РАЗРАБОТКА ВЕБ-САЙТОВ*, КЛИЕНТСКАЯ ОПТИМИЗАЦИЯ*, REACTJS*, NODE.JS*

Create React App (aka React Scripts) и серверный рендеринг с Redux и Router

dfuse 16 марта 2017 в 09:34 14,8k

Из комментариев к статье стало понятно, что очень многие люди склоняются в сторону экосистемы Create React App (он же React Scripts). Это вполне разумно, т.к. это самый популярный и простой в использовании продукт (благодаря отсутствию конфигурации и поддержке ведущих людей React-сообщества), в котором, к тому же, есть почти все необходимое — сборка, режим разработки, тесты, статистика покрытия. Не хватает только серверного рендеринга.

В качестве одного из способов в официальной документации предлагается либо вбивать начальные данные в шаблон либо воспользоваться статическими слепками. Первый подход не позволит поисковикам нормально индексировать статичный HTML, а второй — не поддерживает проброс никаких начальных данных, кроме HTML (фраза из документации: this doesn't pass down any state except what's contained in the markup). Поэтому если используется Redux, то придется для рендеринга использовать что-то другое.

Я адаптировал пример из статьи для использования с Create React App, теперь он называется Create React Server и умеет запускать серверный рендеринг командой:

```
create-react-server --createRoutes src/routes.js --createStore
src/store.js
```

При таком запуске никакой особой конфигурации не требуется, все делается через параметры командной строки. Если нужно — можно подсунуть так же свои шаблоны и обработчики.

Небольшое лирическое отступление. Как говорят авторы React Router — их сайты индексируются Гуглом без проблем и без всякого серверного рендеринга. Может это и так. Но одной из главных проблем является не только Гугл, но и быстрая доставка контента юзеру, и это может даже поважнее индексации, которую можно обмануть.

Установка

Для начала установим требующийся для этого примера пакеты:

```
npm install create-react-server --save-dev
```

Добавим файл `.babelrc` или секцию `babel` в файл `package.json`

```
{
    "presets": [
      "react-app"
    ]
}
```

Пресет `babel-preset-react-app` ставится вместе с `react-scripts`, но для серверного рендеринга нам надо явно на него сослаться.

Страница (т.е. конечная точка React Router)

Как и прежде, суть серверного рендеринга довольно проста: на сервере нам нужно определить на основе правил роутера, какой компонент будет показан на странице, выяснить, какие данные ему нужны для работы, запросить эти данные, отрендерить HTML, и выслать этот HTML вместе с данными на клиент.

Сервер берет конечный компонент, вызывает у него `getInitialProps`, внутри которого можно сделать диспатч экшнов Redux'a и вернуть начальный набор `props` (на случай, если Redux не используется). Метод вызывается как на клиенте, так и на сервере, что позволяет сильно упростить начальную загрузку данных.

```
// src/Page.js

import React, {Component} from "react";
import {connect} from "react-redux";
import {withWrapper} from "create-react-server/wrapper";
import {withRouter} from "react-router";

export class App extends Component {

    static async getInitialProps({location, query, params, store}) {
        await store.dispatch(barAction());
        return {custom: 'custom'}; // это станет начальным набором
props при рендеринге
    };

    render() {
```

```
        const {foo, bar, custom, initialError} = this.props;
        if (initialError) return (<pre>Ошибка в функции
getInitialProps: {initialError.stack}</pre>);
        return (
            <div>Foo {foo}, Bar {bar}, Custom {custom}</div>
        );
    }

}

// подключаемся к Redux Provider как обычно
App = connect(state => ({foo: state.foo, bar: state.bar})(App);

// подключаемся к WrapperProvider, который тянет initialProps с
сервера
App = withWrapper(App);

// до кучи подключаемся к React Router
App = withRouter(App);

export default App;
```

Переменная `initialError` будет иметь значение, если в функции `getInitialProps` возникла ошибка, причем не важно где — на клиенте или на сервере, поведение одинаково.

