Введение в React

Кирилл Талецкий

TeachMeSkills
18 сентября 2023

Анализируем наше JS приложение

Анализ нашего подхода к созданию приложения

- Перерисовка innerHTML целиком требует от браузера:
  - Распарсить строку в валидный HTML
  - Вычислить Render Tree
  - Вычислить расположение элементов и их размер (Layout)
  - Раскрасить слои (Paint)
  - Совместить слои друг с другом (Layer composition)

- Всё это нужно сделать за примерно 16 мс (если мы хотим плавные 60 кадров в секунду)
- Если на странице много интерактивных элементов, которые часто меняются, то если работать через переприсваивание innerHTML просто не вывезет по производительности

- **Вывод: нужно обновлять элементы адресно, меняя только те из них, что действительно подверглись изменению**

Решение “в лоб”

- Нам нужно обновлять элементы адресно, меняя только те из них, что действительно подверглись изменению

- Как это сделать:
  - Берём старый innerHTML из body
  - Сравниваем с новым, который мы только что сгенерили
  - Находим разницу, например:
    - Изменилось содержание элемента h1
    - Изменилось значение `class` у элемента div в заголовку
  - Находим эти элементы в DOM
  - Применяем к ним эти изменения

Проблема 1 - большие строки

- JS очень плох в работе со строками:
  - Он довольно медленно их сравнивает
  - Он не умеет быстро менять их содержимое

- Берём старый innerHTML из body
- Сравниваем с новым, который мы только что сгенерили
- Находим разницу, например:
  - Изменилось содержание элемента h1
  - Изменилось значение `class` у элемента div в заголовку
- Находим эти элементы в DOM
- Применяем к ним эти изменения

- Сравнение двух больших HTML документов - так себе идея, особенно когда это часто повторяющаяся операция

Проблема 2 - DOM

- Сравнение HTML строк — не вариант
- Тогда давайте сравнивать два экземпляра DOM!

- Берём старый innerHTML из body
- Сравниваем с новым, который мы только что сгенерили
- Находим разницу, например:
  - Изменилось содержание элемента h1
  - Изменилось значение `class` у элемента div в заголовку
- Находим эти элементы в DOM
- Применяем к ним эти изменения

- Проблема в том, что DOM никогда не был предназначен для сложных операций по работе с UI
- В нём нет API для быстрого сравнения двух DOM деревьев и вывода результатов

Решение

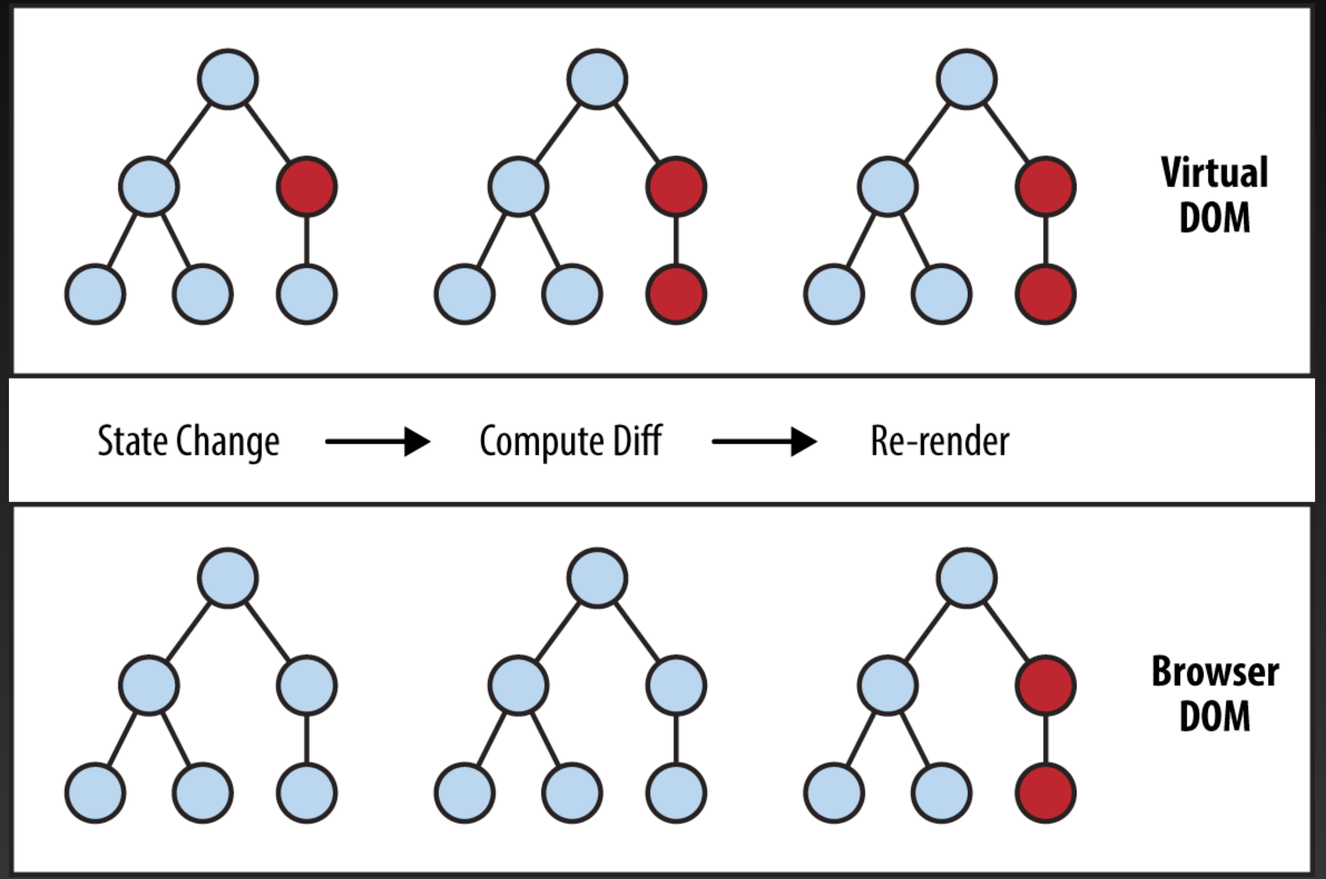
- DOM плохо подходит для хранения текущего состояния интерфейса
- В строку всё записывать тоже не вариант

