



REDUX

Zsolt German
zsolt_german@epam.com

MARCH 14, 2018

Agenda

- About
- Basics
- Best Practices
- Usage with React
- Advanced techniques

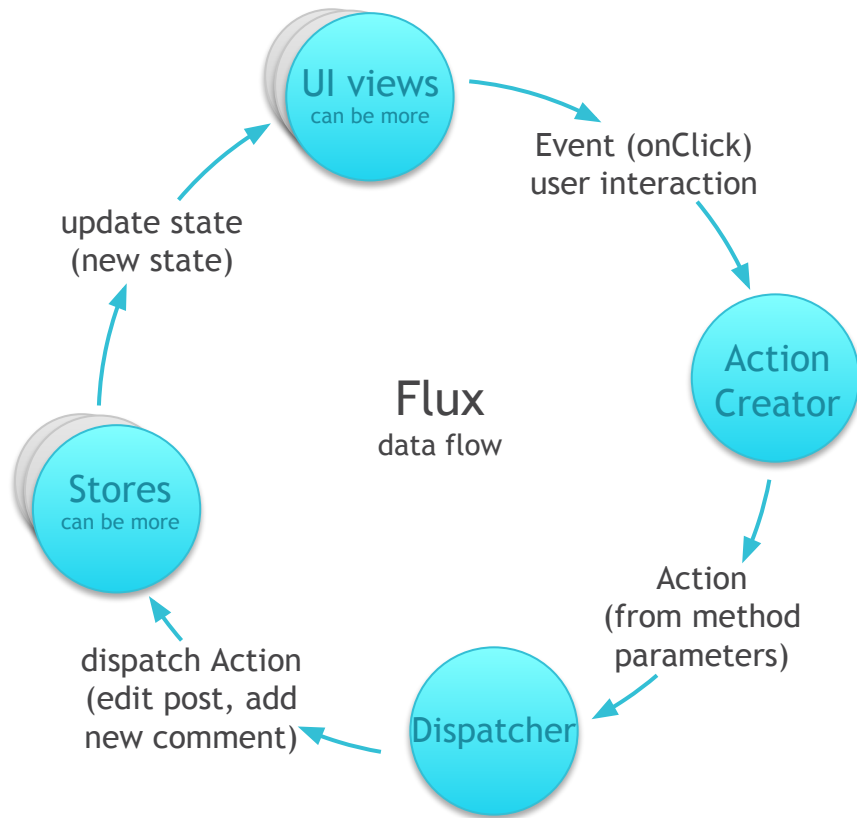


Redux

ABOUT FLUX AND REDUX

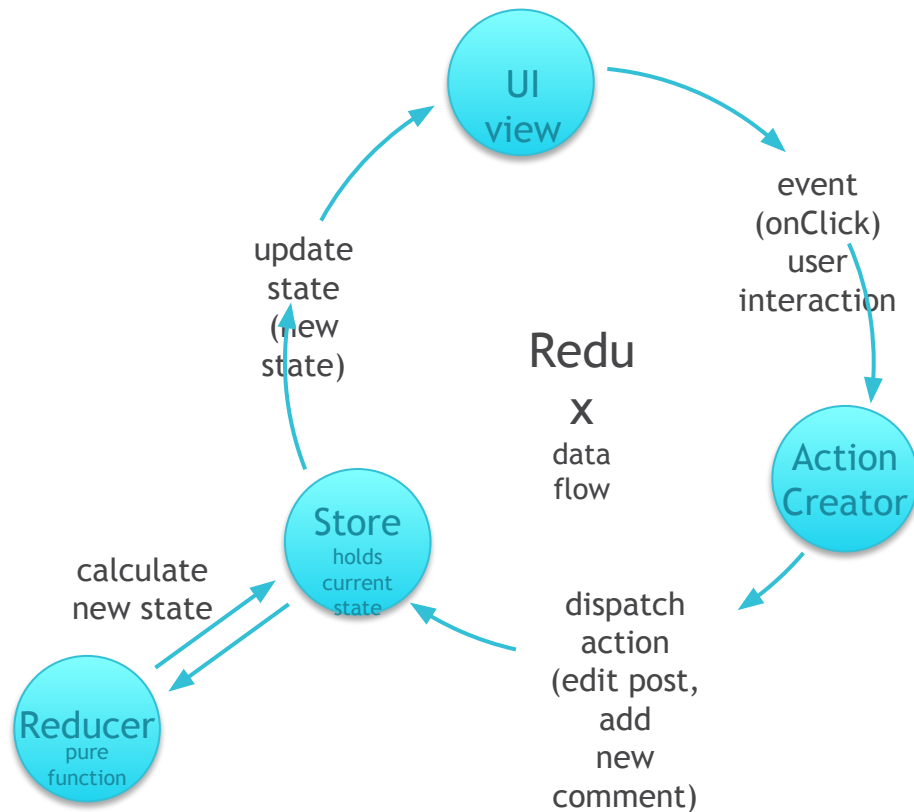
FLUX

- Application architecture
- Pattern
- By Facebook
- Replace MVC model
- Predictable data
- Unidirectional data flow
- Complements React UI



