

Оптимизация и производительность React приложений

№ урока: 6 **Курс:** React Advanced

Средства обучения: Текстовый редактор или IDE, браузер, Node.js, терминал

Обзор, цель и назначение урока

В этом уроке мы, неожиданно для себя, узнаем, что React может быть медленным – поймем когда это может происходить и как с этим бороться. Вместе с этим мы познакомимся с различными инструментами, которые позволяют выявить наличия узких мест в ваших React приложениях, инструментов для оптимизации и анализа. Также мы узнаем о таком понятии как «нормализация» хранилища в Redux, узнаем чем хороши иммутабельные структуры данных в JavaScript и узнаем как сделать билд вашего приложения стройнее и быстрее.

Изучив материал данного занятия, учащийся:

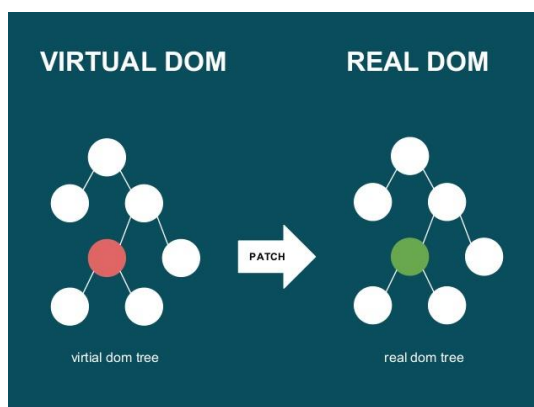
- Узнает о принципе согласования в React.
- Сможет оптимизировать и улучшать производительность своих приложений.
- Узнает о преимуществах иммутабельных структур данных в JavaScript.
- Научится использовать инструменты для оптимизации и анализа приложений при разработке.
- Узнает о принципах нормализации хранилища в Redux.
- Научится делать хорошие билды.

Содержание урока

1. Принцип согласования в React / React Reconciliation
2. Применение `shouldComponentUpdate` и `PureComponent` / Избежание ре-рендера компонентов
3. Иммутабельность в JavaScript
4. Инструменты оптимизации React приложений
5. Нормализация хранилища / `Normalizr`
6. Правила хорошего билда
7. Инструменты анализа билда.

Резюме

React, вместо того, чтобы взаимодействовать с DOM-деревом напрямую – работает с его легковесной копией, объектом на JavaScript.



Мы производим какие-то манипуляции, эти изменения VirtualDOM сравнивает (diff) с реальным DOM, и если находятся какие-то расхождения в них, то реальный DOM подвергается изменениям (patch). Такой подход работает значительно быстрее, так как операции с реальным DOM слишком дорогостоящи для браузеров.

Данный подход называется “React Reconciliation” или “Принцип согласования в React” (также вы можете встретить название “React Diff Algorithm”).

Его принцип заключается в том, что подвергая изменениям какое-то DOM-дерево, Реакт попытается найти отличия и точно(!) совершить мутации, используя для этого DOM API.

Но это касается только если DOM-узел не поменялся. Если же DOM-узел меняется, например это был `<div>`, а стал `<main>`, то React в таком случае даже не будет пытаться найти отличия этих двух узлов, он просто удалит старый узел, а на его место вставит новый – соответственно все дети в таком случае тоже перерендерятся.



Проблематика заключается в том, что алгоритм сравнения двух разных узлов оценивается примерно в $O(N^3)$, где N – это количество элементов в дереве. Т.е. на 1000 элементов нам потребуется выполнить 1 млрд. Операций.

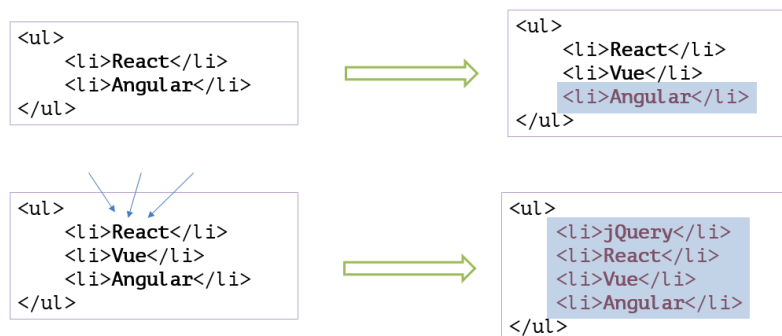
Вместо этого, если просто сделать удаление старого узла и вставки нового – это займет всего лишь $O(N)$. Разница ощутима.



