JavaScript und JavaScript Frameworks

Redux – Eine Übersicht

Übersicht und Zielstellung

- In diesem Abschnitt erfahren Sie:
  - was Redux ist,
  - etwas über die connect-Funktion,
  - wie Redux in mehreren Bereichen arbeitet,
  - wie man mehrere Reducers kombiniert,
  - wie Redux mit mehreren Komponenten arbeitet,

Start in das Thema

Redux

Eine Übersicht

Redux ist eine Bibliothek für JavaScript, die zur Verwaltung des Zustands (State) von Anwendungen verwendet wird.

Es ist besonders nützlich für Anwendungen, die eine große Menge an Daten verwalten müssen und bei denen der Zustand der Anwendung oft geändert wird.

Redux basiert auf dem Konzept der unidirektionalen Datenflussarchitektur, bei der der Zustand der Anwendung in einem zentralen Store gespeichert wird und durch Aktionen (Actions) geändert wird, die vom Benutzer oder der Anwendung selbst ausgelöst werden.

Diese Aktionen werden an einen Reducer weitergeleitet, der den neuen Zustand der Anwendung basierend auf der empfangenen Aktion berechnet und an den Store zurückgibt.

Es wird oft in Verbindung mit React verwendet, kann aber auch mit anderen JavaScript-Frameworks oder Bibliotheken wie Angular oder Vue.js verwendet werden.

Redux

Beschreibung

Angenommen, du hast eine einfache Todo-Liste-Anwendung, die folgende Funktionalitäten bietet: Du kannst Aufgaben hinzufügen, löschen und bearbeiten. Wenn du die Anwendung mit Redux implementierst, würde der Zustand der Anwendung in einem zentralen Store gespeichert werden, der aus einem Objekt mit den folgenden Eigenschaften besteht:

```
{
  todos: [],
  editTodo: null
}
```

todos ist ein Array, das die Liste der Todos enthält, während editTodo das Todo-Objekt enthält, das der Benutzer gerade bearbeitet. Wenn der Benutzer beispielsweise eine Aufgabe hinzufügt, löst die Anwendung eine Aktion aus, die ADD_TODO genannt wird und ein Payload enthält, das das hinzuzufügende Todo-Objekt darstellt:

```
{
  type: 'ADD_TODO',
  payload: {
    id: 1,
    text: 'Eine neue Aufgabe'
  }
}
```

Redux

Beschreibung

Diese Aktion wird an einen Reducer weitergeleitet, der die Anwendung aktualisiert, indem er eine neue Kopie des Zustands der Anwendung erstellt, in der das neue Todo-Objekt im todos Array enthalten ist:

```
function todoReducer(state = initialState, action) {
  switch(action.type) {
    case 'ADD_TODO':
      return {
        ...state,
        todos: [...state.todos, action.payload]
      }
    default:
      return state
  }
}
```

Redux

Beschreibung

Wenn der Benutzer nun eine Aufgabe bearbeiten möchte, wird eine weitere Aktion namens EDIT_TODO ausgelöst, die das Todo-Objekt enthält, das bearbeitet werden soll:

```
{
  type: 'EDIT_TODO',
  payload: {
    id: 1,
    text: 'Eine bearbeitete Aufgabe'
  }
}
```

Redux

Beschreibung

Diese Aktion wird erneut an den Reducer weitergeleitet, der das editTodo Objekt im Zustand der Anwendung auf das bearbeitete Todo-Objekt aktualisiert:

```
function todoReducer(state = initialState, action) {
  switch(action.type) {
    case 'ADD_TODO':
      return {
        ...state,
        todos: [...state.todos, action.payload]
      }
    case 'EDIT_TODO':
      return {
        ...state,
        editTodo: action.payload
      }
    default:
      return state
  }
}
```

Das ist ein sehr einfaches Beispiel, aber es zeigt, wie Redux verwendet werden kann, um den Zustand einer Anwendung zu verwalten und zu aktualisieren. Durch die Verwendung von Redux wird der Zustand der Anwendung konsistent gehalten und ermöglicht es Entwicklern, komplexere Anwendungen mit weniger Fehler zu schreiben.

Redux

React-Redux und die connect()-Funktion

React-Redux ist eine Bibliothek, die speziell für die Verwendung von Redux mit React entwickelt wurde. Es erleichtert die Integration von Redux in React-Anwendungen und bietet eine Reihe von nützlichen Funktionen und Komponenten, die die Verwendung von Redux vereinfachen.

Eine dieser Funktionen ist die connect()-Funktion, die es React-Komponenten ermöglicht, auf den Redux-Store zuzugreifen und ihre Eigenschaften (Props) basierend auf dem Zustand des Stores zu aktualisieren. Die connect()-Funktion ist ein High-Order-Component (HOC), die eine React-Komponente akzeptiert und eine neue Komponente zurückgibt, die Zugriff auf den Redux-Store hat.

Die connect()-Funktion wird wie folgt verwendet:

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

const MyComponent = ({ data }) => {
  return <div>{data}</div>
}

const mapStateToProps = (state) => {
  return {
    data: state.someData
  }
}

export default connect(mapStateToProps)(MyComponent)
```

Redux

React-Redux und die connect()-Funktion

In diesem Beispiel wird die connect()-Funktion verwendet, um die MyComponent-Komponente zu erweitern. Die mapStateToProps-Funktion wird verwendet, um zu definieren, welche Eigenschaften der Redux-Store an die MyComponent-Komponente als Props übergeben werden sollen. In diesem Fall wird data aus dem Redux-Store als data-Prop an die MyComponent-Komponente übergeben.

Durch die Verwendung der connect()-Funktion hat die MyComponent-Komponente nun Zugriff auf den Redux-Store und kann die Eigenschaften des Stores nutzen, um ihre eigenen Eigenschaften (Props) zu aktualisieren. Wenn der Zustand des Redux-Stores geändert wird, wird die MyComponent-Komponente automatisch neu gerendert, um die Änderungen zu reflektieren.

Die connect()-Funktion bietet auch eine Möglichkeit, Aktionen an den Redux-Store zu senden, indem sie die mapDispatchToProps-Funktion verwendet. Diese Funktion kann verwendet werden, um Aktionserzeugerfunktionen in Props zu mappen, die dann von der React-Komponente aufgerufen werden können, um Aktionen an den Redux-Store zu senden und den Zustand der Anwendung zu ändern.

Redux

Redux in mehreren Bereichen

In Redux werden mehrere Bereiche in der Regel als mehrere Reducer-Funktionen implementiert. Ein Reducer ist eine Funktion, die den vorherigen Zustand und eine Aktion akzeptiert und den neuen Zustand der Anwendung zurückgibt. Jeder Reducer ist normalerweise für die Verwaltung des Zustands einer bestimmten Domäne (Bereich) der Anwendung zuständig.

Um mehrere Reducer-Funktionen in einer Redux-Anwendung zu verwenden, wird die combineReducers()-Funktion verwendet. Diese Funktion nimmt ein Objekt von Reducer-Funktionen als Parameter und gibt eine einzelne Reducer-Funktion zurück, die den Zustand jedes einzelnen Bereichs im Redux-Store verwaltet.

Hier ist ein Beispiel, wie combineReducers() verwendet werden kann, um mehrere Reducer-Funktionen zu kombinieren:

```
import { combineReducers } from 'redux';
import usersReducer from './usersReducer';
import productsReducer from './productsReducer';

const rootReducer = combineReducers({
  users: usersReducer,
  products: productsReducer
});

export default rootReducer;
```

Redux

Redux in mehreren Bereichen

In diesem Beispiel wird die combineReducers()-Funktion verwendet, um die Reducer-Funktionen usersReducer und productsReducer zu kombinieren, die jeweils den Zustand der Benutzer- und Produktbereiche der Anwendung verwalten. Das Ergebnis dieser Funktion ist eine einzelne Reducer-Funktion, die den Zustand jedes Bereichs in der Redux-Store-Struktur verwaltet.

Durch die Verwendung von combineReducers() kann jeder Bereich der Anwendung separat verwaltet werden, was zu einer klaren Struktur und einem leichter zu wartenden Code führt. Es ermöglicht auch, den Zustand jedes Bereichs separat zu verwalten, was hilfreich sein kann, wenn Änderungen nur auf einen bestimmten Bereich der Anwendung beschränkt sind.

Redux

Mehrere Reducer kombinieren

In Redux können mehrere Reducer-Funktionen kombiniert werden, um einen einzigen Reducer zu erstellen, der den gesamten Zustand der Anwendung verwaltet. Die combineReducers()-Funktion wird verwendet, um die Reducer-Funktionen zu kombinieren.

Die combineReducers()-Funktion nimmt ein Objekt von Reducer-Funktionen als Parameter und gibt eine einzelne Reducer-Funktion zurück, die den gesamten Zustand der Anwendung verwaltet. Jeder Reducer-Funktion in diesem Objekt ist normalerweise für die Verwaltung des Zustands einer bestimmten Domäne (Bereich) der Anwendung zuständig.

Hier ist ein Beispiel, wie combineReducers() verwendet werden kann, um mehrere Reducer-Funktionen zu kombinieren:

```
import { combineReducers } from 'redux';
import todosReducer from './todosReducer';
import visibilityFilterReducer from './visibilityFilterReducer';

const rootReducer = combineReducers({
  todos: todosReducer,
  visibilityFilter: visibilityFilterReducer
});

export default rootReducer;
```

Redux

Mehrere Reducer kombinieren

In diesem Beispiel wird die combineReducers()-Funktion verwendet, um die Reducer-Funktionen todosReducer und visibilityFilterReducer zu kombinieren, die jeweils den Zustand der Aufgaben- und Sichtbarkeitsfilterbereiche der Anwendung verwalten. Das Ergebnis dieser Funktion ist eine einzelne Reducer-Funktion, die den gesamten Zustand der Anwendung verwaltet.

Durch die Verwendung von combineReducers() kann jeder Bereich der Anwendung separat verwaltet werden, was zu einer klaren Struktur und einem leichter zu wartenden Code führt. Es ermöglicht auch, den Zustand jedes Bereichs separat zu verwalten, was hilfreich sein kann, wenn Änderungen nur auf einen bestimmten Bereich der Anwendung beschränkt sind.

Redux

Redux und mehrere Components

In React-Anwendungen, die Redux verwenden, können mehrere Komponenten auf den Redux-Store zugreifen, um den globalen Zustand der Anwendung zu verwalten.

Um eine Komponente mit dem Redux-Store zu verbinden, wird die connect()-Funktion von react-redux verwendet. Diese Funktion ist ein Higher-Order-Komponente, die eine vorhandene Komponente umschließt und eine neue Komponente zurückgibt, die auf den Redux-Store zugreifen kann.

Die connect()-Funktion wird normalerweise mit zwei Funktionen aufgerufen: mapStateToProps und mapDispatchToProps. mapStateToProps ist eine Funktion, die den Zustand aus dem Redux-Store abruft und als Props an die umschließende Komponente weitergibt. mapDispatchToProps ist eine Funktion, die Aktionserzeuger-Funktionen definiert und als Props an die umschließende Komponente weitergibt. Diese Aktionserzeuger-Funktionen können dann von der Komponente aufgerufen werden, um Aktionen zu erzeugen und an den Redux-Store zu senden.

Redux

Redux und mehrere Components

Hier ist ein Beispiel, wie connect() verwendet werden kann, um eine Komponente mit dem Redux-Store zu verbinden:

```
import { connect } from 'react-redux';
import { fetchTodos } from '../actions/todosActions';
import TodoList from '../components/TodoList';

const mapStateToProps = state => {
  return {
    todos: state.todos
  };
};

const mapDispatchToProps = dispatch => {
  return {
    fetchTodos: () => dispatch(fetchTodos())
  };
};

export default connect(mapStateToProps, mapDispatchToProps)(TodoList);
```

Redux

Redux und mehrere Components

In diesem Beispiel wird die connect()-Funktion verwendet, um die Komponente TodoList mit dem Redux-Store zu verbinden. Die mapStateToProps-Funktion wird verwendet, um den todos-Bereich des Zustands aus dem Redux-Store abzurufen und als Props an die TodoList-Komponente weiterzugeben. Die mapDispatchToProps-Funktion wird verwendet, um die Aktionserzeuger-Funktion fetchTodos() zu definieren und als Prop an die TodoList-Komponente weiterzugeben. Diese Funktion kann dann von der TodoList-Komponente aufgerufen werden, um die fetchTodos-Aktion auszulösen und den Redux-Store zu aktualisieren.

Durch die Verwendung von connect() können mehrere Komponenten auf den Redux-Store zugreifen und den globalen Zustand der Anwendung verwalten. Dies ermöglicht eine klare Trennung von Zustands- und Präsentationslogik und erleichtert das Testen und Warten der Anwendung.

Redux

Redux & React Router

React Router ist eine Bibliothek für React, die es ermöglicht, Routing- und Navigationsfunktionen in einer React-Anwendung zu implementieren. Redux und React Router können in einer Anwendung gemeinsam verwendet werden, um den Zustand der Navigation und die angezeigten Inhalte zu verwalten.

Normalerweise wird der Zustand des React Router in der URL des Browsers gespeichert. Wenn sich der Benutzer auf einer bestimmten Seite befindet, wird die URL geändert, um die aktuelle Seite widerzuspiegeln. Wenn eine andere Seite aufgerufen wird, wird die URL geändert, und die React-Komponenten, die diese Seite darstellen, werden gerendert.

In einer Redux-basierten Anwendung können jedoch der Zustand der Navigation und der angezeigten Inhalte im Redux-Store gespeichert werden. Dies hat den Vorteil, dass der Zustand der Navigation und der angezeigten Inhalte leichter von verschiedenen Komponenten innerhalb der Anwendung abgerufen und aktualisiert werden kann.

Redux

Redux & React Router

```
import React from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';
import { connect } from 'react-redux';
import { setPage } from './actions/navigationActions';

import Home from './components/Home';
import About from './components/About';

const App = ({ currentPage, setPage }) => {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li><a href="/" onClick={() => setPage('home')}>Home</a></li>
            <li><a href="/about" onClick={() => setPage('about')}>About</a></li>
          </ul>
        </nav>
        <Route exact path="/" component={Home} />
        <Route path="/about" component={About} />
        <div>Current Page: {currentPage}</div>
      </div>
    </Router>
  );
};

const mapStateToProps = state => {
  return {
    currentPage: state.navigation.currentPage
  };
};

const mapDispatchToProps = dispatch => {
  return {
    setPage: page => dispatch(setPage(page))
  };
};

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

Redux

Redux & React Router

In diesem Beispiel wird die BrowserRouter-Komponente von React Router verwendet, um die Navigation in der Anwendung zu verwalten. Die connect()-Funktion von React Redux wird verwendet, um die App-Komponente mit dem Redux-Store zu verbinden, damit sie auf den Zustand der Navigation zugreifen kann.

In der App-Komponente wird die nav-Tag verwendet, um Links zur Startseite und zur Über-Seite der Anwendung bereitzustellen. Die onClick-Handler werden verwendet, um die setPage-Aktion auszulösen und den Redux-Store zu aktualisieren. Die Route-Komponenten werden verwendet, um die entsprechenden Komponenten zu rendern, wenn die Benutzer die Links klicken.

Durch die Verwendung von Redux und React Router können der Zustand der Navigation und der angezeigten Inhalte in einer Anwendung effektiv verwaltet werden, was zu einer besseren Struktur und Wartbarkeit führt.

Redux

Einbau von React Router

Um React Router in eine React-Anwendung einzubinden, müssen zunächst die notwendigen Pakete installiert werden. Hierzu können entweder npm oder yarn verwendet werden:

```
npm install react-router-dom
```

oder:

```
npm install react-router-dom
```

Nach der Installation können die Komponenten von React Router in der Anwendung importiert werden:

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
```

Die Router-Komponente ist das Hauptelement von React Router und sollte um alle anderen Komponenten herum eingebettet werden. Die Route-Komponente wird verwendet, um eine bestimmte Route und die zugehörige Komponente zu definieren, die gerendert werden soll. Die Switch-Komponente wird verwendet, um sicherzustellen, dass nur die erste passende Route gerendert wird.

Redux

Einbau von React Router

Die Router-Komponente ist das Hauptelement von React Router und sollte um alle anderen Komponenten herum eingebettet werden. Die Route-Komponente wird verwendet, um eine bestimmte Route und die zugehörige Komponente zu definieren, die gerendert werden soll. Die Switch-Komponente wird verwendet, um sicherzustellen, dass nur die erste passende Route gerendert wird.

Ein Beispiel für die Verwendung von React Router in einer React-Anwendung:

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

import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';
import NotFound from './components/NotFound';

const App = () => {
  return (
    <Router>
      <div>
        <Switch>
          <Route exact path="/" component={Home} />
          <Route path="/about" component={About} />
          <Route path="/contact" component={Contact} />
          <Route component={NotFound} />
        </Switch>
      </div>
    </Router>
  );
};

export default App;
```

Redux

Einbau von React Router

In diesem Beispiel werden vier verschiedene Routen definiert, die die Komponenten Home, About, Contact und NotFound rendern. Wenn keine passende Route gefunden wird, wird die NotFound-Komponente gerendert. Die exact-Eigenschaft wird verwendet, um sicherzustellen, dass nur exakte Treffer auf die Route gerendert werden, um Konflikte mit anderen Routen zu vermeiden.

Durch die Verwendung von React Router kann eine React-Anwendung leicht navigierbar gemacht werden und es können verschiedene Ansichten und Komponenten basierend auf der aktuellen URL gerendert werden.

Redux

Navigation mit dem Route

Um eine Navigation in einer React-Anwendung mit React Router zu erstellen, können Sie eine Komponente erstellen, die Links zu den verschiedenen Routen enthält. Hier ist ein Beispiel für eine einfache Navigationskomponente:

```
import React from 'react';
import { NavLink } from 'react-router-dom';

const Navigation = () => {
  return (
    <nav>
      <ul>
        <li>
          <NavLink exact to="/">Home</NavLink>
        </li>
        <li>
          <NavLink to="/about">About</NavLink>
        </li>
        <li>
          <NavLink to="/contact">Contact</NavLink>
        </li>
      </ul>
    </nav>
  );
};

export default Navigation;
```

Redux

Navigation mit dem Route

In diesem Beispiel wird die NavLink-Komponente von React Router verwendet, um Links zu den verschiedenen Routen zu erstellen. Die exact-Eigenschaft wird verwendet, um sicherzustellen, dass nur exakte Treffer auf die Startseite gerendert werden.

Um die Navigationskomponente in Ihrer Anwendung zu verwenden, können Sie sie in die Hauptkomponente einbinden:

In diesem Beispiel wird die Navigation-Komponente vor der Switch-Komponente platziert, um die Links zu den verschiedenen Routen anzuzeigen. Durch die Verwendung von React Router können Sie schnell und einfach eine Navigationskomponente erstellen und in Ihre Anwendung integrieren.

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

import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';
import NotFound from './components/NotFound';
import Navigation from './components/Navigation';

const App = () => {
  return (
    <Router>
      <div>
        <Navigation />
        <Switch>
          <Route exact path="/" component={Home} />
          <Route path="/about" component={About} />
          <Route path="/contact" component={Contact} />
          <Route component={NotFound} />
        </Switch>
      </div>
    </Router>
  );
};

export default App;
```

VIELEN DANK!