- **Сделаем свой, параллельный DOM!**
  - С блэкджеком и оптимизациями

- Создаём объект-копию DOM
- Он будет единым источником правды для всего UI
  - Если мы хотим поменять UI, то меняем поля в объекте
  - Если поля в объекте поменялись, то мы обновляем реальный DOM

Virtual DOM

Virtual DOM


Virtual DOM
State Change
Compute Diff
Re-render
Browser DOM

React

История React

- JavaScript библиотека для создания пользовательских интерфейсов

- Создана в 2011 году
- В 2013 году был опубликован исходный код*
- В 2017 году было произведено большое обновление — переход на функциональный подход

Зачем нужен React

- Значительно увеличивает скорость работы JS приложений:
  - React оптимизирует работу с DOM, позволяя делать минимальное количество тяжелых операций перетасовки

- Под капотом используется как раз Virtual DOM

Ещё преимущества

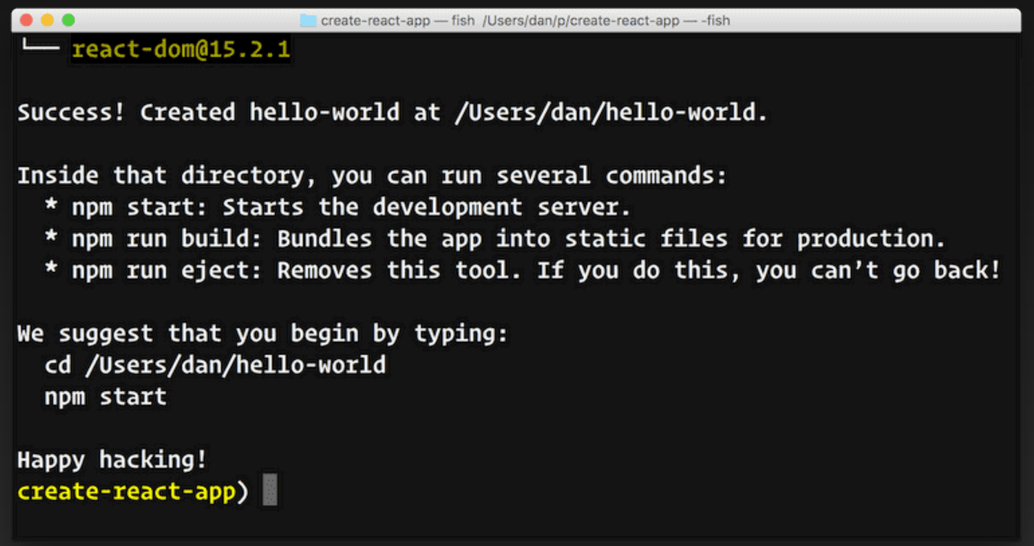
- Предоставляет возможность компонентной разработки
  - Декларативный синтаксис
  - Вы можете инкапсулировать данные, методы и вёрстку в одной компоненте
  - Вы можете создавать переиспользуемые части вёрстки / логики

- **Всё это позволяет создавать и поддерживать очень большие фронтенд-проекты**

Практическая работа с React

Create React App

- Пакет-утилита, которая сделает всю первоначальную настройку Webpack и прочих помощников за вас



Компоненты

- Компоненты делятся на два вида:
  - Встроенные — самые базовые, предоставляются реактом, соответствуют DOM элементам
  - Кастомные — созданные вами или другими разработчиками

JSX

- Специальный синтаксис для создания компонент React
- Создан по образу и подобию HTML с небольшими отличиями
  - JSX может содержать JS выражения — это полезно для условного рендера и рендера списков
  - JSX атрибуты пишутся в camelCase

Кастомные компоненты

- Кастомные компоненты в React — это обычные JS функции
- Они должны возвращать другие компоненты
- Эти функции запускаются реактом на каждое изменение состояния компонент
- Возвращаемое значение используется реактом для отрисовки UI

Стилизация

- Выделяются следующие подходы
  - Инлайн стили через атрибут style
    - Подход CSS in JS
  - Добавление классов через атрибут className

Перенос приложения