Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 22

Provider | connect | mapStateToProps | mapDispatchToProps

План занятия

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

1. Provider
2. Connect
3. mapStateToProps
4. dispatchToProps

Provider


```
import React from 'react'
import { Provider } from 'react-redux'
import store from './redux/store'
import './App.css'
import CakeContainer from './components/CakeContainer'

function App () {
  return (
    <Provider store={store}>
      <div className='App'>
        <CakeContainer />
      </div>
    </Provider>
  )
}

export default App
```

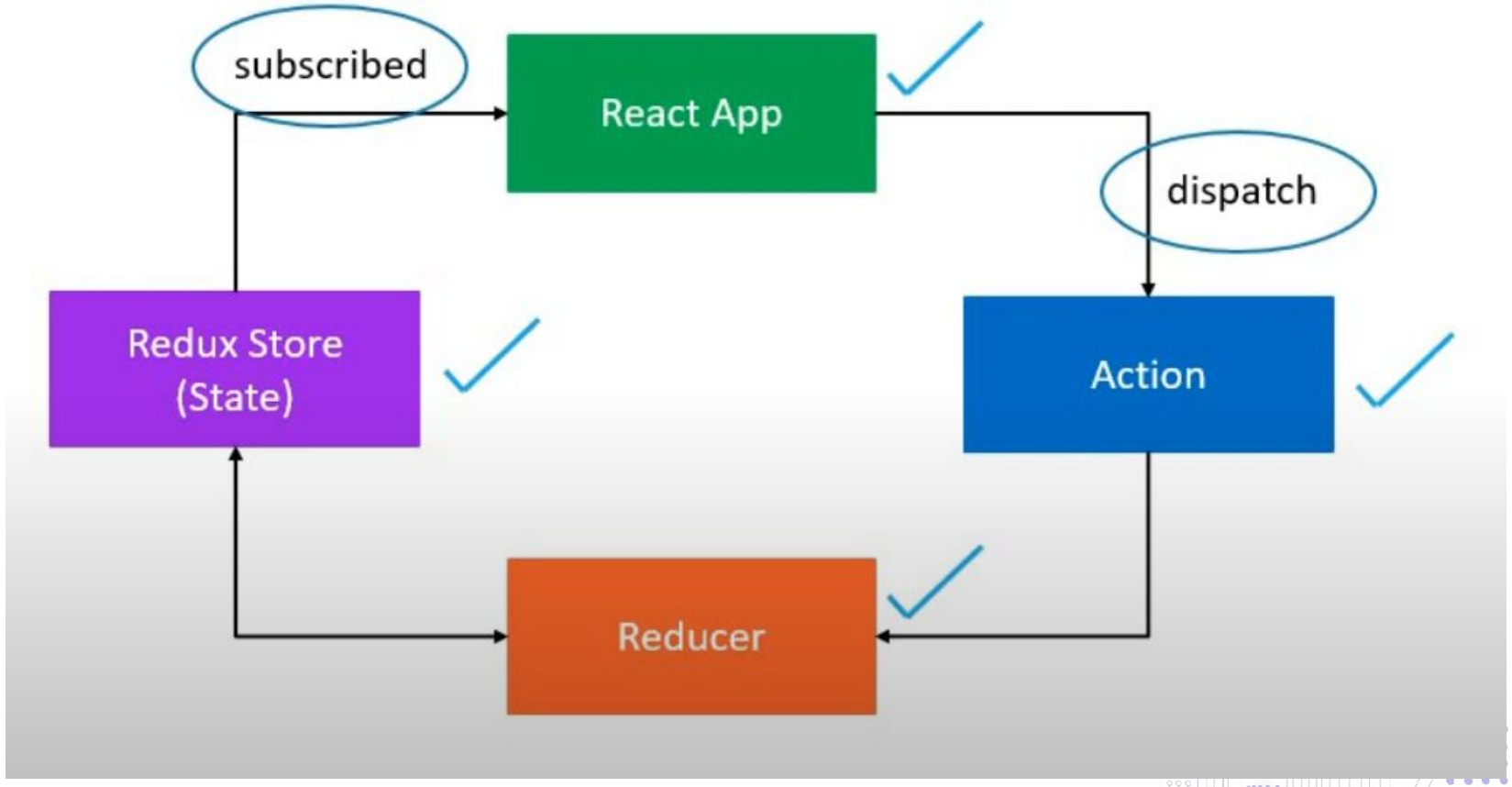
Provider


```
function Provider({
  store,
  context,
  children,
  serverState,
  stabilityCheck = 'once',
  noopCheck = 'once'
}) {
  const contextValue = React.useMemo(() => {
    const subscription = createSubscription(store);
    return {
      store,
      subscription,
      getServerState: serverState ? () => serverState : undefined,
      stabilityCheck,
      noopCheck
    };
  }, [store, serverState, stabilityCheck, noopCheck]);
  const previousState = React.useMemo(() => store.getState(), [store]);
  useIsomorphicLayoutEffect(() => {
    const {
      subscription
    } = contextValue;
    subscription.onStateChange = subscription.notifyNestedSubs;
    subscription.trySubscribe();

    if (previousState !== store.getState()) {
      subscription.notifyNestedSubs();
    }

    return () => {
      subscription.tryUnsubscribe();
      subscription.onStateChange = undefined;
    };
  }, [contextValue, previousState]);
  const Context = context || ReactReduxContext; // @ts-ignore 'AnyAction' is assignable to the constraint of type 'A', but 'A' could be instantiated with a different subtype

  return /*#__PURE__*/React.createElement(Context.Provider, {
    value: contextValue
  }, children);
}

export default Provider;
```

