

Сравнение реализаций реактивного рендеринга клиентских приложений JavaScript

Вихляев Дмитрий Романович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье проводится сравнительный анализ архитектурных паттернов и механизмов реактивности в трёх доминирующих фронтенд решений JavaScript: React, Vue и Angular. Фокус исследования нацелен на этапы жизненного цикла компонентов, определяющие синхронизацию состояния и пользовательского интерфейса. Для визуализации ключевых процессов используются формализованные диаграммы, созданные с помощью языка PlantUML. В работе демонстрируется, как различные реализации реактивности (Virtual DOM, Proxy-based система, Zone.js) формируют уникальные модели жизненных циклов, влияющие на производительность, детерминированность обновлений и сложность разработки.

Ключевые слова: жизненный цикл компонента, реактивность, Virtual DOM, React, Vue, Angular

Comparison of reactive rendering implementations of JavaScript client applications

Vikhlyayev Dmitry Romanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article provides a comparative analysis of architectural patterns and reactivity mechanisms in three dominant JavaScript frontend solutions: React, Vue and Angular. The research focuses on the stages of the component lifecycle that determine the synchronization of state and user interface. Formalized diagrams created using the PlantUML language are used to visualize key processes. The paper demonstrates how different implementations of reactivity (Virtual DOM, Proxy-based system, Zone.js) form unique lifecycle models that affect performance, deterministic updates, and development complexity.

Keywords: component lifecycle, reactivity, Virtual DOM, React, Vue, Angular

1 Введение

1.1 Актуальность

Реактивность – фундаментальный принцип современных фронтенд-фреймворков, обеспечивающий автоматическую синхронизацию состояния

приложения с его представлением. Несмотря на общую цель, реализации этого принципа различаются, что непосредственно отражается на структуре и этапах жизненного цикла компонентов. Данная работа систематизирует эти различия через призму формализованных диаграмм жизненного цикла, что позволяет наглядно сопоставить этапы детекции изменений, рендеринга и побочных эффектов в React, Vue и Angular. Эволюция моделей реактивности (от Virtual DOM к сигналам) демонстрирует стремление к повышению как предсказуемости работы системы, так и её производительности за счёт минимизации избыточных вычислений и обновлений.

1.2 Обзор исследований

В своей статье П.А. Полянская использовала библиотеку React при создании веб-приложения [1]. В.В.Аргасова исследовала web-фреймворк React в разработке информационных систем [2]. Д.А.Мартинovich, С.А.Сатторов описали методы улучшения разработки веб компонентов нативных сайтов с помощью React js [3]. Э.А.Бекирова провела сравнительный анализ производительности и архитектурных подходов в javascript фреймворках: Vue.js и React [4]. Е.В.Ломонов, В.А.Кузьменко исследовали особенности разработки и инструментарий фреймворка React [5]. О.С.Благодельский изучил влияние передовых технологий фронтенд-разработки на улучшение пользовательского интерфейса и повышение уровня взаимодействия с веб-приложениями [6].

1.3 Цель исследования

Цель исследования – привести сравнение ключевых элементов реализации реактивного рендеринга компонентов клиентских приложений JavaScript современных веб-фреймворках.

2 Материалы и методы

В графическом виде представлены процедурные последовательности. Диаграммы построены с помощью инструмента plantuml.

3 Результаты и обсуждения

Одним из ведущих решений использования технологии реактивного рендеринга на стороне клиента является использование библиотеки React.js. В построенных на его основе приложениях, информация передается по направлению от компонента родителя к дочерним компонентам. Когда необходимо зафиксировать изменение состояния, создаются новые, неизменяемые объекты. При вызове методов setState или useState внутренние механизмы библиотеки отмечают компонент как нуждающийся в обновлении. React не отслеживает переменные, а запускает механизм сверки, сравнивая текущую версию виртуального дерева с предыдущим состоянием. В результате сравнения (diffing) определяются изменения, которые необходимо внести в реальную объектную модель документа (DOM) из виртуального, для обновления интерфейса (рис.1).

