РАЗРАБОТКА ВЕБ-САЙТОВ*, ВЕБ-ДИЗАЙН*, REACTJS*, NODE.JS*, JAVASCRIPT*

Универсальные приложения React + Express

apapacy 14 февраля в 04:19 4,1k

В прошлой статье рассматривалась библиотека Next.js, которая позволяет разрабатывать универсальные приложения «из коробки». В обсуждении статьи были озвучены существенные недостатки этой библиотеки. Судя по тому, что https://github.com/zeit/next.js/issues/88 бурно обсуждается с октября 2016 года, решения проблемы в ближайшее время не будет.

Поэтому, предлагаю ознакомится с современным состоянием «экосистемы» React.js, т.к. на сегодняшний день все, что делает Next.js, и даже больше, можно сделать при помощи сравнительно простых приемов. Есть, конечно, и готовые заготовки проектов. Например, мне очень нравится проект, который, к сожалению, базируется на неактульной версии роутера. И очень актуальный, хотя не такой «заслуженный» проект.

Использовать готовые проекты с массой плохо документированных возможностей немного страшно, т.к. не знаешь, где споткнешься, и самое главное — как развивать проект. Поэтому для тех, кто хочет разобраться в современном состоянии вопроса (и для себя), я сделал заготовку проекта с разъяснениями. В ней не будет какого-то моего личного эксклюзивного кода. Просто компиляция из примеров документации и большого количества статей.

В прошлой статье были перечисены задачи которые должно решать универсальное приложение.

1. Асинхронная предзагрузка данных на сервере (React.js как и большинство подобных бибиотек реализует только синхронный рендеринг)и формирование состояния компонента.
2. Серверный рендеринг компонента.
3. Передача состояния компонента на клиент.
4. Воссоздание компонента на клиенте с состоянием, переданным с сервера.
5. «Присоединение» компонента (hydrarte(...)) к полученной с сервера разметке (аналог render(...)).
6. Разбиение кода на оптимальное количество фрагментов (code splitting).

И, конечно, в коде серверной части и клиентской части фронтенда приложения не должно быть различий. Один и тот же компонент должен работать одинаково и при серверном и при клиентском рендеринге.

Начнем с роутинга. В документации React для реализации универсального роутинга предлагается формировать роуты на основании простого объекта. Например так:

```
// routes.js
module.exports = [
  {
    path: '/',
    exact: true,
    // component: Home,
```

```
    componentName: 'home'
  }, {
    path: '/users',
    exact: true,
    // component: UsersList,
    componentName: 'components/usersList',
  }, {
    path: '/users/:id',
    exact: true,
    // component: User,
    componentName: 'components/user',
  },
];
```

Такай форма описания роутов позволяет:

1) сформировать серверный и клиентский роутер на основании единого источника;
2) на сервере сделать предзагрузку данных до создания экземпляра компонента;
3) организовать разбиение кода на оптимальное количество фрагментов (code splitting).

Код серверного роутера очень простой:

```
import React from 'react';
import { Switch, Route } from 'react-router';
import routes from './routes';
import Layout from './components/layout'

export default (data) => (
  <Layout>
    <Switch>
      {
        routes.map(props => {
          props.component = require('./' + props.componentName);
          if (props.component.default) {
            props.component = props.component.default;
          }
```

```
          return <Route key={ props.path } {...props}/>
        })
      }
    </Switch>
  </Layout>
);
```

Отсутсвие возможности использовать полноценный общий `<Layout/>` в Next.js как раз и послужило отправной точкой для написания этой статьи.

Код клиентского роутера немного сложнее:

```
import React from 'react';
import { Router, Route, Switch} from 'react-router';
import routes from './routes';
import Loadable from 'react-loadable';
import Layout from './components/layout';

export default (data) => (
  <Layout>
    <Switch>
      {
        routes.map(props => {
          props.component = Loadable({
            loader: () => import('./' + props.componentName),
            loading: () => null,
            delay: () => 0,
            timeout: 10000,
          });
          return <Route key={ props.path } {...props}/>;
        })
      }
    </Switch>
  </Layout>
);
```

Самая интересная часть заключается в фрагменте кода `() => import('./' + props.componentName)`. Функция import() дает

команду webpack для реализации code splitting. Если бы на странице была обычная конструкция import или require(), то webpack включил бы код компонента в один результирующий файл. А так код будет загружаться при переходе на роут из отдельного фрагмента кода.

Рассмотрим основную точку входа клиентской части фронтенда:

```
'use strict'
import React from 'react';
import { hydrate } from 'react-dom';
import { Provider } from 'react-redux';
import {BrowserRouter} from 'react-router-dom';
import Layout from './react/components/layout';
import AppRouter from './react/clientRouter';
import routes from './react/routes';
import createStore from './redux/store';

const preloadedState = window.__PRELOADED_STATE__;
delete window.__PRELOADED_STATE__;
const store = createStore(preloadedState);

const component = hydrate(
  <Provider store={store}>
    <BrowserRouter>
        <AppRouter />
    </BrowserRouter>
  </Provider>,
  document.getElementById('app')
);
```

Все достаточно обычно и описано в документации React. Воссоздается состояние компонента с сервера и компонент «присоединяется» к готовой разметке. Обращаю внимание, что не все библиотеки позволяют сделать такую операцию в одной строчке кода, как это можно сделать в React.js.

И тот же компонент в серверном варианте:

```
import { matchPath } from 'react-router-dom';
import routes from './react/routes';
import AppRouter from './react/serverRouter';
import stats from '../dist/stats.generated';

...

app.use('/', async function(req, res, next) {
  const store = createStore();
  const promises = [];
  const componentNames = [];
  routes.forEach(route => {
    const match = matchPath(req.path, route);
    if (match) {
      let component = require('./react/' + route.componentName);
      if (component.default) {
        component = component.default;
      }
      componentNames.push(route.componentName);
      if (typeof component.getInitialProps == 'function') {
        promises.push(component.getInitialProps({req, res, next,
match, store}));
      }
    }
    return match;
  })

  Promise.all(promises).then(data => {
    const context = {data};
    const html = ReactDOMServer.renderToString(
      <Provider store={store}>
        <StaticRouter location={req.url} context={context}>
            <AppRouter/>
        </StaticRouter>
      </Provider>
    );
    if (context.url) {
      res.writeHead(301, {
        Location: context.url
      })
      res.end()
    } else {
      res.write(`
        <!doctype html>
```

```
        <script>
         // WARNING: See the following for security issues around
embedding JSON in HTML:
         //
http://redux.js.org/docs/recipes/ServerRendering.html#security-
considerations
         window.__PRELOADED_STATE__ =
${JSON.stringify(store.getState()).replace(/</g, '\\u003c')}
        </script>
        <div id="app">${html}</div>
        <script src='${assets(stats.common)}'></script>
        ${componentNames.map(componentName =>
          `<script src='${assets(stats[componentName])}'></script>`
        )}
      `)
      res.end()
    }
  })
});
```

Наиболее значимая часть — это определение по роуту необходимого компонента:

```
  routes.forEach(route => {
    const match = matchPath(req.path, route);
    if (match) {
      let component = require('./react/' + route.componentName);
      if (component.default) {
        component = component.default;
      }
      componentNames.push(route.componentName);
      if (typeof component.getInitialProps == 'function') {
        promises.push(component.getInitialProps({req, res, next,
match, store}));
      }
    }
    return match;
  })
```

После того как мы находим компонент, мы вызываем его асинхронеый статический метод

`component.getInitialProps({req, res, next, match, store})`. Статический — потому что экземпляр компонента на сервере еще не создан. Этот метод назван по аналогии с Next.js. Вот как этот метод может выглядеть в компоненте:

```
class Home extends React.PureComponent {
  static async getInitialProps({ req, match, store, dispatch }) {
   const userAgent = req ? req.headers['user-agent'] :
navigator.userAgent
   const action = userActions.login({name: 'John', userAgent});
   if (req) {
     await store.dispatch(action);
   } else {
     dispatch(action);
   }
   return;
  }
```

Для хранения состояния объекта исползуется redux, что в данном случае существенно облегчает доступ к состоянию на сервере Без redux это было бы сделать не просто сложно а очень сложно.

Для удобства разработки нужно обеспечить компиляцию клиентского и серверного кода компонентов «на лету» и обновление браузера. Об этом а также о конфигурациях webpack для работы проекта я планирую рассказатьв следующей статье.
https://github.com/apapacy/uni-react

apapacy@gmail.com
14 февраля 2018 года

+6

Поделиться:

Сохранить:

Комментировать

Похожие публикации

React.js: собираем с нуля изоморфное / универсальное приложение. Часть 1: собираем стек 76

ИЗ ПЕСОЧНИЦЫ

yury-dymov • 14 сентября 2016 в 10:45

Разработка javascript приложений на базе Rx.js и React.js (RxReact) 9

MostovenkoAlexander • 5 марта 2015 в 19:01

Инъекция React JS в приложение на Angular JS или борьба за производительность 3

eastbanctech • 5 августа 2014 в 07:34

Популярное за сутки

Наташа — библиотека для извлечения структурированной информации из текстов на русском языке

14

alexkuku • вчера в 16:12

Unit-тестирование скриншотами: преодолеваем звуковой барьер. Расшифровка доклада

4

lahmatiy • вчера в 13:05

Люди не хотят чего-то действительно нового — они хотят привычное, но сделанное иначе

25

ПЕРЕВОД

Smileek • вчера в 10:32

Руководство по SEO JavaScript-сайтов. Часть 2. Проблемы, эксперименты и рекомендации

2

ПЕРЕВОД

ru_vds • вчера в 12:04

Как адаптировать игру на Unity под iPhone X к апрелю

0

P1CACHU • вчера в 16:13

Лучшее на Geektimes

Стивен Хокинг, автор «Краткой истории времени», умер на 77 году жизни

33

HostingManager • вчера в 13:49

Обзор рынка моноколес 2018

70

lozga • вчера в 06:58

«Битва за Telegram»: 35 пользователей подали в суд на ФСБ

40

alizar • вчера в 15:14

Стивен Хокинг и его работа — что дал ученый человечеству?

8

marks • вчера в 14:46

Sunlike — светодиодный свет нового поколения

17

AlexeyNadezhin • вчера в 20:32