Provider


```
import * as React from 'react';
const ContextKey = Symbol.for(`react-redux-context`);
const gT = typeof globalThis !== "undefined" ? globalThis :
/* fall back to a per-module scope (pre-8.1 behaviour) if `globalThis` is not available */
{};

function getContext() {
  var _gT$ContextKey;

  if (!React.createContext) return {};
  const contextMap = (_gT$ContextKey = gT[ContextKey]) != null ? _gT$ContextKey : gT[ContextKey] = new Map();
  let realContext = contextMap.get(React.createContext);

  if (!realContext) {
    realContext = React.createContext(null);

    if (process.env.NODE_ENV !== 'production') {
      realContext.displayName = 'ReactRedux';
    }

    contextMap.set(React.createContext, realContext);
  }

  return realContext;
}

export const ReactReduxContext = /*#__PURE__*/getContext();
export default ReactReduxContext;
```

Connect


subscribed
React App
dispatch
Redux Store
(State)
Action
Reducer

Connect


```
import React from 'react'

function CakeContainer() {
  return (
    <div>
      <h2>Number of cakes</h2>
      <button>Buy Cake</button>
    </div>
  )
}

export default CakeContainer
```

Connect


```
import React from 'react'

function CakeContainer() {
  return (
    <div>
      <h2>Number of cakes</h2>
      <button>Buy Cake</button>
    </div>
  )
}

const mapStateToProps = state => {
  return {
    numOfCakes: state.numOfCakes
  }
}

export default CakeContainer
```

Connect


```
const mapStateToProps = state => {
  return {
    numOfCakes: state.numOfCakes
  }
}

const mapDispatchToProps = dispatch => {
  return {
    buyCake: () => dispatch(buyCake())
  }
}

export default CakeContainer
```

```
function CakeContainer (props) {
  return (
    <div>
      <h2>Number of cakes - {props.numOfCakes}</h2>
      <button onClick={props.buyCake}>Buy Cake</button>
    </div>
  )
}
```