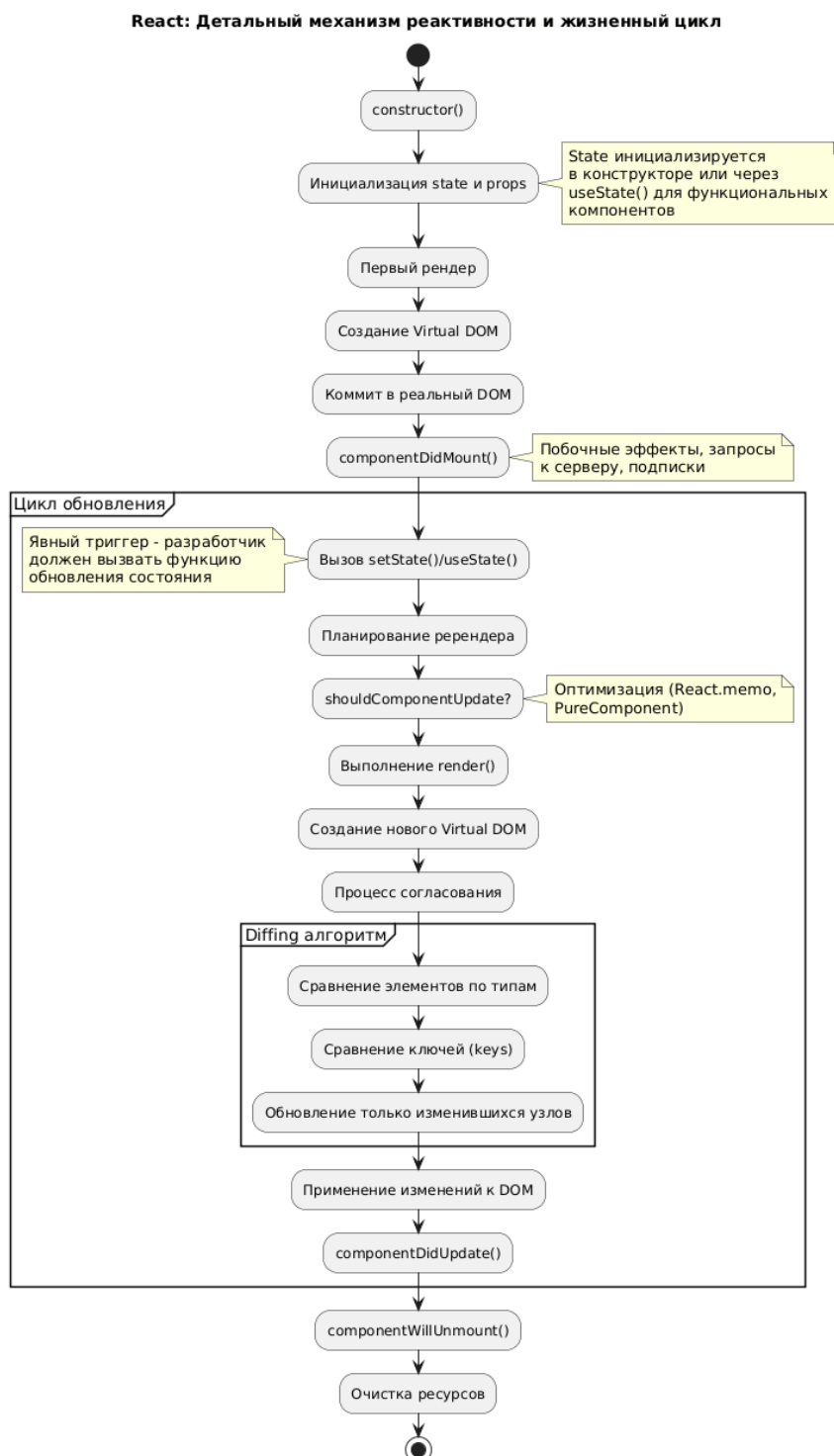


Рис. 1. Жизненный цикл реактивного обновления в React

Жизненный цикл разделён на фазы вычисление разницы и применение изменений (коммита). В состав архитектурного функционала библиотеки React, начиная с версии 16.8, добавляется модель с хуками, основанную на зависимостях (deps), которая декларативно связывает побочные эффекты с данными, но логика изменений остаётся в рамках цикла установки состояния, вычисление разницы и применения изменений.