Redux

- State management library
- Inspired by Flux
- Tiny library
- Small dependency (just using a polyfill)
- Use with React, Angular, etc.
- Centralized Store



Three Principles

1

Single source of truth

- Store saves a single state tree
- Easy debug and inspection
- Undo/redo became trivial

2

State is read-only

- Actions describe what happened
- Changes are centralized
- Easy log, serialize, replay

3

Changes made with pure functions

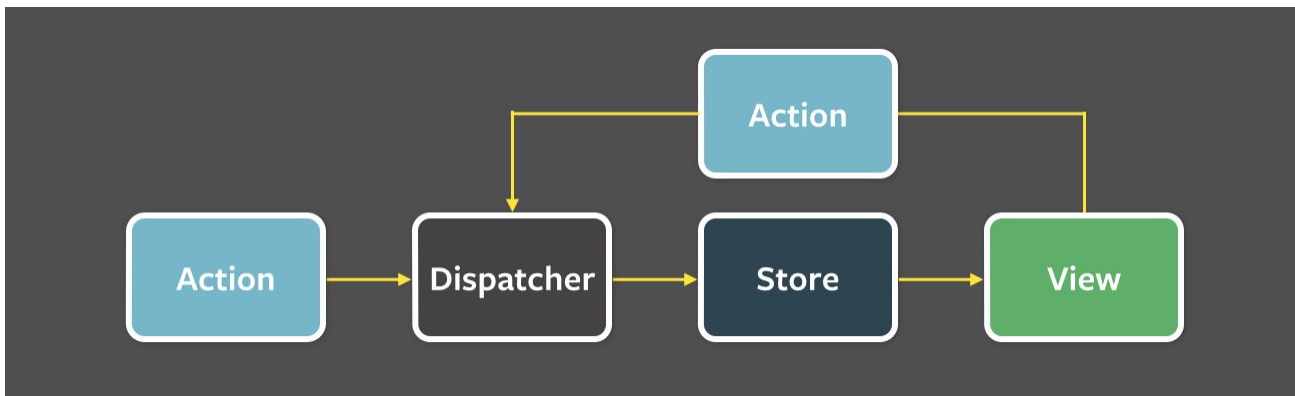
- Reducer transforms the state tree
- New state, no mutation
- Code splitting, reusability



Redux

BASICS

Redux



Install and Import Redux

- Install Redux as development dependency
- Import functions where you use them

Install Redux with npm

```
npm install --save redux
```

Import Redux in code

```
import { createStore } from 'redux';
```

Design State Shape

- All app state
 - Single object
 - Keep minimal
-
- ▣ *Store* provide

State examples

```
// Example when user logged out
{
  logged: false,
  username: ''
}
```

```
// Example when user logged in
{
  logged: true,
  username: 'username'
}
```

Store

- Holds app state
- `createStore()`: init Store
- `.getState()`: get state
- `.dispatch()`: “update” state
- `.subscribe()`: register listener

- Unregister listener via *callback* (that `.subscribe()` returned)

./index.js

```
import { createStore } from 'redux';
import reducer from './reducers/authentication';

// Create store
export const store = createStore(reducer);

// Subscribe state changes
const unreg = store.subscribe(() => {
  // Get actual state
  console.log(store.getState())
});

// Dispatch actions
store.dispatch({ type: 'LOGIN', username: 'username' });
store.dispatch({ type: 'LOGOUT' });

// Unsubscribe listener
unreg();
```

Actions

- Plain objects
- Payloads of information

Action examples

```
// Example when user logs in
{
  type: 'LOGIN',
  username: 'username'
}
```

```
// Example when user logs out
{
  type: 'LOGOUT'
}
```

Reducers

- Specify state change
- Pure function:
 - No mutation
 - No side effect
- Returns new state object
- Default returns previous state

./reducers/authentication.js

```
const initState = { logged: false, username: '' };

const auth = (state = initState, action) => {
  switch (action.type) {
    case 'LOGIN':
      return Object.assign({}, state, {
        logged: true,
        username: action.username
      });
    case 'LOGOUT':
      return Object.assign({}, state, {
        logged: false,
        username: ''
      });
    default:
      return state;
  }
};

export default auth;
```



