無料電子ブック

学習

React

Free unaffiliated eBook created from
Stack Overflow contributors.

#reactjs

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: react

It is an unofficial and free React ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official React.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Reactをいめる

Reactは、なコンポーネントベースのJavaScriptライブラリであり、ユーザーインターフェイスのにされます。

ReactのMVCフレームワークのようなをするために、はFluxフレーバー、えばReduxとみわせてします。

バージョン

バージョン	
0.3.0	2013-05-29
0.4.0	2013-07-17
0.5.0	20131016
0.8.0	2013-12-19
0.9.0	2014-02-20
0.10.0	2014-03-21
0.11.0	2014-07-17
0.12.0	2014-10-28
0.13.0	2015-03-10
0.14.0	2015-10-07
15.0.0	2016-04-07
15.1.0	2016520
15.2.0	2016-07-01
15.2.1	201678
15.3.0	2016-07-29
15.3.1	2016-08-19
15.3.2	2016-09-19
15.4.0	2016-11-16

バージョン	
15.4.1	2016-11-23
15.4.2	2017-01-06
15.5.0	2017-04-07
15.6.0	2017-06-13

Examples

インストールまたはセットアップ

ReactJSは、のHTMLページにめることができるのファイル`react-<version>.js`まれるJavaScriptライブラリです。 々はまた、、React DOMライブラリ`react-dom-<version>.js`をメインのReactファイルとともにインストールします。

なインクルージョン

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <script type="text/javascript" src="/path/to/react.js"></script>
    <script type="text/javascript" src="/path/to/react-dom.js"></script>
    <script type="text/javascript">
        // Use react JavaScript code here or in a separate file
    </script>
    </body>
</html>
```

JavaScriptファイルをするには、のReactドキュメントのインストールページにアクセスしてください 。

ReactはJSXもサポートしています 。 JSXはFacebookによってされたで、JavaScriptにXMLをしています。 JSXをするには、JSXをJavascriptコードにするために、Babelライブラリをみみ、`<script type="text/javascript">`を`<script type="text/babel">`にするがあります。

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <script type="text/javascript" src="/path/to/react.js"></script>
    <script type="text/javascript" src="/path/to/react-dom.js"></script>
    <script src="https://npmcdn.com/babel-core@5.8.38/browser.min.js"></script>
    <script type="text/babel">
        // Use react JSX code here or in a separate file
    </script>
    </body>
</html>
```

npmでインストールする

のようにしてnpmをってReactをインストールすることもできます

```
npm install --save react react-dom
```

JavaScriptプロジェクトでReactをするには、のをします。

```
var React = require('react');
var ReactDOM = require('react-dom');
ReactDOM.render(<App />, ...);
```

をってインストールする

FacebookはYarnというのパッケージマネージャをリリースしました。これはReactのインストールにもできます。 Yarnをインストールしたら、のコマンドをするだけです

```
yarn add react react-dom
```

Reactをnpmでインストールしたとまったくじで、プロジェクトでReactをすることができます。

Hello Worldコンポーネント

Reactコンポーネントは、ベースの`React.Component`クラスをするES6クラスとしてできます。のフォームでは、コンポーネントは、コンポーネントがDOMにレンダリング`render`を`render`メソッドをするがあります。 `render`メソッドはReactノードをします。これは、JSXをしてHTMLのようなタグとしてできます。のは、のComponentをするをしています。

```
import React from 'react'

class HelloWorld extends React.Component {
    render() {
        return <h1>Hello, World!</h1>
    }
}

export default HelloWorld
```

コンポーネントは、 `props`けることもできます。これらは、コンポーネントがることができないをするために、によってされるプロパティです。プロパティには、のイベントがしたにコンポーネントがびすことのできるをめることもできます。たとえば、ボタンがその`onClick`プロパティのをけり、クリックされるたびにびすことができます。コンポーネントをする、その`props`してアクセスすることができる`props`のオブジェクト。

```
import React from 'react'

class Hello extends React.Component {
    render() {
        return <h1>Hello, {this.props.name}!</h1>
    }
```

```
}

export default Hello
```

のは、コンポ—ネントがによって`name` propにされたのをレンダリングするをしています。コンポ—ネントはけったをできないことにしてください。

コンポ—ネントは、 `ReactDOM.render`をして、のコンポ—ネントでレンダリングすることも、`ReactDOM.render`を`ReactDOM.render`するコンポ—ネントとDOMノ—ドのをすることもできます。

```
import React from 'react'
import ReactDOM from 'react-dom'
import Hello from './Hello'

ReactDOM.render(<Hello name="Billy James" />, document.getElementById('main'))
```

ここまでで、なコンポ—ネントをし、 `props`をけれるをっています。これをさらにめ、 `state`をしましょう。

デモのために、Hello Worldアプリケ—ションをしましょう。フルネ—ムがえられているはののみをしてください。

```
import React from 'react'

class Hello extends React.Component {

    constructor(props){

        //Since we are extending the default constructor,
        //handle default activities first.
        super(props);

        //Extract the first-name from the prop
        let firstName = this.props.name.split(" ")[0];

        //In the constructor, feel free to modify the
        //state property on the current context.
        this.state = {
            name: firstName
        }

    } //Look maa, no comma required in JSX based class defs!

    render() {
        return <h1>Hello, {this.state.name}!</h1>
    }
}

export default Hello
```

コンポ—ネントはのをつことができます。または、そのコンポ—ネントがのをとしてけれることができます。

Codepenへのリンク。

こんにちは

JSXなし

ReactのメインAPIをしてReactをし、React DOM APIをしてReactをブラウザにするなをにします。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello React!</title>

    <!-- Include the React and ReactDOM libraries -->
    <script src="https://fb.me/react-15.2.1.js"></script>
    <script src="https://fb.me/react-dom-15.2.1.js"></script>

  </head>
  <body>
    <div id="example"></div>

    <script type="text/javascript">

      // create a React element rElement
      var rElement = React.createElement('h1', null, 'Hello, world!');

      // dElement is a DOM container
      var dElement = document.getElementById('example');

      // render the React element in the DOM container
      ReactDOM.render(rElement, dElement);

    </script>

  </body>
</html>
```

JSXで

からReactをするわりに、JSXJavaScriptでXMLをするためにFacebookによってされたJavascriptをすることができます。これにより、

```
var rElement = React.createElement('h1', null, 'Hello, world!');
```

のものとしてHTMLにしたにとってはみやすい

```
var rElement = <h1>Hello, world!</h1>;
```

JSXをむコードは、 `<script type="text/babel">`タグでむがあり`<script type="text/babel">`。 このタグのすべては、Babelライブラリ ReactライブラリにえてなライブラリをしてプレーンなJavascriptにされます。

に、のはのようになります。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello React!</title>

    <!-- Include the React and ReactDOM libraries -->
    <script src="https://fb.me/react-15.2.1.js"></script>
    <script src="https://fb.me/react-dom-15.2.1.js"></script>
    <!-- Include the Babel library -->
    <script src="https://npmcdn.com/babel-core@5.8.38/browser.min.js"></script>

  </head>
  <body>
   <div id="example"></div>

    <script type="text/babel">

      // create a React element rElement using JSX
      var rElement = <h1>Hello, world!</h1>;

      // dElement is a DOM container
      var dElement = document.getElementById('example');

      // render the React element in the DOM container
      ReactDOM.render(rElement, dElement);

    </script>

  </body>
</html>
```

ReactJSとはですか

ReactJSはオープンソースのコンポーネントベースのフロントエンドライブラリで、アプリケーションのビューレイヤのみをします。それはFacebookによってされています。

ReactJSは、DOMベースのメカニズムをしてHTML DOMのデータビューをめみます。DOMは、なDOMをリロードするのではなく、々のDOMのみをするというをしてにします

> Reactアプリケーションはのコンポーネントでされ、それぞれがさなHTMLをします。コンポーネントはのコンポーネントにネストすることができ、なアプリケーションをなビルディングブロックからすることができます。コンポーネントはをすることもできます。たとえば、TabListコンポーネントには、いているタブにするがされます。

Reactは、JSXというドメインのをってコンポーネントをくことができます。 JSXでは、JavaScriptイベントでミキシングしながら、HTMLをしてコンポーネントをすることができます。Reactはこれをに DOMにし、にHTMLをします。

Reactは、DOMをして、コンポーネントのをかつにHTMLアプリケーションのDOMにレンダリン

グするためにします。DOMは、のDOMのメモリです。 Reactは、ブラウザーのDOMでするのではなく、DOMでほとんどのをうことで、ちにし、のレンダリングサイクルがしてからされたコンポーネントの、、およびのみをいます。

ステートレスをつHello World

ステートレスコンポーネントは、プログラミングからのをています。これはのことをしますは、えられたものとじものをにします。

えば

```
const statelessSum = (a, b) => a + b;

let a = 0;
const statefulSum = () => a++;
```

のからわかるように、statelessSumはにaとbとじをします。ただし、statefulSumはパラメータをしなくてもじをしません。このタイプののは、 ともばれます 。、コンポーネントはそれのかにします。

したがって、ステートレスコンポーネントは、 がなく 、にじをするため、よりにステートレスコンポーネントをすることをおめします。これは、あなたのアプリケーションであなたがにしたいものです。なぜなら、するは、なプログラムののシナリオなのでです。

のもなタイプは、なしのものである。それらののなであり、をとしないコンポーネントは、なJavaScriptとしてすることができます。これらは、 `Stateless Functional Components`あるとわれてい`Stateless Functional Components`なぜなら、これは、 `state`をすることなく、 `props`だけであるからです。

`Stateless Functional Component`をすなをにし`Stateless Functional Component`。

```
// In HTML
<div id="element"></div>

// In React
const MyComponent = props => {
    return <h1>Hello, {props.name}!</h1>;
};

ReactDOM.render(<MyComponent name="Arun" />, element);
// Will render <h1>Hello, Arun!</h1>
```

このコンポーネントがうすべては、 `name` propをむ`h1`をレンダリングすることにしてください。このコンポーネントは、どのもしません。ここにES6のもあります

```
import React from 'react'
```

```
const HelloWorld = props => (
    <h1>Hello, {props.name}!</h1>
)

HelloWorld.propTypes = {
    name: React.PropTypes.string.isRequired
}

export default HelloWorld
```

これらのコンポーネントはをするためにバッキングインスタンスをとしないため、Reactはのがあります。はクリーンですが、ステートレスコンポーネントのはまだされていません 。

リアクションアプリケーションの

create-react-appはFacebookによってされたReactアプリのツールです。のようなのでいやすいようにされたをします。

- ES6とJSXの
- ホットモジュールリロードによるDevサーバ
- コードlinting
- CSSプレフィックス
- JS、CSS、イメージバンドリング、およびソースマップによるスクリプトのビルド
- Jestテストフレームワーク

インストール

まず、ノードパッケージマネージャnpmをしてcreate-react-appをグローバルにインストールします。

```
npm install -g create-react-app
```

に、したディレクトリでジェネレータをします。

```
create-react-app my-app
```

しくされたディレクトリにし、スクリプトをします。

```
cd my-app/
npm start
```

create-react-appは、デフォルトでにできません。 SassなどのコンパイルみCSSをするなど、デフォルトのがなは、ejectコマンドをできます。

```
npm run eject
```

これにより、すべてのファイルをできます。これはなプロセスです。

のはのとおりです。

- エンクレーブ
- nwb
- モーション
- ラック·クリ
- ブドゥー
- rwb
- クイック
- サギ
- ロール

リアクションアプリケーションをする

プロダクションのアプリケーションをビルドするには、のコマンドをします

```
npm run build
```

なコンポーネントをするためのな

コンポーネントと

Reactはアプリケーションのにのみわっているため、Reactののはコンポーネントのです。コンポーネントは、アプリケーションのビューのをします。 "Props"はにJSXノードでされるです `<SomeComponent someProp="some prop's value" />`。アプリケーションがコンポーネントとやりとりするなです。のコードでは、SomeComponentのに、々は、アクセスだろう`this.props`オブジェクトになり、 `{someProp: "some prop's value"}`。

Reactコンポーネントをシンプルなとえることはです。らは「」のでし、をマークアップとしてします。くのシンプルなコンポーネントはこれをさらにさせ、「な」をります。つまり、をこさず、ですがある、コンポーネントはにじをします。このは、にコンポーネントを「クラス」ではなくとしてすることによってにすることができます。 Reactコンポーネントをするは3つあります。

- 「ステートレス」コンポーネント

```
const FirstComponent = props => (
    <div>{props.content}</div>
);
```

- **React.createClass**

```
const SecondComponent = React.createClass({
    render: function () {
        return (
```

```
            <div>{this.props.content}</div>
        );
    }
});
```

- **ES2015**クラス

```
class ThirdComponent extends React.Component {
    render() {
        return (
            <div>{this.props.content}</div>
        );
    }
}
```

これらのコンポ—ネントはまったくじでされます。

```
const ParentComponent = function (props) {
    const someText = "FooBar";
    return (
        <FirstComponent content={someText} />
        <SecondComponent content={someText} />
        <ThirdComponent content={someText} />
    );
}
```

のはすべてじマ—クアップをします。

コンポ—ネントは、そのに「」をつことはできません。したがって、コンポ—ネントにがなは、クラスベ—スのコンポ—ネントにします。については、「コンポ—ネントの」をしてください。

のとして、は、するとできません。つまり、コンポ—ネントからすることはできません。コンポ—ネントのがのをした、Reactハンドルはいをしいものにきえ、コンポ—ネントはしいをしてレンダリングします。

とコンポ—ネントとのについては、「ReactableとReusable Componentsでえる」をしてください。

オンラインでReactをいめるをむ https://riptutorial.com/ja/reactjs/topic/797/reactをいめる

2: AJAXびしにする

Examples

HTTP GETリクエスト

によっては、コンポーネントがリモートエンドポイントREST APIなどからデータをレンダリングするがあるがあります。 なは、 `componentDidMount`メソッドでそのようなびしをうことです。

ここでは、 スーパーエージェントをAJAXヘルパーとしてするをします。

```
import React from 'react'
import request from 'superagent'

class App extends React.Component {
  constructor () {
    super()
    this.state = {}
  }
  componentDidMount () {
    request
      .get('/search')
      .query({ query: 'Manny' })
      .query({ range: '1..5' })
      .query({ order: 'desc' })
      .set('API-Key', 'foobar')
      .set('Accept', 'application/json')
      .end((err, resp) => {
        if (!err) {
          this.setState({someData: resp.text})
        }
      })
  },
  render() {
    return (
      <div>{this.state.someData || 'waiting for response...'}</div>
    )
  }
}

React.render(<App />, document.getElementById('root'))
```

は、 `request`オブジェクトにしてなメソッドをびし、に`.end()`をびしてをすることによってできます。ヘッダーフィールドをするのはです。フィールドとを`.set()`をびします。

`.query()`メソッドはオブジェクトをけります。オブジェクトはGETメソッドとにするとクエリをします。はパス`/search?query=Manny&range=1..5&order=desc`をします。

POSTリクエスト

```
request.post('/user')
  .set('Content-Type', 'application/json')
```

```
  .send('{"name":"tj","pet":"tobi"}')
  .end(callback)
```

は、 Superagentのドキュメントをしてください。

なしのReactのAjax - VanillaJS

はIE9 +でします

```
import React from 'react'

class App extends React.Component {
  constructor () {
    super()
    this.state = {someData: null}
  }
  componentDidMount () {
    var request = new XMLHttpRequest();
    request.open('GET', '/my/url', true);

    request.onload = () => {
      if (request.status >= 200 && request.status < 400) {
        // Success!
        this.setState({someData: request.responseText})
      } else {
        // We reached our target server, but it returned an error
        // Possibly handle the error by changing your state.
      }
    };

    request.onerror = () => {
      // There was a connection error of some sort.
      // Possibly handle the error by changing your state.
    };

    request.send();
  },
  render() {
    return (
      <div>{this.state.someData || 'waiting for response...'}</div>
    )
  }
}

React.render(<App />, document.getElementById('root'))
```

HTTP GETリクエストとデータのループ

のは、リモートソースからしたのデータをコンポーネントにレンダリングするをしています。

ほとんどのブラウザにみまれているfetchをしてAJAXリクエストをします。プロダクションではいブラウザをサポートするためにpolyfillをfetchます。あなたはまた、えばをうためののライブラリをすることができますaxios 、 たSuperAgent 、あるいはプレーンなJavascriptを。

けったデータはコンポーネントのとしてするので、renderメソッドでアクセスできます。そこで

、 `map`をしてデータをループし`map`。 Reactのレンダリングパフォーマンスにとってな、ループしたににの`key` またはをすることをれないでください。

```
import React from 'react';

class Users extends React.Component {
  constructor() {
    super();
    this.state = { users: [] };
  }

  componentDidMount() {
    fetch('/api/users')
      .then(response => response.json())
      .then(json => this.setState({ users: json.data }));
  }

  render() {
    return (
      <div>
        <h1>Users</h1>
        {
          this.state.users.length == 0
            ? 'Loading users...'
            : this.state.users.map(user => (
              <figure key={user.id}>
                <img src={user.avatar} />
                <figcaption>
                  {user.name}
                </figcaption>
              </figure>
            ))
        }
      </div>
    );
  }
}

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

JSBinの

オンラインでAJAXびしにするをむ https://riptutorial.com/ja/reactjs/topic/6432/ajaxびしにする

3: FluxによるReactJSのき

られたと、のをよりにするためのしいコーードのために、フロントエンドのReactJSをするアプリケーーションのがされているとき、Fluxアプローーチをすることはにです。

Fluxは、Facebookがクライアントサイドwebアプリケーーションをするためにするアプリケーーションアーーキテクチャです。これは、デーータフローーをしてReactのなビューーコンポーーネントをします。これはなフレームワークではなく、パターーンであり、しいコーードをくわずにFluxをすぐにすることができます。

Fluxアプリケーーションには、ディスパッチャ 、 ストア 、およびビュー リアクションコンポーーネントの3つのがあります 。これらをModel-View-Controllerとしてはいけません。コントローーラはFluxアプリケーーションにしますが、コントローーラのビューーです。のにあるビューーは、ストアからデーータをし、そのデーータをにすことがよくあります。さらに、アクション - ディスパッチャーーヘルパーーメソッド - は、アプリケーーションでなすべてのをするセマンティックAPIをサポーートするためにされます。 Fluxのサイクルの4のとえることはです。

Fluxは、デーータフローーをして**MVC**をします。ユーーザーーがReactビューーとすると、ビューーはディスパッチャをしてアクションを、アプリケーーションのデーータとビジネスロジックをするさまざまなストアにし、をけるすべてのビューーをします。これは、Reactのプログラミングスタイルでにです。これは、ストアがでビューーをさせるをせずにをできるようにします。

Examples

デーータフローー

これはなのです。

フラックスパターーンは、デーータフローーのをとしています。

1. アクション - アクション`type`とそののデーータをするなオブジェクト。
2. ディスパッチャーー - シングルアクションレシーーバーーとコーールバックコントローーラーー。それがアプリケーーションのなハブであるとします。
3. **Store** - アプリケーーションのとロジックがまれます。ディスパッチャにコーールバックをし、デーータレイヤへのがしたときにするイベントをします。
4. ビューー - ストアからのイベントとデーータをけるコンポーーネントをさせます。かがされたときにレンダリングがします。

Fluxデータフローので、ビューはアクションをし、ディスパッチャーにしてユーザーのをうこともできます。

にす

それをよりにするために、たちはからめることができます。

- なるReactコンポーネント ビュー は、なるストアからをえたデータをします。

  コントローラービューとばれるコンポーネントはほとんどありません。ストアからデータをし、データをのチェーンにすためのグルーコードをするためです。コントローラビューは、ページのなをします。

- ストアは、アプリケーションのビジネスロジックのアクションタイプとのデータをするコールバックとしてされます。

- ディスパッチャーは、アクション·レシーバーとコールバック·コンテナーです。

- アクションは、の`type`プロパティをつなオブジェクトだけです。

  は、アクションタイプとヘルパーメソッド アクション クリエイターとばれるにをしたいとうでしょう。

オンラインでFluxによるReactJSのをむ https://riptutorial.com/ja/reactjs/topic/8158/fluxによるreactjsの

4: jQueryでのReactJSの

Examples

ReactJSとjQuery

まず、jqueryライブラリをインポートするがあります。また、DOMをするときにfindDOmNodeをインポートするもあります。もちろん、Reactもインポートしています。

```
import React from 'react';
import { findDOMNode } from 'react-dom';
import $ from 'jquery';
```

アイコンをクリックするとする「handleToggle」をしています。アイコンをクリックすると、を'toggle'にしてdivをしてすだけです。

```
handleToggle = () => {
    const el = findDOMNode(this.refs.toggle);
    $(el).slideToggle();
};
```

を 'toggle'にしましょう

```
<ul className="profile-info additional-profile-info-list" ref="toggle">
  <li>
    <span className="info-email">Office Email</span>   me@shuvohabib.com
  </li>
</ul>
```

onClickで 'handleToggle'をさせるdiv。

```
 <div className="ellipsis-click" onClick={this.handleToggle}>
   <i className="fa-ellipsis-h"/>
 </div>
```

のなコードをてみましょう。

```
import React from 'react';
import { findDOMNode } from 'react-dom';
import $ from 'jquery';

export default class FullDesc extends React.Component {
    constructor() {
        super();
    }

    handleToggle = () => {
        const el = findDOMNode(this.refs.toggle);
        $(el).slideToggle();
```

```
    };

    render() {
        return (
            <div className="long-desc">
                <ul className="profile-info">
                    <li>
                        <span className="info-title">User Name : </span> Shuvo Habib
                    </li>
                </ul>

                <ul className="profile-info additional-profile-info-list" ref="toggle">
                    <li>
                        <span className="info-email">Office Email</span> me@shuvohabib.com
                    </li>
                </ul>

                <div className="ellipsis-click" onClick={this.handleToggle}>
                    <i className="fa-ellipsis-h"/>
                </div>
            </div>
        );
    }
}
```

たちはですこれが、 **React**コンポーネントで**jQuery**をするです。

オンラインでjQueryでのReactJSのをむ https://riptutorial.com/ja/reactjs/topic/6009/jqueryでのreactjsの

5: JSX

JSXは、JavaScriptにXMLをするプリプロセッサ-ステップです。 ReactをJSXなしでうことはいありませんが、JSXはReactをよりエレガントにします。

XMLとに、JSXタグにはタグ、、があります。がでまれている、はです。それのは、をでみます。はまれたJavaScriptです。

に、JSXは`React.createElement(component, props, ...children)`になをするだけです。

したがって、のJSXコーード

```
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}

ReactDOM.render(<HelloMessage name="Kalo" />, mountNode);
```

のJavaScriptコーードにコンパイルします。

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
}

ReactDOM.render(React.createElement(HelloMessage, { name: "Kalo" }), mountNode);
```

として、 **JSX**ののはでも**HTML**でもないことにしてください。

```
const element = <h1>Hello, world!</h1>;
```

これはJSXとばれ、 **JavaScript**のです。 JSXはテンプレートをいさせるかもしれませんが、JavaScriptのをえています。

Reactチームは、らのドキュメントで、UIをどのようにせるべきかをすることをしています。

Examples

JSXの

JSXでをするはいくつかあります。

JavaScript

の**JavaScript**を`{}`んでとしてすことができます。たとえば、このJSXでは

```
<MyComponent count={1 + 2 + 3 + 4} />
```

`MyComponent`では、`1 + 2 + 3 + 4`がされるため、`props.count`のは`10`になります。

IfとforループはJavaScriptのではないため、JSXですることはできません。

リテラル

もちろん、の`string literal`を`prop`としてすこともできます。これらの2つのJSXはです。

```
<MyComponent message="hello world" />

<MyComponent message={'hello world'} />
```

リテラルをすと、そのはHTMLアンエスケープされます。したがって、これらの2つのJSXはです。

```
<MyComponent message="&lt;3" />

<MyComponent message={'<3'} />
```

このはではありません。これは、のためにここにされています。

デフォルト

にをさないと、`デフォルト`は**`true`**になり**`true`**。これらの2つのJSXはです。

```
<MyTextBox autocomplete />

<MyTextBox autocomplete={true} />
```

しかし、Reactチームは、`このアプローチをしているドキュメントでは、{foo: true}`ではなく`{foo: foo}`であるES6オブジェクトの`{foo}`とするがあるためしていません。らは、このがちょうどそこにあり、HTMLのにマッチするといます。

スプレッドアトリビュート

すでにオブジェクトとしてをっていてJSXですは、スプレッドとして...をしてオブジェクトをすことができます。これらの2つのコンポーネントはです。

```
function Case1() {
  return <Greeting firstName="Kaloyab" lastName="Kosev" />;
}

function Case2() {
  const person = {firstName: 'Kaloyan', lastName: 'Kosev'};
  return <Greeting {...person} />;
}
```

JSXのたち

タグとタグのをむJSXでは、これらのタグのコンテンツはなprop `props.children`としてされます。をすにはいくつかのがあります

リテラル

あなたはタグとタグのにをくことができ、 `props.children`はそのになります。これは、みみのHTMLのくにちます。えば

```
<MyComponent>
    <h1>Hello world!</h1>
</MyComponent>
```

これはなJSXで、MyComponentの`props.children`はに`<h1>Hello world!</h1>`ます。

HTMLはエスケープされていないので、は**HTML**とじようにJSXをくことができます。

これをにいて、この、JSX

- のとのをします。
- をします。
- タグにするしいがされます。
- リテラルのでするしいは、のスペースにされます。

JSXチルドレン

として、よりくのJSXをすることができます。これはネストされたコンポーネントをするのにです

```
<MyContainer>
  <MyFirstComponent />
  <MySecondComponent />
</MyContainer>
```

さまざまなタイプのをさせることができるので、JSXのとにリテラルをできます。これは、JSXがHTMLのようなもので、なJSXとなHTMLのであるのです。

```
<div>
  <h2>Here is a list</h2>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
</div>
```

Reactコンポーネントはの**React**をすことはできませんが、の**JSX**はのをつことができます。したがって、のをレンダリングするは、ののように`div`ラップすることができます。

JavaScript

のJavaScriptを`{}`にんでとしてすことができます。たとえば、これらのはです。

```
<MyComponent>foo</MyComponent>

<MyComponent>{'foo'}</MyComponent>
```

これは、のさのJSXのリストをレンダリングするにです。たとえば、のようにHTMLリストをレンダリングします。

```
const Item = ({ message }) => (
  <li>{ message }</li>
);

const TodoList = () => {
  const todos = ['finish doc', 'submit review', 'wait stackoverflow review'];
  return (
    <ul>
      { todos.map(message => (<Item key={message} message={message} />)) }
    </ul>
  );
};
```

JavaScriptはのタイプのとすることができます。

としての

、JSXにされたJavaScriptは、、React、またはそれらのリストにされます。しかし、`props.children`、Reactがレンダリングするをっているだけでなく、あらゆるのデ―タをすことができるで、のとじようにします。たとえば、カスタムコンポ―ネントをしている、`props.children`としてコ―ルバックをることができます。

```
const ListOfTenThings = () => (
  <Repeat numTimes={10}>
    {(index) => <div key={index}>This is item {index} in the list</div>}
  </Repeat>
);

// Calls the children callback numTimes to produce a repeated component
const Repeat = ({ numTimes, children }) => {
  let items = [];
  for (let i = 0; i < numTimes; i++) {
    items.push(children(i));
  }
  return <div>{items}</div>;
};
```

Reactがレンダリングするにできるものにされるり、カスタムコンポ―ネントにされたはでもかまいません。このいはではありませんが、JSXのをしたいにします。

される

`false`、 `null`、 `undefined`、および`true`がなであることにしてください。しかし、らはにレンダリングしません。これらのJSXはすべてじものにレンダリングされます。

```
<MyComponent />

<MyComponent></MyComponent>

<MyComponent>{false}</MyComponent>

<MyComponent>{null}</MyComponent>

<MyComponent>{true}</MyComponent>
```

これは、Reactをきでレンダリングするのににです。このJSXは、`showHeader`がtrueのにのみaをレンダリングします。

```
<div>
  {showHeader && <Header />}
  <Content />
</div>
```

1つのなは、`0`のような「の」のいくつかはとしてReactによってレンダリングされるということです。たとえば、このコ―ドは、`props.messages`がののに`0`がされるため、どおりにしません。

```
<div>
  {props.messages.length &&
    <MessageList messages={props.messages} />
  }
</div>
```

これをするの1つは、 `&&`のがにブーールであることをすることです。

```
<div>
  {props.messages.length > 0 &&
    <MessageList messages={props.messages} />
  }
</div>
```

に、 `false` 、 `true` 、 `null` 、または`undefined`ようなをにするには、ににするがあることにしてください。

```
<div>
  My JavaScript variable is {String(myVariable)}.
</div>
```

オンラインでJSXをむ https://riptutorial.com/ja/reactjs/topic/8027/jsx

6: React Boilerplate [リアクション+バベル+ウェブパック]

Examples

プロジェクトの

プロジェクトのをインストールするには、ノードパッケージマネージャがです。 Nodejs.orgからオペレーティングシステムのノードをダウンロードしてください。 Node Package Managerにはノードがしています。

ノード·バージョン·マネージャをして、ノードとnpmバージョンのをすることもできます。なるノードのバージョンでプロジェクトをテストするのにです。ただし、にはされません。

システムにノードをインストールしたら、まずバベルとウェブパックをってのReactプロジェクトをするためのパッケージをインストールしてください。

にでコマンドをちめるに。 BabelとWebpackがわれているものをてみましょう。

で`npm init`をすると、プロジェクトをできます。にいます。その、のコマンドをでします。

```
npm install react react-dom --save
```

Dev Dependecies

```
npm install babel-core babel-loader babel-preset-es2015 babel-preset-react babel-preset-stage-0
webpack webpack-dev-server react-hot-loader --save-dev
```

オプションの**Dev**

```
npm install eslint eslint-plugin-react babel-eslint --save-dev
```

このサンプル package.jsonをすることができます

ののプロジェクトルートに`.babelrc`をします。

```
{
  "presets": ["es2015", "stage-0", "react"]
}
```

にじて、ののプロジェクトルートに`.eslintrc`をします。

```
{
  "ecmaFeatures": {
    "jsx": true,
    "modules": true
  },
  "env": {
```

```
    "browser": true,
    "node": true
  },
  "parser": "babel-eslint",
  "rules": {
    "quotes": [2, "single"],
    "strict": [2, "never"],
  },
  "plugins": [
    "react"
  ]
}
```

されたファイルをgitリポジトリにアップロードしないように`.gitignore`ファイルをします。

```
node_modules
npm-debug.log
.DS_Store
dist
```

のの`webpack.config.js`ファイルをします。

```
var path = require('path');
var webpack = require('webpack');

module.exports = {
  devtool: 'eval',
  entry: [
    'webpack-dev-server/client?http://localhost:3000',
    'webpack/hot/only-dev-server',
    './src/index'
  ],
  output: {
    path: path.join(__dirname, 'dist'),
    filename: 'bundle.js',
    publicPath: '/static/'
  },
  plugins: [
    new webpack.HotModuleReplacementPlugin()
  ],
  module: {
    loaders: [{
      test: /\.js$/,
      loaders: ['react-hot', 'babel'],
      include: path.join(__dirname, 'src')
    }]
  }
};
```

に、 `sever.js`ファイルをして、のの`npm start`をできるようにします。

```
var webpack = require('webpack');
var WebpackDevServer = require('webpack-dev-server');
var config = require('./webpack.config');

new WebpackDevServer(webpack(config), {
  publicPath: config.output.publicPath,
```

```
  hot: true,
  historyApiFallback: true
}).listen(3000, 'localhost', function (err, result) {
  if (err) {
    return console.log(err);
  }

  console.log('Serving your awesome project at http://localhost:3000/');
});
```

あなたのReactプロジェクトがかをするのをるために`src/app.js`ファイルを`src/app.js`ます。

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

export default class App extends Component {
  render() {
    return (
      <h1>Hello, world.</h1>
    );
  }
}
```

`package.json start`が`start`すのかをしていれば、ターミナルで`node server.js`または`npm start`します。

スターターキットプロジェクト

このプロジェクトについて

これはなプロジェクトです。このは、ReactJs + Webpack + Bableのをするためのガイドです。

めましょう

たちは、fire up expressサーバーのノード・パッケージ・マネージャーをとし、プロジェクトのをします。ノード・パッケージ・マネージャーをめてごのは、 ここでできます 。ノード・パッケージ・マネージャーのインストールはここでです。

なのフォルダをし、ターミナルまたはGUIからナビゲートします。ターミナルにき、 `npm init`とすると、package.jsonファイルがされます。もないので、プロジェクト、バージョン、エントリポイント、gitリポジトリ、、ライセンスなどがあります。エントリポイントは、プロジェクトをするときにノードがにすためです。に、あなたがするをするようめられます。 *yes*とするか、またはすることができます 。これで、 *package.json*ファイルがです。

Expressサーバーのセットアップ *npm install express @ 4 - save*をします 。これはこのプロジェクトになすべてのです。ここではフラグはですが、 *package.js*ファイルはされません。 *package.json*のなタスクは、のリストをすることです。あなたの*package.json*は`"dependencies": { "express": "^4.13.4", ..... },`ようにえます`"dependencies": { "express": "^4.13.4", ..... },`

なダウンロードの、 *node_modules*フォルダとのサブフォルダがあることがわかります。プロジ

エクトのルートにしいファイル*server.js*ファイルをします。はエクスプレスサーバーをしています。はすべてのコードをわらせ、でします。

```
var express = require('express');
// Create our app
var app = express();

app.use(express.static('public'));

app.listen(3000, function () {
  console.log('Express server is using port:3000');
});
```

*var express = require 'express';*これはあなたにAPIのアクセスをえます。

*var app = express;*としてエクスプレスライブラリをびします。 *app.use;*エクスプレスアプリケーションにをさせてください。 *app.useexpress.static 'public';*たちのWebサーバーにされるフォルダをします。 *app.listenport*、function*{}*ここでたちのポートは*3000*になり、々がびしているはWebサーバーがしくしていることをします。それはいサーバーがセットアップされているということです。

はたちのプロジエクトにき、しいフォルダをし、 *index.html*ファイルをします。 *index.html*はアプリケーションのデフォルト･ファイルで、Expressサーバーはこのファイルをします。 *index.html*は、のようなhtmlファイルです

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8"/>
</head>

<body>
  <h1>hello World</h1>
</body>

</html>
```

ターミナルからプロジエクトパスにし、 ノード*server.js*をします 。 * console.log 'Expressサーバーはポートをしています3000'; *がされます。

ブラウザにし、 *http// localhost3000*とすると、 *hello World*がされます。

すぐパブリックフォルダにし、しいファイル*app.jsx*をします。 JSXはJavaScriptにXMLをするプリプロセッサ･ステップです.ReactをJSXなしですることはいありませんが、JSXはReactをもっとエレガントにします。 *app.jsx*のサンプルコードはのとおりです

```
ReactDOM.render(
  <h1>Hello World!!!</h1>,
  document.getElementById('app')
);
```

*index.html*にき、コードをします。これはのようになります。

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8"/>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23
/browser.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react.js">
</script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react-dom.js">    </script>
</head>

<body>
  <div id="app"></div>

  <script type="text/babel" src="app.jsx"></script>
</body>

</html>
```

これで、あなたはすべてです。はあなたがそれをにつけることをっています。

オンラインでReact Boilerplate [リアクション+バベル+ウェブパック]をむ
https://riptutorial.com/ja/reactjs/topic/5969/react-boilerplate--リアクションplusバベルplusウェブパック-

7: React、WebpackTypescriptのインストール

エディタVSコードなどでのをするには、プロジェクトでするモジュールのをダウンロードするがあります。

たとえば、あなたのプロジェクトでReactとReactDOMをしている、それらのためにハイライトとIntellisenseをしたいとします。のコマンドをして、タイプをプロジェクトにするがあります

```
npm install --save @types/react @types/react-dom
```

あなたのエディタはこのタイピングをにし、これらのモジュールにしてオートコンプリートとIntellisenseをするようになりました。

Examples

webpack.config.js

```
module.exports = {
    entry: './src/index',
    output: {
        path: __dirname + '/build',
        filename: 'bundle.js'
    },
    module: {
        rules: [{
            test: /\.tsx?$/,
            loader: 'ts-loader',
            exclude: /node_modules/
        }]
    },
    resolve: {
        extensions: ['.ts', '.tsx']
    }
};
```

なコンポーネントはのとおりです`entry` 、 `output` 、そののWebpackプロパティにえて。

ローダー

このためには、 `.ts`と`.tsx`ファイルをテストするルールをし、 `ts-loader`としてするがあります。

TSをする

また、 `.ts`と`.tsx`を`resolve`にするがあります。そうしないと、webpackはそれらをしません。

tsconfig.json

これはあなたをかすためののtsconfigです。

```
{
    "include": [
        "src/*"
    ],
    "compilerOptions": {
        "target": "es5",
        "jsx": "react",
        "allowSyntheticDefaultImports": true
    }
}
```

プロパティを1つずつしてみましょう

include

これはソースコードのです。ここでは、 `src`ディレクトリのすべてをコンパイルにめることをする`src/*`エントリが1つしかありません。

compilerOptions.target

ES5ターゲットにコンパイルすることをします。

compilerOptions.jsx

これを`true`すると、TypeScriptはに`React.createElement("div")`を`<div />`から`React.createElement("div")`コンパイルします。

compilerOptions.allowSyntheticDefaultImports

あなたがES6モジュールであるかのようにノードモジュールをインポートできるハンディープロパティー。

```
import * as React from 'react'
const { Component } = React
```

あなたはちょうどできます

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

Reactにデフォルトのエクスポートがないというエラーはされません。

ののコンポーネント

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
```

```
interface AppProps {
    name: string;
}
interface AppState {
    words: string[];
}

class App extends Component<AppProps, AppState> {
    constructor() {
        super();
        this.state = {
            words: ['foo', 'bar']
        };
    }

    render() {
        const { name } = this.props;
        return (<h1>Hello {name}!</h1>);
    }
}

const root = document.getElementById('root');
ReactDOM.render(<App name="Foo Bar" />, root);
```

ReactでTypeScriptをする、React DefinitelyTyped `npm install --save @types/react` をダウンロードする`npm install --save @types/react` 、すべてのコンポーネントでタイプをするがあります。

あなたはこうします

```
class App extends Component<AppProps, AppState> { }
```

ここで、 `AppProps`と`AppState`は、それぞれコンポーネントのやのインタフェースまたはタイプエイリアスです。

オンラインでReact、WebpackTypescriptのインストールをむ
https://riptutorial.com/ja/reactjs/topic/9590/react-webpack-typescriptのインストール

8: React.Componentをする

- ケース1React.createClass{}
- ケース2MyComponentクラスがReact.Component {}をする

`React.createClass` は `React.createClass` でされ、v16でされるです。としてなもののためのドロップインパッケージがあります。それをするをするがあります。

Examples

の

2つのコードをしてのいをべてみましょう。

React.createClass

ここでは、Reactクラスがりてられた`const`をっています。なベースコンポーネントをするには、`render`をけます。

```
import React from 'react';

const MyComponent = React.createClass({
  render() {
    return (
      <div></div>
    );
  }
});

export default MyComponent;
```

React.Component

のReact.createClassをとり、それをES6クラスをするようにしましょう。

```
import React from 'react';

class MyComponent extends React.Component {
  render() {
    return (
      <div></div>
    );
  }
}

export default MyComponent;
```

このでは、ES6クラスをしています。 Reactのでは、React.createClassにアクセスするのではなく、 **MyComponent**というクラスをし、React.Componentからします。このようにして、Reactのや JavaScriptをなくします。

PS、これはのブラウザでするようにES6からES5をコンパイルするために、Babelのようなものでされます。

のと**PropType**をする

デフォルトとそのをしするにはながあります。

React.createClass

このバージョンでは、 `propTypes`プロパティはpropのをできるObjectです。 `getDefaultProps`プロパティは、オブジェクトをしてをするです。

```
import React from 'react';

const MyComponent = React.createClass({
  propTypes: {
    name: React.PropTypes.string,
    position: React.PropTypes.number
  },
  getDefaultProps() {
    return {
      name: 'Home',
      position: 1
    };
  },
  render() {
    return (
      <div></div>
    );
  }
});

export default MyComponent;
```

React.Component

このバージョンでは、 `propTypes`を`createClass`オブジェクトのとしてプロパティではなく、の**MyComponent**クラスのプロパティとしてします。

`getDefaultProps`は、defaultPropsというクラスのObjectプロパティにされました。これはもはやgetではなく、なるObjectです。よりくのReactのをします。これはなJavaScriptです。

```
import React from 'react';

class MyComponent extends React.Component {
```

```
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div></div>
    );
  }
}
MyComponent.propTypes = {
  name: React.PropTypes.string,
  position: React.PropTypes.number
};
MyComponent.defaultProps = {
  name: 'Home',
  position: 1
};

export default MyComponent;
```

さらに、 `propTypes`と`defaultProps`のもあります。これは、あなたのビルドにES7プロパティがになっているのショートカットです

```
import React from 'react';

class MyComponent extends React.Component {
  static propTypes = {
    name: React.PropTypes.string,
    position: React.PropTypes.number
  };
  static defaultProps = {
    name: 'Home',
    position: 1
  };
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div></div>
    );
  }
}

export default MyComponent;
```

をする

のにがあります。

React.createClass

のオブジェクトをにす`getInitialState`があります。

```
import React from 'react';

const MyComponent = React.createClass({
  getInitialState () {
    return {
      activePage: 1
    };
  },
  render() {
    return (
      <div></div>
    );
  }
});

export default MyComponent;
```

React.Component

このバージョンでは、 `getInitialState` をするわりに、すべてのをコンストラクタのなプロパティとしてします。これはなJavaScriptなので、「React API」がなくなっているようにじます。

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      activePage: 1
    };
  }
  render() {
    return (
      <div></div>
    );
  }
}

export default MyComponent;
```

ミックスイン

々はすることができます `mixins` React.createClassので。

React.createClass

このバージョンでは、なミックスインのをるmixinsプロパティをして、コンポーネントに `mixins` をできます。これらはコンポーネントクラスをします。

```
import React from 'react';
```

```
var MyMixin = {
  doSomething() {

  }
};
const MyComponent = React.createClass({
  mixins: [MyMixin],
  handleClick() {
    this.doSomething(); // invoke mixin's method
  },
  render() {
    return (
      <button onClick={this.handleClick}>Do Something</button>
    );
  }
});

export default MyComponent;
```

React.Component

ES6でかれたReactコンポーネントをする、React mixinsはサポートされません。さらに、ReactのES6クラスはサポートされません。らはであるとえられているからです。

"this" Context

React.createClassをすると、にthisコンテキストがしくバインドされますが、ES6クラスをしているはそうではありません。

React.createClass

this.handleClickメソッドがバインドされたonClickにしてください。このメソッドがびされると、ReactはhandleClickしいコンテキストをします。

```
import React from 'react';

const MyComponent = React.createClass({
  handleClick() {
    console.log(this); // the React Component instance
  },
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
});

export default MyComponent;
```

React.Component

ES6クラスの、 `this`はデフォルトでは`null`です。クラスのプロパティはにReactクラスコンポーネントインスタンスにバインドされません。

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  handleClick() {
    console.log(this); // null
  }
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
}

export default MyComponent;
```

`this`では、たちがしいことにることができるいくつかのがあります。

ケース1バインドインライン

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  handleClick() {
    console.log(this); // the React Component instance
  }
  render() {
    return (
      <div onClick={this.handleClick.bind(this)}></div>
    );
  }
}

export default MyComponent;
```

ケース2クラスコンストラクタでバインドする

のアプローチは、のしている`this.handleClick`の`constructor`。このようにして、インラインをけます。 JSXにくれることをけるよりいアプローチとして、くのがえています。

```
import React from 'react';
```

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    console.log(this); // the React Component instance
  }
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
}

export default MyComponent;
```

ケース3ES6のをする

にバインドしなくても、ES6のをすることもできます。

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  handleClick = () => {
    console.log(this); // the React Component instance
  }
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
}

export default MyComponent;
```

ES6 / ajaxに "this"キーワードをさせてサーバーからデータをする

```
import React from 'react';

class SearchEs6 extends React.Component{
    constructor(props) {
        super(props);
        this.state = {
            searchResults: []
        };
    }

    showResults(response){
        this.setState({
            searchResults: response.results
        })
```

```
    }

    search(url){
        $.ajax({
            type: "GET",
            dataType: 'jsonp',
            url: url,
            success: (data) => {
                this.showResults(data);
            },
            error: (xhr, status, err) => {
                console.error(url, status, err.toString());
            }
        });
    }

    render() {
        return (
            <div>
                <SearchBox search={this.search.bind(this)} />
                <Results searchResults={this.state.searchResults} />
            </div>
        );
    }
}
```

オンラインでReact.Componentをするをむ https://riptutorial.com/ja/reactjs/topic/6371/react-componentをする

9: Reactでキーをするとその

き

Reactコンポーネントのリストをレンダリングするときはいつでも、コンポーネントは`key`をつがあります。キーはのにすることができますが、そのリストにしてであるがあります。

Reactがアイテムのリストにをレンダリングするがある、Reactはのリストをにりし、いがあればいつでもをします。にされているキーがない、Reactはをスキャンします。それの、Reactはキーをして、リストにまたはされたキーをします

については、このリンクをしてキーのをください https : //facebook.github.io/react/docs/lists-and-keys.html

このリンクをして、キーをすることがされるをおみください https : //facebook.github.io/react/docs/reconciliation.html#recursing-on-children

Examples

な

クラスレスのReactコンポーネントの

```
function SomeComponent(props){

    const ITEMS = ['cat', 'dog', 'rat']
    function getItemsList(){
        return ITEMS.map(item => <li key={item}>{item}</i>);
    }

    return (
        <ul>
            {getItemsList()}
        </ul>
    );
}
```

このでは、のコンポーネントはのようにされます。

```
<ul>
    <li key='cat'>cat</li>
    <li key='dog'>dog</li>
    <li key='rat'>rat</li>
<ul>
```

オンラインでReactでキーをするとそのをむ https://riptutorial.com/ja/reactjs/topic/9665/reactでキーをするとその

10: Reduxにする

き

では、フロントエンドでアプリケーションレベルのをするためのとなってきており、「アプリケーション」をうゃは、それをうことがよくあります。このトピックでは、ReactアプリケーションでライブラリReduxをするとについてします。

Reactのコンポーネントアーキテクチャは、アプリケーションをモジュールされたカプセルされたさなにするのにらしいものですが、アプリケーションのをするためのいくつかのがあります。Reduxをするは、じデータをのコンポーネントまたはページルートにするがあるです。そので、あるコンポーネントまたはのコンポーネントにローカルなにデータをすることはもうできません。コンポーネントでメッセージをすると、すぐにすることになります。 Reduxをすると、コンポーネントはすべてストアのじデータにサブスクライブされ、はアプリケーションにしてにされます。

Examples

の

*createStore*をしてReduxストアをします。

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp, { inistialStateVariable: "derp"})
```

をしてコンポーネントをReduxストアにし、ストアからコンポーネントにプルプルをプルします。

```
import { connect } from 'react-redux'

const VisibleTodoList = connect(
  mapStateToProps,
  mapDispatchToProps
)(TodoList)

export default VisibleTodoList
```

コンポーネントがReduxストアにメッセージをするためのアクションをします。

```
/*
 * action types
 */

export const ADD_TODO = 'ADD_TODO'
```

```
export function addTodo(text) {
  return { type: ADD_TODO, text }
}
```

これらのメッセージをして、リデューサのストアのしいをします。

```
function todoApp(state = initialState, action) {
  switch (action.type) {
    case SET_VISIBILITY_FILTER:
      return Object.assign({}, state, {
        visibilityFilter: action.filter
      })
    default:
      return state
  }
}
```

オンラインでReduxにするをむ https://riptutorial.com/ja/reactjs/topic/10856/reduxにする

11: TypescriptでのReactJSの

Examples

TypescriptでかれたReactJSコンポーネント

、ReactJSのコンポーネントはTypescriptでfacebookののようにうことができます。 'jsx'ファイルのを 'tsx'にきえてください

```
//helloMessage.tsx:
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});
ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

しかし、Typescriptのなチェックをフルにするためには、

1React.createClassのを**ES6**にするクラス

```
//helloMessage.tsx:
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

2に、とのインターフェースをする

```
interface IHelloMessageProps {
    name:string;
}

interface IHelloMessageState {
  //empty in our case
}

class HelloMessage extends React.Component<IHelloMessageProps, IHelloMessageState> {
  constructor(){
    super();
  }
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
ReactDOM.render(<HelloMessage name="Sebastian" />, mountNode);
```

プログラマがをすことをれた、Typescriptはエラーをします。または、インタフェースにされていないをした。

Typescriptのステートレスなリアクションコンポーネント

それらののなであり、をとしないコンポーネントは、クラスをするわりに、JavaScriptとしてすることができます。

```
import React from 'react'

const HelloWorld = (props) => (
    <h1>Hello, {props.name}!</h1>
);
```

`React.SFC`クラスをって`React.SFC`でじことができます

```
import * as React from 'react';

class GreeterProps {
   name: string
}

const Greeter : React.SFC<GreeterProps> = props =>
    <h1>Hello, {props.name}!</h1>;
```

なお、`React.SFC`です`React.StatelessComponent`のいずれかをすることができますので、。

インストールとセットアップ

ノード・プロジェクトでreactとtypescriptをするには、にnpmでされたプロジェクト・ディレクトリーをっていなければなりません。 `npm init`ディレクトリをするには

npmまたはでのりけ

のようにしてnpmをってReactをインストールすることができます

```
npm install --save react react-dom
```

FacebookはYarnというのパッケージマネージャをリリースしました。これはReactのインストールにもできます。 Yarnをインストールしたら、のコマンドをするだけです

```
yarn add react react-dom
```

Reactをnpmでインストールしたとまったくじで、プロジェクトでReactをすることができます。

Typescript 2.0でののインストール

typescriptをしてコードをコンパイルするには、npmまたはyarnをしてファイルをまたはインストールします。

```
npm install --save-dev @types/react @types/react-dom
```

または、をして

```
yarn add --dev @types/react @types/react-dom
```

Typescriptのいバ―ジョンでののインストール

tsdというのパッケ―ジをするがあります

```
tsd install react react-dom --save
```

Typescriptのまたは

JavaScriptとhtml / xmlをさせたJSXをするには、typescriptコンパイラのをするがあります。プロジェクトのtypescriptファイルは`tsconfig.json`という`tsconfig.json`に、JSXオプションをのようにするがあります。

```
"compilerOptions": {
    "jsx": "react"
},
```

そのコンパイラオプションは、にtypescriptコンパイラに、コードのJSXタグをjavascriptびしにするようにします。

typescriptコンパイラがJSXをプレ―ンなjavascriptびしにするのをけるには、

```
"compilerOptions": {
    "jsx": "preserve"
},
```

ステ―トレスおよびプロパティレスコンポ―ネント

のないもなコンポ―ネントとプロパティは、のようにできます。

```
import * as React from 'react';

const Greeter = () => <span>Hello, World!</span>
```

しかし、そのコンポ―ネントは`this.props`アクセスすることができません。 `this.props`、typescriptはそれがコンポ―ネントかどうかをることができないからです。そのにアクセスするには、をします。

```
import * as React from 'react';

const Greeter: React.SFC<{}> = props => () => <span>Hello, World!</span>
```

コンポ―ネントににプロパティをしていないでも、すべてのコンポ―ネントにはがあるため、**`props.children`**アクセスできるようになりました。

ステートレスとプロパティレスのコンポーネントのもう1つののれたいは、なページテンプレートのです。は、プロジェクトにの`Container`、 `NavTop`、および`NavBottom`コンポーネントがにするとすると、な`Page`コンポーネントの`NavBottom`です。

```
import * as React from 'react';

const Page: React.SFC<{}> = props => () =>
    <Container>
        <NavTop />
        {props.children}
        <NavBottom />
    </Container>

const LoginPage: React.SFC<{}> = props => () =>
    <Page>
        Login Pass: <input type="password" />
    </Page>
```

このでは、 `Page`コンポーネントはでののページでテンプレートとしてできます。

オンラインでTypescriptでのReactJSのをむ https://riptutorial.com/ja/reactjs/topic/1419/typescriptでのreactjsの

12: インストール

Examples

シンプルなセットアップ

フォルダの

このでは、 `src/`コードと`out/`することをしています。このようにフォルダはのようになります

```
example/
|-- src/
|   |-- index.js
|   `-- ...
|-- out/
`-- package.json
```

パッケージの

セットアップnpmをすると、に、Reactコードをes5のコードにするために、babelをするがあります。

```
$npm install --save-dev babel-core babel-loader babel-preset-es2015 babel-preset-react
```

のコマンドは、webpackでするローダーモジュールとコアbabelライブラリをインストールするようnpmにします。また、JSXとes6モジュールコードをするためにes6とするプリセットをインストールします。 プリセットにするは、ここでBabelプリセットをしてください

```
$npm i -D webpack
```

このコマンドはwebpackをのとしてインストールします。 はインストールのであり、 **-D**は--save-devのです

のwebpackパッケージのローダーやwebpack-dev-server extensionなどをインストールすることもできます。

に、のコードがになります

```
$npm i -D react react-dom
```

ウェブパックの

のでは、webpackにをすべきかをえるためにwebpack.config.jsファイルがになります

シンプルなwebpack.config.js

```
var path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'out'),
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /(node_modules)/,
        loader: 'babel-loader',
        query: {
          presets: ['es2015', 'react']
        }
      }
    ]
  }
};
```

このファイルはwebpackにindex.jsファイルsrc /にあるとでまり、 `out`ディレクトリののbundle.jsファイルにするようにします。

`module`ブロックは、にしてつかったすべてのファイルをテストするようにwebpackにし、した、されたロ―ダをびします。 この、 `babel-loader` さらに、 `exclude` regexは、 `node_modules`フォルダのすべてのモジュ―ルのこのなロ―ダ―をするように`node_modules`ます。これは、プロセスのスピ―ドアップにちます。に、 `query`オプションは、webpackにどのパラメ―タをbabelにすかをえ、にインスト―ルしたプリセットをすためにされます。

セットアップのテスト

っているのは、 `src/index.js`ファイルをし、アプリケ―ションをパックすることです

src / index.js

```
'use strict'

import React from 'react'
import { render } from 'react-dom'

const App = () => {
  return <h1>Hello world!</h1>
}

render(
  <App />,
```

```
  document. getElementById('app')
)
```

このファイルは、、な`<h1>Hello world!</h1>`ヘッダーをID「app」のhtmlタグにレンダリングしますが、のところコードをトランスペアレントにするだけでです。

`$./node_modules/.bin/webpack` .ローカルにインストールされたバージョンのwebpackをします-gをしてwebpackをグローバルにインストールしたは`$webpack`します

これは、されたコードを`out/bundle.js`ファイル`out/bundle.js`をにし、そのをするはずです。

webpack-dev-serverの

セットアップ

webpackをするなプロジェクトをした、 `$npm i -g webpack-dev-server`をしてさせると、よりくできるようにhttpサーバーがインストールされます。

webpack.config.jsの

```
var path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'out'),
    publicPath: '/public/',
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /(node_modules)/,
        loader: 'babel',
        query: {
          presets: ['es2015', 'react']
        }
      }
    ]
  },
  devServer: {
    contentBase: path.resolve(__dirname, 'public'),
    hot: true
  }
};
```

は

- `output.publicPath`をするためのパスをします はWebpackファイルをしてください

- `devServer`
    - `contentBase`ファイルをするためのパス `index.html`
    - `hot`は、ディスクのファイルにがえられたときにwebpack-dev-serverをホットリロードするようにします

に、アプリをテストするためになindex.htmlがです。

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>React Sandbox</title>
  </head>
  <body>

    <div id="app" />

    <script src="public/bundle.js"></script>
  </body>
</html>
```

こので`$webpack-dev-server`をすると、ポート8080のローカルhttp `$webpack-dev-server`がし、に`<h1>Hello world!</h1>`をむページがレンダリングされます。

オンラインでインストールをむ https://riptutorial.com/ja/reactjs/topic/6441/インストール

13: コンポーネント

`React.createClass` は `React.createClass` でされ、v16でされるです。としてなもののためのドロップインパッケージがあります。それをするをするがあります。

Examples

コンポーネント

えられたHTMLファイル

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>React Tutorial</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
  </head>
  <body>
    <div id="content"></div>
    <script type="text/babel" src="scripts/example.js"></script>
  </body>
</html>
```

のファイルにのコードをして、コンポーネントをすることができます。

scripts / example.js

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';

class FirstComponent extends Component {
  render() {
    return (
      <div className="firstComponent">
        Hello, world! I am a FirstComponent.
      </div>
    );
  }
}
ReactDOM.render(
  <FirstComponent />, // Note that this is the same as the variable you stored above
  document.getElementById('content')
);
```

のがられます `div#content` にあるものにしてください。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>React Tutorial</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
  </head>
  <body>
    <div id="content">
        <div className="firstComponent">
            Hello, world! I am a FirstComponent.
        </div>
    </div>
    <script type="text/babel" src="scripts/example.js"></script>
  </body>
</html>
```

ネスティングコンポーネント

ReactJSののくは、コンポーネントのネストをにするです。の2つのコンポーネントをします。

```
var React = require('react');
var createReactClass = require('create-react-class');

var CommentList = reactCreateClass({
  render: function() {
    return (
      <div className="commentList">
        Hello, world! I am a CommentList.
      </div>
    );
  }
});

var CommentForm = reactCreateClass({
  render: function() {
    return (
      <div className="commentForm">
        Hello, world! I am a CommentForm.
      </div>
    );
  }
});
```

なるコンポーネントので、これらのコンポーネントをネストしてすることができます。

```
var React = require('react');
var createReactClass = require('create-react-class');

var CommentBox = reactCreateClass({
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
```

```
        <CommentList /> // Which was defined above and can be reused
        <CommentForm /> // Same here
      </div>
    );
  }
});
```

さらにネストするには3つのがあります。いずれもするがあります。

1.をわずにれにする

からき

```
var CommentList = reactCreateClass({
  render: function() {
    return (
      <div className="commentList">
        <ListTitle/>
        Hello, world! I am a CommentList.
      </div>
    );
  }
});
```

これは、AがBをし、BがCをするスタイルです。

- UIをかつに
- コンポ―ネントのにづいてにをにできます

- コンポジション·アーキテクチャーのの
- の

い

- BとCはちょうどプレゼンテーションコンポ―ネントです
- BはCのライフサイクルをするがあります

2.どもをったネスティング

からき

```
var CommentBox = reactCreateClass({
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList>
```

```
            <ListTitle/> // child
          </CommentList>
          <CommentForm />
        </div>
      );
    }
});
```

これは、AがBをし、AがBにCをするようするスタイルです。

- よりいコンポ―ネントライフサイクル
- コンポジションアーキテクチャのの
- よりい

- はしになることができます
- コンポーネントのとの

い

- BはにCのかをするかけれなければならない
- AはCのライフサイクルをするがあります

Bは`this.props.children`をしてCをレンダリングし、Bがそれらのがであるかをるためのされたはありません。したがって、Bはをすることでどものをかにするかもしれませんが、Bがのものをにるがあれば、3がよりいになるかもしれません。

3.をしてネスティングする

からき

```
var CommentBox = reactCreateClass({
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList title={ListTitle}/> //prop
        <CommentForm />
      </div>
    );
  }
});
```

これは、AがBをするスタイルであり、BはAがののためにかをすためのオプションをする。よりされた。

- としての
- な

- よりい

- はしになることができます
- コンポーネントのとの

い

- Bにはかをするためにされたのがあります
- Bはをレンダリングするのかからない

3は、、のをするためのですが、なをし、をにするためのなでもあります。 1はするものをるのにもでいものですが、2と3はさまざまなユースケースでのをもたらすはずです。

コンポーネントの

これはなのです

```
import React, { Component } from 'react';
import { render } from 'react-dom';

class FirstComponent extends Component {
    render() {
        return (
            <div>
                Hello, {this.props.name}! I am a FirstComponent.
            </div>
        );
    }
}

render(
    <FirstComponent name={ 'User' } />,
    document.getElementById('content')
);
```

のは、 をんでいない ステートレスなとばれます。

そのような、 ES6のにづいたステートレスコンポーネントをするがましいことがあります。

ステートレスコンポーネント

くのアプリケーションでは、をするスマートコンポーネントがありますが、にをけり、JSXとしてHTMLをすなコンポーネントをレンダリングします。ステートレスなコンポーネントは、はるかにで、アプリケーションにいパフォーマンスをもたらします。

らは2つのながあります

1. レンダリングされると、それらはされたすべてのでオブジェクトをけります
2. レンダリングするJSXをさなければなりません

```
// When using JSX inside a module you must import React
import React from 'react';
import PropTypes from 'prop-types';

const FirstComponent = props => (
    <div>
        Hello, {props.name}! I am a FirstComponent.
    </div>
);

//arrow components also may have props validation
FirstComponent.propTypes = {
    name: PropTypes.string.isRequired,
}

// To use FirstComponent in another file it must be exposed through an export call:
export default FirstComponent;
```

ステートフルコンポーネント

の「ステートレス」コンポーネントとはに、「ステートフル」コンポーネントには、 `setState` メソッドでできるオブジェクトがあります。を `constructor` でしてからするがあります。

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

class SecondComponent extends Component {
    constructor(props) {
        super(props);

        this.state = {
            toggle: true
        };

        // This is to bind context when passing onClick as a callback
        this.onClick = this.onClick.bind(this);
    }

    onClick() {
        this.setState((prevState, props) => ({
            toggle: !prevState.toggle
        }));
    }

    render() {
        return (
            <div onClick={this.onClick}>
                Hello, {this.props.name}! I am a SecondComponent.
                <br />
                Toggle is: {this.state.toggle}
            </div>
        );
    }
}
```

`Component`はなく`PureComponent`コンポーネントをすると、いとのによる`shouldComponentUpdate()`ライフサイクルメソッドがにされます。これにより、するのないレンダリングのをらすことにより、アプリケーションのパフォーマンスがします。これは、あなたのコンポーネントが「Pure」であり、にじをじとのでレンダリングすることをとしています。

コンポーネント

コンポーネントHOCは、コンポーネントをすることをにします。

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

const PrintHello = ComposedComponent => class extends Component {
    onClick() {
        console.log('hello');
    }

    /* The higher order component takes another component as a parameter
    and then renders it with additional props */
    render() {
        return <ComposedComponent {...this.props } onClick={this.onClick} />
    }
}

const FirstComponent = props => (
    <div onClick={ props.onClick }>
        Hello, {props.name}! I am a FirstComponent.
    </div>
);

const ExtendedComponent = PrintHello(FirstComponent);
```

のコンポーネントは、レンダリングのいにかかわらず、のコンポーネントでロジックをするにされます。

setStateのとし

コンテキストで`setState`をするはがです。たとえば、getのコールバックで`setState`をびすとします。

```
class MyClass extends React.Component {
    constructor() {
        super();

        this.state = {
            user: {}
        };
    }

    componentDidMount() {
        this.fetchUser();
    }
```

```
    fetchUser() {
        $.get('/api/users/self')
            .then((user) => {
                this.setState({user: user});
            });
    }

    render() {
        return <h1>{this.state.user}</h1>;
    }
}
```

これはをびすがあります。 `Component`マウントにコールバックがびされた、 `this.setState`はにはなりません。このようなはず、 `setState`を`setState`ことができるようにするがあります。

このでは、コンポーネントのマウントにXHRをりすことができます。

```
class MyClass extends React.Component {
    constructor() {
        super();

        this.state = {
            user: {},
            xhr: null
        };
    }

    componentWillUnmount() {
        let xhr = this.state.xhr;

        // Cancel the xhr request, so the callback is never called
        if (xhr && xhr.readyState != 4) {
            xhr.abort();
        }
    }

    componentDidMount() {
        this.fetchUser();
    }

    fetchUser() {
        let xhr = $.get('/api/users/self')
            .then((user) => {
                this.setState({user: user});
            });

        this.setState({xhr: xhr});
    }
}
```

メソッドはとしてされます。 `componentWillUnmount`では、XHRをキャンセルすることをめ、すべてのクリーンアップをします。

もっとなことをすることもできます。このでは、としてこのオブジェクトをけり、ぎがしています「stateSetter」`this.setState`がとき`cancel`ばれています

```
function stateSetter(context) {
    var cancelled = false;
    return {
        cancel: function () {
            cancelled = true;
        },
        setState(newState) {
            if (!cancelled) {
                context.setState(newState);
            }
        }
    }
}

class Component extends React.Component {
    constructor(props) {
        super(props);
        this.setter = stateSetter(this);
        this.state = {
            user: 'loading'
        };
    }
    componentWillUnmount() {
        this.setter.cancel();
    }
    componentDidMount() {
        this.fetchUser();
    }
    fetchUser() {
        $.get('/api/users/self')
            .then((user) => {
                this.setter.setState({user: user});
            });
    }
    render() {
        return <h1>{this.state.user}</h1>
    }
}
```

これは、 `cancelled`が、した`setState`クロージャでされるためです。

は、をReactコンポーネントにすです。をむのをつことができます。コールバックとばれることもあります。

JSXでは、をってpropsがされます

```
<MyComponent userID={123} />
```

MyComponentのので、userIDはpropsオブジェクトからアクセスできるようになりました

```
// The render function inside MyComponent
render() {
    return (
        <span>The user's ID is {this.props.userID}</span>
    )
}
```

すべての`props`、その、なはそのデフォルトをすることがです。

```
// defined at the bottom of MyComponent
MyComponent.propTypes = {
    someObject: React.PropTypes.object,
    userID: React.PropTypes.number.isRequired,
    title: React.PropTypes.string
};

MyComponent.defaultProps = {
    someObject: {},
    title: 'My Default Title'
}
```

このでは、prop `someObject`はオプションですが、prop `userID`はです。 `userID`を`MyComponent`にしなかった、にReactエンジンにながされていないことをするコンソ―ルがされます。しかし、このはReactライブラリのにのみされます。でははされません。

`defaultProps`すると、することができます

```
const { title = 'My Default Title' } = this.props;
console.log(title);
```

に

```
console.log(this.props.title);
```

また、 `object` `array`や`functions`をするためのです。オブジェクトにデフォルトのをしないと、がされなかった、のエラ―がします。

```
if (this.props.someObject.someKey)
```

のでは、 `this.props.someObject`は`undefined`であるため、 `someKey`のチェックによってエラ―がスロ―され、コ―ドがします。 `defaultProps`をすると、のチェックをにできます。

コンポ―ネントの - ユ―ザ―インタ―フェイス

しh3があり、それをクリックするとしのをできるように、ボックスにしたいとえています。 Reactは、コンポ―ネントとif elseをしてこれをにでにします。 のコ―ドの

```
// I have used ReactBootstrap elements. But the code works with regular html elements also
var Button = ReactBootstrap.Button;
var Form = ReactBootstrap.Form;
var FormGroup = ReactBootstrap.FormGroup;
var FormControl = ReactBootstrap.FormControl;

var Comment = reactCreateClass({
  getInitialState: function() {
    return {show: false, newTitle: ''};
  },
```

```
  handleTitleSubmit: function() {
     //code to handle input box submit - for example, issue an ajax request to change name in
database
  },

  handleTitleChange: function(e) {
    //code to change the name in form input box. newTitle is initialized as empty string. We
need to update it with the string currently entered by user in the form
    this.setState({newTitle: e.target.value});
  },

  changeComponent: function() {
    // this toggles the show variable which is used  for dynamic UI
    this.setState({show: !this.state.show)};
  },

  render: function() {

    var clickableTitle;

    if(this.state.show) {
        clickableTitle = <Form inline onSubmit={this.handleTitleSubmit}>
                            <FormGroup controlId="formInlineTitle">
                                <FormControl type="text" onChange={this.handleTitleChange}>
                            </FormGroup>
                        </Form>;
    } else {
        clickabletitle = <div>
                           <Button bsStyle="link" onClick={this.changeComponent}>
                               <h3> Default Text </h3>
                           </Button>
                        </div>;
    }

    return (
        <div className="comment">
            {clickableTitle}
        </div>
    );
  }
});

ReactDOM.render(
    <Comment />, document.getElementById('content')
);
```

コードのは**clickableTitle**です。**show**について、FormかButtonのいずれかになります。 Reactはコンポーネントのれをにします。

そこで、{clickableTitle}をレンダリングにすることができます。 clickableTitleをします。 'this.state.show'というについて、するがされます。

ステートレスコンポーネントのバリエーション

```
const languages = [
  'JavaScript',
  'Python',
```

```
  'Java',
  'Elm',
  'TypeScript',
  'C#',
  'F#'
]
```

```
// one liner
const Language = ({language}) => <li>{language}</li>

Language.propTypes = {
  message: React.PropTypes.string.isRequired
}
```

```
/**
* If there are more than one line.
* Please notice that round brackets are optional here,
* However it's better to use them for readability
*/
const LanguagesList = ({languages}) => {
  <ul>
    {languages.map(language => <Language language={language} />)}
  </ul>
}

LanguagesList.PropTypes = {
  languages: React.PropTypes.array.isRequired
}
```

```
/**
 * This syntax is used if there are more work beside just JSX presentation
 * For instance some data manipulations needs to be done.
 * Please notice that round brackets after return are required,
 * Otherwise return will return nothing (undefined)
 */
const LanguageSection = ({header, languages}) => {
  // do some work
  const formattedLanguages = languages.map(language => language.toUpperCase())
  return (
    <fieldset>
      <legend>{header}</legend>
      <LanguagesList languages={formattedLanguages} />
    </fieldset>
  )
}

LanguageSection.PropTypes = {
  header: React.PropTypes.string.isRequired,
  languages: React.PropTypes.array.isRequired
}
```

```
ReactDOM.render(
  <LanguageSection
    header="Languages"
    languages={languages} />,
```

```
  document.getElementById('app')
)
```

ここではそのをつけることができます。

オンラインでコンポーネントをむ https://riptutorial.com/ja/reactjs/topic/1185/コンポーネント

14: コンポーネントライフサイクルの

き

ライフサイクル・メソッドは、コードをし、コンポーネントのさまざまなでコンポーネントとやりとりするためにされます。これらのメソッドは、コンポーネントMounting、Updating、Unmountingをベースにしています。

Examples

コンポーネントの

Reactコンポーネントがされると、いくつかのがびされます。

- `React.createClass` ES5をしているは、5つのユーザーがびされます
- `class Component extends React.Component` ES6をしているは、3つのユーザーがびされます

`getDefaultProps()` ES5のみ

これがにびされるメソッドです。

このがすPropは、コンポーネントがインスタンスされたときにされていない、デフォルトとしてされます。

のでは、にされていない、 `this.props.name`はデフォルトで`Bob`なります。

```
getDefaultProps() {
  return {
    initialCount: 0,
    name: 'Bob'
  };
}
```

`getInitialState()` ES5のみ

これはびされる**2**のメソッドです。

`getInitialState()`のりは、Reactコンポーネントのをします。 Reactフレームワークはこのをびし、りを`this.state`ます。

のでは、 `this.state.count`はthis.props.initialCountので`this.props.initialCount`ます。

```
getInitialState() {
  return {
    count : this.props.initialCount
  };
}
```

componentWillMount() ES5およびES6

これは**3**のメソッドです。

このをして、コンポーネントをDOMにするにコンポーネントをにすることができます。

```
componentWillMount() {
  ...
}
```

render() ES5とES6

これは**4**のメソッドです。

`render()`は、コンポーネントのとのなでなければなりません。これは、レンダリングにコンポーネントをすのをします。ネイティブDOMコンポーネント `<p />` またはコンポーネントのいずれかをすがあります。レンダリングするがないは、 `null`または`undefined`すことができ`null`。

このは、コンポーネントのまたはがされたにびされます。

```
render() {
  return (
    <div>
      Hello, {this.props.name}!
    </div>
  );
}
```

componentDidMount() ES5およびES6

これは**5**のメソッドです。

コンポーネントがマウントされ、コンポーネントのDOMノードにアクセスできるようになりました `refs`。

このメソッドは、のでするがあります。

- タイマーの
- データの

- イベントリスナーの
- DOMの

```
componentDidMount() {
  ...
}
```

ES6の

コンポーネントがES6クラスをしてされている、`getDefaultProps()`および`getInitialState()`はできません。

わりに、 `defaultProps`をクラスのプロパティとしてし、クラスのコンストラクタでのとをします。これらは、にのReactライフサイクルがびされるにクラスのインスタンスにされます。

のは、このアプローチをしています。

```
class MyReactClass extends React.Component {
  constructor(props){
    super(props);

    this.state = {
      count: this.props.initialCount
    };
  }

  upCount() {
    this.setState((prevState) => ({
      count: prevState.count + 1
    }));
  }

  render() {
    return (
      <div>
        Hello, {this.props.name}!<br />
        You clicked the button {this.state.count} times.<br />
        <button onClick={this.upCount}>Click here!</button>
      </div>
    );
  }
}

MyReactClass.defaultProps = {
  name: 'Bob',
  initialCount: 0
};
```

`getDefaultProps()`きえる

コンポーネントのデフォルトは、クラスの`defaultProps`プロパティをしてします。

```
MyReactClass.defaultProps = {
  name: 'Bob',
  initialCount: 0
};
```

`getInitialState()`きえる

コンポーネントのをするなは、 `this.state`をコンストラクタにすること`this.state`。

```
constructor(props){
  super(props);

  this.state = {
    count: this.props.initialCount
  };
}
```

コンポーネントの

`componentWillReceiveProps(nextProps)`

これはプロパティのでびされるのです。

コンポーネントのプロパティがされると 、Reactはこのプロパティをしいプロパティでびします。あなたは*this.props*とし、nextPropsでしいにいにアクセスすることができます。

これらのをすると、いとしいのでいくつかのをうことができます。

```
componentWillReceiveProps(nextProps){
  if (nextProps.initialCount && nextProps.initialCount > this.state.count){
    this.setState({
      count : nextProps.initialCount
    });
  }
}
```

`shouldComponentUpdate(nextProps, nextState)`

これは、 プロパティのとのオンのでびされる**2**のです。

デフォルトでは、のコンポーネント/コンポーネントがコンポーネントのプロパティ/をした、**React**はコンポーネントのしいバージョンをレンダリングします。この、このはにtrueをします。

このをオーバーライドして、コンポーネントをするがあるかどうかをよりにできます。

このはににされます。

が**false**をす、 パイプラインはちにします。

```
componentShouldUpdate(nextProps, nextState){
  return this.props.name !== nextProps.name ||
    this.state.count !== nextState.count;
}
```

componentWillUpdate(nextProps, nextState)

このは、 `componentWillMount()`ようにします。 はDOMにはないため、がされるにいくつかのをうことができます。

/\ this.setStateはできません。

```
componentWillUpdate(nextProps, nextState){}
```

render()

いくつかのがありますので、コンポ―ネントをレンダリングしてください。

componentDidUpdate(prevProps, prevState)

`componentDidMount()`じことです **DOM**はリフレッシュされるので、ここでDOMでいくつかのをうことができます。

```
componentDidUpdate(prevProps, prevState){}
```

の

componentWillUnmount()

このメソッドは、コンポ―ネントがDOMからアンマウントされるにびされます。

のようなクリ―ニングをするのにしています。

- イベントリスナ―の
- タイマ―をクリアします。
- ソケットをします。
- をする。

```
componentWillUnmount(){
  ...
}
```

`componentWillUnMount`で、アタッチされたイベントリスナ―をする

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

export default class SideMenu extends Component {
```

```
  constructor(props) {
    super(props);
    this.state = {
        ...
      };
    this.openMenu = this.openMenu.bind(this);
    this.closeMenu = this.closeMenu.bind(this);
  }

  componentDidMount() {
    document.addEventListener("click", this.closeMenu);
  }

  componentWillUnmount() {
    document.removeEventListener("click", this.closeMenu);
  }

  openMenu() {
    ...
  }

  closeMenu() {
    ...
  }

  render() {
    return (
      <div>
        <a
          href      = "javascript:void(0)"
          className = "closebtn"
          onClick   = {this.closeMenu}
        >
          x
        </a>
        <div>
          Some other structure
        </div>
      </div>
    );
  }
}
```

コンテナ

Reactアプリケーションをする、なにづいてコンポーネントをPresentationalコンポーネントとContainerコンポーネントにけることがましいことがよくあります。
プレゼンテーションコンポーネントは、データのにのみします。モデルをビューにするとなすことができ、しばしばされます。、はされません。コンテナコンポーネントは、データのにします。これは、のをしてにうことも、Reduxのようなのでることもできます。コンテナコンポーネントはデータをするのではなく、データをプレゼンテーションコンポーネントにします。

```
// Container component
import React, { Component } from 'react';
import Api from 'path/to/api';
```

```
class CommentsListContainer extends Component {
    constructor() {
        super();
        // Set initial state
        this.state = { comments: [] }
    }

    componentDidMount() {
        // Make API call and update state with returned comments
        Api.getComments().then(comments => this.setState({ comments }));
    }

    render() {
        // Pass our state comments to the presentational component
        return (
            <CommentsList comments={this.state.comments} />;
        );
    }
}

// Presentational Component
const CommentsList = ({ comments }) => (
    <div>
        {comments.map(comment => (
            <div>{comment}</div>
        )}
    </div>
);

CommentsList.propTypes = {
    comments: React.PropTypes.arrayOf(React.PropTypes.string)
}
```

なるのライフサイクルメソッドびし

このは、ライフサイクルメソッドのとメソッドのびしについてしているのをするものです。

このでは、どのメソッドcomponentWillMount、componentWillReceivePropsなどがびされ、なるのコンポ―ネントにしてどのシ―ケンスがなるかをしています。

コンポ―ネントがされるとき

1. getDefaultProps
2. getInitialState
3. componentWillMount
4. レンダリングする
5. componentDidMount

コンポ―ネントのがされた

1. shouldComponentUpdate
2. componentWillUpdate
3. レンダリングする
4. componentDidUpdate

コンポーネントにがされた

1. componentWillReceiveProps
2. shouldComponentUpdate
3. componentWillUpdate
4. レンダリングする
5. componentDidUpdate

コンポーネントがアンマウントの

1. componentWillUnmount

オンラインでコンポーネントライフサイクルのをむ https://riptutorial.com/ja/reactjs/topic/2750/コンポーネントライフサイクルの

15: コンポーネントの

Examples

ステートレスコンポーネントの

このでは、になります`Redux`して`React Redux`々のアプリケーションのをするためのモジュールを、のためにのコンポーネントの。、そしてofcourseの`React`と`React Dom`

したデモをここでチェックアウトすることができます

のでは、3つのなるコンポーネントと1つのされたコンポーネント

- **UserInputForm** このコンポーネントはフィールドをします。フィールドがされると、コンポーネントがする`props inputChange`メソッドをびし、データがされているはフィールドにそれをします。
- **UserDashboard** このコンポーネントは、なメッセージをし、 `UserInputForm`コンポーネントを`UserInputForm`ます。また、 `inputChange`メソッドを`UserInputForm`コンポーネントに`UserInputForm`ます.Unputはこのメソッドをしてコンポーネントとします。
    - **UserDashboardConnected** このコンポーネントは、 `UserDashboard ReactRedux connect`メソッドをして`UserDashboard`コンポーネントをラップするだけです。これにより、のにコンポーネントのをしてコンポーネントをにできます。
- **App** このコンポーネントは、 `UserDashboardConnected`コンポーネントをレンダリングする`UserDashboardConnected`です。

```
const UserInputForm = (props) => {

  let handleSubmit = (e) => {
    e.preventDefault();
  }

  return(
    <form action="" onSubmit={handleSubmit}>
      <label htmlFor="name">Please enter your name</label>
      <br />
      <input type="text" id="name" defaultValue={props.data.name || ''} onChange={
props.inputChange } />
    </form>
  )

}


const UserDashboard = (props) => {

  let inputChangeHandler = (event) => {
    props.updateName(event.target.value);
```

```
  }

  return(
    <div>
      <h1>Hi { props.user.name || 'User' }</h1>
      <UserInputForm data={props.user} inputChange={inputChangeHandler} />
    </div>
  )

}

const mapStateToProps = (state) => {
  return {
    user: state
  };
}
const mapDispatchToProps = (dispatch) => {
  return {
    updateName: (data) => dispatch( Action.updateName(data) ),
  };
};

const { connect, Provider } = ReactRedux;
const UserDashboardConnected = connect(
  mapStateToProps,
  mapDispatchToProps
)(UserDashboard);



const App = (props) => {
  return(
    <div>
      <h1>Communication between Stateless Functional Components</h1>
      <UserDashboardConnected />
    </div>
  )
}



const user = (state={name: 'John'}, action) => {
  switch (action.type) {
    case 'UPDATE_NAME':
      return Object.assign( {}, state, {name: action.payload}  );

    default:
      return state;
  }
};

const { createStore } = Redux;
const store = createStore(user);
const Action = {
  updateName: (data) => {
    return { type : 'UPDATE_NAME', payload: data }
  },
}


ReactDOM.render(
```

```
  <Provider store={ store }>
    <App />
  </Provider>,
  document.getElementById('application')
);
```

JS Bin URL

オンラインでコンポーネントのをむ https://riptutorial.com/ja/reactjs/topic/6137/コンポーネントの

16: コンポーネントの

Reactコンポーネントのには3つのケースがあります。

- ケース1とのコミュニケーション
- ケース2とのコミュニケーション
- ケース3していないコンポーネントコンポーネントの

Examples

コンポーネントとコンポーネント

Reactのではにであり、チャンスはあなたがすでにっているもなケースです。

コンポーネントにをすことができます。このでは、`message`はコンポーネントにすであり、メッセージはにされています。はです。

```
import React from 'react';

class Parent extends React.Component {
    render() {
        const variable = 5;
        return (
            <div>
                <Child message="message for child" />
                <Child message={variable} />
            </div>
        );
    }
}

class Child extends React.Component {
    render() {
        return <h1>{this.props.message}</h1>
    }
}

export default Parent;
```

ここで、`<Parent />`は、つのレンダリング`<Child />`し、を`message for child`1と5 1。

すると、もう1つをレンダリングしていくつかのをすコンポーネントがあります。

コンポーネントからコンポーネントへ

データをにりすと、コンポーネントからコンポーネントへのとしてをし、コンポーネントはそのをびします。

このでは、Childコンポーネントにをし、ChildコンポーネントでそのをびすことによってParentを

します。

```
import React from 'react';

class Parent extends React.Component {
    constructor(props) {
        super(props);
        this.state = { count: 0 };

        this.outputEvent = this.outputEvent.bind(this);
    }
    outputEvent(event) {
        // the event context comes from the Child
        this.setState({ count: this.state.count++ });
    }

    render() {
        const variable = 5;
        return (
            <div>
                Count: { this.state.count }
                <Child clickHandler={this.outputEvent} />
            </div>
        );
    }
}

class Child extends React.Component {
    render() {
        return (
            <button onClick={this.props.clickHandler}>
                Add One More
            </button>
        );
    }
}

export default Parent;
```

のをするの`outputEvent`メソッドは、のボタン`onClick`イベントによってびされることにしてください。

していないコンポ—ネント

あなたのコンポ—ネントがをたないまたはがありますが、ななのようにそれののは、あるコンポ—ネントがサブスクライブし、もうのコンポ—ネントにシグナルをることです。

これは、イベントシステムの2つのなですするイベントを**/**きり 、イベントを**/**トリガ—/パブリッシュ/ディスパッチして 、にします。

なくとも3つのパタ—ンがあります。 ここでをつけることができます。

なをにします。

- パタ—ン1 **Event Emitter / Target / Dispatcher** リスナ—は、するためにソ—スをするがあ

ります。

    - サブスクライブする `otherObject.addEventListener('click', () => { alert('click!'); });`
    - ディスパッチする `this.dispatchEvent('click');`

- パターン2 パブリッシュ/サブスクライブ イベントをトリガーするソースへののはありません。すべてのイベントをするグローバルオブジェクトがどこからでもアクセスできます。

    - するには `globalBroadcaster.subscribe('click', () => { alert('click!'); });`
    - ディスパッチする `globalBroadcaster.publish('click');`

- パターン3 シグナル Event Emitter / Target / Dispatcherとていますが、ここではランダムなはしません。イベントをさせるのあるオブジェクトには、そののプロパティがです。このようにして、オブジェクトがどのイベントをできるかがにかります。

    - サブスクライブする `otherObject.clicked.add( () => { alert('click'); });`
    - ディスパッチする `this.clicked.dispatch();`

オンラインでコンポーネントのをむ https://riptutorial.com/ja/reactjs/topic/6567/コンポーネントの

17: サーバーレンダリングの

Examples

レンダリングコンポーネント

サーバでコンポーネントをレンダリングするには、 `renderToString`と`renderToStaticMarkup` 2つのオプションがあります。

renderToString

これにより、Reactコンポーネントがサーバーの`HTML`にレンダリングされます。このは`HTML`に`data-react-`も`data-react-`ので、クライアントのReactはをびレンダリングするはありません。

```
import { renderToString } from "react-dom/server";
renderToString(<App />);
```

renderToStaticMarkup

これにより、Reactコンポーネントが`HTML`にレンダリングされますが、 `data-react-`がなければ、コンポーネントはレンダリングされるため、クライアントでレンダリングされるコンポーネントのはおめしません。

```
import { renderToStaticMarkup } from "react-dom/server";
renderToStaticMarkup(<App />);
```

オンラインでサーバーレンダリングのをむ https://riptutorial.com/ja/reactjs/topic/7478/サーバーレンダリングの

18: ステートレスコンポーネント

Reactのステートレスコンポーネントは、されたpropsなです。これらのコンポーネントはにせず、コンポーネントのライフサイクルメソッドのをします。ただし、 propTypesとdefaultPropsすることはできます。

ステートレスコンポーネントのについては、 https://facebook.github.io/react/docs/reusable-components.html#stateless-functionsをしてください 。

Examples

ステートレスコンポーネント

コンポーネントをすると、UIをしたなにできます。これはリアクトのしさです。ページをのさなコンポーネントにけることができます 。

React v14は、 React.Component ES6、またはReact.createClass ES5をしてステートフルなReactコンポーネントをすることができましたデータをするためにをとするかどうかはありません。

React v14では、はステートレスコンポーネントとばれるコンポーネントをするためのながされました。これらのコンポーネントは、プレーンなJavaScriptをします。

えば

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

このは、なReactコンポーネントです。これは、データでのpropsオブジェクトをけり、Reactをすためです。このようなコンポーネントは、りJavaScript なので、これらのコンポーネントは です。

ステートレスコンポーネントは、UIにをいています。は、の「コンテナ」コンポーネント、またはFlux / Reduxなどをしてするがあります。ステートレスコンポーネントは、メソッドまたはライフサイクルメソッドをサポートしていません。

1. クラスオーバーヘッドなし
2. thisキーワードについてするはありません
3. きやすく、しやすい
4. のについてするはありません
5. パフォーマンスの

をとせず、なUIをするReactコンポーネントをする、のReactコンポーネントをするわりに、 ステートレスコンポーネントとしてできます 。

なをえてみましょう

たとえば、ユーザーをしたり、ユーザーをしたり、ユーザーのリストをできるページがあるとします。

これは、アプリケーション`index.js`エントリポイントです。

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

import HomePage from './homepage'

ReactDOM.render(
    <HomePage/>,
    document.getElementById('app')
);
```

`HomePage`コンポーネントは、ユーザーをおよびするためのUIをします。これは、、UI、およびコードをむなReactコンポーネントであることにしてください。ユーザーのリストのデータは`state`にされますが、な`List` にすはリストのUIコードをカプセルします。

homepage.js

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

import List from './list';

export default class Temp extends Component{

    constructor(props) {
        super();
        this.state={users:[], showSearchResult: false, searchResult: []};
    }

    registerClick(){
        let users = this.state.users.slice();
        if(users.indexOf(this.refs.mail_id.value) == -1){
            users.push(this.refs.mail_id.value);
            this.refs.mail_id.value = '';
            this.setState({users});
        }else{
            alert('user already registered');
        }
    }

    searchClick(){
        let users = this.state.users;
        let index = users.indexOf(this.refs.search.value);
        if(index >= 0){
            this.setState({searchResult: users[index], showSearchResult: true});
        }else{
            alert('no user found with this mail id');
        }
    }

    hideSearchResult(){
```

```
        this.setState({showSearchResult: false});
    }

    render() {
        return (
            <div>
                <input placeholder='email-id' ref='mail_id'/>
                <input type='submit' value='Click here to register'
onClick={this.registerClick.bind(this)}/>
                <input style={{marginLeft: '100px'}} placeholder='search' ref='search'/>
                <input type='submit' value='Click here to register'
onClick={this.searchClick.bind(this)}/>
                {this.state.showSearchResult ?
                    <div>
                        Search Result:
                        <List users={[this.state.searchResult]}/>
                        <p onClick={this.hideSearchResult.bind(this)}>Close this</p>
                    </div>
                    :
                    <div>
                        Registered users:
                        <br/>
                        {this.state.users.length ?
                            <List users={this.state.users}/>
                            :
                            "no user is registered"
                        }
                    </div>
                }
            </div>
        );
    }
}
```

に、されるステ―トレスコンポ―ネント `List` 、ユ―ザ―のリストとのをしますが、はされません。

list.js

```
import React from 'react';
var colors = ['#6A1B9A', '#76FF03', '#4527A0'];

var List = (props) => {
    return(
        <div>
            {
                props.users.map((user, i)=>{
                    return(
                        <div key={i} style={{color: colors[i%3]}}>
                            {user}
                        </div>
                    );
                })
            }
        </div>
    );
}

export default List;
```

リファレンス https : //facebook.github.io/react/docs/components-and-props.html

オンラインでステートレスコンポーネントをむ https://riptutorial.com/ja/reactjs/topic/6588/ステートレスコンポーネント

19: どのようになWebpack、し、をする

このビルドパイプラインは、「プロダクションレディ」とはりませんが、しているをるためになものをすることができます。いくつかの々がをめてるアプローチは、Yeoman.ioやののにされたパイプラインをって、それがスタイルにうまでましくないものをりくことです。これにはもはありませんが、おそらくのでは、のアプローチをして、のからすることができます。

あなたがしたいかもしれないかは、テストフレームワークやモカやジャスミンのカルマのようなカバレッジのようなものです。 ESLintをいてをっる。 webpack-dev-serverのホットモジュールのきえにより、Ctrl + S、F5のをることができます。また、のパイプラインはdevモードでしかされないため、プロダクションビルドタスクはです。

ゴッチャ

コンテキストプロパティにしてください`webpack.config.js`たちはではなく、たちのパスをするためにノード·パスモジュールをしている`__dirname`に`/src`ので、これはがslashsesをっています 。したがって、ソリューションをよりクロスプラットフォームにさせるには、ノードをしてしてください。

webpack.config.jsプロパティの

コンテキスト

これはパスをするためにwebpackがルートパスとしてするファイルパスです。したがって、index.jsxでは`require('./index.html')`をしてい`require('./index.html')`ドットはには`src/`ディレクトリにされるため、このプロパティでされています。

エントリ

webpackがにソリューションのバンドルをする。このため、index.jsxではニーズとインポートのソリューションをまとめていることがわかります。

ここでは、webpackがバンドルしたファイルをするをします。バンドルされたjavascriptとスタイルがされるファイルのもしました。

devServer

これらはwebpack-dev-serverにのです。 `contentBase`は、サーバがどこにルートをるべきかをします。ここでは、 `dist/`フォルダをベースとしてしています。 `port`は、サーバがホストされるポートです。 `open`は、webpack-dev-serverにサーバのにデフォルトブラウザをくようにするためにされるものです。

モジュール>ローダー

これは、webpackがなるファイルをつけたときにをするかをるためにするマッピングをします。`test`プロパティは、webpackがこのモジュールをするかどうかをするためにするregexをえます。

ほとんどの、ファイルにマッチします。 `loader`または`loaders`は、ファイルをwebpackにロードし、そのローダーがそのファイルタイプのバンドルをするためにするローダーモジュールのをします。また、javascriptには`query`プロパティがあり`query`。これはにローダにクエリをするため、たちがむならHTMLローダでクエリプロパティをしているがあります。それはをするためののです。

Examples

カスタマイズされた「Hello world」のをってパイプラインをする

ステップ1Node.jsをインストールする

するビルドパイプラインはNode.jsにづいているため、のインスタンスでこれがインストールされていることをするがあります。 Node.jsをインストールするについては、 ここで SOのドキュメントをチェックアウトすることができます

ステップ2プロジェクトをノードモジュールとしてする

コマンドラインでプロジェクトフォルダをき、のコマンドをします。

```
npm init
```

このでは、デフォルトをにすることができます。または、パッケージのにするこの SOのドキュメントをすべてすることができます。

ステップ3なnpmパッケージをインストールする

コマンドラインでのコマンドをして、このでなパッケージをインストールします。

```
npm install --save react react-dom
```

に、devののために、このコマンドをします

```
npm install --save-dev babel-core babel-preset-react babel-preset-es2015 webpack babel-loader
css-loader style-loader file-loader image-webpack-loader
```

に、webpackとwebpack-dev-serverは、プロジェクトのとしてではなく、グローバルにインストールするのあるものです。としてしたいは、そうしてもらえません。するコマンドはのとおりです。

```
npm install --global webpack webpack-dev-server
```

3プロジェクトのルートに.babelrcファイルをする

これで、インストールしたばかりのプリセットをするようにされます。 .babelrcファイルはのようになります。

```
{
  "presets": ["react", "es2015"]
}
```

4プロジェクトのディレクトリをする

ディレクトリのルートにのようなディレクトリをします。

```
|- node_modules
|- src/
  |- components/
  |- images/
  |- styles/
  |- index.html
  |- index.jsx
|- .babelrc
|- package.json
```

`node_modules`、 `.babelrc`、および`package.json`は、のからすでにそこにっているがあります。

ステップ**5**プロジェクトに**Hello World**プロジェクトファイルをりむ

これはパイプラインをするプロセスにとってにではありませんので、はあなたにこれらのコードをえて、それらをりけることができます

src / components / HelloWorldComponent.jsx

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

class HelloWorldComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {name: 'Student'};
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    this.setState({name: e.target.value});
  }

  render() {
    return (
      <div>
        <div className="image-container">
          <img src="./images/myImage.gif" />
        </div>
        <div className="form">
          <input type="text" onChange={this.handleChange} />
          <div>
            My name is {this.state.name} and I'm a clever cloggs because I built a React build
pipeline
          </div>
        </div>
      </div>
    );
  }
```

```
}

export default HelloWorldComponent;
```

src / images / myImage.gif

たちがをにまとめることができるということをするために、あなたがきなでこれをきえることをにしてください。のイメ―ジをしていて、のをけたは、をするために`HelloWorldComponent.jsx`をする`HelloWorldComponent.jsx`があります。に、ファイルがなるをした、webpack.config.jsのロ―ダ―の`test`プロパティを、なでしいファイルにわせてするがあります。

src / styles / styles.css

```
.form {
  margin: 25px;
  padding: 25px;
  border: 1px solid #ddd;
  background-color: #eaeaea;
  border-radius: 10px;
}

.form div {
  padding-top: 25px;
}

.image-container {
  display: flex;
  justify-content: center;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Learning to build a react pipeline</title>
</head>
<body>
  <div id="content"></div>
  <script src="app.js"></script>
</body>
</html>
```

index.jsx

```
import React from 'react';
import { render } from 'react-dom';
import HelloWorldComponent from './components/HelloWorldComponent.jsx';

require('./images/myImage.gif');
require('./styles/styles.css');
require('./index.html');

render(<HelloWorldComponent />, document.getElementById('content'));
```

ステップ**6**ウェブパックをする

プロジェクトのルートにwebpack.config.jsというのファイルをし、このコードをコピーします。

webpack.config.js

```
var path = require('path');

var config = {
  context: path.resolve(__dirname + '/src'),
  entry: './index.jsx',
  output: {
    filename: 'app.js',
    path: path.resolve(__dirname + '/dist'),
  },
  devServer: {
    contentBase: path.join(__dirname + '/dist'),
    port: 3000,
    open: true,
  },
  module: {
    loaders: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        loader: 'babel-loader'
      },
      {
        test: /\.css$/,
        loader: "style!css"
      },
      {
        test: /\.gif$/,
        loaders: [
          'file?name=[path][name].[ext]',
          'image-webpack',
        ]
      },
      { test: /\.(html)$/,
        loader: "file?name=[path][name].[ext]"
      }
    ],
  },
};

module.exports = config;
```

7パイプラインの**npm**タスクをする

これをうには、プロジェクトのルートにあるpackage.jsonファイルでされたJSONのscriptsキーに2つのプロパティをするがあります。スクリプトのキーをのようにします。

```
  "scripts": {
    "start": "webpack-dev-server",
    "build": "webpack",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

`test`スクリプトはすでにそこにあり、あなたはそれをするかどうかをできますが、このではではありません。

8パイプラインをする

コマンドラインから、プロジェクトのルートディレクトリにいる、のコマンドをできるはずです。

```
npm run build
```

これはあなたがしたさなアプリケーションをまとめて、プロジェクトフォルダのルートにする`dist/`ディレクトリにします。

のコマンドをすると

```
npm start
```

に、したアプリケーションは、webpack devサーバインスタンスのデフォルトのWebブラウザにされます。

オンラインでどのようなWebpack、し、をするをむ https://riptutorial.com/ja/reactjs/topic/6294/どのようなwebpack-し-をする

20: パフォーマンス

Examples

- HTML DOMとDOM

HTML DOMはです

Webページは、オブジェクトのツリーとしてにされます。このは、 *Document Object Model*とばれます。さらに、プログラミングJavaScriptなどがHTMLにアクセスできるようにする、にしないインタフェースです。

いえると

HTML DOMは、HTMLを、、、またはするのです。

しかし、これらの**DOM**はにです。

DOMはです

Reactのチームは、 *HTML DOM*をし、の*Virtual DOM*をして、アプリケーションののをするために*HTML DOM*にするがあるのをするというえをいつきました。

Virtual DOMはな**DOM**のからをします。

どのようにですか

で、React は`Virtual DOM`としてされるアプリケーションをち`Virtual DOM`。 アプリケーションのがわるたびに、これはReactがパフォーマンスをするためにするステップです

1. アプリケーションのしいをすしい*DOM*をする
2. いDOMのHTML DOMをすとしいDOMをする
3. 2.のHTML DOMをすいDOMをしいDOMにするのをめます

- それについてもっとるには - ReactのDiffアルゴリズムをんでください

4. これらのがつかると、それらはの*HTML DOM*にマップされます

- *Virtual DOM*は*HTML DOM*のであり、それらのにのがあることをえておいてください

5. これでつかったの*HTML DOM*にされたのが、アプリケーションの*HTML DOM*にされるようになり、 *HTML DOM*をにするがされます 。

Virtual DOMはJavaScriptオブジェクトなので、Virtual DOMにされるはです。

リアクションのアルゴリズム

1つのツリーをのツリーにするためのをすることは、On ^ 3のオーダーのさをする。ここで、nはツリーのノードのである。は、このをでくための2つのにする - On

1. じクラスの2つのコンポーネントがのツリーをし、なるクラスの2つのコンポーネントがなるツリーをします。
2. さまざまなレンダリングでしているにのキーをすることはです。

2つのノードがなるかどうかをするために、Reactが3つのケースをします

1. タイプがなる、2つのノードがなります。

- たとえば、 `<div>...</div>`は`<span>...</span>`とはなり`<span>...</span>`

2. 2つのノードがなるキー

- たとえば、 `<div key="1">...</div>`が`<div key="2">...</div>`

さらに、パフォーマンスをしたいは 、 のことがであり、することがにです

> それらが[2つのノード]がじタイプでない、Reactは、それらがレンダリングしたものとすることをみることさえありません。のものをDOMからし、2のものをするだけです。

はのとおりです

> あるがするもののようにえるDOMをすることはほとんどありません。 Reactは、これら2つのにマッチさせようとをやすわりに、ツリーをからビルドします。

ヒントとコツ

2つのノードがじでない、Reactはそれらをしようとせず、DOMからのノードをし、2のノードをします。これが、のヒントがうです

1. じようなをつ2つのコンポーネントクラスをにているは、じクラスにすることができます。

---

2. コンポーネントがレンダリングされないようにshouldComponentUpdateをします。

```
shouldComponentUpdate: function(nextProps, nextState) {
  return nextProps.id !== this.props.id;
}
```

ReactJSによる

あなたができないものをすることはできません 。 Reactコンポーネントのパフォーマンスをさせるには、それをできるがあります。 ReactJSはパフォーマンスをするアドオンツールをします。 `react-addons-perf`モジュールをインポートしてパフォーマンスをする

```
import Perf from 'react-addons-perf' // ES6
var Perf = require('react-addons-perf') // ES5 with npm
var Perf = React.addons.Perf; // ES5 with react-with-addons.js
```

インポートされた`Perf`モジュールののメソッドをできます。

- Perf.printInclusive
- Perf.printExclusive
- Perf.printWasted
- Perf.printOperations
- Perf.printDOM

ほとんどの、もなのは`Perf.printWasted()`これは々のコンポーネントのなをでしたものです

(index)	Owner > component	Waste
0	"Todos > TodoItem"	102.7

Total time: 132.71 ms

の**Wasted time**にし、の**TipsTricks**セクションをしてComponentのパフォーマンスをさせることができます

React Official GuideとBenchling Enggのれたをしてください。リアクションパフォーマンス

オンラインでパフォーマンスをむ https://riptutorial.com/ja/reactjs/topic/6875/パフォーマンス

21: フォームとユーザー

Examples

されたコンポーネント

されたフォームコンポーネントは、 `value`プロパティでされます。のはReactによってされ、ユーザーはレンダリングされたにしません。わりに、 `value`プロパティのはこのをするがあります。

```
class Form extends React.Component {
  constructor(props) {
    super(props);

    this.onChange = this.onChange.bind(this);

    this.state = {
      name: ''
    };
  }

  onChange(e) {
    this.setState({
      name: e.target.value
    });
  }

  render() {
    return (
      <div>
        <label for='name-input'>Name: </label>
        <input
          id='name-input'
          onChange={this.onChange}
          value={this.state.name} />
      </div>
    )
  }
}
```

のは、 `value`プロパティがののをし、 `onChange`イベントハンドラがコンポーネントのをユーザのでするをしてい`value`。

フォームは、であれば、されたコンポーネントとしてするがあります。これにより、ユーザーのトリガーによってがされても、コンポーネントのとがにします。

されていないコンポーネント

されていないコンポーネントは、 `value`プロパティをたないです。コンポーネントとはに、コンポーネントのとをさせておくのはアプリケーションのです。

```
class Form extends React.Component {
```

```
  constructor(props) {
    super(props);

    this.onChange = this.onChange.bind(this);

    this.state = {
      name: 'John'
    };
  }

  onChange(e) {
    this.setState({
      name: e.target.value
    });
  }

  render() {
    return (
      <div>
        <label for='name-input'>Name: </label>
        <input
          id='name-input'
          onChange={this.onChange}
          defaultValue={this.state.name} />
      </div>
    )
  }
}
```

ここでは、コンポーネントのは、されたコンポーネントとに、 `onChange`イベントハンドラをしてされます。ただし、 `value`プロパティのわりに`defaultValue`プロパティがされています。これは、のレンダリングにのをします。コンポーネントのにするそののは、によってにされません。これがなは、されたコンポーネントをわりにするがあります。

オンラインでフォームとユーザーをむ https://riptutorial.com/ja/reactjs/topic/2884/フォームとユーザー

22: ユーザーインターフェイスソリューション

き

プログラムでされているのユーザーインターフェイスのアイデアにされ、Reactコンポーネントにされたとしましょう。これが「ユーザーインターフェイスソリューション」のトピックです。はappretiatedです。

Examples

ペイン

```
import React from 'react';

class Pane extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {
        return React.createElement(
            'section', this.props
        );
    }
}
```

パネル

```
import React from 'react';

class Panel extends React.Component {
    constructor(props) {
        super(props);
    }

    render(...elements) {
        var props = Object.assign({
            className: this.props.active ? 'active' : '',
            tabIndex: -1
        }, this.props);

        var css = this.css();
        if (css != '') {
            elements.unshift(React.createElement(
                'style', null,
                css
            ));
        }

        return React.createElement(
            'div', props,
```

```
            ...elements
        );
    }

    static title() {
        return '';
    }
    static css() {
        return '';
    }
}
```

シンプルペインとのないはのとおりです。

- パネルがスクリプトによってびされたり、マウスでクリックされたときに、パネルにフォーカスがあります。
- パネルがする`title`コンポーネントごとメソッドを、それがオーバーライドをするのパネルコンポーネントによってすることができる`title` ここでそのは、そのは、にのためにレンダリングにびすことがであるが、こののに`title`をなさない ;
- `css`メソッドでされた々のスタイルシートをむことができます `PANEL.css`からファイルのをあらかじめロードすることができます。

タブ

```
import React from 'react';

class Tab extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {
        var props = Object.assign({
            className: this.props.active ? 'active' : ''
        }, this.props);
        return React.createElement(
            'li', props,
            React.createElement(
                'span', props,
                props.panelClass.title()
            )
        );
    }
}
```

`Tab`インスタンスの`panelClass`プロパティには、にするパネルのクラスがまれているがあります。

PanelGroup

```
import React from 'react';
import Tab from './Tab.js';

class PanelGroup extends React.Component {
```

```
    constructor(props) {
        super(props);
        this.setState({
            panels: props.panels
        });
    }

    render() {
        this.tabSet = [];
        this.panelSet = [];
        for (let panelData of this.state.panels) {
            var tabIsActive = this.state.activeTab == panelData.name;
            this.tabSet.push(React.createElement(
                Tab, {
                    name: panelData.name,
                    active: tabIsActive,
                    panelClass: panelData.class,
                    onMouseDown: () => this.openTab(panelData.name)
                }
            ));
            this.panelSet.push(React.createElement(
                panelData.class, {
                    id: panelData.name,
                    active: tabIsActive,
                    ref: tabIsActive ? 'activePanel' : null
                }
            ));
        }
        return React.createElement(
            'div', { className: 'PanelGroup' },
            React.createElement(
                'nav', null,
                React.createElement(
                    'ul', null,
                    ...this.tabSet
                )
            ),
            ...this.panelSet
        );
    }

    openTab(name) {
        this.setState({ activeTab: name });
        this.findDOMNode(this.refs.activePanel).focus();
    }
}
```

`PanelGroup`インスタンスの`panels`プロパティには、オブジェクトをむがまれているがあります。そこにあるすべてのオブジェクトは、パネルにするなデ―タをします。

- **`name`** - コントロ―ラスクリプトによってされるパネルの。
- **`class`**パネルのクラス。

プロパティをすることをれないでください`activeTab`とタブのに。

タブがダウンしているとき、なパネルがクラスをDOMで`active`していますつまり、されることをします。

`PanelGroup`sをったビュ―の

```
import React from 'react';
import Pane from './components/Pane.js';
import Panel from './components/Panel.js';
import PanelGroup from './components/PanelGroup.js';

class MainView extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {
        return React.createElement(
            'main', null,
            React.createElement(
                Pane, { id: 'common' },
                React.createElement(
                    PanelGroup, {
                        panels: [
                            {
                                name: 'console',
                                panelClass: ConsolePanel
                            },
                            {
                                name: 'figures',
                                panelClass: FiguresPanel
                            }
                        ],
                        activeTab: 'console'
                    }
                )
            ),
            React.createElement(
                Pane, { id: 'side' },
                React.createElement(
                    PanelGroup, {
                        panels: [
                            {
                                name: 'properties',
                                panelClass: PropertiesPanel
                            }
                        ],
                        activeTab: 'properties'
                    }
                )
            )
        );
    }
}

class ConsolePanel extends Panel {
    constructor(props) {
        super(props);
    }

    static title() {
        return 'Console';
    }
}
```

```
class FiguresPanel extends Panel {
    constructor(props) {
        super(props);
    }

    static title() {
        return 'Figures';
    }
}

class PropertiesPanel extends Panel {
    constructor(props) {
        super(props);
    }

    static title() {
        return 'Properties';
    }
}
```

オンラインでユーザーインターフェイスソリューションをむ
https://riptutorial.com/ja/reactjs/topic/8112/ユーザーインターフェイスソリューション

23: リアクションとフロー―の

き

フロー―タイプチェッカー―をしてリアクタンスコンポー―ネントのタイプをチェックする

フロー―|する

Examples

フロー―をしてステ―トレスコンポー―ネントのプロップタイプをチェックする

```
type Props = {
  posts: Array<Article>,
  dispatch: Function,
  children: ReactElement
}

const AppContainer =
  ({ posts, dispatch, children }: Props) => (
    <div className="main-app">
      <Header {...{ posts, dispatch }} />
      {children}
    </div>
  )
```

フロー―をしてのをする

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

type Props = {
  posts: Array<Article>,
  dispatch: Function,
  children: ReactElement
}

class Posts extends Component {
  props: Props;

  render () {
    // rest of the code goes here
  }
}
```

オンラインでリアクションとフロー―のをむ https://riptutorial.com/ja/reactjs/topic/7918/リアクションとフロー―の

24: リアクションルーティング

Examples

Routes.jsファイルの、コンポーネントでのRouter Linkの

ディレクトリにのようなファイルをきます。どのコンポーネントをどのパスにレンダリングするかをします

```
import React from 'react';
import { Route, IndexRoute } from 'react-router';
import New from './containers/new-post';
import Show from './containers/show';

import Index from './containers/home';
import App from './components/app';

export default(
  <Route path="/" component={App}>
    <IndexRoute component={Index} />
    <Route path="posts/new" component={New} />
    <Route path="posts/:id" component={Show} />

  </Route>
);
```

アプリケーションのエントリーポイントであるトップレベルの**index.js**では、のようにこのルータコンポーネントだけをレンダリングするがあります

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Router, browserHistory } from 'react-router';
// import the routes component we created in routes.js
import routes from './routes';


// entry point
ReactDOM.render(
    <Router history={browserHistory} routes={routes} />
  , document.getElementById('main'));
```

これで、アプリケーションで`<a>`タグのわりに`Link`をするだけです。 Linkをすると**React Router**とし、React Routerのをされたリンクにします。これにより**routes.js**でされているしいコンポーネントがレンダリングされます

```
import React from 'react';
import { Link } from 'react-router';

export default function PostButton(props) {
  return (
```

```
    <Link to={`posts/${props.postId}`}>
      <div className="post-button" >
        {props.title}
        <span>{props.tags}</span>
      </div>
    </Link>
  );
}
```

リアクションルーティング

```
import React from 'react';
import { Route, IndexRoute } from 'react-router';

import Index from './containers/home';
import App from './components/app';

//for single Component lazy load use this
const ContactComponent = () => {
  return {
      getComponent: (location, callback)=> {
        require.ensure([], require => {
          callback(null, require('./components/Contact')["default"]);
        }, 'Contact');
      }
    }
};

//for multiple componnets
 const groupedComponents = (pageName) => {
  return {
      getComponent: (location, callback)=> {
        require.ensure([], require => {
          switch(pageName){
            case 'about' :
                callback(null, require( "./components/about" )["default"]);
                    break ;
            case 'tos' :
                callback(null, require( "./components/tos" )["default"]);
                    break ;
          }
        }, "groupedComponents");
      }
    }
};
export default(
  <Route path="/" component={App}>
    <IndexRoute component={Index} />
    <Route path="/contact" {...ContactComponent()} />
    <Route path="/about" {...groupedComponents('about')} />
    <Route path="/tos" {...groupedComponents('tos')} />
  </Route>
);
```

オンラインでリアクションルーティングをむ https://riptutorial.com/ja/reactjs/topic/6096/リアクションルーティング

25: ツール

Examples

リンク

コンポーネントとライブラリをつける。

- のカタログ
- JS.coach

オンラインでツールをむ https://riptutorial.com/ja/reactjs/topic/6595/ツール

26: の

Examples

Reactコンポーネントのは、アプリケーションのデータをしてするためにです。これはJavaScriptオブジェクトとしてされ、 コンポーネントレベルのスコープをち、 コンポーネントのプライベートデータとえることができます。

のでは、コンポーネントの`constructor`でいくつかのをし、それを`render`でしてい`render`。

```
class ExampleComponent extends React.Component {
  constructor(props){
    super(props);

    // Set-up our initial state
    this.state = {
      greeting: 'Hiya Buddy!'
    };
  }

  render() {
    // We can access the greeting property through this.state
    return(
      <div>{this.state.greeting}</div>
    );
  }
}
```

setState

Reactアプリケーションの UIをうなは、 `setState()`をびすことです。このは、したしいとののとでいマージをし、コンポーネントとすべてののをトリガーします。

パラメーター

1. `updater` これは、にマージされるべきのキーとのペア、またはそのようなオブジェクトをすをつオブジェクトです。
2. `callback (optional)` `setState()`がにされたにされる。 `setState()`びしは、Reactによってアトミックになることがされていないため、 `setState()`がにされたにらかのアクションをしたいにです。

`setState`メソッドは、にマージするがあるいくつかのキーとのペアをつオブジェクトか、`prevState`と`props`からされたそのようなオブジェクトをすのどちらかである`updater` `setState`をけれます。

`updater`としてObjectをした`setState()`

```
//
// An example ES6 style component, updating the state on a simple button click.
// Also demonstrates where the state can be set directly and where setState should be used.
//
class Greeting extends React.Component {
    constructor(props) {
        super(props);
        this.click = this.click.bind(this);
        // Set initial state (ONLY ALLOWED IN CONSTRUCTOR)
        this.state = {
            greeting: 'Hello!'
        };
    }
    click(e) {
        this.setState({
              greeting: 'Hello World!'
        });
    }
    render() {
        return(
            <div>
                <p>{this.state.greeting}</p>
                <button onClick={this.click}>Click me</button>
            </div>
        );
    }

}
```

updaterとしてのでのsetState()

```
//
// This is most often used when you want to check or make use
// of previous state before updating any values.
//

this.setState(function(previousState, currentProps) {
  return {
    counter: previousState.counter + 1
  };
});
```

これは、 `setState()`へののびしがされるオブジェクトをするよりもです。これは、のびしをReactでしてにすることができ、のをしてをするのましいです。

```
this.setState({ counter: this.state.counter + 1 });
this.setState({ counter: this.state.counter + 1 });
this.setState({ counter: this.state.counter + 1 });
```

これらのびしは、Reactによって`Object.assign()`をしてバッチされ、カウンターが3ではなく1ずつインクリメントされます。

また、アプローチをして、ロジックをコンポーネントにすることもできます。これにより、ロジックのとがになります。

```
// Outside of component class, potentially in another file/module

function incrementCounter(previousState, currentProps) {
    return {
        counter: previousState.counter + 1
    };
}

// Within component

this.setState(incrementCounter);
```

オブジェクトとコールバックをしてsetState()をびす

```
//
// 'Hi There' will be logged to the console after setState completes
//

this.setState({ name: 'John Doe' }, console.log('Hi there'));
```

のパターン

あなたはstate propsをすべきではありません。それはパターンとなされます 。えば

```
export default class MyComponent extends React.Component {
    constructor() {
        super();

        this.state = {
            url: ''
        }

        this.onChange = this.onChange.bind(this);
    }

    onChange(e) {
        this.setState({
            url: this.props.url + '/days=?' + e.target.value
        });
    }

    componentWillMount() {
        this.setState({url: this.props.url});
    }

    render() {
        return (
            <div>
                <input defaultValue={2} onChange={this.onChange} />

                URL: {this.state.url}
            </div>
        )
    }
}
```

prop `url`は`state`でされ、されます。わりに、をにしてから、 `state`と`props`をしてなパスをすることをします。

```
export default class MyComponent extends React.Component {
    constructor() {
        super();

        this.state = {
            days: ''
        }

        this.onChange = this.onChange.bind(this);
    }

    onChange(e) {
        this.setState({
            days: e.target.value
        });
    }

    render() {
        return (
            <div>
                <input defaultValue={2} onChange={this.onChange} />

                URL: {this.props.url + '/days?=' + this.state.days}
            </div>
        )
    }
}
```

なぜなら、Reactアプリケーションでは、ののソースをちたいからです。つまり、すべてのデータは1つのコンポーネントのであり、1つのコンポーネントだけです。データをそのにし、データをのコンポーネントにをしてするのは、このコンポーネントのです。

のでは、MyComponentクラスとそののが、そののので「url」をしています。 MyComponentでstate.urlをすると、これらのはにされません。々はののをってしまい、アプリケーションをじてデータのれをすることがますますになっています。これを2のとすると、urlはコンポーネントののみされ、MyComponentのとしてされるため、ののソースをします。

、イベント、およびされたコントロール

に、「された」フィールドをつReactコンポーネントのをします。フィールドのがされるたびに、フィールドのしいでコンポーネントのをするイベントハンドラがびされます。イベントハンドラで`setState`をびすと、 `setState`のコンポーネントのを`render`するびしがトリガされ`render`。

```
import React from 'react';
import {render} from 'react-dom';


class ManagedControlDemo extends React.Component {

  constructor(props){
    super(props);
```

```
    this.state = {message: ""};
  }

  handleChange(e){
    this.setState({message: e.target.value});
  }

  render() {
    return (
      <div>
        <legend>Type something here</legend>
          <input
            onChange={this.handleChange.bind(this)}
            value={this.state.message}
            autoFocus />
        <h1>{this.state.message}</h1>
      </div>
   );
  }
}


render(<ManagedControlDemo/>, document.querySelector('#app'));
```

のにすることはにです。ユーザーがフィールドのをするたびに

- `handleChange`がびされます
- `setState`がびされます
- `render`がびされる

ポップクイズ、フィールドにをした、どのDOMがされるか

1. これらのすべて - トップレベルのdiv、、、h1
2. とh1のみ
3. も
4. DOMはですか

これをここでしてえをつけることができます

オンラインでのをむ https://riptutorial.com/ja/reactjs/topic/1816/の

27: の

React 15.5では、PropTypesコンポーネントはのnpmパッケージ、つまり'prop-types'にし、PropTypeをするときはのimportがです。のについては、のをしてください https://facebook.github.io/react/blog/2017/04/07/react-v15.5.0.html

Examples

き

`props`コンポーネントにコンポーネントからのデータとメソッドをすためにされます。

---

propsについてのいこと

1. らはです。
2. これにより、なコンポーネントをすることができます。

な

```
class Parent extends React.Component{
  doSomething(){
     console.log("Parent component");
  }
  render() {
    return <div>
         <Child
           text="This is the child number 1"
           title="Title 1"
           onClick={this.doSomething} />
         <Child
           text="This is the child number 2"
           title="Title 2"
           onClick={this.doSomething} />
       </div>
  }
}

class Child extends React.Component{
  render() {
    return <div>
       <h1>{this.props.title}</h1>
       <h2>{this.props.text}</h2>
      </div>
  }
}
```

このでわかるように、 `props`おかげでなコンポーネントをできます。

デフォルトの

defaultPropsでは、コンポーネントのpropsデフォルトまたはフォールバックをできます。defaultPropsは、でなるビューからコンポーネントをびすときにですが、いくつかのビューではなるをすがあります。

ES5

```
var MyClass = React.createClass({
  getDefaultProps: function() {
    return {
      randomObject: {},
      ...
    };
  }
}
```

ES6

```
class MyClass extends React.Component {...}

MyClass.defaultProps = {
    randomObject: {},
    ...
}
```

ES7

```
class MyClass extends React.Component {
    static defaultProps = {
        randomObject: {},
        ...
    };
}
```

getDefaultProps()またはdefaultPropsのはキャッシュされ、 this.props.randomObjectがコンポーネントによってされていないにをつことをするためにされます。

PropType

propTypesすると、コンポーネントになpropsとそのをすることができます。あなたのコンポーネントはpropTypesをせずにしますが、コンポーネントをよりみやすくし、コンポーネントをんでいるのにドキュメンテーションとしてさせるようにし、にReactはあなたにしますあなたがしたとなるタイプのをしてください。

いくつかのなpropTypesおよびになpropTypesは、

```
  optionalArray: React.PropTypes.array,
  optionalBool: React.PropTypes.bool,
  optionalFunc: React.PropTypes.func,
```

```
  optionalNumber: React.PropTypes.number,
  optionalObject: React.PropTypes.object,
  optionalString: React.PropTypes.string,
  optionalSymbol: React.PropTypes.symbol
```

`isRequired`をの`propType`アタッチするは、そのコンポーネントのインスタンスをするにそのプロップをするがあります。 な `propTypes`をしないと、コンポーネントインスタンスをできません。

ES5

```
var MyClass = React.createClass({
  propTypes: {
    randomObject: React.PropTypes.object,
    callback: React.PropTypes.func.isRequired,
    ...
  }
}
```

ES6

```
class MyClass extends React.Component {...}

MyClass.propTypes = {
    randomObject: React.PropTypes.object,
    callback: React.PropTypes.func.isRequired,
    ...
};
```

ES7

```
class MyClass extends React.Component {
     static propTypes = {
        randomObject: React.PropTypes.object,
        callback: React.PropTypes.func.isRequired,
        ...
    };
}
```

よりなの

に、 `PropTypes`すると、よりなをすることができます

オブジェクトの

```
...
    randomObject: React.PropTypes.shape({
        id: React.PropTypes.number.isRequired,
        text: React.PropTypes.string,
    }).isRequired,
...
```

オブジェクトのの

```
...
    arrayOfObjects: React.PropTypes.arrayOf(React.PropTypes.shape({
        id: React.PropTypes.number.isRequired,
        text: React.PropTypes.string,
    })).isRequired,
...
```

スプレッドオペレーターをしてをす

のわりに

```
var component = <Component foo={this.props.x} bar={this.props.y} />;
```

プロパティは、することができ、のとしてすがあるは...すべてのをするES6でのためのサポートを。コンポーネントはこのようになります。

```
var component = <Component {...props} />;
```

すオブジェクトのプロパティは、コンポーネントのにコピーされることにしてください。

はです。のはのをきします。

```
var props = { foo: 'default' };
var component = <Component {...props} foo={'override'} />;
console.log(component.props.foo); // 'override'
```

のケースでは、スプレッドをしてコンポーネントにのだけをすことができます。に、からびをできます。

のコンポーネントがたくさんのをとするが、それらをつずつすことはましくないときににです。

```
const { foo, bar, other } = this.props // { foo: 'foo', bar: 'bar', other: 'other' };
var component = <Component {...{foo, bar}} />;
const { foo, bar } = component.props
console.log(foo, bar); // 'foo bar'
```

との

コンポーネントの「」コンポーネントは、なprops.childrenできます。このは、コンポーネントをまとめて「」するのににち、JSXマークアップをDOMのされたをよりにさせることができます。

```
var SomeComponent = function () {
    return (
        <article className="textBox">
            <header>{this.props.heading}</header>
            <div className="paragraphs">
                {this.props.children}
```

```
            </div>
        </article>
    );
}
```

でコンポーネントをするときにののサブをめることができます。

```
var ParentComponent = function () {
    return (
        <SomeComponent heading="Amazing Article Box" >
            <p className="first"> Lots of content </p>
            <p> Or not </p>
        </SomeComponent>
    );
}
```

Props.childrenは、コンポーネントによってすることもできます。 props.children はであってもでなくてもよいので 、ReactはReact.Childrenとしてユーティリティをします。 をの<section>でりしたいは、のをえてみましょう

```
var SomeComponent = function () {
    return (
        <article className="textBox">
            <header>{this.props.heading}</header>
            <div className="paragraphs">
                {React.Children.map(this.props.children, function (child) {
                    return (
                        <section className={child.props.className}>
                            React.cloneElement(child)
                        </section>
                    );
                })}
            </div>
        </article>
    );
}
```

React.cloneElementをしての<p>タグからをすることにしてください。はなので、これらのはできません。わりに、これらののないクローンをするがあります。

さらに、ループにをするときは、ReactがRerender にどのようにをするかをし、ループにされたにグローバルにな<key>をめることをくしてください。

コンポーネントのタイプの

コンポーネントをするときにコンポーネントのタイプをることははなもあります。のコンポーネントをするためにReact `Children.map` utilをうことができます

```
React.Children.map(this.props.children, (child) => {
  if (child.type === MyComponentType) {
    ...
  }
});
```

オブジェクトは、のコンポーネントとできる`type`プロパティをします。

オンラインでのをむ https://riptutorial.com/ja/reactjs/topic/2749/の

28: の

き

のキーは、にじのDOMのリストをするためにされます。

したがって、をりしてliのリストをする、liはkeyプロパティでされたのをとします。これは、データベースアイテムのIDまたはのインデックスになります。

アレイインデックスをキーとしてすることは、アレイがとともにするにはにされません。 React Docsから

> のとして、のアイテムのインデックスをキーとしてすことができます。アイテムのべえがもわれないでも、リオーダーはくなります。

これにするい https : //medium.com/@robinpokorny/index-as-a-key-is-an-anti-pattern-e0349aece318

Examples

のidをう

ここでは、コンポーネントのにされるToDoのリストがあります。

ToDoには、textプロパティとidプロパティがあります。 idプロパティがバックエンドデータストアからており、のであるとします。

```
todos = [
  {
    id: 1,
    text: 'value 1'
  },
  {
    id: 2,
    text: 'value 2'
  },
  {
    id: 3,
    text: 'value 3'
  },
  {
    id: 4,
    text: 'value 4'
  },
];
```

がにそれをできるように、リストのキーを`todo-${todo.id}`しました

```
render() {
  const { todos } = this.props;
  return (
    <ul>
      { todos.map((todo) =>
        <li key={ `todo-${todo.id}` }>
          { todo.text }
        </li>
      }
    </ul>
  );
}
```

インデックスの

のデータベースIDをっていないは、のようにのインデックスをすることもできます。

```
render() {
  const { todos } = this.props;
  return (
    <ul>
      { todos.map((todo, index) =>
        <li key={ `todo-${index}` }>
          { todo.text }
        </li>
      }
    </ul>
  );
}
```

オンラインでのをむ https://riptutorial.com/ja/reactjs/topic/9805/の

29: フォーム

Examples

されたコンポーネント

コンポーネントはにバインドされ、そのはイベントベースのコールバックをしてコードでされます。

```
class CustomForm extends React.Component {
constructor() {
    super();
    this.state = {
        person: {
            firstName: '',
            lastName: ''
        }
    }
}

handleChange(event) {
    let person = this.state.person;
    person[event.target.name] = event.target.value;
    this.setState({person});
}

render() {
    return (
        <form>
            <input
            type="text"
            name="firstName"
            value={this.state.firstName}
            onChange={this.handleChange.bind(this)} />

            <input
            type="text"
            name="lastName"
            value={this.state.lastName}
            onChange={this.handleChange.bind(this)} />
        </form>
    )
}
```

}

このでは、のpersonオブジェクトでをします。その、2つののをpersonオブジェクトの々のキーにバインドします。ユーザーがタイプすると、 `handleChange`でをします。コンポーネントのはにバインドされているので、 `setState()`びすことでユーザーがしたときにレンダリングできます。

コントロールされたコンポーネントをうときに`setState()`を`setState()`ないと、ユーザーはするようになりますが、Reactのみがをするのでがされません。

のはpersonオブジェクトのキーのとじであることにすることもです。これにより、ここにすのをできます。

```
handleChange(event) {
    let person = this.state.person;
    person[event.target.name] = event.target.value;
    this.setState({person});
}
```

`person[event.target.name]`とじです`person.firstName || person.lastName`。 もちろん、これはどのがされているかによってなります。ユーザーがする、をしてをキーのとさせることができないため、ユーザーなしonChangeがどこからびされているのか。

オンラインでフォームをむ https://riptutorial.com/ja/reactjs/topic/8047/フォーム

30: の

Examples

のコンポーネントをコンパイルし、たちのWebページにレンダリングできるようにしたい

ファイル src / index.jsx

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

class ToDo extends React.Component {
    render() {
        return (<div>I am working</div>);
    }
}

ReactDOM.render(<ToDo />, document.getElementById('App'));
```

すべてのをインストールする

```
# install react and react-dom
$ npm i react react-dom --save

# install webpack for bundling
$ npm i webpack -g

# install babel for module loading, bundling and transpiling
$ npm i babel-core babel-loader --save

# install babel presets for react and es6
$ npm i babel-preset-react babel-preset-es2015 --save
```

Webpackをする

ディレクトリのルートに`webpack.config.js`ファイルをします。

ファイル webpack.config.js

```
module.exports = {
    entry: __dirname + "/src/index.jsx",
    devtool: "source-map",
    output: {
        path: __dirname + "/build",
        filename: "bundle.js"
    },
    module: {
        loaders: [
            {test: /\.jsx?$/, exclude: /node_modules/, loader: "babel-loader"}
        ]
```

```
    }
}
```

ベイベルの

ディレクトリのルートに`.babelrc`ファイルをします。

ファイル .babelrc

```
{
    "presets": ["es2015","react"]
}
```

コンポーネントをするHTMLファイル

プロジェクトディレクトリのルートになhtmlファイルをする

ファイル index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <div id="App"></div>
    <script src="build/bundle.js" charset="utf-8"></script>
  </body>
</html>
```

コンポーネントをトランスパイルしてバンドルする

webpackをすると、コンポーネントをバンドルできます。

```
$ webpack
```

これにより、 `build`ディレクトリにファイルがされます。

ブラウザでHTMLページをき、のコンポーネントをする

オンラインでのをむ https://riptutorial.com/ja/reactjs/topic/7480/の

31: コンポーネント

き

コンポーネントして「HOC」は、なコードをしてコンポーネントをするためにされるアプリケーションのパターンです。のコンポーネントクラスにやをすることができます。

HOCは、としてコンポーネントをけれ、されたをつしいコンポーネントをするな javascriptです。

HOCは、サードパーティのライブラリでよくされています。 Redux など。

Examples

コンポーネント

コンポーネントがマウントされるたびにconsole.logにしたいとしましょう

hocLogger.js

```
export default function hocLogger(Component) {
  return class extends React.Component {
    componentDidMount() {
      console.log('Hey, we are mounted!');
    }
    render() {
      return <Component {...this.props} />;
    }
  }
}
```

あなたのコードでこのHOCをしてください

MyLoggedComponent.js

```
import React from "react";
import {hocLogger} from "./hocLogger";

export class MyLoggedComponent extends React.Component {
    render() {
        return (
            <div>
                This component get's logged to console on each mount.
            </div>
        );
    }
}

// Now wrap MyLoggedComponent with the hocLogger function
export default hocLogger(MyLoggedComponent);
```

をするコンポーネント

たとえば、ユーザーがログインしているにのみされるコンポーネントがあるとします。

そこで、renderのをチェックするHOCをします。

AuthenticatedComponent.js

```
import React from "react";

export function requireAuthentication(Component) {
    return class AuthenticatedComponent extends React.Component {

        /**
         * Check if the user is authenticated, this.props.isAuthenticated
         * has to be set from your application logic (or use react-redux to retrieve it from
global state).
         */
        isAuthenticated() {
            return this.props.isAuthenticated;
        }

        /**
         * Render
         */
        render() {
            const loginErrorMessage = (
                <div>
                    Please <a href="/login">login</a> in order to view this part of the
application.
                </div>
            );

            return (
                <div>
                    { this.isAuthenticated === true ? <Component {...this.props} /> :
loginErrorMessage }
                </div>
            );
        }
    };
}

export default requireAuthentication;
```

ユーザーからにするがあるこのコンポーネントのコンポーネントをします。

MyPrivateComponent.js

```
import React from "react";
import {requireAuthentication} from "./AuthenticatedComponent";

export class MyPrivateComponent extends React.Component {
    /**
     * Render
     */
    render() {
```

```
        return (
            <div>
                My secret search, that is only viewable by authenticated users.
            </div>
        );
    }
}

// Now wrap MyPrivateComponent with the requireAuthentication function
export default requireAuthentication(MyPrivateComponent);
```

このののは、 ここでします 。

オンラインでコンポーネントをむ https://riptutorial.com/ja/reactjs/topic/9819/コンポーネント

クレジット

S. No		Contributors
1	Reactをいめる	Adam, Adrián Daraš, Alex, Alex Young, Anuj, Bart Riordan, Cassidy, Community, Daksh Gupta, Dave Kaye, diabolicfreak, DMan, Donald, Everettss, Gianluca Esposito, himanshuIIITian, hyde, Ilya Lyamkin, Inanc Gumus, ivarni, jengeb, jolyonruss, Jon Chan, JordanHendrix, juandemarco, Kaloyan Kosev, Konstantin Grushetsky, Maksim, Marty, MaxPRafferty, Md. Nahiduzzaman Rose, Md.Sifatul Islam, Ming Soon, MMachinegun, Nick Bartlett, orvi, paqash, Prakash, rossipedia, Shabin Hashim, Simplans, Sunny R Gupta, TheShadowbyte, Timo, Tushar Khanna, user2314737
2	AJAXびしにする	adamboro, Fabian Schultz, Jason Bourne, lifeiscontent, McGrady, Sunny R Gupta
3	FluxによるReactJSの	vintproykt
4	jQueryでのReactJSの	Kousha, Shuvo Habib
5	JSX	Kaloyan Kosev, Ming Soon
6	React Boilerplate [リアクション+バベル+ウェブパック]	Mihir, parlad neupane, Tien Do
7	React、Webpack Typescriptのインストーール	Aron
8	React.Componentをする	Kaloyan Kosev, leonardoborges, Michael Peyper, pwolaq, Qianyue, sqzaman
9	Reactでキーをするとその	Sammy I.
10	Reduxにする	Jim
11	TypescriptでのReactJSの	Everettss, John Ruddell, kevgathuku, Leone, Rajab Shakirov

12	インストール	Rene R, Ruairi O'Brien
13	コンポーネント	akashrajkn, Anuj, Bart Riordan, Bond, Brandon Roberts, Denis Ivanov, Diego V, DMan, Evan Hammer, Everettss, goldbullet, GordyD, hmnzr, Ilya Lyamkin, ivarni, Jagadish Upadhyay, jbmartinez, John Ruddell, jolyonruss, Jon Chan, jonathangoodman, JordanHendrix, justabuzz, k170, Kousha, Kyle Richardson, m_callens, Maayan Glikser, Michael Peyper, Paul Graffam, philpee2, QoP, Radu Brehar, Sai Vikas, sjmarshy , Timo, Vlad Bezden, WooCaSh, Zakaria Ridouh, zurfyx
14	コンポーネントライフサイクルの	Alex Young, Alexg2195, Anuj, Ashari, Everettss, F. Kauder, irrigator, John Ruddell, QoP, Salman Saleem, Saravana, Siddharth, skav, Timo, ultrasamad, Vivian, WitVault
15	コンポーネントの	Random User
16	サーバーレンダリングの	Adrián Daraš, MauroPorrasP
17	ステートレスコンポーネント	Adam, Mark Lapierre, Mayank Shukla, Valter Júnior
18	どのようにな Webpack、し、をする	Bart Riordan, Tien Do, Zac Braddy
19	パフォーマンス	Aditya Singh, lustoykov, thibmaek
20	フォームとユーザー	Everettss, Henrik Karlsson, ivarni, Timo
21	ユーザーインターフェイスソリューション	vintproykt
22	リアクションとフローの	JimmyLv, lifeiscontent, Rifat, Rory O'Kane
23	リアクションルーティング	abhirathore2006, Robeen
24	ツール	brillout
25	の	Alex Young, Alexander, Brad Colthurst, Everettss, Kousha, Kyle Richardson, QoP, skav, Timo
26	の	Ahmad, Anuj, Danillo Corvalan, Everettss, Faktor 10, Fellow Stranger, hansn, Ilya Lyamkin, Jack7, Jagadish Upadhyay,

		JimmyLv, MaxPRafferty, QoP, Sergii Bishyr, vintproykt, WitVault, zbynour
27	の	Dennis Stücken, thibmaek
28	フォーム	promisified
29	の	ghostffcode, Tien Do
30	コンポーネント	Dennis Stücken