В отличие от библиотеки React, фреймворк Vue.js реализует метод реактивности с помощью обёртки JavaScript Proxy. Описываемый фреймворк

отслеживает чтение и запись свойств реактивных объектов. При новой перерисовке компонент автоматически собирает зависимости от реактивных свойств. Работа идёт на уровне объекта, а не на уровне отдельных свойств, поэтому глубоко вложенные структуры данных обрабатываются без предварительного обходного обхода и инициализации (рис.2).

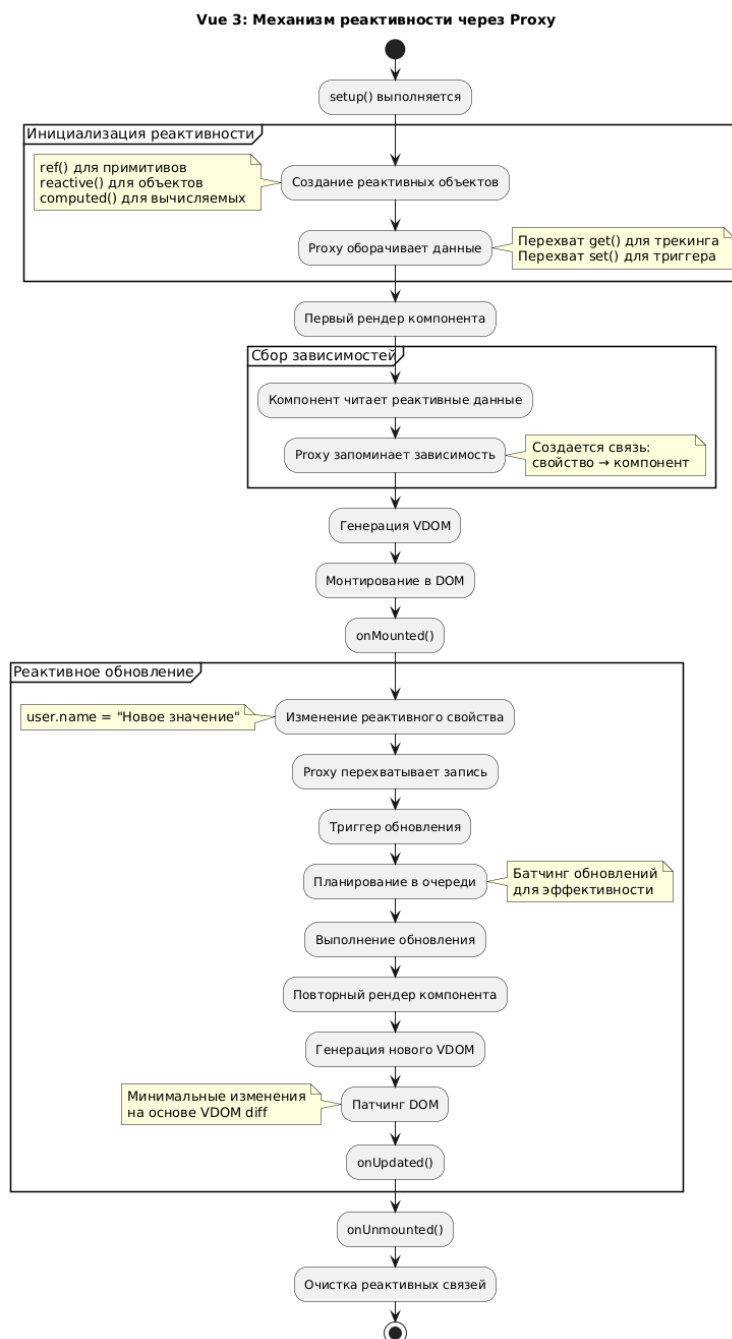


Рис. 2. Жизненный цикл реактивного обновления в Vue.js

В Vue.js обновление не запускается явно, например, через механизм `setState` в React. Перехват операций присваивания происходит с помощью `Proxy`. Благодаря этому обновляются только те компоненты, которые зависят от изменившихся данных. Главная разница заключается в том, что

жизненный цикл компонента становится реактивным на данные, а не на вызов метода обновления по умолчанию.

Angular использует собственную систему детекции изменений (Change Detection), которая в стандартной конфигурации запускается асинхронно в ответ на любые асинхронные события (клики, таймеры, HTTP-запросы). Этот процесс обеспечивается библиотекой Zone.js, которая "монопатрирует" асинхронные API. По умолчанию используется стратегия Default, проверяющая всё дерево компонентов от корня. Разработчик может переключить компонент на стратегию OnPush, которая проверяет компонент только при изменении его Input-свойств или событий (events), либо при явной пометке на проверку (рис.3).

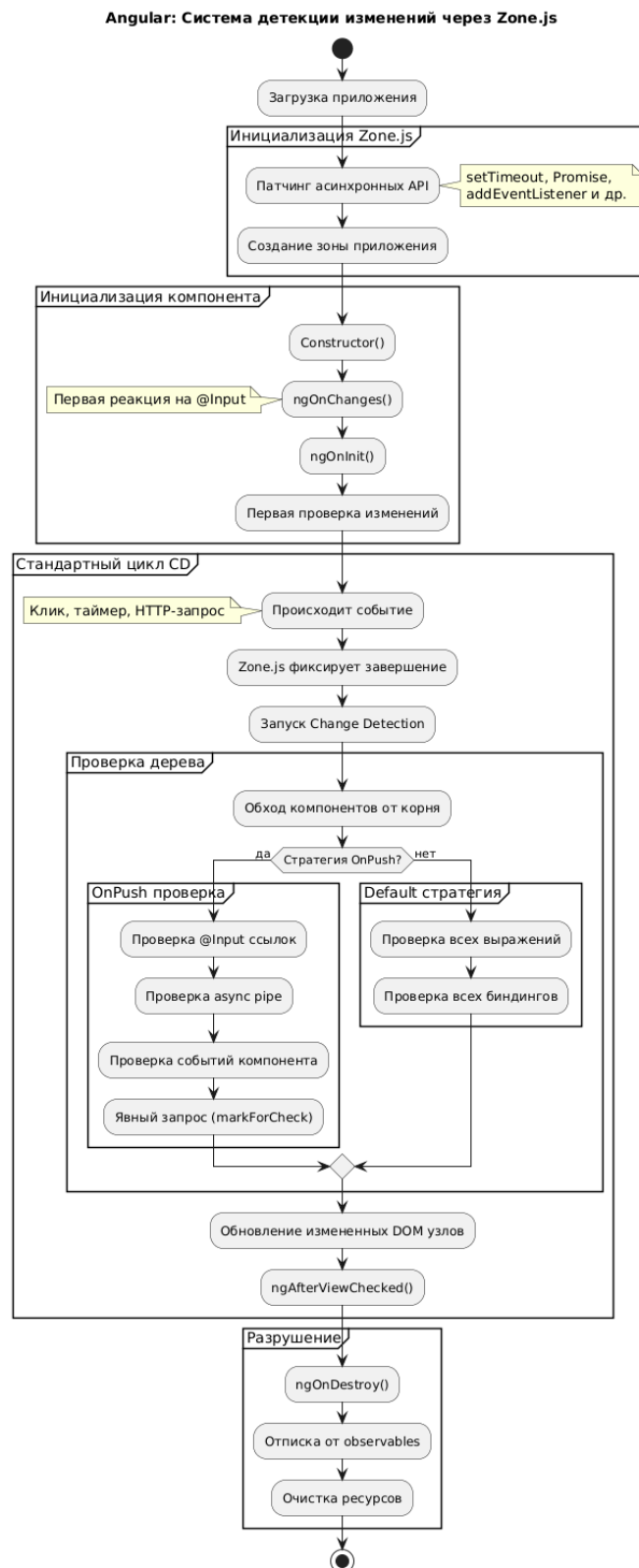


Рис. 3. Жизненный цикл реактивного обновления в Angular

Система обнаружения изменений тесно связана с системой жизненного цикла компонента, а Zone.js регулирует и контролирует её как основная централизованная система. Основная структура построена на реакции компонента на асинхронное событие, а не на изменение конкретной переменной. Стратегия OnPush обеспечила детализацию в применении

обнаружения изменений, поэтому Angular приблизился к модели React или Vue.

Система, использующая «явные обновления» в сочетании с неизменяемой структурой данных, обеспечит предсказуемый способ принятия решений о том, как обновлять состояние, что очень важно для систем со сложной логикой или с частыми изменениями состояния. Система с таким подходом потребует от разработчика постоянного использования методов кэширования для оптимизации производительности, а также ручного предотвращения ненужных «повторных рендеров», что значительно увеличит умственную нагрузку на разработчиков и создаст больше лишнего кода, чем потребовалось бы для реализации решения.

Модель работы, использующая прозрачные автоматические реакции через прокси, обеспечит сбалансированную полезность. Точное отслеживание зависимостей позволяет минимизировать требования к визуальному обновлению пользовательского интерфейса без использования пользовательского ввода, обеспечивая тем самым оптимально высокую производительность с минимальной необходимой настройкой «из коробки» и быстрый темп разработки. Основным недостатком этого подхода заключается в меньшей прозрачности процесса и может повлечь за собой дополнительные накладные расходы при построении больших структур данных во время инициализации.

4 Вывод

Реактивный подход определяется мышлением разработчика и тем, как он хочет оптимизировать код. В решениях на основе React существует несколько способов проектирования исходного кода, но оптимизацию придётся выполнять вручную с помощью кэширования. Аналогичное решение (React), используемое в Vue.js для создания систем, которые автоматически оптимизируются, является сбалансированным и не доставляет хлопот. На платформе Angular можно создать цикл событий, который сложно оптимизировать, но при этом есть доступ к управлению циклом событий и обширным инструментам оптимизации. Понимая жизненный цикл инструмента, можно определить, какой инструмент лучше всего подходит для вашей задачи.

Библиографический список

1. Полянская П.А. Использование библиотеки React при создании веб-приложений // Студенческий. 2024. № 1-2 (255). С. 60-61.
2. Аргасова В.В. Применение web-фреймворка React при разработке информационных систем // Аллея науки. 2018. Т. 2. № 7 (23). С. 526-528.
3. Мартинович Д.А., Саторов С.А. Методы улучшения разработки веб-компонентов нативных сайтов с помощью React js // Славянский форум. 2024. № 2 (44). С. 338-344.
4. Бекирова Э.А. Сравнительный анализ производительности и

- архитектурных подходов в javascript фреймворках: Vue.js и React // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2024. № 3 (45). С. 18-23.
5. Ломонов Е.В., Кузьменко В.А. Особенности разработки и инструментарий фреймворка React // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2023. № 1 (39). С. 26-30.
 6. Благодельский О.С. Влияние передовых технологий фронтенд-разработки на улучшение пользовательского интерфейса и повышение уровня взаимодействия с веб-приложениями // Актуальные исследования. 2024. № 20-1 (202). С. 73-79.
 7. Чепурной М.П., Дяченко А.С. Сравнительный анализ фреймворков для фронтенд-разработки // Международный студенческий научный вестник. 2025. № 1. С. 12.
 8. Шадров И.А., Гаев Л.В. React в современной веб-разработке: ключевые задачи и решения // Инновационная наука. 2025. № 5-2. С. 88-89.