eBook Gratuit

APPRENEZ React Router

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#react-router

Table des matières

À propos

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

It is an unofficial and free React Router 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 Router.

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

Chapitre 1: Démarrer avec React Router

Remarques

React Router est une bibliothèque de routage populaire et complète pour React.js qui maintient l'interface utilisateur synchronisée avec l'URL. Il prend en charge le chargement de code paresseux, la correspondance de route dynamique et la gestion de la transition de lieu. Il a été initialement inspiré du routeur Ember.

TODO: cette section devrait également mentionner tous les grands sujets dans react-router, et établir un lien avec les sujets connexes. La documentation de react-router étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Date de sortie
1.0.0	2015-11-09
1.0.1	2015-12-05
1.0.2	2015-12-08
1.0.3	2015-12-23
2.0.0	2016-02-10
2.0.1	2016-03-09
2.1.0	2016-04-11
2.1.1	2016-04-11
2.2.0	2016-04-13
2.2.1	2016-04-14
2.2.2	2016-04-14
2.2.3	2016-04-15
2.2.4	2016-04-15
2.3.0	2016-04-18
2.4.0	2016-04-28
2.4.1	2016-05-19

Version	Date de sortie
2.5.0	2016-06-22
2.5.1	2016-06-23
2.5.2	2016-07-01
2.6.0	2016-07-18
2.6.1	2016-07-29
2.7.0	2016-08-20
2.8.0	2016-09-09
2.8.1	2016-09-13
3.0.0	2016-10-24
3.0.1	2017-01-12
3.0.2	2017-01-18
3.0.3	2017-03-28
3,0.4	2017-04-09
3.0.5	2017-04-10
4.0.0	2017-04-12
4.1.0	2017-04-17
4.1.1	2017-04-12

Examples

Installation et configuration

Pour installer React Router, lancez simplement la commande npm

```
npm install --save react-router
```

Et tu as fini. C'est littéralement tout ce que vous avez à faire pour installer le routeur de réaction.

Veuillez noter que `react-router` dépend de la `react` . Assurez-vous donc que vous installez également `react` .

Installer:

en utilisant un transpiler ES6, comme babel

```
import { Router, Route, Link } from 'react-router'
```

ne pas utiliser un transpiler ES6

```
var Router = require('react-router').Router
var Route = require('react-router').Route
var Link = require('react-router').Link
```

Installation à l'aide de la construction UMD

Une version est également disponible sur npmcdn . Vous pouvez inclure le script comme ceci:

```
<script src="https://npmcdn.com/react-router/umd/ReactRouter.min.js"></script>
```

La bibliothèque sera disponible globalement sur `window.ReactRouter` .

Bonjour tout le monde avec React et React Router

Une fois que vous avez installé `react` et `react-router` , il est temps d'utiliser les deux ensemble.

La syntaxe est très simple, vous spécifiez l' `url` et le `component` vous souhaitez rendre lorsque cette URL est ouverte

```
<Route path="hello" component={ HelloComponent } />
```

Cela signifie que lorsque le chemin d'URL est `hello` , `HelloComponent` le composant `HelloComponent`

FILENAME: `app.js`

```
'use strict';

import React from 'react';
import { render } from 'react-dom';
import { Router, browserHistory, Link } from 'react-router';

// These are just demo components which render different text.

let DashboardPage = () => (
  <div>
    <h1>Welcome User</h1>
    <p>This is your dashboard and I am an example of a stateless functional component.</p>
    <Link to="/settings">Goto Settings Page</Link>
  </div>
)

let SettingsPage = () => (
  <div>
    <h1>Manage your settings</h1>
    <p>display the settings form fields here...or whatever you want</p>
    <Link to="/">Back to Dashboard Page</Link>
```

```
  </div>
)

let AuthLoginPage = () => (
  <div>
    <h1>Login Now</h1>
    <div>
      <form action="">
        <input type="text" name="email" placeholder="email address" />
        <input type="password" name="password" placeholder="password" />
        <button type="submit">Login</button>
      </form>
    </div>
  </div>
)

let AuthLogoutPage = () => (
  <div>
    <h1>You have been successfully logged out.</h1>
    <div style={{ marginTop: 30 }}>
      <Link to="/auth/login">Back to login page</Link>
    </div>
  </div>
)

let ArticlePage = ({ params }) => (
  <h3>Article {params.id}</h3>
)

let PageNotFound = () => (
  <div>
    <h1>The page you're looking for doesn't exist.</h1>
  </div>
)

// Here we pass Router to the render function.
render( (
    <Router history={ browserHistory }>

      <Route path="/" component={ DashboardPage } />
      <Route path="settings" component={ SettingsPage } />

      <Route path="auth">
        <IndexRoute component={ AuthLoginPage } />
        <Route path="login" component={ AuthLoginPage } />
        <Route path="logout" component={ AuthLogoutPage } />
      </Route>

      <Route path="articles/:id" component={ ArticlePage } />

      <Route path="*" component={ PageNotFound } />

    </Router>
  ), document.body );
```

Paramètres d'itinéraire : Le chemin du routeur peut être configuré pour prendre des paramètres afin que nous puissions lire la valeur du paramètre au composant. Le chemin dans `<Route path="articles/:id" component={ ArticlePage } />` a un `/:id` . Cette variable `id` pour objet le paramètre path et est accessible au composant `ArticlePage` en utilisant `{props.params.id}` .

Si nous visitons `http://localhost:3000/#/articles/123` alors `{props.params.id}` au composant `ArticlePage` sera résolu à 123. Mais la visite de l'url `http://localhost:3000/#/articles` , ne sera pas travailler car il n'y a pas de paramètre id.

Le paramètre route peut être rendu **facultatif** en l'écrivant entre une paire de parenthèses:
`<Route path="articles(/:id)" component={ ArticlePage } />`

Si vous souhaitez utiliser des **sous-routes** , vous pouvez le faire

```
<Route path="path" component={ PathComponent }>
  <Route path="subpath" component={ SubPathComponent } />
</Route>
```

- quand `/path` est accédé, `PathComponent` sera rendu
- lorsque `/path/subpath` est accédé, `PathComponent` sera rendu et `SubPathComponent` sera transmis en tant que `props.children`

Vous pouvez utiliser `path="*"` pour intercepter tous les itinéraires inexistants et rendre la `404 page not found` .

Commencer

Cette prise *en main* suppose que vous travaillez avec create-react-app ou quelque chose de similaire en utilisant Babel et tous les autres produits disponibles.

Consultez également la documentation excellente ici .

Tout d'abord, installez react-router-dom:

`npm install react-router-dom` ou `yarn add react-router-dom` .

Ensuite, créez un composant qui contient une barre de navigation de base avec deux éléments et des pages de base:

```
import React from 'react'
import { BrowserRouter, Route, Link } from 'react-router-dom'

const Home = () => (
  <div>
    <p>We are now on the HOME page</p>
  </div>
)

const About = () => (
  <div>
    <p>We are now on the ABOUT page</p>
  </div>
)

const App = () => (
  <BrowserRouter>
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
```

```
        <li><Link to="/about">About</Link></li>
      </ul>
      <hr/>
      <Route path="/" component={Home}/>
      <Route path="/about" component={About}/>
    </div>
  </BrowserRouter>
)
export default App
```

Passons pas à pas dans ce code:

- `import React from 'react'` : Assurez-vous d'importer `React`
- `import { BrowserRouter as Router, Route, Link } from 'react-router-dom'` divisé:
- `BrowserRouter` est le routeur proprement dit. Veillez à envelopper votre composant dans le composant `BrowserRouter` .
- `Route` est un itinéraire particulier qui peut être navigué vers
- `Link` est un composant qui génère une balise `<a href="...">` que vous pouvez utiliser comme lien hypertexte.

- `const Home` est une fonction qui retourne la page d'accueil.
- `const About` est une fonction qui renvoie la page About.

- `const App` est le composant principal:
- `<BrowserRouter>` est le composant JSX qui encapsule les composants dans lesquels vous souhaitez utiliser le composant `<Route>` .
- ' `is a single element to wrap all JSX inside the` BrowserRouter`.
- `<ul>` est la barre de navigation. Il contient un lien vers Home et un lien vers About.
- `<li><Link to="/">Home</Link></li>` liens vers la page d'accueil. Vous pouvez voir que, puisque le lien fait référence à "/", un chemin relatif vide rend la page d'accueil.
- `<li><Link to="/about">About</Link></li>` liens vers la page À propos.
- `<Route path="/" component={Home}/>` décrit quel composant doit être rendu si le chemin relatif est `"/"` .
- `<Route path="/about" component={About}/>` décrit quel composant doit être rendu si le chemin relatif est `"/about"` .

Beaucoup d'apprendre d'ici, mais j'espère que cela explique les principes fondamentaux, alors à partir de là, vous pouvez continuer vos apprentissages.

Lire Démarrer avec React Router en ligne: https://riptutorial.com/fr/react-router/topic/5546/demarrer-avec-react-router

Chapitre 2: Réactivez le routeur 4 avec TypeScript

Introduction

Quelques exemples d'intégration de TypeScript avec `react-router` 4.x.

L'objectif est de préserver autant de sécurité que possible.

Comment faire cela avec TypeScript n'est pas évident lorsque vous suivez la documentation du projet.

Examples

Routage de base

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Route, BrowserRouter as Router, Link } from 'react-router-dom';

class Home extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>HOME</div>
        <div><Link to='/one'>Goto Page One</Link></div>
        <div><Link to='/two'>Goto Page Two</Link></div>
      </div>);
  }
}

class One extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>ONE</div>
        <Link to='/'>Goto Home</Link>
      </div>
    );
  }
}

class Two extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>TWO</div>
        <Link to='/'>Goto Home</Link>
      </div>
    );
  }
}
```

```
ReactDOM.render(
  <Router>
    <div>
      <Route exact path="/" component={Home} />
      <Route exact path="/one" component={One} />
      <Route exact path="/two" component={Two} />
    </div>
  </Router>

  , document.getElementById('root')
);
```

Cela représente un routage de `react-router` très basique avec les classes TypeScript `React.Component` .

Routage avec des paramètres typés

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Route, BrowserRouter as Router, Link, match } from 'react-router-dom';

// define React components for multiple pages
class Home extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>HOME</div>
        <div><Link to='/details/id123'>Goto Details</Link></div>
      </div>);
  }
}

interface DetailParams {
  id: string;
}

interface DetailsProps {
  required: string;
  match?: match<DetailParams>;
}

class Details extends React.Component<DetailsProps, any> {
  render() {
    const match = this.props.match;
    if (match) {
      return (
        <div>
          <div>Details for {match.params.id}</div>
          <Link to='/'>Goto Home</Link>
        </div>
      );
    } else {
      return (
        <div>
          <div>Error Will Robinson</div>
          <Link to='/'>Goto Home</Link>
        </div>
      )
```

```
    }
  }
}

ReactDOM.render(
  <Router>
    <div>
      <Route exact path="/" component={Home} />
      <Route exact path="/details/:id" component={(props) => <Details required="some string"
{...props} />} />
    </div>
  </Router>

  , document.getElementById('root')
);
```

Afin de préserver la sécurité de type pour le composant `Details` qui a une propriété requise appelée `required` , la définition `<Route>` définit un composant basé sur une fonction anonyme qui compose un nouveau composant de type `<Details>` et spécifie la propriété `required` .

L'opérateur spread est utilisé pour `props` les `props` transmises au composant basé sur la fonction anonyme sur le composant `<Details>` composé.

La propriété de `match` est définie comme facultative, car elle est remplie dynamiquement par `react-router` . Nous ne pouvons malheureusement pas la définir comme propriété requise. Cela signifie qu'un protecteur de type est requis lors de l'accès aux valeurs ultérieurement.

Routage avec paramètres typés et propriétés injectées

Cette solution est plus complexe, exploitant des décorateurs TypeScript personnalisés qui injectent des données de `match` , d' `history` et / ou de `location` dans votre classe `React.Component` , ce qui vous permet de `React.Component` une sécurité de type complète sans protection de type, comme dans l'exemple précédent.

// `Routed.ts` - définit les décorateurs

```
import { RouteComponentProps, match } from 'react-router';
import { History, Location } from 'history';

// re-export for convenience, uppercase match to be in line with everything else
export { History, Location, match as Match };

// names for the three types we support injecting
type InjectionPropType = 'location' | 'history' | 'match';

// holder for a given property to be injected as a specific type
class InjectionProp {
  prop: string;
  type: InjectionPropType;
}

// a store, key = class name (constructor.name) and array of InjectionProp's for that class
// this will be filled in by the three property decorators @RoutedMatch, @RoutedLocation and
@RoutedHistory
class InjectionStore {
```

```
  [key: string]: InjectionProp[];
}

// instance of the store
const store: InjectionStore = {};

// type guard for RouteComponentProps
function instanceOfRouteProps<P>(object: any): object is RouteComponentProps<P> {
  return 'match' in object && 'location' in object && 'history' in object;
}

// class level decorator, wraps the constructor with custom one which injects
// values into instances based on the InjectionStore instance
export function Routed<T extends { new (...args: any[]): {} }>(constructor: T) {

  // get the class name from the constructor
  const className = (constructor as any).name;

  // return a new class with a new constructor which calls super(..)
  return class extends constructor {

    constructor(...args: any[]) {
      super(args);

      // if there is a React props passed as arg[0]
      if (args.length >= 1) {

        const routeProps = args[0];

        // check type guard to see if the React props is enriched with RouteComponentProps by
react-router
        if (instanceOfRouteProps(routeProps)) {
          // check if the current class has any registered properties to be injected
          if (store[className]) {
            const injectionProps = store[className];
            // iterate over properties to inject
            for (let i = 0; i < injectionProps.length; i++) {
              const injectionProp = injectionProps[i];
              // inject the specified property with the appropriate type
              switch (injectionProp.type) {
                case 'match':
                  (this as any)[injectionProp.prop] = routeProps.match;
                  break;
                case 'history':
                  (this as any)[injectionProp.prop] = routeProps.history;
                  break;
                case 'location':
                  (this as any)[injectionProp.prop] = routeProps.location;
                  break;
              }
            }
          }
        }
      }
    }
  }
}

// generic property decorator, registers a classes property for inject in the store above
function RoutedInjector(proto: any, prop: string, type: InjectionPropType): any {
  const className = proto.constructor.name;
```

```
  if (!store.hasOwnProperty(className)) {
    store[className] = [];
  }
  store[className].push({
    prop: prop,
    type: type
  });
}

// property decorator for Match instances
export function RoutedMatch(proto: any, prop: string): any {
  RoutedInjector(proto, prop, 'match');
}

// property decorator for Location instances
export function RoutedLocation(proto: any, prop: string): any {
  RoutedInjector(proto, prop, 'location');
}

// property decorator for History instances
export function RoutedHistory(proto: any, prop: string): any {
  RoutedInjector(proto, prop, 'history');
}
```

// index.ts

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Route, BrowserRouter as Router, Link, match } from 'react-router-dom';

import { History, Location, Match, Routed, RoutedHistory, RoutedLocation, RoutedMatch } from
'./Routed';

// define React components for multiple pages
class Home extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>HOME</div>
        <div><Link to='/details/id123'>Goto Details</Link></div>
      </div>);
  }
}

interface DetailParams {
  id: string;
}

interface DetailsProps {
  required: string;
}

@Routed
class Details extends React.Component<DetailsProps, any> {

  @RoutedMatch
  match: Match<DetailParams>;

  @RoutedLocation
  location: Location;
```

```
  @RoutedHistory
  history: History;

  render() {
    return (
      <div>
        <div>Details for {this.match.params.id} on location {this.location.pathname}</div>
        <span
          onClick={(e) => this.history.push('/')}
          style={{ textDecoration: 'underline', cursor: 'pointer' }}
        >Goto Home</span>
      </div>
    );

  }
}

ReactDOM.render(
  <Router>
    <div>
      <Route exact path="/" component={Home} />
      <Route exact path="/details/:id" component={(props) => <Details required="some string"
{...props} />} />
    </div>
  </Router>

  , document.getElementById('root')
);
```

Cet exemple utilise des décorateurs personnalisés pour injecter des instances de données spécifiques à `@RoutedMatched` `react-router` l'aide des décorateurs de propriétés `@RoutedMatched` , `@RoutedLocation` et `@RoutedHistory` sur un `React.Component` décoré avec `@Routed` .

Le résultat final est que les types de données spécifiques à `React.Component` `react-router` sont maintenant entièrement découplés des propriétés et de l'état personnalisés de `React.Component` . Un avantage supplémentaire est qu'ils ne sont plus des propriétés facultatives, ce qui signifie que vous n'avez pas besoin de gardes de type pour accéder en toute sécurité à leurs valeurs.

Le paramètre `history` affiche l'accès à l'objet `History` partir du package `history` (une dépendance de `react-router`), qui peut être utilisé, comme indiqué, pour manipuler par programme l'historique du navigateur.

Le paramètre d' `location` porte des informations sur l'emplacement actuel

Lire Réactivez le routeur 4 avec TypeScript en ligne: https://riptutorial.com/fr/react-router/topic/9815/reactivez-le-routeur-4-avec-typescript

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec React Router	Alex, Community, Diego V, imdzeeshan, Ming Soon, Random User, Sventies, Vishnu Y S, YasserKaddour
2	Réactivez le routeur 4 avec TypeScript	Alex