Redux

BEST PRACTICES

Redux

ACTION

CREATORS

Action Types and Action Creators

- Keep all action types in a separate file
 - All existing actions in one place
- Create action creators
 - More verbose code

actions/actionTypes.js

```
// Authentication
export const LOGIN = 'LOGIN';
export const LOGOUT = 'LOGOUT';

// User management
export const USER_ADD = 'USER_ADD';
export const USER_DELETE = 'USER_DELETE';
```

actions/authentication.js

```
import { LOGIN, LOGOUT } from './actionTypes';

export const login = (username) => ({
  type: LOGIN,
  username
});

export const logout = () => ({
  type: LOGOUT
});
```


Generate Action Creator

- You can generate action creators with factories
- You can use libraries for that like *redux-act* or *redux-actions*

actions/todo.js with action creator factory

```
function makeActionCreator(type, ...argNames) {
  return function (...args) {
    let action = { type };
    argNames.forEach((arg, index) => {
      action[argNames[index]] = args[index];
    })
    return action;
  }
}
```

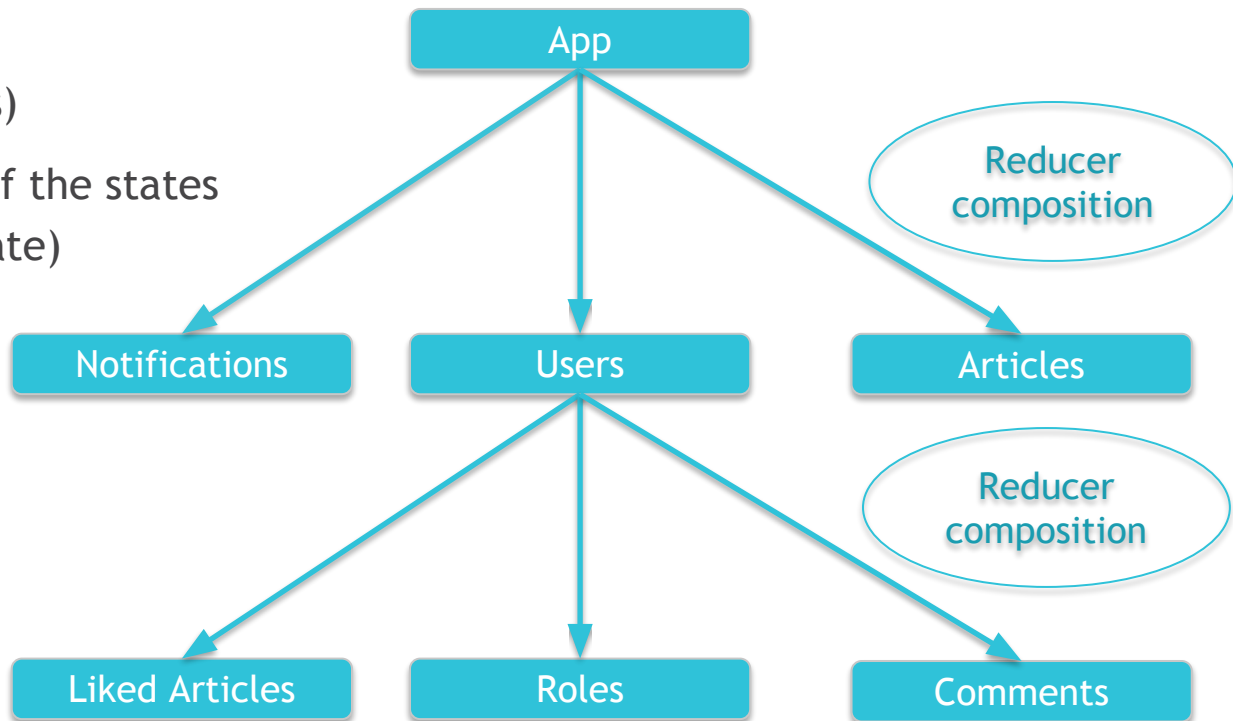
```
export const addTodo =
  makeActionCreator(ADD_TODO, 'text');
export const editTodo =
  makeActionCreator(EDIT_TODO, 'id', 'text');
export const toggleTodo =
  makeActionCreator(TOGGLE_TODO, 'id');
export const removeTodo =
  makeActionCreator(REMOVE_TODO, 'id');
```

Redux

REDUCERS

Reducer Composition

- App state has hierarchy (properties of properties)
- Write reducers on part of the states (reducer state != app state)
- Combine them (down to up)



Splitting Reducers

- When fields are independent
- Reducer composition with:
combineReducers()
- Reducers could split into files
- *state* is not the app state!

reducers/notifications.js

```
export const notifications = (state = 0, action) => {  
  switch (action.type) {  
    case SET_NOTIFICATIONS:  
      return action.notifications;  
    default:  
      return state;  
  }  
};
```

reducer.js

```
import notifications from 'reducers/notifications.js';  
import users from 'reducers/users.js';  
import articlesReducer from 'reducers/articlesReducer.js';  
  
export const reducer = combineReducers({  
  notifications,  
  users,  
  articles: articlesReducer  
});
```

