[Lesson 5-7]

[DAN.IT EDUCATION]

[Что мы рассмотрим:]

- Что такое middleware
- Написание своего простейшего Middleware
- redux-logger
- Ассинхронные запросы с использованием redux-thunk

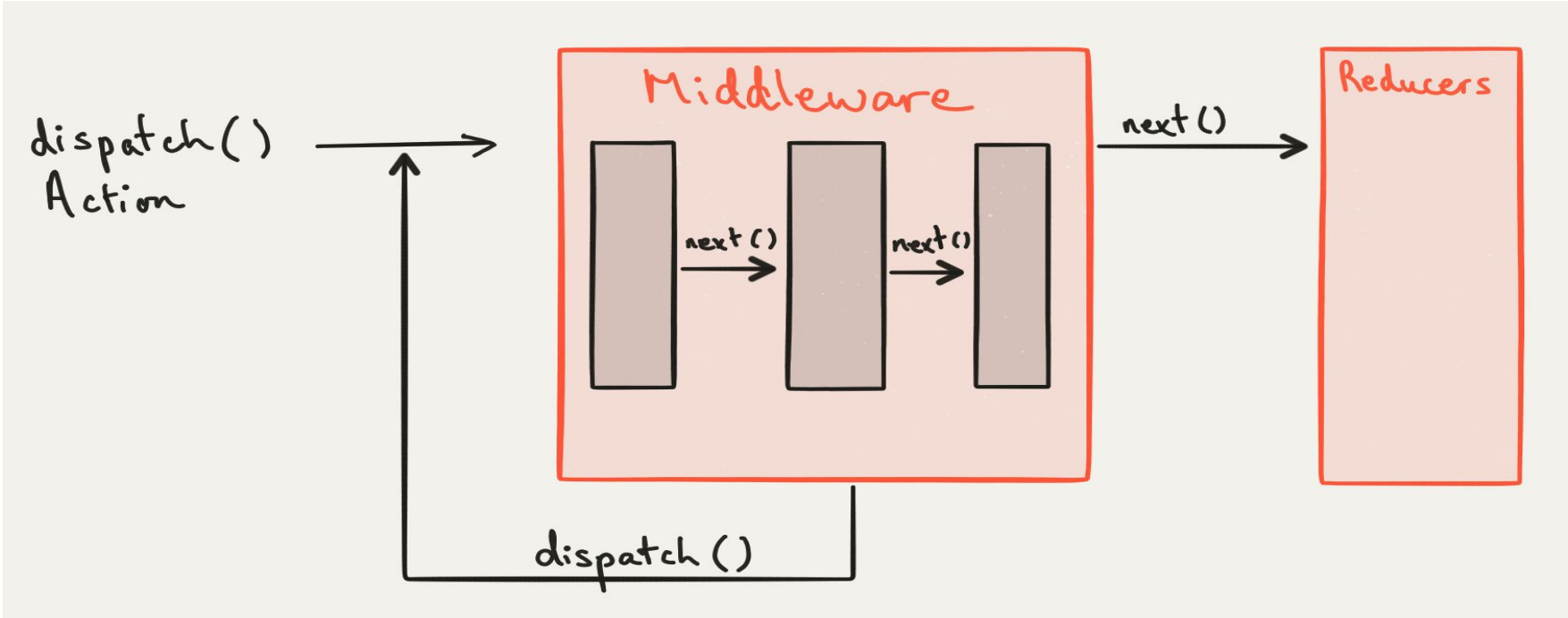
[DAN.IT EDUCATION]

[Redux Middleware]

Middlewares (посредники) - это промежуточная функция которая берет исходные данные, производит какие-либо действия с ними, и передает полученный результат далее.

Другими словами, суть middleware функций, взять входные данные, добавить что-то и передать дальше.

[Redux]


dispatch()
Action
Middleware
next()
next()
next()
Reducers
dispatch()

[Redux]

./src/store.js

```
import {createStore, applyMiddleware} from 'redux'
import rootReducer from './reducers/index’
import {myLogger} from ‘./middlewares/myLogger’

const store = createStore(rootReducer, applyMiddleware(myLogger))

export default store;
```

[DAN.IT] EDUCATION

[Redux]

./src/middlewares/myLogger.js

```
export const myLogger = store => next => action => {
   console.log(`Событие: ${action.type}, payload данные:`, action.payload )
   return next(action)
}
```

[Redux]

```
npm i redux-logger --S
```

./src/store.js

```
import {createStore, applyMiddleware} from 'redux'
import rootReducer from './reducers/index’
import logger from ‘redux-logger’

const store = createStore(rootReducer, applyMiddleware(logger))

export default store;
```

Асинхронные actions

Синхронный Action

- Пользователь кликнул на кнопку
- dispatch action {type: ТИП_ДЕЙСТВИЯ, payload: доп.данные}
- интерфейс обновился

Асинхронный Action

- Пользователь кликнул на кнопку
- dispatch action {type: ТИП_ДЕЙСТВИЯ_ЗАПРОС}
- запрос выполнился успешно
  dispatch action {type: ТИП_ДЕЙСТВИЯ_УСПЕШНО, payload: доп.данные}
- запрос выполнился неудачно
  dispatch action {type: ТИП_ДЕЙСТВИЯ_НЕУДАЧНО, error: true, payload: доп.данные ошибки}

[Redux]

```
npm i redux-thunk --S
```

./src/store.js

```
import {createStore, applyMiddleware} from 'redux'
import rootReducer from './reducers/index’
import logger from ‘redux-logger’
import thunk from 'redux-thunk'

const store = createStore(rootReducer, applyMiddleware(thunk, logger))

export default store;
```

[Redux]

./src/reduceres/mails/index.js

```
const initialState = {
  ...,
    isFetching: false
}
```

```
case DEL_MAIL_REQUEST:
   return { ...state, isFetching: true }
case DEL_MAIL_SUCCESS:
    return {...state, ...action.payload, isFetching: false }
```

[DAN.IT] EDUCATION

[Redux]

./src/actions/mails.js

```
export const DEL_MAIL_REQUEST = 'DEL_MAIL_REQUEST'
export const DEL_MAIL_SUCCESS = 'DEL_MAIL_SUCCESS'
export function delEmail(delID, mailList) {
  return dispatch => {
     dispatch({
     type: DEL_MAIL_REQUEST,
   })
     setTimeout(() => {
       let newMailList = {...mailList }
       for(let key in newMailList){
           newMailList[key] = newMailList[key].filter( (item) => {
                return item.id != delID ? item : null
           })
       }
     dispatch({ type: DEL_MAIL_SUCCESS, payload: {mailList: newMailList} })
           }, 1000)
     }
}
```

[DAN.IT] EDUCATION

[Redux]

./src/components/MailList/index.js

```
const mapDispatchToProps = dispatch => {
  return {
    delEmail: (delID, mailList) => dispatch(delEmail(delID, mailList)),
  }
}
```

```
const mapStateToProps = (state) => {
     return {
          mails: state.mails.mailList,
          isFetching: state.mails.isFetching
     }
}
```

[DAN.IT] EDUCATION

[Redux]

./src/components/MailList/index.js

```
return (
   <Fragment>
      <div className="mail-list">
          <ul>
            {this.props.isFetching ? <li>Загрузка...</li> : mailList}
          </ul>
      </div>
   </Fragment>
);
```