最短で学ぶReactとReduxの基礎から実践まで

やまもとじゅん

Version 0.1, 2018/12/22

目次

1. 環境構築とHello world

1.1. yarn のインストール

1.1.1. yarn とは

YarnはFacebook、Google、Exponent、Tildeによって開発された新しいJavaScriptパッケージマネージャー

- npm はインストールパッケージの速度および一貫性が不十分
- npmではパッケージがインストール時にコードを実行することを許可しているため、セキュリティー上の問題がある

yarmインストール @yarnpkg.com

1.1.2. yarnでインストールするパッケージのバージョンについて

```
axios@0.16.2
babel-core@6.25.0
babel-loader@7.1.1
babel-preset-es2015@6.24.1
babel-preset-react@6.24.1
css-loader@0.28.4
extract-text-webpack-plugin@3.0.0
geolib@2.0.22
import-glob-loader@1.1.0
lodash@4.17.4
node-sass@4.5.3
prop-types@15.5.10
query-string@5.0.0
react@15.6.1
react-dom@15.6.1
react-google-maps@7.2.0
react-redux@5.0.6
react-router-dom@4.1.2
redux@3.7.2
redux-devtools@3.4.0
redux-devtools-extension@2.13.2
redux-thunk@2.2.0
sass-loader@6.0.6
style-loader@0.18.2
webpack@3.3.0
webpack-dev-server@2.5.1
eslint@3.19.0
eslint-config-airbnb@15.0.2
eslint-plugin-import@2.7.0
eslint-plugin-jsx-a11y@5.1.1
eslint-plugin-react@7.1.0
```

1.1.3. yarn をインストールする

paccage.json を作成

```
yarn -v
yarn init
```

1.1.4. ES2015に必要なパッケージをインストールする

Udemyのコースが作成されたタイミングで利用されたバージョンを指定している

```
yarn add webpack@3.3.0
yarn add webpack-dev-server@2.5.1
yarn add babel-core@6.25.0
yarn add babel-loader@7.1.1
yarn add babel-preset-react@6.24.1
yarn add babel-preset-es2015@6.24.1
```

webpack-dev-server

開発サーバをローカルで動かすツール

bebel-*

Javascript のトランスコンパイラ

babel-preset-react

リアクトをコンパイルするためのBabelプリセット

babel-preset-es2015

ES2015 で書かれたソースコードをコンパイルするためのBabelプリセット

1.2. ES2015でのHello world

↓webpack.config.js

```
var publidDir = __dirname + '/public';
module.exports = {
  entry: [
    './src/index.js'
  ],
  output: {
    path: publidDir,
    publicPath: '/',
    filename: 'bundle.js'
  },
  module: {
    loaders: [{
      exclude: /node_modules/,
      loader: 'babel-loader',
      query: {
        presets: ['react', 'es2015']
      }
    }]
  },
  resolve: {
    extensions: ['.js', '.jsx']
  },
  devServer: {
    historyApiFallback: true,
    contentBase: publidDir
  }
};
```

↓public/index.js

```
<!DOCTYPE html>
<html lang="ja" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>udemy react</title>
  </head>
  <body>
    <div class="container">
        Hello World
    </div>
    <script src="bundle.js" charset="utf-8"></script>
  </body>
</html>
```

↓src/index.js

```
// とりあえず空
```

1.2.1. 開発サーバを起動する

```
./node_modules/.bin/webpack-dev-server
```

ソースコードが変更されると、自動的にコンパイル、更新までを自動的に行ってくれる。
実際にはファイルの実体を生成しない。

webpack.config.js の publicPath + filename にアクセスがあったとき、コンパイル結果を返す

コマンドを登録する

↓package.jsonに追記

```
"scripts": {
  "start" : "./node_modules/.bin/webpack-dev-server"
},
```

起動

```
yarn run start
```

1.2.2. ビルドを実行する

```
./node_modules/.bin/webpack
```

webpack.config.js の path + publicPath + filename にコンパイルしたファイルを生成する

1.3. Eslint のインストール

文法のチェックツール

```
yarn add eslint@3.19.0
yarn add eslint-plugin-react@7.1.0
```

1.3.1. 設定ファイルを作成する

```
./node_modules/.bin/eslint --init
```

NOTE	./node_modules/.bin/eslint --init を実行すると、./node_modules/.bin/eslint の実行ファイルが消えてしまい、次の操作でNo such file or directoryのエラーが発生する init実行後に$ yarn install を実行すると復活

なんか足らないようなので以下を実行

```
yarn add eslint-plugin-react@7.1.0
yarn add eslint-plugin-jsx-a11y@5.1.1
yarn add eslint-plugin-import@2.7.0
yarn add eslint-config-airbnb@15.0.2
yarn add circular-json@0.3.3
```

チェック

```
./node_modules/.bin/eslint src/index.js
```

Atomのパッケージと連携するとリアルタイムに検証してくれる

1.3.2. Atom のプラグイン

- es6-javascript
- intentions
- busy-signal
- linter
- linter-ui-default
- linter-eslint

NOTE	インストール後はリフレッシュする

1.4. SCSS を使えるようにする

```
yarn add node-sass(@4.5.3)
yarn add style-loader@0.18.2
yarn add css-loader@0.28.4
yarn add sass-loader@6.0.6
yarn add import-glob-loader@1.1.0
yarn add extract-text-webpack-plugin@3.0.0
```

↓webpack.config.js

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

const publidDir = path.join(__dirname, '/public');
module.exports = [
  {
    entry: [
```

```
      './src/index.js',
    ],
    output: {
      path: publidDir,
      publicPath: '/',
      filename: 'bundle.js',
    },
    module: {
      loaders: [{
        exclude: /node_modules/,
        loader: 'babel-loader',
        query: {
          presets: ['react', 'es2015'],
        },
      }],
    },
    resolve: {
      extensions: ['.js', '.jsx'],
    },
    devServer: {
      historyApiFallback: true,
      contentBase: publidDir,
    },
  },
  {
    entry: {
      style: './stylesheets/index.scss',
    },
    output: {
      path: publidDir,
      publicPath: '/',
      filename: 'bundle.css',
    },
    module: {
      loaders: [
        {
          test: /\.css$/,
          loader: ExtractTextPlugin.extract({ fallback: 'style-loader', use: 'css-
loader' }),
        },
        {
          test: /\.scss$/,
          loader: ExtractTextPlugin.extract({ fallback: 'style-loader', use: 'css-
loader!sass-loader' }),
        },
      ],
    },
    plugins: [
      new ExtractTextPlugin('bundle.css'),
    ],
  },
```

```
];
```

scss のために追加された entry, output に合わせて....

↓./stylesheets/index.scss

```
/* 一旦空 */
```

↓./public/indexhtml に追記

```
<link rel="stylesheet" href="bundle.css">
```

1.5. Reactを使ったHello world

```
yarn add react@15.6.1
yarn add react-dom@15.6.1
```

↓src/index.js

```
import React from 'react';
import ReactDom from 'react-dom';

ReactDom.render(<div>Hello React</div>, document.querySelector('.container'));
```

2. Reactの基礎

2.1. Component の作り方

↓/src/index.js → /src/index.jsx

```
import React from 'react';
import ReactDom from 'react-dom';
import App from './components/app';

ReactDom.render(<App />, document.querySelector('.container'));
```

webpack.config.js の /src/index.js → /src/index.jsx

2.1.1. ESLint を設定する

Atom にJSXを解釈させるプラグインを追加
language-javascript-jsx

document 等にチェックエラーが入るが、ブラウザなのでOK、という設定
↓.eslintrc.js に追加

```
"env": {
  "browser" : true
}
```

2.1.2. Functional Component

↓/src/components/app.jsx

```
import React from 'react';

function App(props){
  return (<div>Hello App</div>);
}

export default App;
```

2.1.3. Class Component

↓/src/components/app.jsx

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (<div>Hello Component</div>);
  }
}

export default App;
```

2.2. propsを用いたcomponent間の情報伝達

ステートレスなコンポーネントを作ってみる
↓/src/components/greeting.jsx

```
import React, { PropTypes } from 'react';

function Greeting(props) {
  return (<div>Hi, {props.name}</div>);
}

Greeting.propTypes = {
  name: PropTypes.string.isRequired,
};

export default Greeting;
```

↓/src/components/index.jsx

```
import React, { Component } from 'react';
import Greeting from './greeting';

class App extends Component {
  render() {
    return (<Greeting name="June" />);
  }
}

export default App;
```

2.3. stateを用いてcomponentに状態を持たせる

```
import React, { Component } from 'react';
import Greeting from './greeting';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Jhon',
    };
  }

  handleMouseOver() {
    this.setState({
      name: 'Bob',
    });
  }

  handleMouseOut() {
    this.setState({
      name: 'Jhon',
    });
  }

  render() {
    return (
      <div
        onMouseOver={() => this.handleMouseOver()}
        onMouseOut={() => this.handleMouseOut()}
      >
        <Greeting name={this.state.name} />
      </div>
    );
  }
}

export default App;
```

2.4. ユーザーの入力情報を取得する

```
import React, { Component } from 'react';
import Greeting from './greeting';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Jhon',
    };
  }

  handleChangeName(name) {
    this.setState({
      name,
    });
  }

  render() {
    return (
      <div>
        <input
          type="text"
          value={this.state.name}
          onChange={e => this.handleChangeName(e.target.value)}
        />
        <button onClick={() => this.handleChangeName('Bob')}>Bob</button>
        <Greeting name={this.state.name} />
      </div>
    );
  }
}

export default App;
```

3. Reactを使った実践的なWEBアプリケーションの作成

3.1. 構成

prace

入力された文字列

address

住所

lat

緯度

lng

経度

3.1.1. JSXでも補完を効かせるAtomプラグイン

emet

keymap.cson (File > keymap...) に追記

```
'atom-text-editor[data-grammar="source js jsx"]':
  'tab': 'emmet:expand-abbreviation-with-tab'
```

3.1.2. ファイルの命名規則

キャメルケース（ アッパーキャメルケース ）としているサンプルが多いようなのでそれに合わせる
app.jsx → App.jsx（index.jsxからの参照も修正する）

3.2. SearchForm コンポーネント

↓/component/SearchForm.jsx

```
import React, { Component, PropTypes } from 'react';

class SearchForm extends Component {
  constructor(props) {
    super(props);
    this.state = {
      place: '大阪',
    };
  }

  handlePlaceChange(place) {
    this.setState({
      place,
    });
  }

  handleSubmit(e) {
    e.preventDefault();
    this.props.onSubmit(this.state.place);
  }

  render() {
    return (
      <form onSubmit={e => this.handleSubmit(e)}>
        <input
          type="text"
          value={this.state.place}
          onChange={e => this.handlePlaceChange(e.target.value)}
        />
        <input type="submit" value="検索" />
      </form>
    );
  }
}

SearchForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
};

export default SearchForm;
```

↓/component/App.jsx

```
import React, { Component } from 'react';
import SearchForm from './SearchForm';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Jhon',
    };
  }

  handlePlaceSubmit(place) {
    console.log(place);
  }

  render() {
    return (
      <div>
        <h1>緯度経度検索</h1>
        <SearchForm onSubmit={place => this.handlePlaceSubmit(place)} />
      </div>
    );
  }
}

export default App;
```

3.3. GeocodeResultコンポーネント

```
import React, { PropTypes } from 'react';

const GeocodeResult = ({ address, lat, lng }) => (
  <ul className="geocode-result">
    <li>住所：{ address }</li>
    <li>緯度：{ lat }</li>
    <li>経度：{ lng }</li>
  </ul>
);

GeocodeResult.propTypes = {
  address: PropTypes.string,
  lat: PropTypes.number,
  lng: PropTypes.number,
};

GeocodeResult.defaultProps = {
  address: '',
  lat: 0,
  lng: 0,
};

export default GeocodeResult;
```

↓/component/App.jsx

```
import React, { Component } from 'react';
import SearchForm from './SearchForm';
import GeocodeResult from './GeocodeResult';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      address: '',
      lat: 0,
      lng: 0,
    };
  }

  handlePlaceSubmit(place) {
    console.log(place);
  }

  render() {
    return (
      <div>
        <h1>緯度経度検索</h1>
        <SearchForm onSubmit={place => this.handlePlaceSubmit(place)} />
        <GeocodeResult
          address={this.state.address}
          lat={this.state.lat}
          lng={this.state.lng}
        />
      </div>
    );
  }
}

export default App;
```

3.4. Google Geocoding API に問い合わせ、結果を表示する

3.4.1. axios ライブラリを追加する

ブラウザや node.js で動く Promise ベースのHTTPクライアントである。REST-API を実行したいときなど、これを使うと実装が簡単にできる。

```
yarn add axios@0.16.2
```

3.4.2. Google Geocoding API

Google Geocoding API

エンドポイント

outputFormat

outputFormat

json or xml

パタメタ:**address**

住所

パラメタ:**key**

APIキーが必須になった

↓/component/App.jsx

```
import axios from 'axios';
import React, { Component } from 'react';
import SearchForm from './SearchForm';
import GeocodeResult from './GeocodeResult';

const GEOCODE_ENDPOINT = 'https://maps.googleapis.com/maps/api/geocode/json';
const GOOGLE_MAP_APIKEY = 'AIzaSyCINYzcjOFN4ChmBlhWaWOsKwkA4UQeHn4';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      address: '',
      lat: 0,
      lng: 0,
    };
  }

  setErrorMessage(message) {
    this.setState({
      address: message,
      lat: 0,
      lng: 0,
    });
  }

  handlePlaceSubmit(place) {
    axios
      .get(GEOCODE_ENDPOINT, {
        params: {
          key: GOOGLE_MAP_APIKEY,
          address: place,
        },
      })
      .then((results) => {
        const data = results.data;
```

```
        const result = data.results[0];
        switch (data.status) {
          case 'OK': {
            const location = result.geometry.location;
            this.setState({
              address: result.formatted_address,
              lat: location.lat,
              lng: location.lng,
            });
            break;
          }
          case 'ZERO_RESULTS': {
            this.setErrorMessage('結果が見つかりませんでした');
            break;
          }
          default: {
            this.setErrorMessage('結果が見つかりませんでした');
          }
        }
      })
      .catch((error) => {
        // console.log(error);
        this.setErrorMessage('通信に失敗しました');
      });
  }

  render() {
    return (
      <div>
        <h1>緯度経度検索</h1>
        <SearchForm onSubmit={place => this.handlePlaceSubmit(place)} />
        <GeocodeResult
          address={this.state.address}
          lat={this.state.lat}
          lng={this.state.lng}
        />
      </div>
    );
  }
}

export default App;
```

3.4.3. Google Mapの導入

react-google-maps が無駄にややこしいので Static Map にした

↓/components/Map.jsx

```
import React, { PropTypes } from 'react';

const GOOGLE_MAP_APIKEY = 'AIzaSyCINYzcjOFN4ChmBlhWaWOsKwkA4UQeHn4';

const Map = ({ lat, lng, width, height, zoom }) => (
  <img
src={`https://maps.googleapis.com/maps/api/staticmap?center=${lat},${lng}&size=${width
}x${height}&zoom=${zoom}&key=${GOOGLE_MAP_APIKEY}`} alt="map" />
);

Map.propTypes = {
  lat: PropTypes.number,
  lng: PropTypes.number,
  width: PropTypes.number,
  height: PropTypes.number,
  zoom: PropTypes.number,
};

Map.defaultProps = {
  lat: 0,
  lng: 0,
  width: 400,
  height: 400,
  zoom: 18,
};

export default Map;
```

↓/components/Apps.jsx の render 部分

```
render() {
  return (
    <div>
      <h1>緯度経度検索</h1>
      <SearchForm onSubmit={place => this.handlePlaceSubmit(place)} />
      <GeocodeResult
        address={this.state.address}
        lat={this.state.lat}
        lng={this.state.lng}
      />
      <Map lat={this.state.lat} lng={this.state.lng} />
    </div>
  );
}
```

3.5. リファクタリング

- lat lng をまとめて location にする
- /src/components/App.jsx のややこしいトコを外だしして、シンプルに保つ

↓/src/domain/Geocoder.js

```
import axios from 'axios';

const GEOCODE_ENDPOINT = 'https://maps.googleapis.com/maps/api/geocode/json';
const GOOGLE_MAP_APIKEY = 'AIzaSyCINYzcjOFN4ChmBlhWaWOsKwkA4UQeHn4';

export const geocode = place =>
  axios
    .get(GEOCODE_ENDPOINT, {
      params: {
        key: GOOGLE_MAP_APIKEY,
        address: place,
      },
    })
    .then((results) => {
      const data = results.data;
      const status = data.status;
      const result = data.results[0];
      if (typeof result === 'undefined') {
        return { status };
      }

      const address = result.formatted_address;
      const location = result.geometry.location;
      return { status, address, location };
    });

export const staticMap = (location, width, height, zoom) =>
`https://maps.googleapis.com/maps/api/staticmap?center=${location.lat},${location.lng}
&size=${width}x${height}&zoom=${zoom}&key=${GOOGLE_MAP_APIKEY}`;
```

4. Component のライフサイクル