Страница, которая будет использоваться как заглушка для 404 ошибок должна иметь статическое свойство `notFound`:

```
// src/NotFound.js

import React, {Component} from "react";
import {withWrapper} from "create-react-server/wrapper";

class NotFound extends Component {
    static notFound = true;
    render() {
        return (
            <div>404 Not Found</div>
        );
    }
```

```
}

export default withWrapper(NotFound);
```

Router

Функция `createRoutes` должна возвращать правила роутера, асинхронные роуты тоже поддерживаются, но для простоты это пока опустим:

```
// src/routes.js

import React from "react";
import {IndexRoute, Route} from "react-router";
import NotFound from './NotFound';
import App from './Page';

export default function(history) {
    return <Route path="/">
        <IndexRoute component={App}/>
        <Route path='*' component={NotFound}/>
    </Router>;
}
```

Redux

Функция `createStore` должна принимать начальное состояние в качестве параметра и возвращать новый `Store`:

```
// src/store.js

import {createStore} from "redux";

function reducer(state, action) { return state; }

export default function (initialState, {req, res}) {
```

```
    if (req) initialState = {foo: req.url};
    return createStore(
        reducer,
        initialState
    );
}
```

Когда функция вызывается на сервере, второй параметр будет иметь объекты Request и Response из NodeJS, можно вытащить некую информацию и вложить ее в начальное состояние.

Главная входная точка

Соберем все воедино, а также добавим специальную обертку для получения `initialProps` с сервера:

```
// src/index.js

import React from "react";
import {render} from "react-dom";
import {Provider} from "react-redux";
import {browserHistory, match, Router} from "react-router";
import {WrapperProvider} from "react-router-redux-
middleware/wrapper";

import createRoutes from "./routes";
import createStore from "./store";

const Root = () => (
    <Provider store={createStore(window.__INITIAL_STATE__)}>
        <WrapperProvider initialProps={window.__INITIAL__PROPS__}>
            <Router history={browserHistory}>{createRoutes()}
</Router>
        </WrapperProvider>
    </Provider>
);

render((<Root/>), document.getElementById('root'));
```

Запуск простого сервера через консольную утилиту

Добавим скрипты в секцию `scripts` файла `package.json`:

```
{
    "build": "react-scripts build",
    "server": "create-react-server --createRoutes src/routes.js --
createStore src/store.js
}
```

И запустим

```
npm run build
npm run server
```

Теперь если мы откроем `http://localhost:3000` в браузере — мы увидим страницу, подготовленную на сервере.

В этом режиме результат сборки сервера нигде не хранится и каждый раз вычисляется на лету.

Запуск сервера через API и сохранение результатов сборки

Если возможностей командной строки стало мало, или требуется хранить результаты сборки сервера, то всегда можно создать сервер не через CLI, а через API.

Установим в дополнение к предыдущим пакетам `babel-cli`, он понадобится для сборки сервера:

```
npm install babel-cli --save-dev
```

Добавим скрипты в секцию `scripts` файла `package.json`:

```
{
    "build": "react-scripts build && npm run build-server",
    "build-server": "NODE_ENV=production babel --source-maps --out-
dir build-lib src",
    "server": "node ./build-lib/server.js"
}
```

Таким образом клиентская часть будет по-прежнему собираться Create React App (React Scripts), а серверная — с помощью Babel, который заберет все и `src` и положит в `build-lib`.

```
// src/server.js

import path from "path";
import express from "express";
import {createExpressServer} from "create-react-server";
import createRoutes from "./createRoutes";
import createStore from "./createStore";

createExpressServer({
    createRoutes: () => (createRoutes()),
    createStore: ({req, res}) => (createStore({})),
    outputPath: path.join(process.cwd(), 'build'),
    port: process.env.PORT || 3000
}));
```

Запустим:

```
npm run build
npm run server
```

Теперь если мы снова откроем `http://localhost:3000` в браузере — то мы опять увидим ту же страницу, подготовленную на сервере.

Полный код примера можно посмотреть тут: https://github.com/kirill-konshin/react-router-redux-middleware/tree/master/examples/create-react-app.

Похожие публикации

Архитектура модульных React + Redux приложений 2. Ядро

3

marshinov • 24 апреля 2017 в 10:04

Упрощаем универсальное/изоморфное приложение на React + Router + Redux + Express

22

dfuse • 9 марта 2017 в 09:36

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

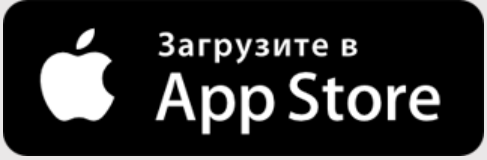
139

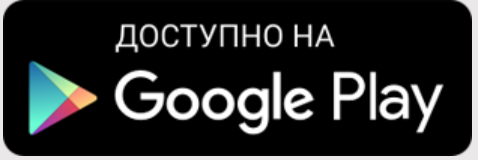
marks • вчера в 14:19

Дела шпионские (часть 1)

16

TashaFridrih • вчера в 13:16





Полная версия