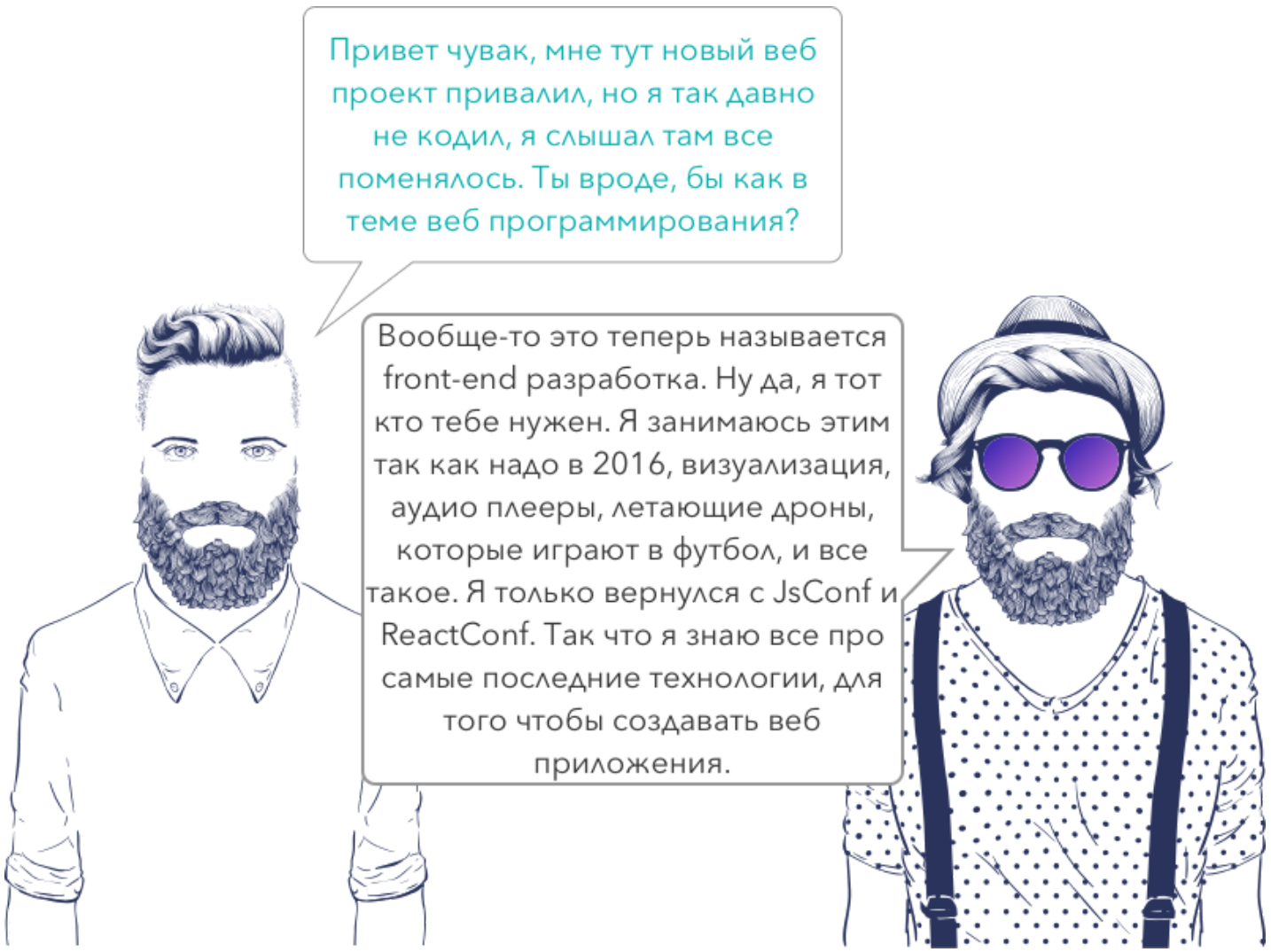
РАЗРАБОТКА ВЕБ-САЙТОВ*, REACTJS*, JAVASCRIPT*

Введение в React и Redux для бекенд-разработчиков

marshinov 11 апреля 2017 в 01:40 34,8k



Если вы как я долгое время считали, что JavaScript – это такой «игрушечный» язык на котором пишут анимашки для менюшек и падающие снежинки на форумах под новый год, а потом очнулись в 2016 году с мыслями WTF: react, flux redux, webpack, babel,… не отчаивайтесь. Вы не одиноки. Материалов по современному

фронтенду в сети много, даже слишком много. Под катом еще одно альтернативное мнение о том, каково это учить JavaScript в 2016 году.