Redux

MIDDLEWAR

E

Middleware

There can be several middleware entities, each performing its own useful role in an Application

Middleware is a *curried* function which receives current store, next middleware in the chain, and current action

They are connected during the creation of store:

```
const logMiddleware = store => next => action => {  
  console.log(action);  
  next(action);  
};
```

Redux Thunk

```
const INCREMENT_COUNTER = 'INCREMENT_COUNTER';

function increment() {
  return {
    type: INCREMENT_COUNTER
  };
}

function incrementAsync() {
  return dispatch => {
    setTimeout(() => {
      // Yay! Can invoke sync or async actions with `dispatch`
      dispatch(increment());
    }, 1000);
  };
}
```

Using Middleware

- 3rd party extension point
- Between action and reducer
- For logging, routing, etc.
- Async middleware, e.g.: *redux-thunk*

./middleware/logger.js

```
export default store => next => action => {
  console.log('dispatching', action);

  let result = next(action);
  console.log('next state', store.getState());

  return result;
}
```

./index.js

```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import logger from './middleware/logger.js';
import appReducer from './appReducer';

let store = createStore(
  appReducer,
  applyMiddleware(logger, thunk)
);
```


Action Creators with Validation

- *redux-thunk* middleware
 - **Thunk**: a subroutine used to inject additional calculation into another subroutine
 - Action creators could also **return a callback function**
 - Provide Store's *dispatch()* and *getState()* for callback functions
 - Allows **async** calculation
- For **validation** use callback function instead of action object

actions/authenticationWithValidation.js

```
import { LOGIN } from './actionTypes';

const loginWithoutCheck = (username) => ({
  type: LOGIN,
  username
});

export const login = (username) =>
  (dispatch, getState) => {
    if (getState().logged) {
      return;
    }
    dispatch(loginWithoutCheck(username));
  };

dispatch(login(username));
```

Redux

SUMMAR

Y

Summary of Data Flow with Best Practices

1

**You
call
dispatch**

- Create Action via Action Creator
- Action describes what happened

2

**Store
calls
reducer**

- In: *Previous state* and *Action*
- Out: Next state

3

**Root reducer
combines
output tree**

- *combineReducers()*

4

**Store
saves
whole state**

- Listeners invoked
- Bind to UI (later): *react-redux*

A busy city street scene at sunset. The sun is low in the sky, creating a warm, golden glow. The street is filled with pedestrians, some carrying shopping bags. On the left, a large, ornate building with a Union Jack flag is visible. A sign for 'Superdry' is mounted on the building. In the foreground, a large blue banner with white text reads 'USAGE WITH REACT'. To the left of the banner, the word 'Redux' is written in white on a smaller blue background. The street is lined with buildings, some with scaffolding, and a Starbucks logo is visible on the right. A red awning is also present on the right side of the street.