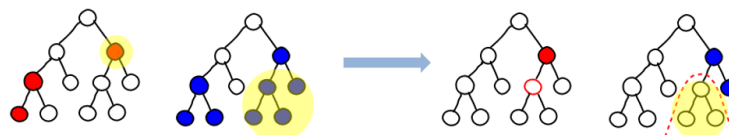
Для того, чтобы сделать правильное согласование детей в DOM-дереве – React использует ключи (keys). Поэтому используя уникальные ключи для каждого ребенка – React без проблем сможет выполнить операции удаления/вставки/замены используя внутренний хеш-мэп.

```
<ul>
  <li key="john_resig">
    jQuery
  </li>
  <li key="facebook">
    React
  </li>
  <li key="evan_you">
    Vue
  </li>
  <li key="google">
    Angular
  </li>
</ul>
```

Иначе, та же вставка в начало списка сбивала бы React с толку, из-за чего безобидное добавление в начало перерендеривало бы все дерево целиком.



Почему в некоторых случаях React может быть медленным? Дело в том, что при изменении props в родительском компоненте – во всех дочерних компонентах React также попытается отыскать различия. Если таких компонентов достаточно много, то это может занять какое-то время и привести к торможениям в интерфейсе. Но благодаря методу `shouldComponentUpdate` мы можем сравнить старые props/state с новыми, и если в них ничего не поменялось, то избежать ненужного рендера.



Также мы можем использовать для этого встроенный в React `PureComponent` (<https://facebook.github.io/react/docs/react-api.html#react.purecomponent>). Но отличие заключается в том, что `PureComponent` использует неглубокую проверку props и state по ссылке (используя оператор строгого равенства «`===`»).

Глубокое сравнение — очень затратная операция. Если бы PureComponent каждый раз ее вызывал, то он бы приносил больше вреда, чем пользы.

Также можно использовать immutable данные. Сравнение в таком случае становится очень простым, так как имеющиеся переменные не изменяются, а всегда создаются новые. В ES2015 появилось много возможностей использования Immutable данных — это, например, spread-оператор, Object.assign. Для массивов можно использовать старые добрые методы concat, map, filter, reduce — которые возвращают новый массив, вместо мутаций в старом.

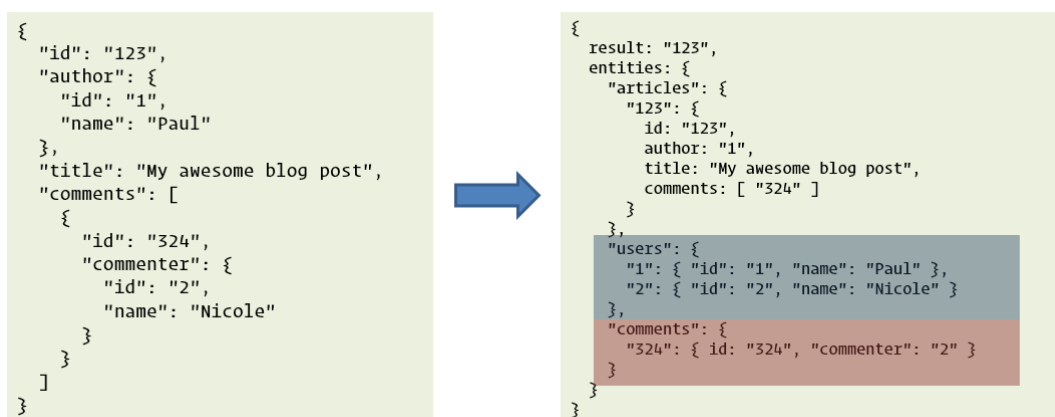
Также вы можете использовать сторонние библиотеки вроде Immutable.js (<https://facebook.github.io/immutable-js/>)

В процессе разработки вы можете использовать библиотеку <https://github.com/garbles/why-did-you-update> (обертка над React) и выявить узкие места, где происходит ненужный рендер компонентов.

Работая с каким-то сторонним API вы часто будете получать JSON-объекты с большой вложенностью. Используя Redux управлять таким состоянием не всегда удобно, поэтому для Redux зачастую этот state нормализуют.

Это значит, что с помощью схем в normalizr (<https://github.com/paularmstrong/normalizr>) вы преобразовываете ваш стейт в плоский вид и обращаетесь к данным по ключам, которые дублируют их ID.

Пример:



Теперь касательно оптимизация вашего webpack-бандла.

Для production билда всегда запускайте вебпак с ключем “-p”.

Этот ключ автоматически назначит переменной окружения (process.env.NODE_ENV) значение ‘production’, благодаря которому многие библиотеки станут намного легче за счёт удаления фрагментов предназначенных для разработки. Например, React, в продакшн-моде отключает проверку PropTypes и удаляет из бандла излишние сообщения и предупреждения, которые нужны для разработки. Это ускорит ваше приложение и сделает ваш бандл меньше. Также при включенном флаге ‘-p’ задействуется UglifyJsPlugin, который минифицирует ваш JS-код.

Вы всегда можете переопределить конфиг вебпака, благодаря переменной окружения (process.env.NODE_ENV) и можете добавить свои опции/плагины. Например, реальным юз-кейсом для продакшн-мода является использование ExtractTextPlugin (<https://github.com/webpack-contrib/extract-text-webpack-plugin>), которые “выкусывает” ваш инлайновый CSS из бандла и помещает его в отдельный файл. У плагина достаточно понятная документация и есть примеры как настроить SASS, Less и пр. Советую поиграться.

После того как вы настроите как следует ваш webpack-конфиг вы можете воспользоваться инструментами анализа вашего бандла и получить более детальную информацию.

Для этого можно использовать стандартный анализатор от Webpack -

<https://webpack.github.io/analyse/> или такой вот тул <https://github.com/th0r/webpack-bundle-analyzer>, который более информативный и наглядный.

Для того, чтобы получить JSON с мета-информацией по бандлу нужно использовать такую команду ‘webpack –json > output.json’

Материалы ко всем урокам видеокурса:

<https://github.com/fnnzzz/react-advanced-itvdm>

Чат в telegram, где вы можете задать интересующие вас вопросы:

<https://t.me/joinchat/AAAAAA3GLHawuWAWXBOccQ>

Закрепление материала

- Что React делает с разными DOM-узлами и почему именно так он делает?
- Когда может пригодиться shouldComponentUpdate?
- В чем различия shouldComponentUpdate и PureComponent?
- Какие преимущества у иммутабельных структур данных?
- Зачем нормализовать нужно хранилище и как это делать?
- Назовите несколько вариантов оптимизации бандла.

Рекомендуемые ресурсы

React reconciliation

<https://react-cn.github.io/react/docs/reconciliation.html>

<http://buildwithreact.com/article/in-depth-diffing>

<http://tftf.ru/stati/javascript/reactjs/reference/reconciliation/>

<https://www.youtube.com/watch?v=2TYstiGDJnc>

shouldComponentUpdate / PureComponent

<https://60devs.com/pure-component-in-react.html>

<http://jamesknelson.com/should-i-use-shouldcomponentupdate/>

<https://facebook.github.io/react/docs/react-api.html#react.purecomponent>

Immutability in JS

<https://www.youtube.com/watch?v=9M-r8p9ey8U>

<https://habrahabr.ru/company/devexpress/blog/302118/>

Оптимизация производительности в React

<https://facebook.github.io/react/docs/optimizing-performance.html>

<https://habrahabr.ru/post/327364/>

Полезный tool chain

<https://github.com/garbles/why-did-you-update>

<https://github.com/acdlite/recompose>

<https://github.com/th0r/webpack-bundle-analyzer>

<https://github.com/thejameskyle/babel-react-optimize>

Normalizr

<https://github.com/paularmstrong/normalizr>

<http://redux.js.org/docs/recipes/reducers/NormalizingStateShape.html>

<https://egghead.io/lessons/javascript-redux-normalizing-api-responses-with-normalizr>

<https://tonyhb.gitbooks.io/redux-without-profanity/content/normalizer.html>

<https://www.robinwieruch.de/the-soundcloud-client-in-react-redux-normalizr/>