Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 13

Virtual DOM. React Fiber React Reconciliation

План занятия

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

1. Стратегии рендеринга
2. Virtual DOM
3. React Fiber
4. React Reconciliation

Стратегии рендеринга

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Варианты стратегий

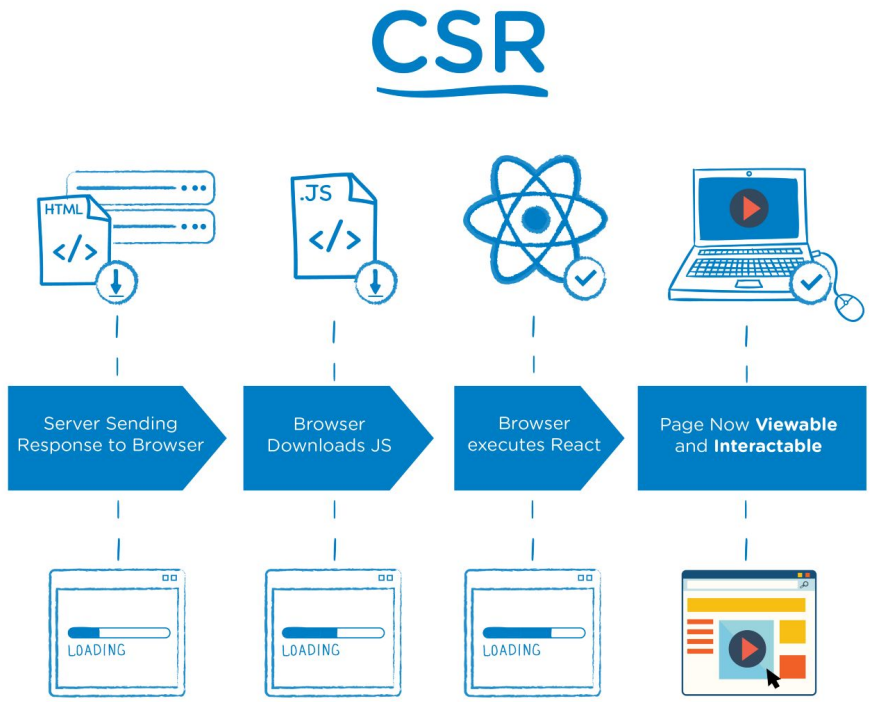
- CSR – Client Side Rendering
- SSR – Server Side Rendering
- RSC – React Server Components
- SSG – Static Site Generation
- ISR – Incremental Static Regeneration

Стратегии рендеринга

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

CSR – Client Side Rendering

- Обычный способ для SPA – приложений
- Полностью весь рендеринг выполняется на клиента (в браузере)
- Для получения данных инициируется запрос с клиента на получения динамических данных
- При первой загрузке страницы сервер предоставляет node ( ) или несколько таких node
- Поисковые пауки от Yandex и Google и других поисковиках не могут индексировать информацию с таких сайтов, или если могу, то информация не корректно индексируется

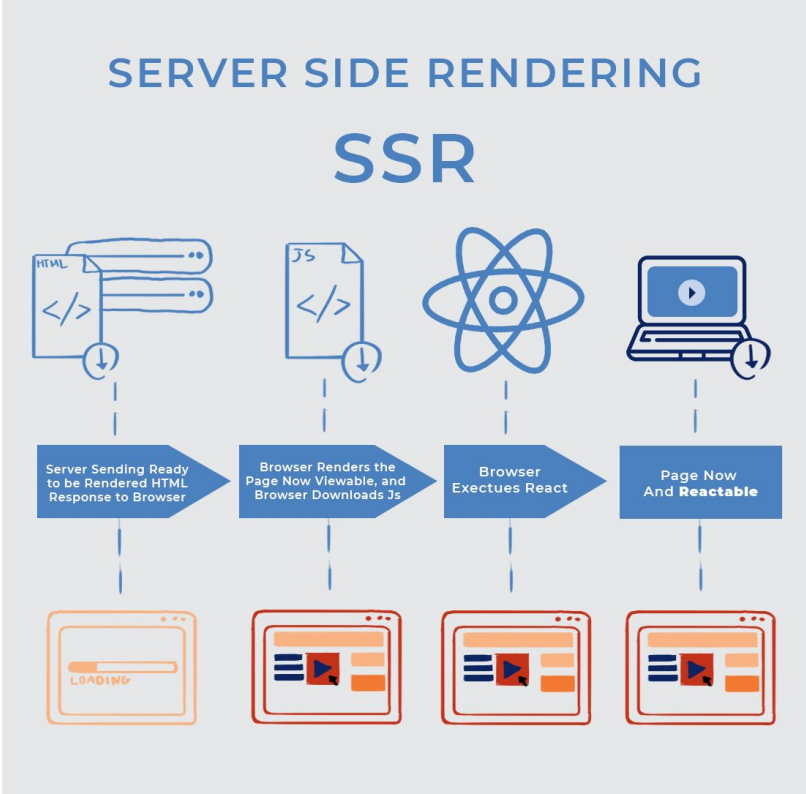


Стратегии рендеринга

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

SSR – Server Side Rendering

- При загрузке страницы пользователь получает готовую, уже срендереную HTML страницу сервером
- Клиент со страницей получает JS файлы, которые встраиваются в поставленный HTML. Этот процесс называется Hydration
- Далее при загрузке остальных страниц во время переходов между ними работает CSR
- Т.к. HTML-страница отдается полностью происходит индексация поисковыми системами

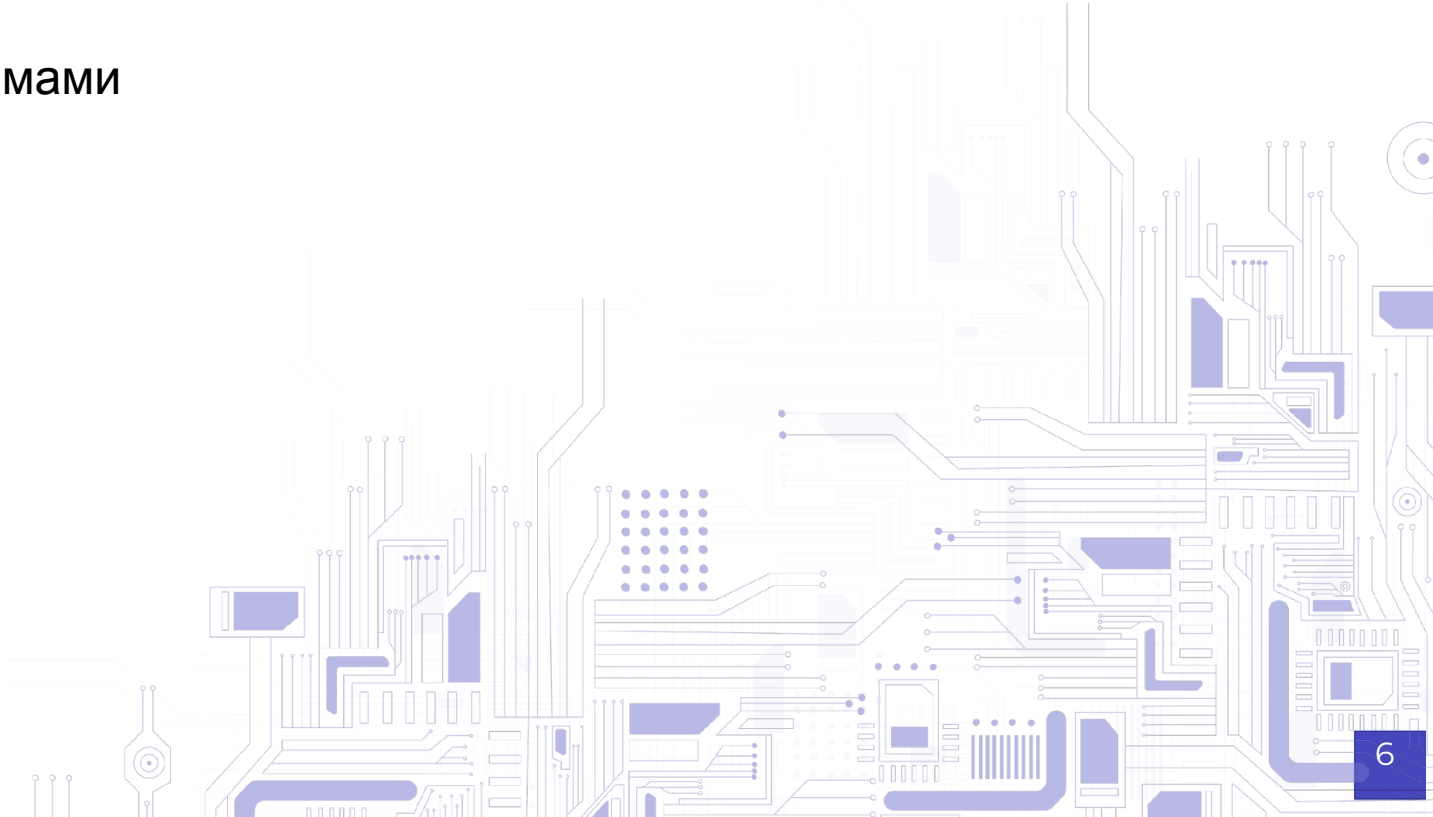


Стратегии рендеринга

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

RSC – React Server Components

- Способ выполнять рендер содержимого на сервере и стримить его на клиент без дополнительной гидрации
- Стримить HTML-статику при любом запросе, а не только при первом
- Отлично индексируется поисковыми системами



Стратегии рендеринга

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

SSG – Static Site Generation

- Генерация HTML-страниц выполняется на сервере
- Рендеринг страниц происходит во время сборки (build) приложения, а не во время выполнения (run time)
- Страницы статические не требуют гидрации

Стратегии рендеринга

ISR – Incremental Static Regeneration

- Позволяет перерендеривать статические страницы
- Является комбинацией подходов SSG и SSR / RSC
- Обновление страниц происходит по таймеру или по событию (click, save …)

DOM

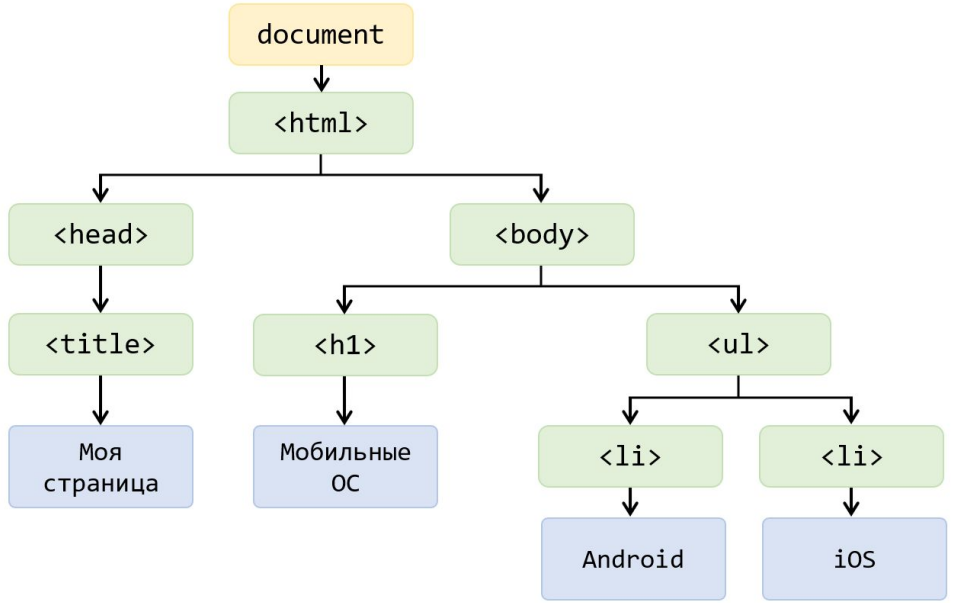
ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

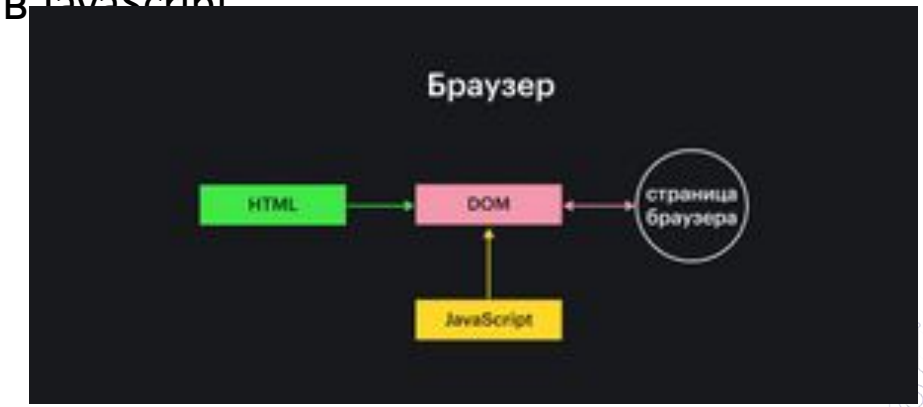
DOM (Document Object Model) — это специальная древовидная структура, которая позволяет управлять HTML-разметкой из JavaScript-кода

Управление состоит из добавления и удаления элементов, изменения их стилей и содержимого

Браузер создаёт DOM при загрузке страницы, складывает его в переменную document
Затем сообщает, что DOM создан, с помощью события DOMContentLoaded

С переменной document начинается любая работа с HTML-разметкой в JavaScript





DOM

DOM спроектирован таким образом, чтобы быть независимым от любого конкретного языка программирования, обеспечивая структурное представление документа согласно единому и последовательному API

Существует некоторое количество различных типов данных, которые используются в API

Для простоты узлы это как elements, массивы узлов как на nodeLists (либо просто elements) и атрибуты узла, просто как на attributes

DOM

document
Возвращается объект типа document (например, свойство элемента ownerDocument возвращает документ к которому он относится), этот объект document является собственным корневым объектом. В DOM document Reference разделе описан объект document.

element
Обозначает элемент или узел типа element, возвращаемый DOM API. Объекты element реализуют DOM element интерфейс и также более общий Node интерфейс.

NodeList
Массив элементов возвращается методом Document.getElementsByTagName(). Конкретные элементы в массиве доступны по индексу двумя способами:

- list.item(1)
- list[1]

DOM

attribute

Возвращается членом API (например, метод createAttribute()) - это будет ссылка на объект, которая предоставляет специальный интерфейс для атрибутов. Атрибуты - это узлы в DOM, как и элементы

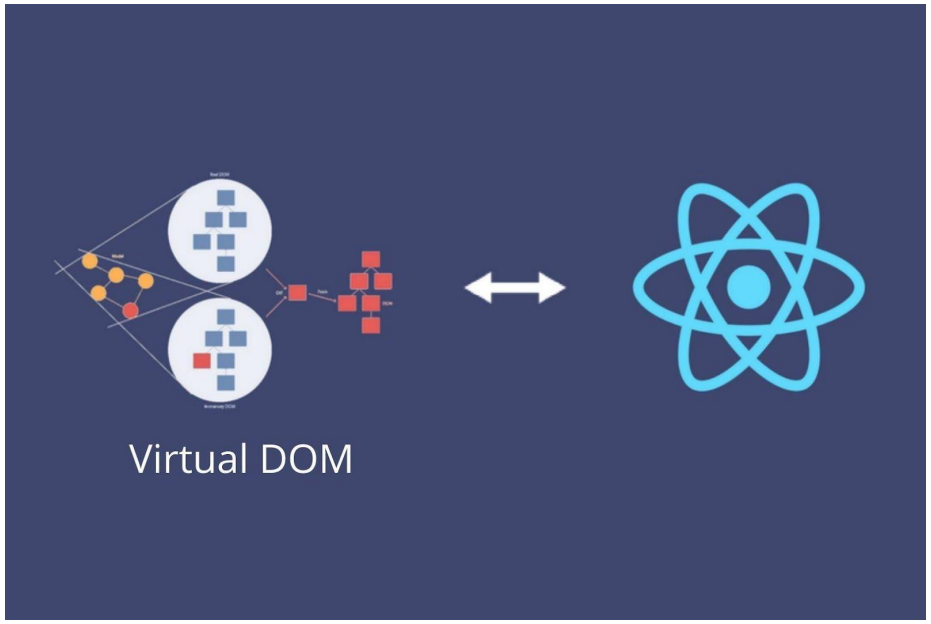
namedNodeMap

Подобна массиву, но элементы доступны по имени или индексу. Доступ по индексу для удобства перечисления, т.к. элементы не имеют определённого порядка в списке. Этот тип данных имеет метод item() для этих целей и возможно добавлять и удалять элементы из namedNodeMap

Virtual DOM

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Виртуальный DOM (VDOM) - это концепция программирования, в которой идеальное, или “виртуальное”, представление пользовательского интерфейса хранится в памяти и синхронизируется с “реальным” DOM с помощью библиотеки, такой как ReactDOM



Virtual DOM

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Виртуальный DOM - это представление DOM.

Созданием реального dom будут заниматься браузеры.

Современные фреймворки, такие как react, vue и т.д., создадут в памяти дерево элементов, подобное реальному dom, это называется virtual DOM.

DOM

```
<ul class="fruits">
    <li>Apple</li>
    <li>Orange</li>
    <li>Banana</li>
</ul>
```

Virtual DOM

Virtual DOM

```
{
  type: "ul",
  props: {
    "class": "fruits"
  },
  children: [
    {
      type: "li",
      props: null,
      children: [
        "Apple"
      ]
    },
    {
      type: "li",
      props: null,
      children: [
        "Orange"
      ]
    },
    {
      type: "li",
      props: null,
      children: [
        "Banana"
      ]
    }
  ]
}
```

Virtual DOM

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Зачем нам нужен Virtual DOM?

Раньше, когда SPA не был особо популярен, рендеринг выполнялся на стороне сервера. Таким образом, при каждом взаимодействии пользователя / запросе сервер отправляет новую страницу после рендеринга

В случае SPA будет только один документ, и в этом же документе будут выполняться все манипуляции с DOM. Таким образом, для сложных проектов может использоваться много неоптимизированных операций с DOM.

Virtual DOM

Допустим, хотим отобразить список из массива

```
function generateList(fruits) {
    let ul = document.createElement('ul');
    document.getElementByClassName('.fruits').appendChild(ul);

    fruits.forEach(function (item) {
        let li = document.createElement('li');
        ul.appendChild(li);
        li.innerHTML += item;
    });

    return ul
}

let fruits = ['Apple', 'Orange', 'Banana']
document.getElementById('#list').innerHtml = generateList(fruits)
```

Virtual DOM

Если список изменится, вышеуказанный метод будет вызван снова для генерации списка

```
fruits = ['Pineapple', 'Orange', 'Banana']
document.getElementById('#list').innerHtml = generateList(fruits)
```

Проблема с этим подходом заключается только в изменении текста single fruit, но генерируется новый список, который обновляется до DOM.

Эта операция выполняется медленно в DOM

Virtual DOM

Изменим неоптимизированный код. Это уменьшит количество операций в DOM.

```
document.querySelector('li').innerText = fruits[0]
```

Конечный результат как неоптимизированного, так и оптимизированного кода одинаков, но стоимость неоптимизированной работы DOM - производительность.

Если размер списка большой тогда можно увидеть разницу.
Это проблема старых фреймворках, таких как backbone js и прочих

Virtual DOM

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Насколько виртуальный DOM быстрее реального DOM?

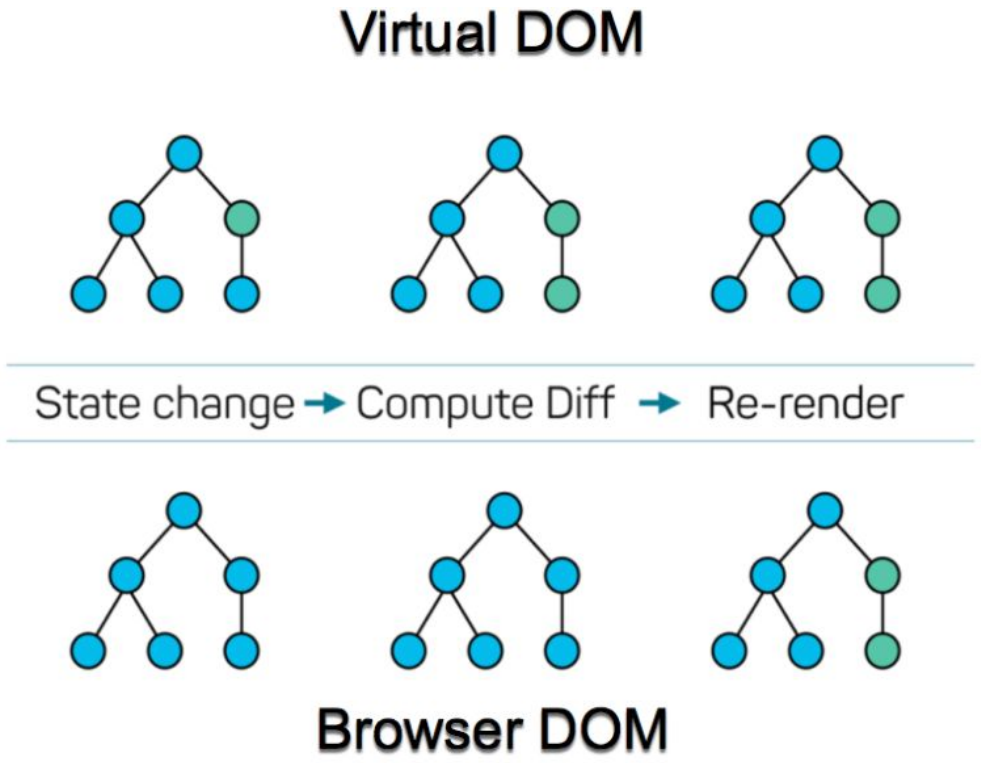
Нет, виртуальный DOM не быстрее реального DOM.

Под капотом Virtual DOM также использует Real DOM для отображения страницы или контента. Таким образом, виртуальный DOM никак не может быть быстрее реального DOM.

Почему говорят, что виртуальный DOM быстрее?

Дело не в том, что виртуальный DOM быстрее.

Используя Virtual DOM, можем узнать, что изменено, и применить только эти изменения к реальному DOM вместо замены всего DOM



Virtual DOM

Является ли Virtual DOM единственным способом сократить дорогостоящие операции DOM?

Другие фреймворки, такие как ember js, angular и svelte, используют разные подходы для решения одной и той же проблемы.

React Fiber

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

React Fiber это продолжающееся переделывание ключевых алгоритмов React и кульминация двух лет исследований проводимых командой разработки.

Цель React Fiber — улучшить приемлемость React в таких областях как анимация, лейаут и жесты.

Его главное свойство — инкрементальная отрисовка — способность разделить работу по отрисовке на части и распределить ее на несколько кадров.

React Fiber

Ключевые функции React Fiber:

- способность приостановить, прекратить или переиспользовать результат работы при поступлении новых обновлений
- способность назначить приоритет разным типам обновлений
- новые примитивы для параллельной выполнения

React Fiber

Реконсилиация — алгоритм, используемый React-ом, чтобы вычислить разницу между двумя деревьями для определения того, какие части должны быть обновлены.

Обновление — изменение в данных используемых для отрисовки приложения. Обычно, это результат вызова setState. В конечном счете обновление приводит к переотрисовке.

Центральная идея API React-а — считать что обновления вызывают переотрисовку всего приложения. Это позволяет разработчикам рассуждать декларативно, вместо того, чтобы беспокоиться о том, как эффективно перевести приложение из одного состояния в другое.

React Fiber

React использует оптимизации, которые создают видимость полного обновления приложения, но в тоже время поддерживают отличную производительность.

Основная масса этих оптимизаций это часть процесса называемого **реконсилиацией**.

Реконсилиация это алгоритм лежащий в основе виртуального DOM.

Высокоуровневое описание примерно такое: когда отрисовывается React приложение, генерируется дерево структур описывающих приложение и сохраняется в памяти. Это дерево затем отправляется среде отрисовки — например, в случае браузерного приложения, дерево переводится в набор операций над DOM.
Когда приложение обновляется (обычно из-за setState), генерируется новое дерево. Новое дерево сравнивается с предыдущим для определения, какие операции нужны, чтобы обновить отрисованное приложение.

React Fiber

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Fiber это попытка переписать с нуля реконсилятор

- Предполагается, что компоненты разных типов производят существенно разные деревья. React не будет пытаться их сравнивать, просто заменит старое дерево на новое.

- Определение разницы между двумя списками элементов осуществляется с помощью ключей. Ключи должны быть “стабильные, предсказуемые и уникальные”.

React Fiber

Планирование

Планирование — процесс определения когда работа должна быть выполнена.

Работа — любые вычисления, которые должны быть выполнены. Работа это обычно результат обновления (например, по причине вызова setState).

React Fiber

Принципы Архитектуры React – цитата из документации
В текущей реализации React обходит дерево элементов рекурсивно и вызывает функцию отрисовки всего обновленного дерева в течение одного тика. Однако, в будущем React может откладывать некоторые обновления, чтобы избежать падения частоты кадров.

Это основной мотив архитектуры React. Некоторые популярные библиотеки реализуют push-подход, когда вычисления происходят по мере поступления новых данных. Однако React придерживается pull-подхода, когда вычисления могут быть отложены до того момента, когда они необходимы.

React это не универсальная библиотека для обработки данных, это библиотека для построения пользовательских интерфейсов. Мы думаем, что React занимает однозначное место в приложении, чтобы знать, какие вычисления сейчас уместны, а какие нет.

Если что-то находится за пределами экрана, мы можем отложить логику связанную с этим. Если данные прибывают быстрее, чем частота кадров, мы можем объединять обновления в группы. Чтобы избежать падение частоты кадров, мы можем отдавать приоритет работе связанной со взаимодействием пользователя с интерфейсом (например, анимация вызванная нажатием на кнопку), чем менее важной фоновой

React Fiber

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Цель Fiber — дать возможность React воспользоваться преимуществами планирования. В частности, нам нужно иметь возможность:

- приостанавливать работу и возвращаться к ней позже.
- назначить приоритет разным типам работы.
- переиспользовать результат ранее выполненной работы.
- прервать работу, если результат ее больше не нужен.