Redux

USAGE WITH REACT

Redux

DESIGN

Interworking between React and Redux

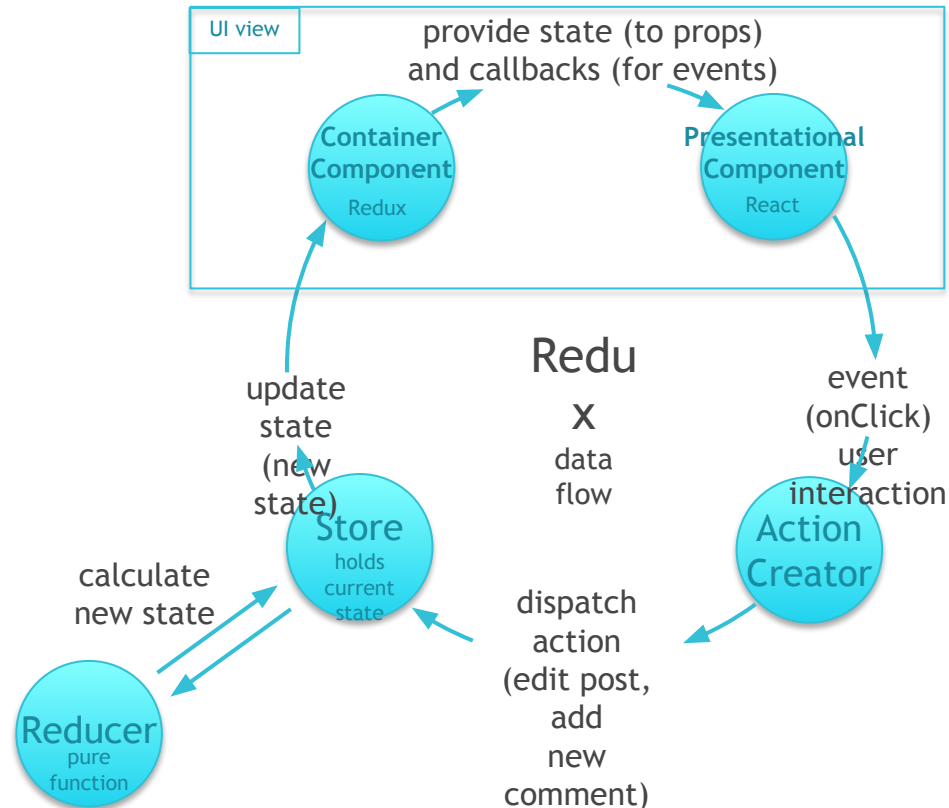
- Split UI view: logic and rendering

- *Container Component*

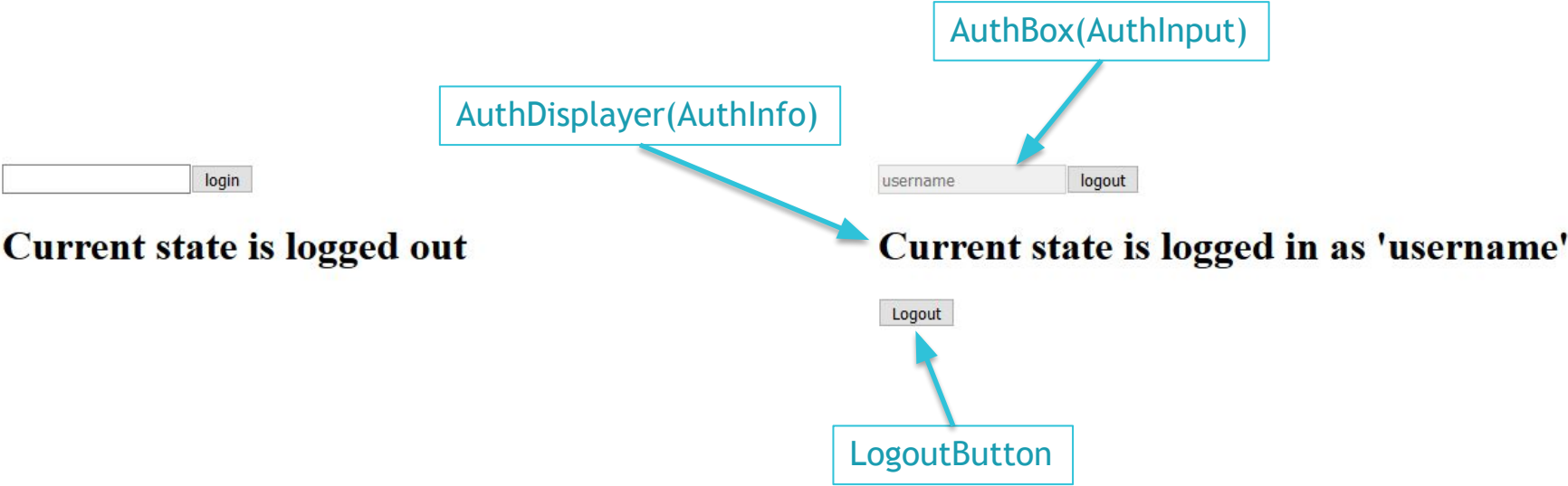
- Provide state parts via *props*
- Dispatch events via *callbacks*

- *Presentational Component*

- Using *props* to observe state changes
- Invoke *callbacks* on events



Application for design



Redux

BASIC COMPONENTS

Presentational Components

□ *Redux* not used here

□ Properties provided by
its container component
from state

components/AuthInfo.js

```
import React from 'react';

export const AuthInfo =
  ({ logged, username }) => (
    <h1>
      Current state is {
        'logged ' + (logged
          ? `in as '${username}'`
          : 'out')
        }
    </h1>
  );

export default AuthInfo;
```

Container Components

- ❑ *React* not used here
- ❑ These are just data providing
- ❑ No visual elements
- ❑ Use them in the *App.js* (later)

containers/AuthDisplayer.js

```
import { connect } from 'react-redux';
import AuthInfo from '../components/AuthInfo';

const mapStateToProps = state => ({
  logged: state.logged,
  username: state.username
});

const AuthDisplayer =
  connect(mapStateToProps)(AuthInfo);
export default AuthDisplayer;
```

