. Framework Design: A Role Modeling Approach. riehle.org, 2000. URL: riehle.org/computer-science/research/dissertation/diss-a4.pdf.

2. Rutenberg G. Pull vs. Push MVC Architecture. guyrutenberg.com, 208. URL: guyrutenberg.com/2008/04/26/pull-vs-push-mvc-architecture.

3. Dalling T. Model View Controller Explained. tomdalling.com, 2009. URL: tomdalling.com/blog/software-design/model-view-controller-explained.

4. Vanilla JS Vs React JS: What To Choose For Your Development? Tagline Infotech LLP, 2023. URL: taglineinfotech.com/react-js-vs-vanilla-js.

5. Baer E. What React Is and Why It Matters. O'Reilly, 2017. URL: oreilly.com/library/view/what-react-is/9781491996744/ch01.html.

6. Your First Component. React Official Documentation, 2023. URL: react.dev/learn/your-first-component.

7. Minnick C. Beginning reactjs foundations building user interfaces with reactjs: an approachable guide. Hoboken, New Jersey: Wiley, 2022. 512 p.

8. Caputa K. A look inside React Fiber. Makers Den, 2017. URL: makersden.io/blog/look-inside-fiber.

9. Omar A. React Fiber. Scaler Topics, 2023. URL: scaler.com/topics/react/react-fiber.

10. Clark A. React Fiber architecture. GitHub, 2016. URL: github.com/acdlite/react-fiber-architecture.