Чтобы это сделать нужно разделить работу на блоки.

Файбер представляет собой **единицу работы**.

Файбер — это реализация стека вызовов, специализированная для React-компонентов.

Единичный файбер можно считать **виртуальным фреймом стека**.

React Fiber

Польза от переделывания стека в том, что можно сохранить фреймы стека в памяти и выполнять их как и когда захочется.

Это критично для достижения требований к планированию.

Помимо планирования, ручное управление фреймами стека раскрывает потенциал для параллелизма и границ ошибок.

React Fiber

Структура файбера

Файбер это JS-объект, который содержит информацию о компоненте, его входных данных и выходном результате.

Файбер это JS-объект, который содержит информацию о компоненте, его входных данных и выходном результате.

Важные поля файбера:
type и key служат такой же цели, как и в случае React-элемента. Фактически, когда файбер создается из элемента, эти два поля просто копируются из него.

type ссылается на компонент соответствующий файберу. Для составных компонентов, type это функция или класс компонента. Для хост-компонентов (div, span и так далее), type это строка.

Концептуально, type это функция (как в v = f(d)), выполнение которой отслеживается стековым фреймом.

Вместе с полем type, поле key используется во время реконсилиации для определения того, можно ли переиспользовать результат файбера.

React Fiber

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

child и sibling поля ссылаются на другие файберы и описывают древовидную структуру файбера.
Поле child (потомок) ссылается на файбер, соответствующий результату функции render компонента

```
function Parent() {
  return <Child />
}
```

здесь, Child будет файбером-потомком для Parent

Поле sibling (элемент того же уровня) учитывает случай, когда render возвращает несколько потомков:

```
function Parent() {
  return [<Child1 />, <Child2 />]
}
```

Поле child ссылается на первый файбер в односвязном списке потомков.
Child1 является файбером-потомком для Parent, а
Child2 - одноуровневый элемент (sibling) для Child1.

React Fiber

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

return

Поле return ссылается на файбер, к которому нужно вернуться после обработки текущего файбера. Концептуально, это тоже самое, что адрес возврата в стековом фрейме. Можно считать этот файбер родительским.

Если у файбера есть несколько потомков, return каждого потомка ссылается на родителя. В примере, для Child1 и Child2 return будет ссылаться на Parent.

```
function Parent() {
   return [<Child1 />, <Child2 />]
}
```

React Fiber

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

pendingProps и memoizedProps

Концептуально, пропсы (props) это аргументы функции.
Файберу устанавливаются pendingProps в начале выполнения, а memoizedProps устанавливаются в конце.

Когда входящие pendingProps равны memoizedProps, это сигнал к тому, что предыдущий результат выполнения файбера может быть переиспользован, чтобы избежать ненужную работу.

React Fiber

pendingWorkPriority

Число означающее приоритет работы соответствующего файбера.
В модуле ReactPriorityLevel перечислены разные уровни приоритетов и то, что они представляют.

За исключением приоритета NoWork, равного нулю, большее число соответствует меньшему приоритету.

Например, можно было бы использовать следующую функцию, чтобы проверить, что приоритет файбера выше или равен заданному:

```
function matchesPriority(fiber, priority) {
  return fiber.pendingWorkPriority !== 0 &&
    fiber.pendingWorkPriority <= priority
}
```

Планировщик использует значение приоритета, чтобы найти следующий блок работы на выполнение.

React Fiber

Это только маленькая часть о файбере

Еще есть:

- как планировщик находит следующий блок работы на выполнение
- как отслеживается приоритет и распространяется по дереву файберов
- как планировщик узнает, когда приостановить и продолжить работу
- как файберы очищаются и помечаются как выполненные
- как работают побочные эффекты (как например методы жизненного цикла)

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание