Redux

Redux là gì?

Redux là một kiến trúc cho một hệ thống dựa trên kiến trúc Flux của Facebook. Redux thường được sử dụng trong React, tuy nhiên Redux có thể sử dụng ở nhiều framework hay ngôn ngữ khác không nhất thiết là với React. Để thuận tiện cho việc xây dựng kiến trúc này, Dan Abramov (cha đẻ của Redux) đã làm ra một thư viện cùng tên, nhằm hỗ trợ dev xây dựng kiến trúc Redux nhanh hơn.

Redux dùng để làm gì?

Khi làm việc với state, điển hình là một Modal. Nếu muốn bật tắt một modal ta thường phải truyền các state qua props chồng chéo lẫn nhau. Gây cho việc rối loạn props. Chưa kể việc truyền những hàm để `setState` cho component cha thông qua component con, khiến việc debug trở nên khó khăn hơn rất nhiều.

Để dễ hiểu hơn thì như sau. Giả sử ta có 2 Modal. Một Modal A và Modal B. Ở Modal A có thể bật / tắt Modal B và tương tự cho Modal B có thể bật / tắt Modal A. Cái hay nữa là trong Modal A có một Form A. Form A này có thể bật tắt Modal B. Ngoài ra ở trên Header của App có 2 icon giúp click vào bật Modal A và Modal B.

Đọc thì dễ dàng ta viết những hàm để bật tắt Modal A/B và truyền qua props như sau:

```
// Modal component
class Modal extends Component {
  render() {
    const {
      children,
      onClose, isVisible,
      modalTitle,
    } = this.props;

    const modalClass = [
      'modal',
      isVisible ? 'visible' : 'hidden',
    ];
    return (
      <div className={modalClass.join('')}>
        <div
          className="modal modal-header"
        >
          <div>
            {modalTitle}
          </div>
          <i
            className="modal-btn modal-close-btn"
            onClick={onClose}
          >
            X
          </i>
        </div>
        <div className="modal-body">
          {children}
        </div>
      </div>
    )
  }
}

// FormA
class FormA extends Component {
  handleCancel = () => {
    const { closeModal } = this.props;
    closeModal();
  }

  handleSubmit = () => {
    const { closeModal } = this.props;
    // submit form
    closeModal();
  }

  render() {
    return (
      <div>
        <input type="text" name="sample" />
        <button onClick={openModalB}>
```

```
          Next Step
        </button>
      </div>
    )
  }
}

// App.js
class App extends Component {
  //
  render() {

    return (
      <div>
        <Header
          openModalA={this.openModalA}
          openModalB={this.openModalB}
        />
        <Modal
          isVisible={isModalAOpen}
          onClose={this.closeModalA}
          modalTitle="Modal A"
        >
          <FormA
            // truyền vào closeModal
            // để formA có nút submit hoặc cancel điền form
            closeModal={this.closeModalA}
            openModalB={this.openModalB}
            closeModalB={this.closeModalB}
          />
        </Modal>
        <Modal
          isVisible={isModalBOpen}
          onClose={this.closeModalB}
          modalTitle="Modal B"
        >
          <FormB
            closeModal={this.closeModalB}
            openModalA={this.openModalA}
            closeModalA={this.closeModalA}
          />
        </Modal>
      </div>
    )
  }
}
```

Đọc sơ qua cũng thấy gọi là các hàm được truyền vào rất nhiều qua các component khác nhau, rất khó theo dõi luồng chạy của code trên. Như nếu click vào `Next Step` -> Modal B hiện lên. Nhưng trong Modal B lại mở được Modal A. Vậy khi ở Modal A mình nhấn `closeModalA` thì liệu Modal A đã mở Modal B ban đầu có đóng không?

Modal A -> Modal B -> Modal A -> Close Modal A
Câu hỏi: Modal A ban đầu có close không? Rồi Modal A close thì Modal B sẽ như thế nào?

Rất rất phức tạp để giải quyết vấn đề trên.

Ngoài ra còn một vấn đề nữa như sau. Form A có một input box để nhập tên Quiz. Form A có một nút là **Thêm câu trả lời**. Cứ nhấn nút đó một input box được hiện ra để người dùng nhập vào câu trả lời mới và có một nút **Lưu** kế bên để lưu câu trả lời mới nhập. Sau khi xong thì có một nút **Thêm câu hỏi**. Nhấn nút đó thì câu hỏi được nhập và toàn bộ câu trả lời sẽ được thêm vào danh sách câu hỏi.

Xem code dùng state nhanh

```
class AnswerBox extends Component {
  // some methods
  render() {
    // get state
    return (
      <div>
        <input
          type="text"
          name="Answer"
          value={answer}
          onChange={onAnswerChange}
        />
        <button
          onClick={onSaveAnswer}
        >
          Save
        </button>
      </div>
    )
  }
}

class QuizForm extends Component {
  // some methods
  addQuestion = () => {
    const { addNewQuiz } = this.props;
    const { question } = this.state;

    addNewQuiz(question);
  }

  render() {
    // get state
    return (
      <div>
        <input
          type="text"
          name="Quiz"
          ...
        />

        <AnswerBox
          answer={answerText} // get from state
          onAnswerChange={this.handleAnswerOnChange}
          onSaveAnswer={this.handleAnswerSaving}
        />
        <button
          onClick={this.addAnswer}
        >
          Add answer
        </button>
      </div>
    )
  }
```

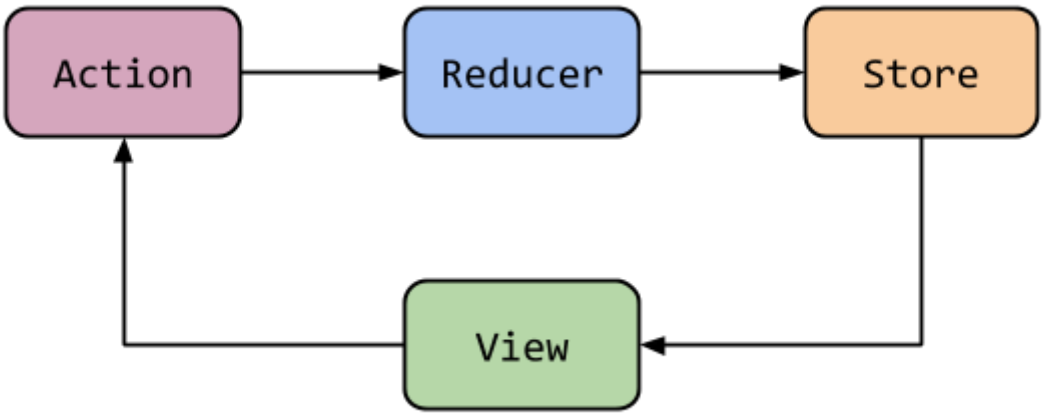
```
}

class Question extends Component {
  //
}

class MainLayout extends Component {
  render() {
    return (
      <div>
        {questionList.map((question) => (
          <Question
            question={question}
          />
        ))}
        <QuizForm
          addNewQuiz={this.addNewQuiz}
        />
      </div>
    )
  }
}
```

Lại một lần nữa ta thấy được việc truyền các state và setState methods khá là xếp tầng. Như từ MainLayout muốn lây được câu hỏi được thêm vào từ QuizForm thì phải truyền một hàm addNewQuiz vào để có thể setState từ trong QuizForm. Trong QuizForm do để dễ quản lý việc thêm câu trả lời nên cũng đã tách component quản lý việc thêm mới câu trả lời vào một component riêng. Từ đó phải truyền các value từ state và các hàm để quản lý tương ứng. Nôm na là rối rắm và mệt ơi là mệt. Cứ thử tưởng tượng, sau này ứng dụng có hơn chục component. Từ cái component này muốn xài state của component kia thì cứ truyền qua props như vậy có ngày nhiễu loạn.

Thấy được sự hỗn loạn của Giang hồ như vậy. Redux đã được ra đời để giải quyết mớ hỗn độn trên.

Redux Nhập môn

Vì việc mỗi một component có một state riêng, rồi ai muốn setState cho component thì phải viết một hàm rồi truyền qua props như vậy rất bất tiện. Thế nên ông chú Dan mới kiểu. "Ồ, tại sao mình không **gộp tất cả state lại thành một state chung** nhỉ? Sau đó **muốn thay đổi state ở chỗ Redux mình thì chỉ cần đưa ra những hiệu lệnh**". Với hai câu trên, thì đã mô tả được gần hết kiến trúc Redux. Để kỹ hơn thì cùng xem Diagram ở dưới nha



Nhìn vào Diagram trên, thì dễ dàng nhất nhận thấy được khối **View** chính là những gì đã học suốt những tuần qua. Đó chính là phần React. Còn ba phần còn lại thì cùng nhau tìm hiểu thêm cùng với 2 câu mô tả ở trên.

Với phần thứ nhất: **gộp tất cả state lại thành một state chung**. Thì đó chính là **store** trong sơ đồ trên. Store chính là nơi Redux tập trung và lưu trữ toàn bộ state của từng component lại.

Tuy nhiên bên React có setState thì redux có hàm nào tương tự vậy hok ta? 🤔 Hờ hờ, tất nhiên là méo. Thay vào đó, Redux dùng một cơ chế để update store khác gọi là action dispatching. Và đó là khối **action** màu hường dễ thương trong hình vẽ trên. Mỗi action sẽ chứa những thông tin cần thiết để Redux có thể biến đổi state cũ thành state mới chứa những dữ liệu mới.

Ủa vậy còn cái cục **Reducer**. Còn mỗi cục đó chưa nói, vậy cục đó công dụng là gì thế? 😮 Nếu bạn đọc phần action và chắc cũng sẽ có xíu thắc mắc rằng: *"Ủa, Action chứa thông tin không mà trời, sao mà biến đổi state được?? Biến đổi thông tin gì gì phải làm trong code hay mình bỏ nó dô cái hàm chớ?!?!?! ??* 😃 *??"* Thì đúng rồi đó, action bản thân chỉ chứa thông tin thôi chứ không biến đổi được state trong store. Muốn biến đổi state trong store thì đây. Bé **Reducer**, khối màu xanh dương là người giúp mọi người làm điều trên. **Reducer là một hàm mà Redux sẽ dùng để biến đổi state dựa trên những action đưa truyền vào.** Thế nên **reducer nhận hai tham số** đó là **state hiện tại của store và action** mà coder muốn reducer hiện thực để biến đổi state hiện tại thành một state mới.

Redux Nâng cao

Vì redux là một kiến trúc như kiến trúc căn nhà. Từng khối kia chính là từng package hay module để xử lý từng việc trong hệ thống đó tương tự là như những căn phòng trong kiến trúc của những căn nhà. Tuy nhiên, nãy giờ chúng ta chỉ nói về căn phòng này làm gì, có chúc năng gì nhưng chưa hề nói về luồng chạy cũng như cách kết nối từng căn phòng của redux lại với nhau. Phần này cùng tìm hiểu về dòng chảy dữ liệu trong redux và cách mà mọi thứ được liên kết với nhau nhé.

Để dễ dàng hơn trong việc kết nối các React Component với Redux Module. Ta sẽ dùng một thư viện nhỏ để hỗ trợ gọi là `react-redux`.

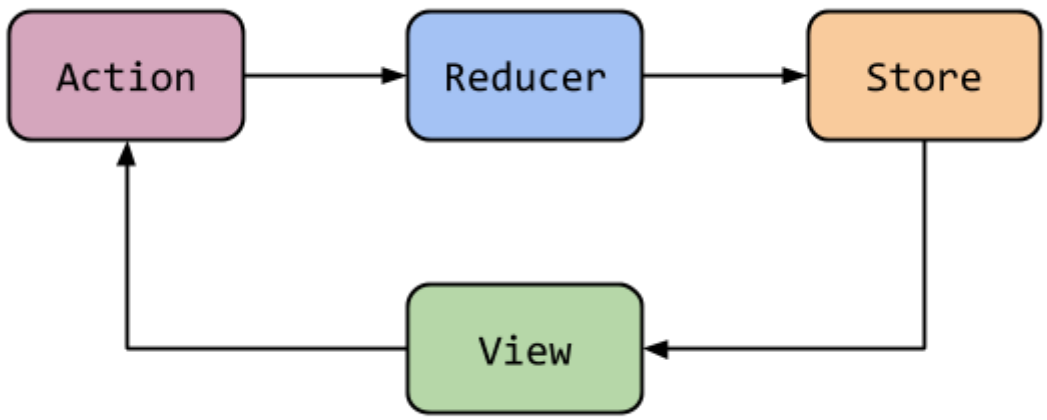
Bắt đầu nào!!
Để hiểu được sâu hơn về Redux, hãy liên tưởng bạn đang mở một hiệu sách có hàng trăm, hàng ngàn cuốn sách. Vì hiệu sách bạn rất lớn và nhiều sách hay từ những cuốn sách cũ kỹ nhất cũng có luôn. Nên khi mở cửa, có rất nhiều người tới để tham quan cxung như mượn sách từ hiệu sách của bạn. Nhưng bạn gặp một vấn đề khiến bạn không vui như tưởng 😟 Đó là vì quá nhiều người mượn sách cùng lúc, chen chúc, không xếp hàng, làm bạn lộn xộn hết cả lên. Và thế là đông thì đông nhưng lộn xộn vậy, làm hư hết sách mà còn bị thất thu mà không biết ai để mà bắt đền. Quyết tâm triệt để việc này, tối hôm đó, bạn viết một bảng thông báo

> NGƯỜI MƯỢN SÁCH, VUI LÒNG ĐIỀN VÀO GIẤY MƯỢN
> NGƯỜI MUA SÁCH, VUI LÒNG ĐIỀN VÀO PHIẾU MUA
> NGƯỜI THAM QUAN, VUI LÒNG ĐIỀN THÔNG TIN VÀO PHIẾU ĐĂNG KÝ THAM QUAN
>
> SAU ĐÓ VUI LÒNG XẾP HÀNG VÀ ĐƯA PHIẾU TƯƠNG ỨNG CHO QUẢN LÝ ĐỂ ĐƯỢC GIẢI QUYẾT **TỪNG NGƯỜI MỘT**

Và không nằm ngoài dự đoán. Với thông báo trên và việc xử lý từng người một thông qua các phiếu thông tin. Bạn bây giờ đã hoàn toàn xử lý được khối lượng công việc khổng lồ một cách trơn tru hơn. Ngoài ra bạn còn biết được ai làm gì để mà sách bị hư là túm cổ ngay.

Vâng, đó chính xác 100% cách Redux hiện thực. Với các tủ sách, thì đó chỉnh là store của redux. Với các phiếu thông tin, đó chính là những action của redux. Còn reducer? À chính là người quản lý đó! Người quản lý tiếp nhận thông tin từ các phiếu thông tin, rồi làm những hành động như lấy sách, lấy tiền của khách, trao sách cho khách, update doanh thu từ tiền của khách. Thì với mỗi hành động từ quản lý, đã làm thay đổi state của hiệu sách của bạn, và đó chính là công việc của một reducer. Và khi đó khách hàng sẽ có những cái họ muốn nè, như mượn được sách còn hiệu sách được tiền.

Dễ hiểu phải không nào?



Bên React cũng vậy. React chính là những khách hàng. React chính là nơi phát sinh ra những phiếu thông tin để đưa cho quản lý. Khi React gửi những phiếu thông tin cho quản lý. Quản lý khi này là reducer sẽ đọc thông tin gửi từ React xem cần làm gì nè. Sau đó hiện thực những yêu cầu mà React muốn để update Store của Redux. Khi update store xong, thay vì khi này quản lý sẽ gửi lại những yêu cầu về cho React như hiệu sách. Thì thằng Redux sẽ làm giúp reducer việc này, vì reducer bây giờ bận lắm rồi 😟. Redux phải phụ xíu, đó chính là re-render React component gửi một phiếu thông tin về Redux.

Hiểu sơ sơ được rồi phải không? Bây giờ sang những từ chuyên ngành hơn nhé.

Những phiếu thông tin khi này gọi là **Action**
React sẽ điều thông tin vào những Action này và gửi lên cho quản lý. Việc gửi thông tin cho quản lý gọi là **Dispatch Action**
Và quản lý khi này sẽ được gọi là **Reducer**
Quản lý update thông tin sách, thông tin tiền bạc. Nơi lưu trữ những thông tin đó được gọi là **Store**
Sau khi quản lý update tủ sách và doanh thu. Khi này Redux mới đưa sách cho khách. Hành động trao sách này bên React Redux khoong phải là trao sách mà là **Re-render**

Một lần nữa nhưng ngắn gọn hơn

React sẽ tạo ra những **Action**
Sau khi tạo ra những Action xong, React sẽ **Dispatch** những Action này vào **Reducer**
Reducer cập **Store** với những thông tin từ Action
Sau khi cập nhật xong, Redux sẽ tiến hành **Re-render** lại React component đã dispatch action. Tạo thành một vòng khép kín. OvOb

Đó là phần lý thuyết về luồng hoạt động của Redux. Tuy nhiên làm sao mọi thứ có thể kết dính với nhau được? Thì cái keo kết dính đó chính là phần `react-redux`. React redux là một thư viện giúp kết dính React component với Redux một cách nhanh chóng và dễ dàng nhất nhát nhất có thể.

Để kết dính với Redux, `react-redux` sẽ dùng một hàm đặc biệt gọi là `connect(mapStateToProps, mapDispatchToProps)(React_Component)` để làm việc đó.

Thì công việc của hàm `connect` này là gì? Connect sẽ nhận vào hai tham số.

Tham số thứ nhất là `mapStateToProps`, đây là một **hàm**. Giúp `react-redux` lấy ra những state mình muốn lấy ra từ store của redux. Sau đó? Sau đó `react-redux` sẽ nhét cái mớ này vào props của React Component mà mình muốn lấy state ra thông qua việc return một **object**. Vd cho nóng:

```
// hàm này nhận vào là store chứa toàn bộ của redux
// tuy nhiên để dễ hiểu thì coi store = 1 state khổng lồ
function mapStateToProps(state) {
  // lấy field counter từ state ra
  // rất giống React state đúng không?
  const { counter } = state;

  // khi lấy xong rồi, thì mình BẮT BUỘC PHẢI RETURN MỘT OBJECT
  // object này chứa các props mà mình muốn đưa vào component
  return {
    counter,
  }
  // dòng return này có thể xem là
  // <Component counter={counter} />

  // nếu return như sau:
  // return {
  //   c: counter,
  // }
  // thì nó sẽ thành
  // <Component c={counter} />
}
```

Lấy được thông tin từ state trong store của redux rồi nè. Hay ghê. Bây giờ làm sao mình gửi cho reducer cái action ta? Thì đây là phần kết dính thứ hai, `mapDispatchToProps` . Hàm này sẽ giúp tạo ra những hàm để dispatch một action, và tống những hàm này vào props của Component muốn thực hiện những action này. Và giống như `mapStateToProps` , `mapDispatchToProps` cũng bắt buộc trả về **một object** chứa những props để dispatch một action.

```
// hàm này sẽ nhận một tham số gọi là dispatch
// dispatch là 1 hàm sẽ làm nhiệm vụ gửi một action vào redux
function mapDispatchToProps(dispatch) {
  // bằng việc return một object
  // có key lfa increase
  // key này là một hàm
  // hàm này khi gọi bên React component sẽ dispatch một action
  // để tăng giá trị counter lên 10 đơn vị
  return {
    increase: fucntion() {
      // action là một object
      // và thường có 2 fields
      // type để biến action này làm gì
      // và mọi thông tin thêm thường được bỏ vào payload
      dispatch({
        type: COUNTER_INCREASE,
        payload: {
          value: 1,
        },
      });
    } // end increase function
  }; // end return

  // cái này tương đương với việc nó ở trên
  // <Component increase={function() { ... }} />
}
```

Vậy là ta đã biết được cách làm sao gửi đơn rồi, và làm sao lấy những giá trị từ state thông qua hàm connect. Tuy nhiên có ai thấy lạ không?

```
// sao lại là 2 cặp dấu ()() nhỉ?
connect(mapStateToProps, mapDispatchToProps)(Component);
```

Sao lại là 2 cặp dấu ()() nhỉ? Bởi vì connect khi nhận vào 2 tham số kia. Sẽ trả về một hàm khác. Thì hàm này mới đảm nhận việc connect thật sự. Chứ ở lần đầu, connect làm việc khác, đó là lấy các giá trị từ Redux ra. Sau đó trả về một hàm, hàm này có toàn bộ giá trị của Redux được lấy ra thông qua 2 tham số `mapStateToProps` và `mapDispatchToProps` . Thì bước cuối cùng mới là connect hàm có những giá trị từ Redux vào component thông qua props. Và sau khi connect xong, sẽ return về một React Component chứa đầy đủ những thông tin đã lấy hoặc muốn thêm vào từ 2 hàm `mapStateToProps` và `mapDispatchToProps` ;

```
const reactComponentConnector = connect(mapStateToProps, mapDispatchToProps);

const ComponentWithState = reactComponentConnector(Component);
// ComponentWithState sẽ có toàn bộ props từ `mapStateToProps` và `mapDispatchToProps`
// nếu chỉ xài Component không thì sẽ không có
```

Tuy nhiên còn reducer và store của tui đâu 😟
Đây đây đừng nóng vội. Store và reducer đi chung với nhau, nên mới để tới đây nói đây.

Để kết nối store với REdux. Một lần nữa ta phải nhờ sự trợ giúp của captain React redux. `react-redux` lần này sẽ đưa cho ta một component gọi là `Provider`. Component này sẽ giúp connect toàn bộ redux vào react app của chúng ta. Cách sử dụng cũng cực kỳ đơn giản. Chỉ cần bọc component `Root` hoặc cái `App` của React với `Provider` này là xong. Như thế này

```
<Provider store={store}>
// toàn bộ react component phải ở trong đây
// bình thường sẽ là <App />
  <App />
</Provider>
```

Ủa store kìa? Làm sao để tạo store đây? Cực kỳ dễ luôn. Chỉ cần lấy hàm `createStore` trong thư viện của Redux ra xài là xooong.

```
const initialStoreValue = {};
const store = createStore(reducer, initialStoreValue);
```

Hàm này sẽ nhận vào 2 tham số. Cái thứ nhất chính là người quản lý của chúng ta, là reducer, cái thứ hai chính là giá trị ban đầu của store này. Mặc định ta sẽ khởi tạo store là một object rỗng như trên.

Và khi này ta chỉ cần tạo ra ông chú reducer nữa là xong. Và như nói ở tuốt tuốt trên. Ông chú reducer chính là một hàm, nhận vào state hiện tại, và action để chỉnh sửa state.

```
function reducer(state, action) {
  switch(action.type) {
    // a mấy người muốn tui tăng giá trị counter lên nè
    case COUNTER_INCREASE: {
      // giá trị muốn tăng lấy ra từ payload trong action
      const { payload } = action;
      // copy qua state khác để tránh chỉnh sửa state cũ
      const newState = {
        ...state, // cú pháp copy nhanh
      };
      // update counter
      newState.counter += payload.value;
      // update counter xong rồi
      // ok return state mới thôi
      return newState;
    }
    // default là tui ko biết mấy người làm action gì
    // tui ko hiểu
    // nên tui không sửa gì hết
    default:
      return state;
  }
}
```

Và xoooong! Chúng ta đã học được cách viết một reducer đơn giản giản giản, tạo một store thông qua hàm **createStore** và kết nối reducer vào store. Sau đó từ store vào **Provider** để connect Redux vào React app của mình. Ngoài ra ở mỗi component, mình cũng đã học được cách làm sao lấy giá trị trong store và làm sao để dispatch một action vào redux thông qua hàm **connect** với 2 tham số **mapStateToProps** và **mapDispatchToProps**.

Redux in action

Khi hiểu lý thuyết rồi thì tiến hành nhảy vào code thôi nào. Và nhớ cài `react-redux` trước khi tiến hành nhé. Để nhanh gọn thì sẽ giới thiệu cho mọi người một app cộng trừ một số dùng redux.

Khi làm với Redux. **Luôn luôn làm phần View trước**.

```
// Counter.js
class Counter extends Component {
  render() {
    // biết vì sao counter lấy qua props ko ;)
    const {
      counter,
      increase, // increase là gì đây :o
    } = this.props;

    return (
      <div>{counter}</div>
      <button onClick={increase}>+</button>
    );
  }
}


// App.js
import { Provider } from 'react-redux';
// tách qua 1 file cho dễ quản lý nè
import store from './store';

class App extends Component {
  render() {
    return (
      <Provider store={store}>
        <Counter />
      </Provider>
    );
  }
}


// store.js
// thư viện redux giúp code Redux nhanh như bị ysl
import { createStore } from 'redux';
// reducer của chúng ta đây
import rootReducer from './reducers';

const store = createStore(rootReducer, {});

export default store;
```

```
// reducer.js
// anh quản lý reducer đây rồi
function reducer(state, action) {
  switch(action.type) {
    case COUNTER_INCREASE: {
      const { payload } = action;
      const newState = {
        ...state, // cú pháp copy nhanh
      };
      newState.counter += payload.value;
      return newState;
    }
    default: {
      return state;
    }
  }
}

export default reducer;
```

Mà hình như nãy giờ thiếu thiếu cái gì :? À đúng rồi. Là connect. Khi mình connect một component với Redux. Thì file giúp mình làm điều đó gọi là **Container**. Đặt tên theo dạng [Tên_Component]Container. Vd: CounterContainer

```
// CounterContainer.js
import Counter from './Counter';

// arrow function nè
const mapStateToProps = (state) => {
  return {
    counter: state.counter,
  }
}

const mapDispatchToProps = (dispatch) => {
  return {
    increase: () => dispatch({
      type: COUNTER_INCREASE,
      payload: {
        value: 10,
      }
    }),
  }
}

// connect()() trả về một React component có props lấy từ 2 hàm trên
// mà vẫn render ra y chang cái Counter cũ
// Vì nó là React component nên xài như HTML thoải mái :D
export default connect(mapStateToProps, mapDispatchToProps)(Counter);
```

Ùi ui, chết tí quên. Sửa lại App.js thôi

```
// App.js
import { Provider } from 'react-redux';
// container là nơi để connect đó mấy ông
import CounterContainer from './CounterContainer';
// tách qua 1 file cho dễ quản lý nè
import store from './store';

class App extends Component {
  render() {
    return (
      <Provider store={store}>
        <CounterContainer />
      </Provider>
    );
  }
}
```

Vậy là hoàn thành. Có thể hơi khó hiểu. Thế nên đừng ngại note lại và hỏi để được giải đáp nha.