Итак, нам потребуются: React, React Dev Tools, React-Hot-Loader, React-Router, Thunk, Redux, Redux Dev Tools, Semantic-UI, Webpack, Babel и npm.

На первый взгляд много. Сравним с бекендом: MVC-фреймворк, ORM, Data Mapper, IOC-контейнер, логер, профайлер, очереди, управление конфигурациями, сборка и выкладка… Список можно продолжить, но думаю идея понятна и так: с ростом сложности решаемых задач растет и сложность инструментов. Все чаще мы употребляем термин Web App вместо Web Site, акцентируя внимание на богатых возможностях современных веб-приложений.

Почему именно этот стек?

Если вдуматься и отбросить все лишнее, то единственный ресурс постиндустриальной эпохи, через который можно выразить все остальные – это время. Освоение каждой новой технологии требует затрат времени, а значит более перспективны инвестиции в технологии, которые не устареют в ближайшие пару месяцев. Дополнительный плюс получают технологии с пологой кривой обучения.

React + Redux VS Angular VS Yet Another JS Framework

Только ленивый не сравнил Angular с React'ом (приписав при этом,

дескать, сравнение не корректно, Angular – фреймворк, React – библиотека). Пойдем от обратного. Почему бы не выбрать что-нибудь эдакое, типа Vue, Ember или, упаси боже, Elm?

1. Поддержка крупных вендоров
2. Размер сообщества

Благодаря этим факторам, вероятность выживания Angular и React выше. Простите, другие замечательные хипстерские решения, нам не по пути. Итак, почему React? Лично для меня выбор был не простым:

1. Меня пугали jsx-файлы
2. Я кое-что умел на Angular 1.x и переходить на другую технологию было психологически не комфортно
3. ng2 по-умолчанию идет в комплекте с TypeScript, что мне как сторонникy статической типизации ближе
4. ng подкупал подходом «работает из коробки». Копаться в многообразии npm-пакетов решительно не хотелось.

Короче, я заставил себя изучить все статьи-сравнилки и написать Todo App на React, что склонило чашу весов в противоположную сторону. Ключевыми факторами для меня стали:

1. HTML-шаблоны Angular — ужасны и их синтаксис меняется от версии к версии. В React шаблон – это JavaScript, потому что компонент – не более чем View. Сообщения об ошибках в React лучше.

2. Как ни странно, TypeScript. При более детальном изучении оказалось, что не все так прекрасно. Во-первых TypeScript – это не полноценный язык со статической типизацией, а транспилер. Это сильно ограничивает возможности использования шаблонов и мета-программирования. Во-вторых, далеко не все npm-пакеты идут в комплекте с d.ts-файлами. Короче, Flow показался проще в прикручивании. В-третьих, у TypeScript есть как ярые фанаты, так и противники. Если фанаты TS сравнительно лояльны к ES6, то обратное – не верно. ES6 получает дополнительное очко к Bus Factor.

   > Если вам нравится TypeScript, ничто не мешает использовать его вместе с React. Просто конкретно мне он пока не дал критического объема преимуществ, чтобы заставить тратить время на еще один элемент в стеке.

3. Доклад Дэна Абрамова про «путешествия во времени». Если ваш опыт в бекенде похож на мой, то вы без труда увидите, что новомодный flux – это CQRS и Event Sourcing «вид в профиль». Вместо проекций – редюсеры, а вместо команд и доменных событий – экшны. А если вы работали, например, с WPF, то разобраться с React – вообще дело пары вечеров.

   > Да, Redux можно использовать и с Angular, он никак не привязан к React, но для React уже есть react-redux и react-hot-loader. Наверное, для Angular тоже есть, но мейнтейнер Redux'а явно на стороне React.

Для React и Redux доступно два расширения Chrome. Рекомендую поставить оба, чтобы сделать отладку приятной.

Таким образом, связка React + Redux:

1. Более-менее проста в изучении, потому что в основе лежит простая идея `ui = f(state => props)`, где f — это реакт-компонент, state — redux, а `state => props` — это react-redux.
2. Не тащит за собой дополнительных зависимостей
3. Обладает лучшим на данный момент Tool Support (IDE и плагины для Chrome)

> Есть еще всякие ништяки, вроде React Native, но я им не пользовался, поэтому поделиться на эту тему мне, к сожалению, нечем.

А flux и все эти модные словечки. Как это работает?

Возможно, для фронтенда flux — это некое откровение. Для бекендщика разница между CQRS и flux — не велика. React — это наше представление. Оно может зависеть от props (read-only) или state (mutable). В зависимости от state и props компонент может отображаться по-разному. Эта логика содержится в методе `render`. Компонент сам себя не перерисовывает. Вместо этого, он полагается на экосистему React. Мы можем либо изменить свое состояние (с помощью метода `setState`), либо быть перерисованными извне (переданы новые props). Обработчики событий для UI-элементов передаются через props. Получается такой код
`<button onClick={this.props.handleClick} />`.
Состояние приложение хранится в Redux и представляет собой json-объект. Redux следит за изменением объекта. Изменением считается изменение ссылки, поэтому вместо изменения текущего состояния необходимо конструировать новое путем копирования и

модификации старого. Проще всего это сделать через spread-оператор:

```
const newState = {...prevState, somethingNew: 'blah-blah'}
```

Пакет react-redux осуществляет односторонний байндинг `redux state => react component` с помощью метода connect. При изменении состояния Redux сам перерисует необходимые компоненты, передав в props dispatch и часть общего state, хранимого в Redux. Какую часть состояния и какие функции на основании `store.dispatch` передавать — решать вам. Я рекомендую передать все обработчики событий компонента и не «светить» dispatch в компонентах.

State содержится в Redux, но и у компонентов есть свой state. Какой из них использовать?

Разработчик Redux предлагает делать как удобнее. Это не совсем формальный совет. У нас сложилась практика использовать state компонента только для форм или в целях оптимизации.

JSX

JSX – это не JavaScript. Да, React можно писать без JSX, но проще тогда без React'а. Вообще ситуация с HTML-шаблонами напоминает мне засилье шаблонизаторов для PHP лет десять назад. Самым монструозным из всех был конечно Smarty. Мне казалось, что люди сошли с ума. Как иначе можно было объяснить желание написать шаблонизатор… для шаблонизатора?

JSX – простой и понятный способ использовать JavaScript в шаблонах. Вам не нужно учить дополнительный ЯП, просто оберните теги в `(скобки)`, а код внутри тегов `{в другие скобки}`. И все. Да, так просто. Если вас беспокоит разделение логики и представления, перестаньте беспокоиться прямо сейчас. React — это View-библиотека. За состояние приложения (в т.ч. поведение) отвечает Redux. React может «диспатчить» сообщения в store, а Redux будет их обрабатывать либо через редьюсеры (чистые функции), либо через специальные middleware (побочные эффекты).

React позволяет создавать functional stateless-компоненты:

```
const StatelessComponent = props => (<div>Hello, {props.name}!
</div>)
```

Или компоненты-классы:

```
class Hello extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Функциональная запись короче и лучше читается. Но я рекомендую не увлекаться экономией строк и вместо:

```
export default props => (<div>{props.title}</div>)
```

использовать чуть менее лаконичное, но более безопасное:

