Программные системы и вычислительные методы
Правильная ссылка на статью:
Бондаренко О.С. Анализ методов обновления DOM в современных веб-фреймворках: Virtual DOM и Incremental DOM // Программные системы и вычислительные методы. 2025. № 2. DOI: 10.7256/2454-0714.2025.2.74172 EDN: JCILDR URL: https://nbpublish.com/library_read_article.php?id=74172

Анализ методов обновления DOM в современных веб-фреймворках: Virtual DOM и Incremental DOM

Бондаренко Олеся Сергеевна

ORCID: 0009-0003-5477-6947

магистр; Факультет программной инженерии и компьютерной техники; федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО».

195299, Россия, г. Санкт-Петербург, Калининский р-н, пр-кт Просвещения, д. 99

rancerenly@gmail.com

Статья из рубрики "Операционные системы "

DOI:

10.7256/2454-0714.2025.2.74172

EDN:

JCILDR

Дата направления статьи в редакцию:

20-04-2025

Дата публикации:

06-05-2025

Аннотация: Статья представляет собой анализ современных методов обновления структуры Document Object Model (DOM) в популярных клиентских веб-фреймворках, таких как Angular, React и Vue. Основное внимание уделяется сравнению концепций Virtual DOM и Incremental DOM, которые лежат в основе архитектурных решений соответствующих фреймворков. Virtual DOM, применяемый в React и Vue, оперирует виртуальным деревом, сравнивает его версии с целью выявления различий и минимизации изменений в реальном DOM. Такой подход обеспечивает относительную простоту реализации реактивного интерфейса, однако сопровождается дополнительными затратами на вычисления и использование ресурсов. В отличие от него, Angular использует Incremental DOM, при котором отсутствует создание промежуточных структур:

изменения применяются напрямую через механизм Change Detection. Этот подход позволяет добиваться высокой производительности за счёт точечных обновлений DOM-элементов без необходимости в виртуальном представлении. В исследовании применяется сравнительный анализ архитектурных подходов к обновлению DOM, основанный на изучении официальной документации, практических экспериментов с кодом и визуализации процессов рендеринга в Angular и React. Методология включает теоретическое обоснование, пошаговый разбор механизмов обновлений и оценку их влияния на производительность. Научная новизна статьи заключается в систематическом сопоставлении архитектурных подходов к обновлению DOM в ведущих фреймворках, с акцентом на внедрение сигнальной модели в Angular версии 17+. Подробно проанализировано влияние использования сигналов на отказ от библиотеки Zone.js и формирование более предсказуемой, детерминированной модели рендеринга, а также возможности управления производительностью на более низком уровне. Статья содержит не только теоретическое описание, но и практические примеры, раскрывающие поведение обновлений в реальных сценариях. Также рассматриваются нюансы шаблонной компиляции, работы функций effect() и computed(). Проведённое сравнение Virtual DOM и Incremental DOM позволяет выявить ключевые различия, оценить применимость подходов в зависимости от задач и уровня сложности проекта, а также предложить направления оптимизации фронтенд-архитектур.

Ключевые слова:

DOM, Virtual DOM, Incremental DOM, Angular, React, Vue, сигналы, рендеринг, диффинг, компиляция шаблонов

Введение

В современном веб-разработке критически важно обеспечение высокой производительности и отзывчивости интерфейса. Усиливающаяся динамичность данных, масштабируемость приложений и сложность управления состоянием требуют от инструментов разработки инновационных решений для работы с DOM – основной структурой любой веб-страницы. Фреймворк Angular отличается уникальным подходом: вместо использования Virtual DOM, как в React и Vue, он применяет механизм обнаружения изменений, который в последних версиях (v17+) дополняется поддержкой сигналов (signals). Именно эти инновационные решения позволяют создавать быстро работающие и предсказуемые интерфейсы.

Актуальность исследования обосновывается развитием фронтенд-разработки. Постоянное развитие технологий требует обновления подходов к рендерингу интерфейсов. Использование сигналов и инкрементального обновления DOM является важным этапом на пути повышения интерактивности и масштабируемости веб-приложений.

Целью данной статьи является рассмотреть методы взаимодействия с DOM в Angular, выделить отличия от Virtual DOM-решений (как в React и Vue), описать основные процессы рендеринга и выполнения инструкций обновления, а также показать преимущества реактивного программирования в Angular.