Connect


```
import React from 'react'
import { connect } from 'react-redux'
import { buyCake } from '../redux'
```

```
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(CakeContainer)
```

mapStateToProps


```jsx
<Provider store={store}>
  <div className='App'>
    <ItemContainer cake />
    <ItemContainer />
    <HooksCakeContainer />
    <CakeContainer />
    <IceCreamContainer />
    <NewCakeContainer />
  </div>
</Provider>
```

```jsx
function ItemContainer (props) {
  return (
    <div>
      <h2>Item - {props.item} </h2>
    </div>
  )
}

const mapStateToProps = (state, ownProps) => {
  const itemState = ownProps.cake
    ? state.cake.numOfCakes
    : state.iceCream.numOfIceCreams

  return {
    item: itemState
  }
}
```

```jsx
export default connect(mapStateToProps)(ItemContainer)
```

mapDispatchToProps


```
const mapDispatchToProps = (dispatch, ownProps) => {
  const dispatchFunction = ownProps.cake
  ? () => dispatch(buyCake())
  : () => dispatch(buyIceCream())

  return {
    buyItem: dispatchFunction
  }
}
```

Hooks

Hooks React - это API, которые предоставляют функциональным компонентам возможность управлять состоянием, обрабатывать побочные эффекты и многое другое, сохраняя при этом чистый и сжатый код.

В react-redux до появления перехватчиков, API-интерфейс *connect()* компонент более высокого порядка, который считывает значения из хранилища Redux при каждом обновлении.

API *connect()* принимает два необязательных аргумента:

- *mapStateToProps*: всякий раз, когда происходят обновления состояния хранилища Redux, оно получает полное состояние и возвращает объект данных, необходимый компоненту.
- *mapDispatchToProps*: это может быть функция или объект. Получает dispatch в качестве аргумента и возвращает объект функций, которые *отправляют* actions. Как объект, она содержит создателей действий, которые обращаются к props, которые автоматически отправляют actions при вызове.

Hooks

С появлением hooks команда React создала *useSelector*, *useDispatch* и другие hooks, которые упрощают работу с React-Redux и заменяют API *connect()*

Проведем аналогию между hooks и mapStateToProps и mapDispatchToProps

Hooks

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Что такое useSelector?

Hook *useSelector* аналогичен *mapStateToProps*.

Он подписывается на хранилище Redux, запускает предоставленную функцию после каждой отправки и повторно обновляет компонент на основе обновлений состояния.

Существенное различие между ними заключается в том:

- *mapStateToProps* передает несколько значений в качестве реквизитов
- *useSelector* принимает текущее состояние в качестве аргумента, возвращает требуемые данные и сохраняет возвращаемое значение в виде отдельной переменной вместо props.

Hooks

Как использовать hook useSelector в React

приложение-счетчик с hook *useSelector*

Структура папок приведена:

```
todo-app-demo/
├─ node_modules/
├─ public/
├─ src/
│  ├─ App.js
│  ├─ App.test.js
│  ├─ index.css
│  ├─ index.js
│  ├─ reportWebVitals.js
│  ├─ setUpTests.js
├─ .gitignore
├─ package.json
├─ postcss.config.js
├─ README.md
├─ tailwind.config.js
├─ yarn.lock
```

Hooks


```
import { useSelector } from 'react-redux';

function App() {
  const value = useSelector((state) => state.value);

  return (
    <div className="flex flex-col space-y-5 items-center py-10">
      <h1 className="text-7xl font-bold">{value}</h1>
      <button className="p-2 bg-blue-700 hover:bg-blue-800 hover:shadow
font-semibold text-white text-xl rounded-lg"
      >
        Click me
      </button>
    </div>
  );
}

export default App;
```

Hooks

Что такое useDispatch?

hook *useDispatch* аналогичен в *API connect() mapDispatchToProps*.
Позволяет отправлять любое действие в хранилище, добавляя действие в качестве аргумента в переменную *dispatch*.

```
import { useDispatch, useSelector } from 'react-redux';
import increment from './actions/increment';

function App() {
  const value = useSelector((state) => state.value);

  const dispatch = useDispatch();

  const handleClick = () => {
    dispatch(increment);
  };

  return (
    <div className="flex flex-col space-y-5 items-center py-10">
      <h1 className="text-7xl font-bold">{value}</h1>
      <button
        onClick={handleClick}
        className="p-2 bg-blue-700 hover:bg-blue-800 hover:shadow font-semibold text-white text-xl rounded-lg"
      >
        Click me
      </button>
    </div>
  );
}

export default App;
```