```
const StatelessComponent = props => (<div>{props.title}</div>)
StatelessComponent.propTypes = {…}
export default StatelessComponent
```

Во-первых, если вы не передадите параметры, то React недвусмысленно намекнет в консоли, что вы не правы. Во-вторых, *WebStorm* умеет анализировать `PropTypes` и при авто-дополнении заботливо вставит все required props.

Babel

Если вы не поняли на каком языке примеры кода выше, не расстраивайтесь. Это не JavaScript, ну не совсем JavaScript. Это ES6 + JSX. С JSX мы разобрались в параграфе выше – это просто синтаксический сахар для шаблонизации (почти как `<?=$var?>` в PHP или `@Model.Param` в Razor).

С ES6 ситуация чуть сложнее и запутаннее. Если коротко:

1. JavaScript собрали на коленке под нужны тогдашнего интернета, который представлял собой действительно в основном гипер-текст.
2. Прошло некоторое время и в сайты начали пихать все что угодно, кроме текста.
3. Язык в существующем виде перестал удовлетворять нуждам рынка. Консорциум ECMA стал придумывать всякие новые фичи и стандарты языка, да с такой скоростью, что браузеры не успели все внедрить.
4. В сообществе психанули и написали Babel – транспилер из JavaScript в… JavaScript. Ну в смысле из «нового» JS в «старый», который браузеры поддерживают.

Babel умеет транспилить не только JS, но и JSX, что позволяет писать React-приложения на ES6.

> Да, есть `React.createElement`. Можно писать на React-приложения и на ES5, но зачем?

Стоит отметить, что ES6 – это не истина в последней инстанции. Некоторые фичи до сих пор являются экспериментальными (например, генераторы) и для их использования потребуются полифиллы (библиотеки, реализующие экспериментальные фичи стандарта). Частично из-за этого мы решили отказаться от redux-saga в пользу redux-thunk, хотя и идея диспатчить функции до сих пор не кажется мне изящной (она просто работает).

Webpack

Так, то есть пишем мы на ES6 + JSX, а в бразуере выполняется минифицированный JS. Все это напоминает историю изобретения высокоуровневых ЯП. Люди могли писать более эффективные программы на ассемблере, но предпочли удобство и продуктивность. Раз есть исходники и компилятор (транспилятор в нашем случае), то потребуется и система сборки. Если в вашей пещере было достаточно тепло и уютно, возможно, названия grunt и gulp вам ничего не говорят. Что к лучшему. На данный момент, можно считать (слава богу), что для JS есть только один сборщик – Webpack — оставивший конкурентов позади. Можно считать, что webpack — аналог maven или msbuild (кому что ближе) в мире фронтэнда. Не смотря на то что, конфиги webpack'а на первый взгляд напоминают некромантские свитки, через какое-то время

привыкаешь. Наверное, каждый любитель фронтенда должен хотя-бы раз в своей жизни написать tutorial по настройке webpack, также как каждый фанат ФП – tutorial по монадам.

Что нужно знать про webpack:

1. Вам потребуется кто-то, кто умеет его настраивать
2. npm start — для запуска dev-сервера
3. npm run build для сборки фронта на продакшн

> Вообще Webpack собирает не только JS, но еще и sass, svg, шрифты и вообще все что душе угодно, но я пока еще не готов написать полноценный туториал, так что поищите на просторах интернета.

Npm

У Ruby есть gem'ы, у php – composer, у .NET – nuget. Короче, JavaScript тоже потребовался пакетный менеджер. Изначально npm использовался в nodejs-разработке (отсюда и название — node package manager), а для фронта использовался bower. Потребность в последнем как-то отпала сама собой с повсеместным переходом использованием ES6, Webpack и TypeScript. Этот параграф добавлен лишь для того чтобы отметить, что npm использует файл package.json, внутри которого можно написать:

```
"scripts": {
    "build": "webpack --config webpack/config.js -p",
    "start": "webpack-dev-server --config webpack/config.js"
  }
```

Без этих строк `npm run build` и `npm start` не заработают.

Было бы логичнее вызывать `npm build`, а не `npm run build`, но эта команда зарезервирована для внутренних целей npm, так что ничего не выйдет.

React-Hot-Loader

Ключ `--hot` запускает dev-сервер webpack'а с «горячей заменой». Согласитесь, билдить при каждом изменении – довольно уныло. HMR (hot moudle replacement) делает это за вас. react-hot-loader позволяет избежать при этом перезагрузки страницы и потери текущего состояния. Настройка hot-loader'а правда, довольно тонкая работа и вообще фича — экспериментальная и работает не всегда. Особенно сложные отношения у горячей замены с react-router. Но к хорошему быстро привыкаешь и рано или поздно вам захочется написать `if(module.hot)` для того, чтобы страница не перезагружалась.

React-Router и Thunk

Основная ниша React'а в Web – это конечно SPA-сайты. А какой SPA-сайт без навигации и общения с сервером. Первую задачу решает react-router. Здесь альтернатив нет. Из неприятных сюрпризов:

1. Четвертая версия не совместима с третьей из-за чего у нее проблемы с пакетом history.

2. Для hot-reload нужно соблюдать некоторые нюансы, иначе в консоли будут появляется предупреждения о том, что нельзя заменить route. К счастью ошибки достаточно информативные и исправить их просто.
3. `RouterMiddleware` может сломать Redux плагин при совместном использовании.
4. Не очевидно, что вместо тегов `<Route />` можно использовать JavaScript-объекты и передать их в компонент роутера `<Router routes={routes} />`. Это бывает полезно, при разработке модульных приложений, когда структура маршрутов не известна заранее.

Thunk – это middleware для redux, позволяющее диспатчить функции вместо объектов. Чаще всего используется для запросов к серверу. В компоненте делать запросы к серверу – не комильфо. Редьюсеры – вообще дожны быть чистыми функциями. Ничего не остается, как делегировать это middleware.

Альтернатива thunk – redux-saga. Сага предоставляет больше возможностей, но у нее довольно крутая кривая обучения, и она тащит за собой полифиллы для генераторов. При правильном проектировании логики в компонентах нет, а connect к стейту Redux в основном производится через фабричный метод. В общем, мне показалось, что нет большой разницы, как именно управление перейдет к `fetch` и будет ли написано `yield` или `then`. На сайте Redux пример с thunk, так что по совокупности причин сагу я отложил до лучших времен.

Semantic-UI

Раз мы заговорили про SPA-приложения, то кроме навигации и запросов к серверу нужны еще компоненты, которые будут ту самую серверную информацию отображать. Для React есть обвязки Bootstrap, Material UI, Syncfusion Web Essentials (хотя эти обвязки не честные – там внутри jQuery). Наш выбор остановился на Sematic-UI. Решение удалось принять очень быстро – сначала отмели платные компоненты. Material UI не стали использовать из-за обилия анимации (сложнее модифицировать). Остались Bootstrap и Semantic. На Бутстрапе уже пол интернета сделано и в целом, Семантик показался более визуально-привлекательным. В общем, остановились на нем. Сразу оговорюсь, что использование Semantic UI – строго опционально, потому что минифицированная версия весит около 500кб.

Так что разрабатывать фронтенд в 2016 году вполне себе комфортно. Да инструментов много, многие библиотеки не совместимы, новые версии выходят очень часто. Это разумная плата за гигантский скачок в качестве фронтенд-стека.

Комментарии (181)

Похожие публикации

Эволюция на React+Redux

28

Fen1kz • 7 апреля 2017 в 10:21

React Redux. Получение доступа к state из функции mapDispatchToProps()

10

ИЗ ПЕСОЧНИЦЫ

gnv_cor • 7 ноября 2016 в 14:22

Универсальный (Изоморфный) проект на Koa 2.x + React + Redux + React-Router

6

BoryaMogila • 30 сентября 2016 в 09:43

Популярное за сутки

Яндекс открывает Алису для всех разработчиков. Платформа Яндекс.Диалоги (бета)

69

BarakAdama • вчера в 10:52

Почему следует игнорировать истории основателей успешных стартапов

20

ПЕРЕВОД

m1rko • вчера в 10:44

Как получить телефон (почти) любой красотки в Москве, или интересная особенность MT_FREE

24

ИЗ ПЕСОЧНИЦЫ

cab404 • вчера в 20:27

Java и Project Reactor

10

zealot_and_frenzy • вчера в 10:56

Пользовательские агрегатные и оконные функции в PostgreSQL и Oracle

6

erogov • вчера в 12:46

Лучшее на Geektimes

Как фермеры Дикого Запада организовали телефонную сеть на колючей проволоке

31

NAGru • вчера в 10:10

Энтузиаст сделал новую материнскую плату для ThinkPad X200s

49

alizar • вчера в 15:32

Кто-то посылает секс-игрушки с Amazon незнакомцам. Amazon не знает, как их остановить

85

Pochtoycom • вчера в 13:06

Илон Маск продолжает убеждать в необходимости создания колонии людей на Марсе

139

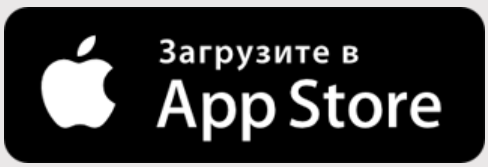
marks • вчера в 14:19

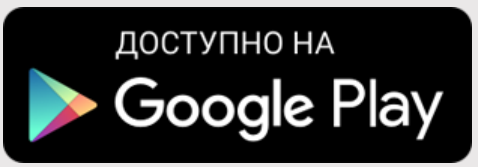
Дела шпионские (часть 1)

16

TashaFridrih • вчера в 13:16

Мобильное приложение





Полная версия