РАЗРАБОТКА ВЕБ-САЙТОВ*, REACTJS*, JAVASCRIPT*

Универсальные приложения React + Express (продолжение)

арарасу 14 февраля в 21:56 2,4k

В предыдущей статье был рассмотрен простой проект универсального приложения на React.js, в котором используются только стандартные средства и фрагменты кода из официальной документации React.js. Но этого недостаточно для удобной разработки. Нужно сформировать окружение так, чтобы были стандартные возможности (например «горячая» перегрузка компонентов) в равной степени как для серверной, так и для клиентской части фронтенда.

Проект из предыдущей статьи построен на описании роутов в виде простого объекта:

```
// 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',
  },
];
```

Этот объект задает также разбиение кода на фрагменты (code splitting). Во так это сконфигурировано для клиентского webpack:

```
const webpack = require('webpack'); //to access built-in
const HtmlWebpackPlugin = require('html-webpack-plugin');
//installed via npm
const path = require('path');
const CommonsChunkPlugin = webpack.optimize.CommonsChunkPlugin;
const nodeEnv = process.env.NODE_ENV || 'development';
const port = Number(process.env.PORT) || 3000;
const isDevelopment = nodeEnv === 'development';
const routes = require('../src/react/routes');
const hotMiddlewareScript = `webpack-hot-middleware/client?
path=/__webpack_hmr&timeout=20000`;

const entry = {};
for (let i = 0; i < routes.length; i++ ) {
  entry[routes[i].componentName] = [
    '../src/client.js',
    '../src/react/' + routes[i].componentName + '.js',
  ];
  if (isDevelopment) {
    entry[routes[i].componentName].unshift(hotMiddlewareScript);
  }
}

module.exports = {
  name: 'client',
  target: 'web',
  cache: isDevelopment,
  devtool: isDevelopment ? 'cheap-module-source-map' : 'hidden-
source-map',
  context: __dirname,
  entry,
  output: {
    path: path.resolve(__dirname, '../dist'),
    publicPath: isDevelopment ? '/static/' : '/static/',
    filename: isDevelopment ? '[name].bundle.js': '[name].
[hash].bundle.js',
```

```
    chunkFilename: isDevelopment ? '[name].bundle.js': '[name].
[hash].bundle.js',
  },
  module: {
    rules: [{
        test: /\.jsx?$/,
        exclude: /node_modules/,
        loader: "babel-loader",
        options: {
          cacheDirectory: isDevelopment,
          babelrc: false,
          presets: [
            'es2015',
            'es2017',
            'react',
            'stage-0',
            'stage-3'
          ],
          plugins: [
            "transform-runtime",
            "syntax-dynamic-import",
          ].concat(isDevelopment ? [
              ["react-transform", {
                "transforms": [{
                  "transform": "react-transform-hmr",
                  "imports": ["react"],
                  "locals": ["module"]
                }]
              }],
            ] : [
            ]
          ),
        }
      }
    ]
  },
  plugins: [
    new webpack.optimize.OccurrenceOrderPlugin(),
    new webpack.HotModuleReplacementPlugin(),
    new webpack.NoEmitOnErrorsPlugin(),
    new webpack.NamedModulesPlugin(),
    //new webpack.optimize.UglifyJsPlugin(),
    function(compiler) {
                this.plugin("done", function(stats) {
                    require("fs").writeFileSync(path.join(__dirname,
"../dist", "stats.generated.js"),
           'module.exports=' +
JSON.stringify(stats.toJson().assetsByChunkName) +
```

```
';console.log(module.exports);\n');
      });
    }
  ].concat(isDevelopment ? [
        ] : [
      new CommonsChunkPlugin({
        name: "common",
        minChunks: 2
      }),
    ]
  ),
};
```

В каждый фрагмент результирующего кода включается общая точка входа `client.js`, основной компонент для соответсвующего имени роута, а для окружения development еще и `webpack-hot-middleware/client`.

Для рабочего билда дополнительно формируется модуль с общим для всех компонгентов кодом:

```
new CommonsChunkPlugin({
    name: "common",
    minChunks: 2
}),
```

Значение `minChunks` позволяет управлять рамером фрагментов. При значении 2 любой участок одинакового кода, который используется в двух фрагментах будет перемещен в файл с именем `common.bundle.js`. Увеличение значения позволяет уменьшить размер модуля `common.bundle.js`. И увеличивает размер других фрагментов.

Для билда серверного фронтенда используется другой файл с

конфигурацией webpack:

```
const webpack = require('webpack');
const path = require('path');
const nodeExternals = require('webpack-node-externals');
const externalFolder = new RegExp(`^${path.resolve(__dirname,
'../src')}/(react|redux)/.*$`)
const nodeEnv = process.env.NODE_ENV || 'development';
const isDevelopment = nodeEnv === 'development';

module.exports = {
  name: 'server',
  devtool: isDevelopment ? 'eval' : false,
  entry: './src/render.js',
  target: 'node',
  bail: !isDevelopment,
  externals: [
    nodeExternals(),
    function(context, request, callback) {
      if (request == module.exports.entry
        || externalFolder.test(path.resolve(context, request))){
        return callback();
      }
      return callback(null, 'commonjs2 ' + request);
     }
  ],
  output: {
    path: path.resolve(__dirname, '../src'),
    filename: 'render.bundle.js',
    libraryTarget: 'commonjs2',
  },
  module: {
    rules: [{
      test: /\.jsx?$/,
      exclude: [/node_modules/],
      use: "babel-loader?retainLines=true"
    }]
  }
};
```

Он значительно проще т.к. нам не нужно разбивать серверный код на фрагменты, а также обеспечивать поддержку старых версий браузеров (которые не поддерживают ES2017).

Опция `devtool: 'eval'` для режима разработчика показывает в сообщении об ошибке реальный файл и номер строки исходного кода.

Функция определяющая каталоги не входящие в билд:

```
const externalFolder = new RegExp(`^${path.resolve(__dirname,
'../src')}/(react|redux)/.*$`);
...
   function(context, request, callback) {
      if (request == module.exports.entry
        || externalFolder.test(path.resolve(context, request))){
        return callback();
      }
      return callback(null, 'commonjs2 ' + request);
     }
```

Предполагается что все модули кроме react и redux будут написаны с учетом возможностей node.js и не будут преобразовываться в legacy JavaScript.

Теперь рассмотрим код сервера, который может работать в режиме разработчика с hot reload, и в режиме продакшна:

```
'use strict';
const path = require('path');
const createServer = require('http').createServer;
const express = require('express');
const port = Number(process.env.PORT) || 3000;
const api = require('./src/api/routes');
const app = express();
const serverPath = path.resolve(__dirname,
'./src/render.bundle.js');
let render = require(serverPath);
let serverCompiler
```

```
const nodeEnv = process.env.NODE_ENV || 'development';
const isDevelopment = nodeEnv === 'development';
app.set('env', nodeEnv);

if (isDevelopment) {
  const webpack = require('webpack');
  serverCompiler = webpack([require('./webpack/config.server')]);
  const webpackClientConfig = require('./webpack/config.client');
  const webpackClientDevMiddleware = require('webpack-dev-
middleware');
  const webpackClientHotMiddleware = require('webpack-hot-
middleware');
  const clientCompiler = webpack(webpackClientConfig);
  app.use(webpackClientDevMiddleware(clientCompiler, {
    publicPath: webpackClientConfig.output.publicPath,
    headers: {'Access-Control-Allow-Origin': '*'},
    stats: {colors: true},
    historyApiFallback: true,
  }));
  app.use(webpackClientHotMiddleware(clientCompiler, {
    log: console.log,
    path: '/__webpack_hmr',
    heartbeat: 10 * 1000
  }));
  app.use('/static', express.static('dist'));
  app.use('/api', api);
  app.use('/', (req, res, next) => render(req, res, next));
} else {
  app.use('/static', express.static('dist'));
  app.use('/api', api);
  app.use('/', render);
}

app.listen(port, () => {
  console.log(`Listening at ${port}`);
});

if (isDevelopment) {
  const clearCache = () => {
    const cacheIds = Object.keys(require.cache);
    for (let id of cacheIds) {
      if (id === serverPath) {
        delete require.cache[id];
        return;
      }
    }
  }
```

```
  const watch = () => {
    const compilerOptions = {
      aggregateTimeout: 300,
      poll: 150,
    };
    serverCompiler.watch(compilerOptions, onServerChange);
    function onServerChange(err, stats) {
      if (err || stats.compilation && stats.compilation.errors &&
stats.compilation.errors.length) {
        console.log('Server bundling error:', err ||
stats.compilation.errors);
      }
      clearCache();
      try {
        render = require(serverPath);
      } catch (ex) {
        console.log('Error detecded', ex)
      }
      return;
    }
  }
  watch();
}
```

Если со слушателями изменения клиенской части фронтенда все понятно и хорошо описано в документации, то с серверной частью рендеринга я нашел решение в статье и немного упростил его. Суть такая, что в режиме разработчика функция рендеринга оборачивается другой функцией, которая вызывает всегда самый актуальный вариант функции рендеринга. При этом, после того как компилятор обнаруживает изменения в исходных файлах, происходит очстка кэша require и повторная загрузка скомпилирванного модуля:

```
    clearCache();
    try {
        render = require(serverPath);
    } catch (ex) {
        console.log('Error detecded', ex)
    }
```

Теперь при изменении исходного текста компонентов будет скомпилирована как серверная, так и клиентская часть кода, после чего компонент в браузере перегрузится. Параллельно перегрузится и код серверного рендеринга компонента.

Как это часто бывает, проделанная работа уперлась в непредвиденный момент. Code splitting это хорошо. Но как же ведет себя асинрхронно загружаемый компонент в реальной жизни? Увы, весь код роутинга и рендеринга React.js синхронный, и на время первой загрузки компонента отображается прелоадер (его можно сделать кастомным). Но для этого ли я все начинал? Все же решение нашлось. На основании стандартного компонента Link можно создать асинхронный компонента AsyncLink:

```
import React from "react";
import PropTypes from "prop-types";
import invariant from "invariant";
import { Link, matchPath } from 'react-router-dom';
import routes from './routes';

const isModifiedEvent = event =>
  !!(event.metaKey || event.altKey || event.ctrlKey ||
event.shiftKey);

class AsyncLink extends Link {
  handleClick = (event) => {
    if (this.props.onClick) this.props.onClick(event);
    if (
      !event.defaultPrevented && // onClick prevented default
      event.button === 0 && // ignore everything but left clicks
      !this.props.target && // let browser handle "target=_blank"
etc.
      !isModifiedEvent(event) // ignore clicks with modifier keys
    ) {
      event.preventDefault();
      const { history } = this.context.router;
```

```
      const { replace, to } = this.props;
      function locate() {
        if (replace) {
          history.replace(to);
        } else {
          history.push(to);
        }
      }
      if (this.context.router.history.location.pathname) {
        const route = routes.find((route) =>
matchPath(this.props.to, route) ? route : null);
        if (route) {
          import(`${String('./' +
route.componentName)}`).then(function() {locate();})
        } else {
          locate();
        }
      } else {
        locate();
      }
    }
  };
}
export default AsyncLink;
```

Вобщем все достаточно гладко после этого начало работать.

https://github.com/apapacy/uni-react

apapacy@gmail.com

14 февраля 2018 года

Комментарии (17)

Похожие публикации

Клонирование объектов в Node.js: Быстрее, глубже, нежнее! 15

wickedweasel • 28 декабря 2012 в 13:13

Node.js + Chromium = AppJS: один из перспективных вариантов второго шага эволюции веборазработчика 58

Mithgol • 1 октября 2012 в 16:04

Node.JS for Windows: node.exe 20

Emile_de_Sad • 4 августа 2011 в 00:23

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