Hooks

Hooks Redux (useSelector, useDispatch) против API connect()
Согласно официальной документации React-Redux, рекомендуется использовать hooks React-Redux по умолчанию; однако есть некоторые случаи, когда API *connect()* реализует функцию лучше из-за ее зрелости. При их использовании следует иметь в виду следующие вещи:

- connect() Устаревшие Props и “Children-зомби”: API имеет встроенный класс подписки, гарантирующий, что подключенные компоненты получают уведомления об обновлении хранилища только тогда, когда был обновлен ближайший подключенный предок. API connect() переопределяет внутренний контекст React и отображает <ReactReduxContext.Provider> с новым значением. Невозможно отобразить поставщика контекста в hooks Redux, поэтому useSelector может запускаться до появления новых обновленных реквизитов, что приводит к устареванию props и появлению “children”. Это крайний случай.

Hooks

- **Развязка:** логика контейнера (то, как данные из хранилища Redux вводятся в компонент) отделена от логики представления (рендеринга компонента) с помощью *mapStateToProps. useSelector* предоставляет свежий и отличный подход к подключенным компонентам, что компоненты являются автономными и разделение компонентов более важно.

- **Типы:** Использовать TypeScript в API *connect()* может быть непросто. Намного проще использовать TypeScript с hooks Redux.

- **Опыт разработчика:** hooks Redux не требуют написания большого количества кода. Они намного проще в использовании, чем *connect()*, что требует при создании компонента контейнера для каждого подключенного компонента. Hooks также более понятны новичкам, чем *connect()*.

- **Размер пакета:** hooks Redux имеют гораздо меньший размер пакета, что ускоряет их установку и реализацию в коде.

- **Будущее:** поскольку React больше ориентируется на функциональные компоненты и hooks, hooks Redux более перспективны, чем API *connect()*.

Hooks

- useSelector использует по умолчанию строгое равенство для сравнения объектов, которые возвращает селектор (из-за этого в случае возврата нового объекта компонент постоянно будет перерисовываться) и нужно использовать свой метод для сравнения. Или можно написать свой хук:

```
import { useSelector, shallowEqual } from 'react-redux';

export function useShallowEqualSelector(selector) {
   return useSelector(selector, shallowEqual);
}
```

И использовать его:

```
export const AwesomeReduxComponent = () => {
    // Хук необходим, если селектор возвращает новый объект
    const { count } = useShallowEqualSelector(state => {
        count: state.counter.count;
    });
    const dispatch = useDispatch();

    return <div />;
};
```

Hooks

В отличие от connect, хук useSelector не предотвращает повторный ререндер компонента, когда перерисовывается родитель, даже если props не изменились. Поэтому для оптимизации стоит использовать React.memo():

```
export const AwesomeReduxComponent = React.memo(() => {
    // Хук необходим, если селектор возвращает новый объект
    const { count } = useShallowEqualSelector(state => {
        count: state.counter.count;
    });
    const dispatch = useDispatch();

    return <div />;
});
```

Hooks

При передаче *callback-а* с *dispatch* дочерним компонентам следует оборачивать метод в *useCallback*, что бы дочерние компоненты не рендерились без необходимости:

```
export const AwesomeReduxComponent = React.memo(() => {
    // Хук необходим, если селектор возвращает новый объект
    const { count } = useShallowEqualSelector(state => {
        count: state.counter.count;
    });
    const dispatch = useDispatch();
    const onClick = useCallback(
        () => dispatch(incrementCount()),
        [dispatch]
    );

    return <div />;
});
```

Hooks

- **Усложнение тестирования**. Для тестирования компонента придется всегда создавать стор и оборачивать компонент в *ReduxProvider*, т.е. придется писать интеграционные тесты. В случае с *connect*, мы можем экспортировать компонент и тестировать его независимо.

- **Нарушение принципа единой ответственности**. Компонент становится ответственным за слишком многое, тем самым становится более сложным. Дядюшка Боб будет недоволен.

- **Дебаг**. В своем тестовом приложении я могу изменять значения пропсов компонента в *dev tools* (которые приходят из *connect-a*) и смотреть, как компонент будет выглядеть в таком случае.

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание