React学习笔记（一）—— React几种基本配置方案

1.只使用React，不使用JSX

1)引入文件：react.js、react-dom.js

2)例子

```
<body>
    <div id="app"></div>
    <script>
        var HelloMessage = React.createClass ({
            displayName: "HelloMessage",
            render: function render () {
                return React.createElement("div", null, "Hello
",this.props.name);
            }
        });
        ReactDOM.render(React.createElement(HelloMessage, {name: "John"}),
document.getElementById("app"));
    </script>
</body>
```

2.通过browser.js转换JSX/ES 2015(非生产设置)

1)引入文件：react.js、react-dom.js、brower.js

2)例子

```
<body>
    <div id="app"></div>
    <script type="text/babel">
        const HelloMessage = React.createClass({
            render: function (){
                return <div>Hello {this.props.name}</div>;
            }
        });
        ReactDOM.render(<HelloMessage name="Jhon" />,
document.getElementById("app"));
    </script>
</body>
```

*注：*script元素设置type属性的值为type="text/babel"。当browser.js加载后将找到有关于type="text/babel"的脚本，并且将JSX/ES2015转换成ES5 JavaScript。

3.通过system.js/browser.js转换JSX/ES 2015(非生产设置)

1)引入文件：SystemJS

2)例子

通过jspm CDN加载SystemJS：

```
<script src="https://jspm.io/system@0.19.js"></script>
<script>
    System.config({
        transpiler: "babel",
        babelOptions: {}
    });
    System.import('./main.js');
</script>
```

在*main.js*文件中引入所需要依赖的文件：

```
import React from "react"
import ReactDOM from "react-dom"

const Hello = ({name}) => <h1>Hello {name}!</h1>

ReactDOM.render(
    <Hello name={"dude"} />,
    document.body.appendChild(document.createElement("div"))
)
```

4：使用在线编辑器创建React

5：在开发过程中使用Babel-cli和npm转换JSX/ES 2015

`step1:`确定安装了最新稳定版的node.js和npm，然后安装全局packages。

```
npm install webpack browser-sync -g
```

`step2:`创建目录和文件。

```
├── build
├── index.html
├── package.json
├── src
|   ├── app.js
|   ├── app.css
|   └── math.js
└── webpack.config.js
```

打开*package.json*文件，并且创建一个空的JSON对象：

```
{

}
```

`step3:`通过npm安装依赖关系。

在第二步创建的根目录下运行下面的npm命令安装所需要的依赖关系:

```
npm install babel-core babel-loader babel-preset-es2015 babel-preset-react
babel-preset-stage-0 browser-sync css-loader extract-text-webpack-plugin
style-loader webpack --save-dev

npm install react react-dom @telerik/kendo-react-buttons --save
```

运行上面两行命令将会安装必要的npm包。现在根目录中增加了一个*node_modules*文件夹，并且将需要的npm包都放在这个文件夹下：

```
├── build
├── index.html
├── node_modules
|   ├── @telerik
|   ├── babel-core
|   ├── babel-loader
|   ├── babel-preset-es2015
|   ├── babel-preset-react
|   ├── babel-preset-stage-0
|   ├── browser-sync
|   ├── css-loader
|   ├── extract-text-webpack-plugin
|   ├── react
|   ├── react-dom
|   ├── style-loader
|   └── webpack
├── package.json
├── src
|   ├── app.js
|   ├── app.css
|   └── math.js
└── webpack.config.js
```

`step4:`更新app.js，app.css，math.js和index.html。

打开*app.js*文件，并且添加下面的代码:

```
import React from 'react';
import ReactDOM from 'react-dom';
import * as KendoReactButtons from '@telerik/kendo-react-buttons';
import '@telerik/kendo-react-buttons/dist/npm/css/main.css';
import { square, diag } from './math.js';
import './app.css';

console.log(square(11)); // 121
console.log(diag(4, 3)); // 5

class ButtonContainer extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            disabled: false
        };
    }
    onClick = () => {
        this.setState({ disabled: !this.state.disabled });
    }
    render() {
```

```
        return (
            <div>
                <KendoReactButtons.Button onClick={this.onClick}>Button
1</KendoReactButtons.Button>
                <KendoReactButtons.Button disabled=
{this.state.disabled}>Button 2</KendoReactButtons.Button>
            </div>
        );
    }
}

ReactDOM.render(
    <div>
        <p>Button</p>
        <KendoReactButtons.Button>Button test</KendoReactButtons.Button>
        <p>Disabled Button</p>
        <KendoReactButtons.Button disabled>Button</KendoReactButtons.Button>
        <p>Primary Button</p>
        <KendoReactButtons.Button primary>Primary
Button</KendoReactButtons.Button>
        <p>Button with icon</p>
        <KendoReactButtons.Button
icon="refresh">Refresh</KendoReactButtons.Button>
        <p>Button with icon (imageUrl)</p>
        <KendoReactButtons.Button imageUrl="http://demos.telerik.com/kendo-
ui/content/shared/icons/sports/snowboarding.png">Snowboarding</KendoReactBut
tons.Button>
        <p>Button with a custom icon (iconClass) [FontAwesome icon]</p>
        <KendoReactButtons.Button iconClass="fa fa-key fa-fw">FontAwesome
icon</KendoReactButtons.Button>
        <p>Toggleable Button</p>
        <KendoReactButtons.Button togglable>Togglable
button</KendoReactButtons.Button>
        <p>onClick event handler</p>
        <ButtonContainer />
    </div>,
    document.getElementById('app')
);
```

打开*app.css*文件，并且添加下面的代码：

```
body{
    margin:50px;
}
```

打开*math.js*文件，并且添加下面的代码：

```
export const sqrt = Math.sqrt;

export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}
```

打开*index.html*文件，并且添加下面的代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Webpack</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-
awesome/4.5.0/css/font-awesome.min.css">
    <link rel="stylesheet" type="text/css" href="./build/style.css">
</head>
<body>
    <div id="app"></div>
    <script src="./build/appBundle.js"></script>
</body>
</html>
```

`step5:`更新webpack.config.js。

打开*webpack.config.js*文件，并且添加下面的代码：

```
var path = require('path');
var ExtractTextPlugin = require("extract-text-webpack-plugin");

module.exports = {
    entry: ['./src/app.js'],
    output: {
        path: path.resolve(__dirname, 'build'),
        filename: 'appBundle.js'
    },
    module: {
        loaders: [{
            test: /\.css$/,
            loader: ExtractTextPlugin.extract("style-loader", "css-loader")
        }, {
            loader: 'babel-loader',
            exclude: /node_modules/,
            test: /\.js$/,
            query: {
                presets: ['es2015', 'react', 'stage-0'],
            },
        }]
    },
    plugins: [
        new ExtractTextPlugin("style.css", {
            allChunks: true
        })
    ]
};
```

`step6:`更新package.json。

打开*package.json*文件，你可以看到文件中包括像下面这样的代码：

```
{
  "devDependencies": {
    "babel-core": "^6.9.0",
    "babel-loader": "^6.2.4",
    "babel-preset-es2015": "^6.9.0",
    "babel-preset-react": "^6.5.0",
    "babel-preset-stage-0": "^6.5.0",
    "browser-sync": "^2.12.8",
    "css-loader": "^0.23.1",
    "extract-text-webpack-plugin": "^1.0.1",
    "style-loader": "^0.13.1",
    "webpack": "^1.13.1"
  },
  "dependencies": {
    "@telerik/kendo-react-buttons": "^0.1.0",
    "react": "^15.1.0",
    "react-dom": "^15.1.0"
  }
}
```

并且在*package.json*文件中添加scripts相关配置：

```
{
    "scripts": {
        "webpack": "webpack --watch",
        "server": "browser-sync --port 4000 start --server --files
\"**/*.html\" \"build/**/*.css\" \"build/**/*.js\" "
    },
    "devDependencies": {
        "babel-core": "^6.9.0",
        "babel-loader": "^6.2.4",
        "babel-preset-es2015": "^6.9.0",
        "babel-preset-react": "^6.5.0",
        "babel-preset-stage-0": "^6.5.0",
        "browser-sync": "^2.12.8",
        "css-loader": "^0.23.1",
        "extract-text-webpack-plugin": "^1.0.1",
        "style-loader": "^0.13.1",
        "webpack": "^1.13.1"
    },
    "dependencies": {
        "@telerik/kendo-react-buttons": "^0.1.0",
        "react": "^15.1.0",
        "react-dom": "^15.1.0"
    }
}
```

`step7:` 运行webpack和server。

在你项目根目录下运行下面的npm命令：

```
npm run server
```

注： 如果正确遵循所有的步骤操作，那么Browsersync应该会打开浏览器在http://localhost:4000这个地址上加载index.html和app.js。Webpack和Browsersync会运行所做的修改。当然，这只是webpack的基本配置。

6：通过Webpack和Babel-core在开发过程中转换JSX/ES 2015（作者认为最好的方案）

`step1:` 确定安装了最新稳定版的node.js和npm，然后安装全局packages。

```
npm install jspm browser-sync -g
```

`step2:` 创建文件夹和文件。

```
├── app.js
├── index.html
├── js
|   └── math.js
├── package.json
└── style
    └── app.css
```

打开*package.json*文件，并且创建一个空的JSON对象：

```
{

}
```

`step3:` 安装npm依赖关系。

在根目录下运行下面的命令：

```
npm install jspm browser-sync --save-dev
```

运行上面命令将会安装必要的npm包。现在项目根目录下添加了node_modules和相应的npm包：

```
├── app.js
├── index.html
├── js
|   └── math.js
├── node_modules
|   ├── browser-sync
|   └── jspm
├── package.json
└── style
    └── app.css
```

`step4:`初始化SystemJS/JSPM设置。

在项目根目录下运行下面的jspm-cli命令：

```
jspm init
```

将会问你9个问题，每个问题你只需要按回车键就行了。 将会在根目录创建一个config.js文件和一个jspm_packagees文件夹。这时项目的目录结构像这样：

```
├── app.js
├── config.js
├── index.html
├── js
|   └── math.js
├── jspm_packages
|   ├── github
|   ├── npm
|   ├── system-csp-production.js
|   ├── system-csp-production.js.map
|   ├── system-csp-production.src.js
|   ├── system-polyfills.js
|   ├── system-polyfills.js.map
|   ├── system-polyfills.src.js
|   ├── system.js
|   ├── system.js.map
|   └── system.src.js
├── node_modules
|   ├── browser-sync
|   └── jspm
├── package.json
└── style
    └── app.css
```

打开*config.js*文件，并且更新babelOptions对象,将下面的代码：

```
babelOptions: {
    "optional": [
        "runtime",
        "optimisation.modules.system"
    ]
},
```

更新为：

```
babelOptions: {
    "optional": [
      "runtime",
      "optimisation.modules.system"
    ],
    "stage": 0
},
```

`step5:`更新app.js，app.css，math.js和index.html。

打开*app.js*文件，并且添加下面的代码：

```
import './style/app.css!'; //note, had to add the !
import React from 'react';
import ReactDOM from 'react-dom';
import * as KendoReactButtons from '@telerik/kendo-react-buttons';
import '@telerik/kendo-react-buttons/dist/npm/css/main.css!'; //note, had to
add the !
import { square, diag } from './js/math.js';

console.log(square(11)); // 121
console.log(diag(4, 3)); // 5

class ButtonContainer extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            disabled: false
        };
    }
    onClick = () => {
        this.setState({ disabled: !this.state.disabled });
    }
    render() {
        return (
            <div>
```

```
                <KendoReactButtons.Button onClick={this.onClick}>Button
1</KendoReactButtons.Button>
                <KendoReactButtons.Button disabled=
{this.state.disabled}>Button 2</KendoReactButtons.Button>
            </div>
        );
    }
}
ReactDOM.render(
    <div>
        <p>Button</p>
        <KendoReactButtons.Button>Button test</KendoReactButtons.Button>
        <p>Disabled Button</p>
        <KendoReactButtons.Button disabled>Button</KendoReactButtons.Button>
        <p>Primary Button</p>
        <KendoReactButtons.Button primary>Primary
Button</KendoReactButtons.Button>
        <p>Button with icon</p>
        <KendoReactButtons.Button
icon="refresh">Refresh</KendoReactButtons.Button>
        <p>Button with icon (imageUrl)</p>
        <KendoReactButtons.Button imageUrl="http://demos.telerik.com/kendo-
ui/content/shared/icons/sports/snowboarding.png">Snowboarding</KendoReactBut
tons.Button>
        <p>Button with a custom icon (iconClass) [FontAwesome icon]</p>
        <KendoReactButtons.Button iconClass="fa fa-key fa-fw">FontAwesome
icon</KendoReactButtons.Button>
        <p>Toggleable Button</p>
        <KendoReactButtons.Button togglable>Togglable
button</KendoReactButtons.Button>
        <p>onClick event handler</p>
        <ButtonContainer />
    </div>,
    document.getElementById('app')
);
```

打开*app.css*文件，并且添加下面的代码：

```
body{
    margin:50px;
}
```

打开*math.js*文件，并且添加下面的代码：

```
export const sqrt = Math.sqrt;

export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}
```

打开*index.html*文件，并且添加下面的代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>systemJS/jspm</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-
awesome/4.5.0/css/font-awesome.min.css">
</head>
<body>
    <div id="app"></div>
    <script src="jspm_packages/system.js"></script>
    <script src="config.js"></script>
    <script>
            System.import('app.js');
    </script>
</body>
</html>
```

`step6:`使用jspm-cli安装开发包。

在你项目根目录下运行下面的jspm-cli命令：

```
jspm install react react-dom css npm:@telerik/kendo-react-buttons
```

上面的命令将在jspm_packagees目录下安装React，react-dom，jspm css插件和Kendo UI React按钮。这些依赖关系将会放到package.json文件中。另外，jspm配置文件也将会对应更新，以便安装需要的安装包，而且不需要手动更新config.js文件。更新后的jspm_packagees目录结构看起来像下面这样：

```
├── jspm_packages
|   ├── github
|   |   ├── jspm
|   |   └── systemjs
```

```
|   ├── npm
|   |   ├── @telerik
|   |   ├── Base64@0.2.1
|   |   ├── Base64@0.2.1.js
|   |   ├── asap@2.0.3
|   |   ├── asap@2.0.3.js
|   |   ├── assert@1.3.0
|   |   ├── assert@1.3.0.js
|   |   ├── babel-core@5.8.38
|   |   ├── babel-core@5.8.38.js
|   |   ├── babel-runtime@5.8.38
|   |   ├── base64-js@0.0.8
|   |   ├── base64-js@0.0.8.js
|   |   ├── browserify-zlib@0.1.4
|   |   ├── browserify-zlib@0.1.4.js
|   |   ├── buffer@3.6.0
|   |   ├── buffer@3.6.0.js
|   |   ├── classnames@2.2.5
|   |   ├── classnames@2.2.5.js
|   |   ├── core-js@1.2.6
|   |   ├── core-js@1.2.6.js
|   |   ├── core-util-is@1.0.2
|   |   ├── core-util-is@1.0.2.js
|   |   ├── domain-browser@1.1.7
|   |   ├── domain-browser@1.1.7.js
|   |   ├── encoding@0.1.12
|   |   ├── encoding@0.1.12.js
|   |   ├── events@1.0.2
|   |   ├── events@1.0.2.js
|   |   ├── fbjs@0.6.1
|   |   ├── fbjs@0.6.1.js
|   |   ├── fbjs@0.8.2
|   |   ├── fbjs@0.8.2.js
|   |   ├── https-browserify@0.0.0
|   |   ├── https-browserify@0.0.0.js
|   |   ├── iconv-lite@0.4.13
|   |   ├── iconv-lite@0.4.13.js
|   |   ├── ieee754@1.1.6
|   |   ├── ieee754@1.1.6.js
|   |   ├── inherits@2.0.1
|   |   ├── inherits@2.0.1.js
|   |   ├── is-stream@1.1.0
|   |   ├── is-stream@1.1.0.js
|   |   ├── isarray@0.0.1
|   |   ├── isarray@0.0.1.js
|   |   ├── isarray@1.0.0
|   |   ├── isarray@1.0.0.js
|   |   ├── isomorphic-fetch@2.2.1
|   |   ├── isomorphic-fetch@2.2.1.js
|   |   ├── js-tokens@1.0.3
|   |   ├── js-tokens@1.0.3.js
```

```
|   |   ├── loose-envify@1.2.0
|   |   ├── loose-envify@1.2.0.js
|   |   ├── node-fetch@1.5.2
|   |   ├── node-fetch@1.5.2.js
|   |   ├── object-assign@4.1.0
|   |   ├── object-assign@4.1.0.js
|   |   ├── pako@0.2.8
|   |   ├── pako@0.2.8.js
|   |   ├── path-browserify@0.0.0
|   |   ├── path-browserify@0.0.0.js
|   |   ├── process-nextick-args@1.0.7
|   |   ├── process-nextick-args@1.0.7.js
|   |   ├── process@0.11.3
|   |   ├── process@0.11.3.js
|   |   ├── promise@7.1.1
|   |   ├── promise@7.1.1.js
|   |   ├── punycode@1.3.2
|   |   ├── punycode@1.3.2.js
|   |   ├── querystring@0.2.0
|   |   ├── querystring@0.2.0.js
|   |   ├── react-dom@0.14.8
|   |   ├── react-dom@0.14.8.js
|   |   ├── react-dom@15.0.2
|   |   ├── react-dom@15.0.2.js
|   |   ├── react@0.14.8
|   |   ├── react@0.14.8.js
|   |   ├── react@15.0.2
|   |   ├── react@15.0.2.js
|   |   ├── readable-stream@1.1.14
|   |   ├── readable-stream@1.1.14.js
|   |   ├── readable-stream@2.1.2
|   |   ├── readable-stream@2.1.2.js
|   |   ├── stream-browserify@1.0.0
|   |   ├── stream-browserify@1.0.0.js
|   |   ├── string_decoder@0.10.31
|   |   ├── string_decoder@0.10.31.js
|   |   ├── ua-parser-js@0.7.10
|   |   ├── ua-parser-js@0.7.10.js
|   |   ├── url@0.10.3
|   |   ├── url@0.10.3.js
|   |   ├── util-deprecate@1.0.2
|   |   ├── util-deprecate@1.0.2.js
|   |   ├── util@0.10.3
|   |   ├── util@0.10.3.js
|   |   ├── whatwg-fetch@1.0.0
|   |   └── whatwg-fetch@1.0.0.js
|   ├── system-csp-production.js
|   ├── system-csp-production.js.map
|   ├── system-csp-production.src.js
|   ├── system-polyfills.js
|   ├── system-polyfills.js.map
```

```
|   ├── system-polyfills.src.js
|   ├── system.js
|   ├── system.js.map
|   └── system.src.js
```

`step7:`更新package.json。

打开*package.json*文件，代码看起来像下面这样：

```
{
  "devDependencies": {
    "browser-sync": "^2.12.8",
    "jspm": "^0.16.35"
  },
  "jspm": {
    "dependencies": {
      "@telerik/kendo-react-buttons": "npm:@telerik/kendo-react-
buttons@^0.1.0",
      "css": "github:systemjs/plugin-css@^0.1.22",
      "react": "npm:react@^15.1.0",
      "react-dom": "npm:react-dom@^15.1.0"
    },
    "devDependencies": {
      "babel": "npm:babel-core@^5.8.24",
      "babel-runtime": "npm:babel-runtime@^5.8.24",
      "core-js": "npm:core-js@^1.1.4"
    }
  }
}
```

在*package.json*文件中添加scripts相关配置：

```
{
    "scripts": {
        "bundle": "jspm bundle app.js --inject",
        "unBundle": "jspm unbundle",
        "server": "browser-sync --port 4000 --no-inject-changes start --
server --files \"**/*.html\" \"style/**/*.css\" \"js/**/*.js\" "
    },
    "devDependencies": {
        "browser-sync": "^2.12.8",
        "jspm": "^0.16.35"
    },
    "jspm": {
        "dependencies": {
            "@telerik/kendo-react-buttons": "npm:@telerik/kendo-react-
buttons@^0.1.0",
            "css": "github:systemjs/plugin-css@^0.1.22",
            "react": "npm:react@^15.1.0",
            "react-dom": "npm:react-dom@^15.1.0"
        },
        "devDependencies": {
            "babel": "npm:babel-core@^5.8.24",
            "babel-runtime": "npm:babel-runtime@^5.8.24",
            "core-js": "npm:core-js@^1.1.4"
        }
    }
}
```

`step8:`运行服务器。

在你项目根目录下运行下面的jspm-cli命令：

```
npm run server
```

`step9:`捆绑模式。

SystemJS/jspm提供了一个捆绑模式。打开一个新的命令窗口，并且在项目根目录下运行下面的npm命令：

```
npm run bundle
```

如果要解除捆绑模式，可以执行下面的命令：

```
npm run unBundle
```