Presentational Components with Local State

- Presentational components could have local state

components/AuthInput.js

```
export class AuthInput extends Component {
  constructor(props) {
    super(props);
    this.state = { username: '' };
  }
  handleChange = (event) => {
    this.setState({ [event.target.name]: event.target.value });
  }
  loginClick = () => {
    this.props.handleLogin(this.state.username);
  }
  logoutClick = () => {
    this.props.handleLogout();
    this.setState({ username: '' });
  }

  render = () => {
    const btnLabel = this.props.logged ? 'logout' : 'login';
    const btnClick = this.props.logged
      ? this.logoutClick : this.loginClick;
    return (
      <div>
        <input type="text" name="username" disabled={this.props.logged}
          onChange={this.handleChange} value={this.state.username} />
        <button type="button" onClick={btnClick}>{btnLabel}</button>
      </div>
    )
  }
}

export default AuthInput;
```

Container Components with Callbacks

□ Here we provided callbacks

containers/AuthBox.js

```
import { connect } from 'react-redux';
import { LOGIN, LOGOUT } from '../actions/actionTypes';
import AuthInput from '../components/AuthInput';

const mapStateToProps = (state, ownProps) => ({
  logged: state.logged
});
const mapDispatchToProps = (dispatch, ownProps) => ({
  handleLogin: (username) => {
    dispatch({ type: LOGIN, username });
  },
  handleLogout: () => {
    dispatch({ type: LOGOUT });
  }
});

export const AuthBox = connect(
  mapStateToProps, mapDispatchToProps
)(AuthInput);

export default AuthBox;
```

Redux

MIXED COMPONENTS

Presentational Container Components

- Use *React* and *Redux* also
- Got *dispatch* in *props*
- ▣ *connect* provides to the *presentational component part*

Presentational Component part

- Use only when logic is small
- Split as component grows

Container Component part

containers/LogoutButton.js

```
import React from 'react';
import { connect } from 'react-redux';
import { LOGOUT } from '../actions/actionTypes';

const LogoutButtonLayout = ({dispatch, logged}) => {
  if (!logged) {
    return false;
  }
  const handleLogout = () => {
    dispatch({ type: LOGOUT });
  }
  return (
    <button type="button"
      onClick={handleLogout}>Logout</button>
  )
}

const mapStateToProps =
  state => ({ logged: state.logged });
export const LogoutButton =
  connect(mapStateToProps)(LogoutButtonLayout)
export default LogoutButton;
```

Redux

SUMMARY OF COMPONENTS

Compare Presentational and Container Components

□ Presentational Components

- Design with *React*
- Using props
- Invoke prop callbacks
- Should implement

□ Container Components

- Business logic with *Redux*
- Subscribe state
- Dispatch actions
- Generated by *react-redux*

Install react-redux

```
npm install --save react-redux
```


Redux

CONNECT REDUX AND REACT

Passing the Store

- Create the *App* component
- Use *Provider* from *react-redux*

components/App.js

```
export const App = () => (  
  <div>  
    <AuthBox />  
    <AuthDisplayer />  
    <LogoutButton />  
  </div>  
);  
export default App;
```

index.js

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import { createStore } from 'redux';  
import { Provider } from 'react-redux';  
import auth from './reducers/auth';  
import App from './components/App';  
const store = createStore(auth);  
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>,  
  document.getElementById('root')  
);
```



Redux

ADVANCED TECHNIQS

Redux

IMMUTABLE.J

S

Benefits of Immutable.js

- Functional Programming
- Avoid bugs
- Performance
- Rich API

Bad things:

- Interoperate is hard
(avoid `.toJS()` calls)
(never mix with plain objects)
- No destructuring and spread operator
(more verbose code)
- Slower on small often changed values
(not the case in Redux)
- Harder debug
(use object formatter)

Redux

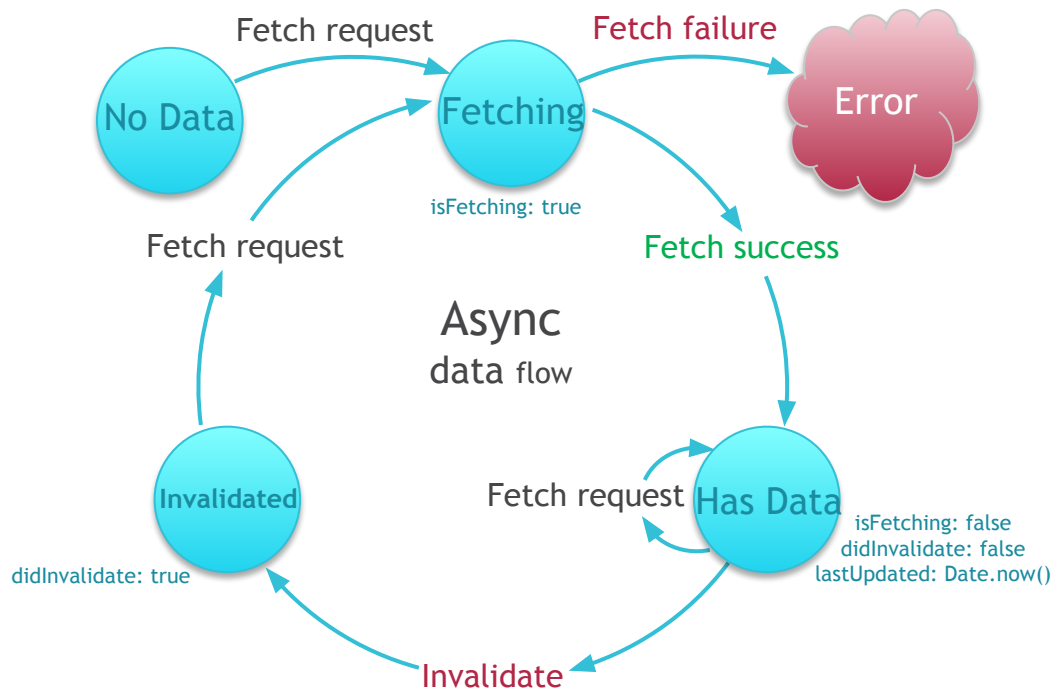
ASYNC DATA

FLOW

Async Data Flow

Actions:

- POSTS_FETCH_REQUEST:
 - `isFetching`: true
- POSTS_FETCH_SUCCESS:
 - `isFetching`: false
 - `didInvalidate`: false
 - `lastUpdated`: Date.now()
- POSTS_FETCH_FAILURE:
 - Handle error
- POSTS_INVALIDATE:
 - `didInvalidate`: true



Async State Shape

- Some state variable needed for store async process status:
 - *isFetching*: the fetch has begun
 - *didInvalidate*: refresh needed
 - *lastUpdated*: last fetch time
- These should be handled in the reducer.

Example of Async State Shape