Механизм обнаружения изменений

Одной из ключевых особенностей Angular является использование собственного

механизма обнаружения изменений (Change Detection), который обеспечивает эффективную синхронизацию состояния компонентов с отображаемым в браузере DOM. В отличие от таких фреймворков, как React или Vue, которые опираются на концепцию Virtual DOM и используют алгоритмы диффинга, Angular применяет иной подход: он отслеживает изменения в данных и напрямую обновляет только те части DOM, которые действительно нуждаются в этом. Такой способ минимизирует избыточные действия и позволяет добиться высокой производительности, особенно в сложных и масштабируемых приложениях.[6]

Механизм Change Detection работает по следующему принципу. Каждый компонент в Angular связан с моделью данных и шаблоном (template), отображающим эту модель. Когда происходит изменение данных — будь то изменение переменной, результат выполнения асинхронной операции или взаимодействие с пользователем — Angular инициирует проверку, чтобы определить, какие именно данные были изменены, и какие части DOM следует обновить. Это и есть процесс, называемый «dirty checking» — проверка «загрязнённости» или устаревания данных в компоненте.[7]

Во время этого процесса Angular последовательно проходит по дереву компонентов, начиная с корневого, и проверяет каждое свойство, связанное с шаблоном. Если фреймворк обнаруживает расхождение между текущим значением свойства и его предыдущим значением, он запускает процедуру обновления соответствующего элемента в DOM. Такой подход гарантирует актуальность отображаемой информации при любых изменениях, происходящих в приложении.

Особенности dirty checking

Важно отметить, что в Angular процесс dirty checking не полагается на прямое сравнение старого и нового дерева, как это происходит в Virtual DOM. Вместо этого Angular опирается на заранее известные связи между данными и шаблоном, определённые во время компиляции компонента. Благодаря этому фреймворк может обходить только те участки дерева, где потенциально могли произойти изменения, тем самым экономя ресурсы.

Поскольку Angular отслеживает изменения в каждом компоненте и их зависимостях, он способен эффективно обновлять интерфейс даже в очень больших приложениях. Однако важно правильно организовать логику компонентов, чтобы избежать избыточных вызовов Change Detection и непреднамеренных рендеров. Например, если компонент использует тяжёлые вычисления в шаблоне, они могут выполняться повторно при каждом проходе Change Detection, что негативно скажется на производительности.

Angular традиционно использует библиотеку Zone.js для автоматического триггера процесса Change Detection при возникновении асинхронных событий — таких как таймеры, HTTP-запросы или события DOM. Эта библиотека патчит стандартные API браузера (например, setTimeout, Promise) таким образом, что Angular узнаёт, когда завершилась асинхронная операция, и автоматически инициирует проверку на изменения.[10]

Такой подход удобен, поскольку позволяет разработчику не думать о вручную вызовах обновления интерфейса. Однако использование Zone.js может приводить к избыточному количеству запусков Change Detection, особенно в приложениях с интенсивным взаимодействием с пользователем или большим количеством асинхронных событий.

Для повышения эффективности Angular предоставляет механизм стратегий обнаружения

изменений. По умолчанию Angular использует стратегию Default, при которой проверяется весь компонент при любом изменении. Однако, при указании стратегии ChangeDetectionStrategy.OnPush, Angular будет обновлять компонент только в случае, если изменился один из его входных параметров (@Input) или произошло событие внутри компонента.

Кроме того, начиная с версии 17, Angular предлагает использовать сигналы (signals) как новый реактивный механизм работы с состоянием. Сигналы позволяют явно описывать зависимости данных, а также автоматически инициировать обновление только в тех местах, где действительно произошли изменения. Это делает процесс Change Detection более предсказуемым и гранулярным. [7] Вместе с функцией effect() сигналы позволяют создавать логически связанные реакции на изменение состояния, не затрагивая лишние компоненты или шаблоны.

С выходом версии 17 был введён принцип работы с сигналами. Сигналы – это реактивные переменные, обновление которых происходит в автоматически отслеживаемом контексте. Это позволяет избавиться от громоздких подписок и ручного управления состоянием. Основные преимущества сигналов таковы:

Автоматическое отслеживание изменений. Функция effect() регистрирует блоки кода, реагирующие на изменения сигнала, что упрощает реализацию логики обновления.

Простота и предсказуемость. Применяя сигналы, разработчик получает компактный и легко понимаемый код, где все изменения в данных немедленно отражаются в DOM.

Оптимизация производительности. Благодаря точечному обновлению DOM обновляются только те элементы, данные которых изменились, что минимизирует избыточные операции и ускоряет рендеринг.

Рассмотрим базовый подход применения сигналов в Angular на рисунке 1:

Рисунок 1 — Использование Input-сигнала в Angular

В этом примере реализован компонент ExampleInputComponent, который ожидает входной параметр exampleInput типа string, в шаблоне выводится текст «Введенный текст: %переданный текст в компонент%». Родительский компонент PlaygroundComponent импортирует дочерний и определяет его в своем шаблоне, передавая туда текст «Привет, мир!».

Результат выполнения представлен на рисунке 2.

Рисунок 2 — Демонстрация работы сигналов

Реактивное программирование: сигналы, computed и effect

С появлением сигналов разработчики получают инструмент для создания реактивных переменных, значения которых можно легко отслеживать и изменять. Сигналы значительно упрощают управление состоянием: при изменении данных автоматически запускается обновление зависимых частей интерфейса. Это повышает предсказуемость кода и сокращает вероятность возникновения ошибок.

Пример использования сигналов приведён выше в компоненте ExampleInputComponent. Такой подход устраняет необходимость управления подписками вручную и позволяет Angular самостоятельно контролировать обновления.

Вычисляемые значения с computed()

Помимо сигналов Angular предлагает функцию computed(), которая позволяет создавать вычисляемые значения на основе нескольких сигналов. Это удобно для агрегации или преобразования данных без дублирования логики. Рассмотрим пример:

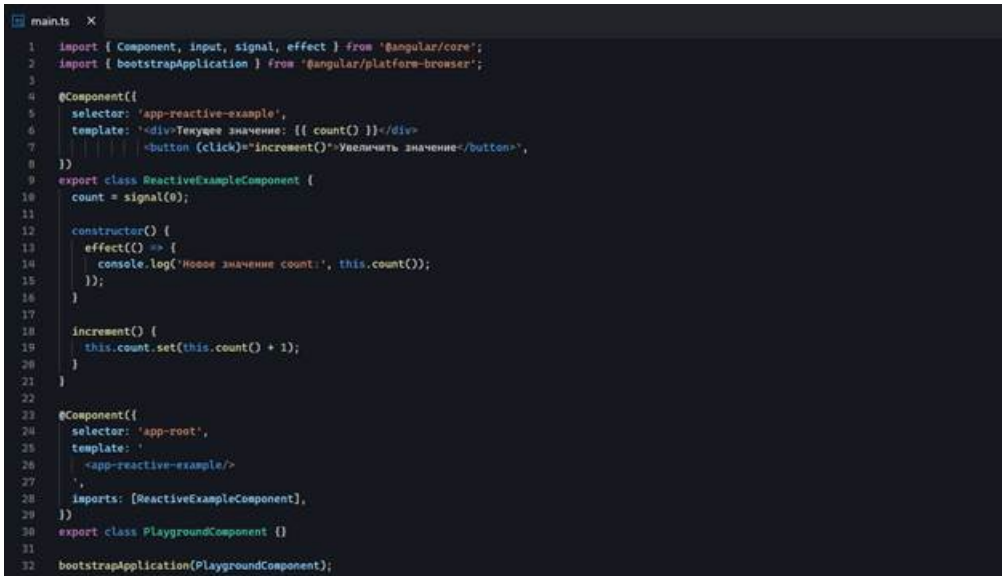

Рисунок 3 — Использование сигнала в effect

В данном примере функция effect() автоматически реагирует на изменения сигнала count и выводит новое значение в консоль. Такой подход устраняет необходимость явно подписываться на изменения и позволяет фреймворку самостоятельно управлять обновлением значений.

При клике видим изменения значения.

Рисунок 4 — Демонстрация работы на сайте

Использование computed для производных значений

Помимо сигналов Angular предлагает использовать функцию computed(), позволяющую создавать вычисляемые значения на основе других сигналов. Данный метод позволяет агрегировать или преобразовывать данные, минимизируя дублирование логики и повышая читаемость кода. Пример использования computed():

Рисунок 5 — Функция computed для выполнения манипуляций со значением

В этом примере любое изменение в сигналах firstName или lastName автоматически приведет к пересчету fullName, а следовательно, и к обновлению представления.

Сравнениеподходов: Virtual DOM vs Incremental DOM

Virtual DOM — это концепция, применяемая в React и Vue, при которой создается виртуальное представление текущего состояния реального DOM. При изменении данных формируется новая версия виртуального дерева, после чего выполняется сравнительный (диффинг) анализ с предыдущей версией, что позволяет выявить минимальный набор изменений.[1] Затем, только обнаруженные отличия применяются к реальному DOM. Такой подход обеспечивает оптимизацию обновлений, поскольку изменяются лишь необходимые части интерфейса, а также абстрагирует сложность работы с DOM, избавляя разработчика от необходимости вручную управлять изменениями.[4][5] Однако постоянное создание и сравнение виртуальных деревьев требует дополнительных вычислительных ресурсов, что может стать проблемой при работе с большими и сложными интерфейсами.

В отличие от этого, Angular использует принцип инкрементального обновления DOM. Здесь нет необходимости создавать виртуальную копию: фреймворк напрямую анализирует, какие свойства изменились, и обновляет только соответствующие элементы. Главное преимущество данного подхода заключается в прямом обновлении элементов, так как Angular, анализируя структуру приложения, обновляет только изменившиеся участки DOM.[2]При этом происходит компиляция шаблонов, когда на этапе сборки исходные шаблоны преобразуются в высокопроизводительный код с набором готовых инструкций для обновления DOM, а использование сигналов позволяет

точно определить зависимости, обновляя только те узлы, которые действительно изменились, что значительно снижает количество лишних проверок.[6][3]

Процесс рендеринга в Angular включает несколько этапов. Сначала происходит компиляция шаблонов, когда исходный код преобразуется в набор инструкций для обновления DOM. Затем Angular выполняет проверку изменений посредством механизма Change Detection, что позволяет определить, какие части интерфейса нуждаются в обновлении. После этого применяется инкрементальное обновление: обновляются лишь измененные узлы, что способствует экономии ресурсов. Наконец, сгенерированные инструкции обновляют атрибуты, стили, классы и события, гарантируя корректное и эффективное обновление интерфейса.[8]

Таким образом, использование Virtual DOM позволяет оптимизировать обновления за счет изменения только необходимых частей, однако требует дополнительных вычислительных затрат на создание и сравнение виртуальных деревьев. Angular же достигает высокой эффективности путем непосредственного анализа изменений и обновления DOM «на лету», что исключает необходимость в создании виртуальной копии и существенно снижает нагрузку на систему.

Таблица 1 — Сравнение Incremental DOM и Virtual DOM

Критерий	**Инкрементальное обновление**	**Виртуальный DOM**
Принцип работы	Прямое обновление измененных узлов через механизм Change Detection	Создание виртуального дереве, диффинг и патчинг DOM
Шаблоны	Генерируются инструкции	Шаблоны обновляются через сравнение Virtual DOM
Изменения	Обновляются только изменившиеся части (*Зависит от использования стратегии)[7][8]	Обновляются только изменившиеся элементы, затраты на создание копий и сравнение их с реальным DOM
Привязка данных	Двунаправленное связывание	Однонаправленное связывание

Будущие направления развития Angular

В будущих версиях Angular планируется дальнейшее использование сигналов для управления изменениями, что позволит отказаться от Zone.js – библиотеки, автоматизирующей обнаружение изменений, но создающей накладные расходы. Переход к полному ручному управлению рендерингом с применением сигналов может сделать приложения ещё более оптимизированными и предсказуемыми.

Реактивное программирование становится основой современной разработки. В будущем Angular вероятно расширит функционал для работы с потоками данных, улучшит интеграцию с асинхронными процессами и упростит создание вычисляемых значений. Это позволит разработчикам писать более декларативный и масштабируемый код.

Заключение

Angular предлагает уникальный подход к работе с DOM, основанный на механизме

обнаружения изменений и использовании сигналов. Вместо создания виртуального представления, как в React или Vue, Angular напрямую обновляет только изменившиеся участки интерфейса. Такой инкрементальный подход, в сочетании с компиляцией шаблонов и заранее сгенерированными инструкциями, обеспечивает высокую производительность, минимальные затраты на обновление и прозрачное управление состоянием приложения.

Применение реактивных сигналов, функций effect() и computed(), а также использование сервисов вроде Renderer2 позволяют создавать безопасные, масштабируемые приложения, отвечающие требованиям современного веб-разработчика.

Таким образом, современный подход Angular к обновлению DOM не только способствует повышению производительности и удобству отладки, но и открывает новые возможности для оптимизации работы сложных и динамичных веб-приложений. Продолжая эволюцию в направлении полной интеграции реактивного программирования и отказа от традиционных методов, Angular остаётся мощным инструментом в арсенале разработчиков, позволяющим создавать быстрые и отзывчивые решения для удовлетворения растущих потребностей рынка.

Библиография

1. Бетеев К.Ю., Муратова Г.В. Концепция virtual dom в библиотеке react.js // Инженерный вестник Дона. - 2022. - № 3. - С. 170-180. EDN: LHOOOS.
2. Incremental DOM [Электронный ресурс]. URL: https://github.com/google/incremental-dom (дата обращения: 16.04.2025).
3. Introducing Incremental DOM [Электронный ресурс]. URL: https://medium.com/google-developers/introducing-incremental-dom-e98f79ce2c5f (дата обращения: 12.04.2025).
4. Understanding Angular Ivy: Incremental DOM and Virtual DOM [Электронный ресурс]. URL: https://blog.nrwl.io/understanding-angular-ivy-incremental-dom-and-virtual-dom-243be844bf36 (дата обращения: 16.04.2025).
5. Разбираемся в Angular Ivy: Incremental DOM и Virtual DOM [Электронный ресурс]. URL: https://habr.com/ru/articles/448048/ (дата обращения: 03.03.2025).
6. Virtual DOM vs Incremental DOM in Angular [Электронный ресурс]. URL: https://www.angularminds.com/blog/virtual-dom-vs-incremental-dom-in-angular (дата обращения: 17.02.2025).
7. Angular Документация [Электронный ресурс]. URL: https://angular.dev/api/core/ChangeDetectionStrategy (дата обращения: 16.04.2025).
8. Довженко М.И., Готская И.Б. Анализ способов реализации алгоритма отслеживания изменений в одностраничных веб-приложениях // Альманах научных работ молодых ученых XLVII научной и учебно-методической конференции Университета ИТМО. Том 7. - 2018. - С. 123-126. EDN: YXNFSH.
9. Исходный код примеров [Электронный ресурс]. URL: https://stackblitz.com/edit/6meb5pyu?file=src%2Fmain.ts (дата обращения: 16.04.2025).
10. Zone.js [Электронный ресурс]. URL: https://github.com/angular/angular/tree/main/packages/zone.js (дата обращения: 16.04.2025).

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Статья посвящена анализу методов обновления DOM в современных веб-фреймворках, с акцентом на сравнение подходов Virtual DOM (используемого в React и Vue) и Incremental DOM (реализованного в Angular). Особое внимание уделено механизму обнаружения изменений (Change Detection) в Angular, включая нововведения, такие как сигналы (signals), и их влияние на производительность и управление состоянием приложений.

Автор применяет сравнительный анализ, детально рассматривая принципы работы Virtual DOM и Incremental DOM, их преимущества и недостатки. В статье приведены конкретные примеры использования сигналов, функций `effect()` и `computed()` в Angular, что демонстрирует практическую значимость исследования. Методология включает анализ документации, исходного кода и научных публикаций, что обеспечивает достоверность выводов.

Тема статьи крайне актуальна в контексте современной веб-разработки, где производительность и отзывчивость интерфейсов являются критически важными. Автор обоснованно подчеркивает необходимость оптимизации работы с DOM, особенно в условиях роста сложности и масштабируемости веб-приложений. Упоминание новых возможностей Angular (например, сигналов в версии 17+) добавляет работе практическую ценность для разработчиков.

Научная новизна исследования заключается в детальном анализе инкрементального подхода Angular к обновлению DOM, включая последние изменения, такие как внедрение сигналов. Автор не только сравнивает два основных метода (Virtual DOM и Incremental DOM), но и предлагает взгляд на будущее развитие Angular, например, отказ от Zone.js в пользу ручного управления рендерингом. Это делает статью полезной как для теоретиков, так и для практиков.

Статья написана четким и логичным языком, с соблюдением академических норм. Структура работы хорошо продумана: от введения и постановки проблемы до детального анализа и выводов. Использование таблиц (например, сравнение Incremental DOM и Virtual DOM) и рисунков (примеры кода) enhances наглядность материала. Библиография включает актуальные источники, что подкрепляет аргументацию автора.

Автор приходит к обоснованному выводу, что инкрементальный подход Angular, особенно с использованием сигналов, обеспечивает высокую производительность и предсказуемость обновлений DOM. Подчеркивается, что такой метод минимизирует избыточные операции и упрощает управление состоянием приложения. Статья также указывает на перспективы дальнейшего развития Angular, что делает её ценной для долгосрочного планирования проектов.

Материал будет интересен широкому кругу читателей: от студентов и исследователей в области веб-технологий до практикующих разработчиков, которые стремятся оптимизировать свои приложения. Статья сочетает теоретическую глубину с практическими примерами, что делает её полезной для применения в реальных проектах.

Статья «Анализ методов обновления DOM в современных веб-фреймворках: Virtual DOM и Incremental DOM» представляет собой качественное исследование, соответствующее высоким академическим стандартам. Работа обладает научной новизной, актуальностью и практической значимостью. Рекомендую статью к публикации.