```
{
  selectedUser: 'user1',
  posts: {
    12: { id: 12, post: '...' }
  },
  postsByUsers: {
    'user1': {
      items: [12],
      isFetching: false,
      didInvalidate: false,
      lastUpdated: 1439478405547
    }
  }
}
```


Fetch in Redux

- Create action creators
(*same as before*)
- Create reducer (*same as before*)
- Create thunk that use action creators
("async action creator")
- Dispatch with the thunk

- This code always fetches
 - Create another thunk that use ***fetchPosts()*** when needed

Fetch in Redux

```
const requestPosts = (user) => ({
  type: REQUEST_POSTS,
  user
});
const receivePosts = (user, json) => ({
  type: RECEIVE_POSTS,
  user,
  posts: json.data.children.map(child => child.data),
  receivedAt: Date.now()
});
export const invalidateSubreddit = (user) => ({
  type: INVALIDATE_POSTS,
  user
});

const fetchPosts = user => dispatch => {
  dispatch(requestPosts(user));
  return fetch(`https://www.reddit.com/r/${user}.json`)
    .then(response => response.json())
    .then(json => dispatch(receivePosts(user, json)))
};

// store.dispatch(fetchPosts('reactjs'))
// .then(() => console.log(store.getState()));
```

Fetch with Checks

- Create a function for the **condition**
- Create the new thunk that **invoke** the previous thunk when condition true
- Dispatch is the same
- Use the thunk in the UI as used action creators before

Async Actions

```
const shouldFetchPosts = (state, user) => {
  const posts = state.postsByUser[user];
  if (!posts) {
    return true;
  } else if (posts.isFetching) {
    return false;
  } else {
    return posts.didInvalidate;
  }
};

const fetchPostsIfNeeded = user => (dispatch, getState) => {
  if (shouldFetchPosts(getState(), user)) {
    return dispatch(fetchPosts(user));
  } else {
    return Promise.resolve();
  }
};

// store.dispatch(fetchPostsIfNeeded('reactjs'))
//   .then(() => console.log(store.getState()));
```

Redux

RESELECT

Computing Derived Data

- Render todos with filter
- Re-render without change:
 - every time *todos* are same value,
 - but different reference
- It makes change detection inefficient
- Performance issue because rendering

- Solve this with *reselect* library

containers/VisibleTodoList.js

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

const getVisibleTodos = (todos, filter) => {
  switch (filter) {
    case 'SHOW_ALL':
      return todos;
    case 'SHOW_COMPLETED':
      return todos.filter(t => t.completed);
    case 'SHOW_ACTIVE':
      return todos.filter(t => !t.completed);
  }
}

const mapStateToProps = state => ({
  todos: getVisibleTodos(state.todos, state.filter)
});

export const VisibleTodoList = connect(
  mapStateToProps
)(TodoList);
```

Efficiently Compute Derived Data with *reselect* Library

□ *createSelector()*:

creates memorized selector

□ When

- related state values are same (via input-selectors)
- then result is the same (via transform function)

Could split to:
selectors/*.js

□ **Problem:** couldn't reuse the selector

□ **Solution:** Make factories for selector and *mapStateToProps*

containers/VisibleTodoList.js

```
import { createSelector } from 'reselect';

const getFilter = state => state.filter;
const getTodos = state => state.todos;
export const getVisibleTodos = createSelector(
  [getFilter, getTodos],
  (filter, todos) => {
    switch (filter) {
      case 'SHOW_ALL':
        return todos;
      case 'SHOW_COMPLETED':
        return todos.filter(t => t.completed);
      case 'SHOW_ACTIVE':
        return todos.filter(t => !t.completed);
    }
  }
);

const mapStateToProps = state => ({
  todos: getVisibleTodos(state)});
export const VisibleTodoList = connect(
  mapStateToProps)(TodoList);
```

Redux

REDUX UNDO

Understanding Undo History

□ Use 3 variable in the root state:

- past: Array<T>
- present: T
- future: Array<T>

□ Cases:

- Undo
- Redo
- Handle other action

Example of Counter State with Undo History

```
// Count 0 to 9:  
past = [0,1,2,3,4,5,6,7,8]  
present = 9  
future = []  
  
// Undo 4 times:  
past = [0,1,2,3,4]  
present = 5  
future = [9,8,7,6]  
  
// Redo 2 times:  
past = [0,1,2,3,4,5,6]  
present = 7  
future = [9,8]  
  
// Decrement 4 times:  
past = [0,1,2,3,4,5,6,7,6,5,4]  
present = 3  
future = []
```

Undo History with *redux-undo*

- Use *redux-undo* library
- *distinctState()*: ignore actions that didn't result state change
- Dispatch actions with *ActionCreators.undo()*, *ActionCreators.redo()*, etc.

Install redux-undo

```
npm install --save redux-undo
```

reducers/undoableTodos.js

```
import undoable from 'redux-undo';  
import todos from '../reducers/todos';  
  
export const undoableTodos = undoable(  
  todos, { filter: distinctState() }  
);
```

index.js

```
import { ActionCreators } from 'redux-undo';  
  
store.dispatch(ActionCreators.undo());  
store.dispatch(ActionCreators.redo());  
  
store.dispatch(ActionCreators.jump(-2));  
store.dispatch(ActionCreators.jump(5));  
  
store.dispatch(ActionCreators.clearHistory());
```


Redux

REACT ROUTER

React Router

□ Redux:

source of truth of data

□ React Router:

source of truth of url

□ Cannot change URL in actions

□ *Cannot time travel*

□ *Cannot rewind action*

Redux and React Router

```
// Connect React Router with Redux App:
const Root = ({ store }) => (
  <Provider store={store}>
    <Router>
      <Route path="/:filter?" component={App} />
    </Router>
  </Provider>);

// Navigating with React Router:
const Links = () => <div>
  <Link to="/">Show All</Link>
  <Link to="/SHOW_ACTIVE">Show Active</Link>
  <Link to="/SHOW_COMPLETED">Show Completed</Link>
</div>;

// App gets the matched URL parameters,
// and provide components to their props
const App = ({ match: { params } }) => <div>
  <VisibleTodosList filter={params.filter || 'SHOW_ALL'} />
  <Links />
</div>;

// In container components you could use
// the matched parameter from ownProps
const mapStateToProps = (state, ownProps) => ({
  todos: getVisibleTodos(state.todos, ownProps.filter)
});
```

Redux

SUB-APP APPROACH

Isolating SubApps

□ Independent SubApps

- Won't share data
- Won't share actions
- Won't communicate each other

□ Useful for large teams

□ Each component have own store

subapps/SubApp.js

```
import subAppReducer from './subAppReducer.js';
export class SubApp extends Component {
  constructor(props) {
    super(props);
    this.store = createStore(subAppReducer);
  }
  render = () => (
    <Provider store={this.store}>
      <App />
    </Provider>
  )
}
```

app.js

```
import SubApp from './subapps/SubApp.js';
export const BigApp = () => (<div>
  <SubApp />
  <OtherSubApp2 />
  <OtherSubApp3 />
</div>);
```



Redux

QUESTIONS?



Redux

RESOURCES

A photograph of a busy city street at sunset. The sun is low in the sky, creating a warm, golden glow and long shadows. The street is filled with pedestrians, some carrying shopping bags. On the left, a large, ornate building with a Union Jack flag is visible. A sign for 'Superdry' is mounted on a pole. In the background, a building is under construction with scaffolding. A Starbucks logo is visible on a building to the right. A large teal banner with white text is overlaid across the middle of the image.

Redux

THANKS FOR YOUR
ATTENTION!