



React Native

Day 4





Redux

Motivação para o uso do Redux

- Nossos aplicativos têm sido relativamente simples, mas quando a complexidade aumenta, começamos a receber alguns erros
 - Estados ou propriedades aninhadas (*deeply nested state/props*)
 - Informações duplicadas nos estados
 - Não atualizar todas propriedades dependentes
 - Componentes com um grande numero de propriedades
 - A incerteza se o dado está realmente atualizado e quando foi atualizado

Complexidade

- Escala de complexidade: Facebook
 - O Facebook achou a arquitetura MVC muito complexa para suas aplicações
 - A complexidade se manifestou em bugs
 - O Facebook começou a pesquisar sobre fluxo de dados unidirecional (*Flux*)
- Flux é uma arquitetura usada pelo Facebook, que junto com o framework React é usado para construir aplicações web no client-side que trabalhem de forma reativa. Basicamente, uma forma de fluxo unidirecional de dados entre eventos e ouvintes

FLUX

- As Views mudam com base nas alterações na **store**
- A única coisa que pode atualizar dados em uma **store** é o **dispatcher**
- A única maneira de acionar o **dispatcher** é invocando **actions**
- **Actions** são acionadas a partir das Views

Redux

- Uma biblioteca de gerenciamento de dados inspirada no Flux
- Conceitos do Redux
 - Única fonte de verdade para dados
 - O estado só pode ser atualizado por uma **action** que aciona uma recomputação dos dados
 - Atualizações são feitas usando funções puras (*pure functions*)
 - Action → Reducer → Update Store

O estado central

O **redux** existe para solucionar dois problemas graves...

A injeção desnecessária de props

- Nativamente , o React Native só suporta a passagem de informações por **props** e **state**, que só podem ser passados de pai pra filho. Logo...
- Se o filho do filho do filho de um componente precisar de uma informação que só ele capta, então você vai ter que passar a informação:

Componente 1 > Componente 2 > Componente 3 > Componente 4

A origem não muito confiável de dados

Especialmente em apps maiores (abre o Facebook ou o YouTube), pode acontecer que:

- Você precisa de uma informação que está sendo captada em algum lugar... agora é só você lembrar onde
- Caso algum componente intermediário na linha de passagem de informação altere o dado por acidente... lá se foi 4 horas da sua vida até você descobrir porque não está funcionando o que você queria

A solução

Um único banco central que guarda todas as informações do seu app

- Evita a passagem de dados por uma hierarquia de componentes
- Evita a passagem de dados por componentes não relacionados
- Evita a ocultação da origem dos dados
- Enfim, simplifica sua vida na hora de debugar

Uma mudança de estado

Nada acontece magicamente...

Uma mudança de estado acontece em várias etapas.

0) O banco central é criado

```
// UserRedux.js
```

```
export const INITIAL_STATE =
```

```
Immutable({
```

```
    username: null,
```

```
    password: null,
```

```
})
```

```
state.user.username
```

```
state.user.password
```

```
// NavigationRedux.js
```

```
export const INITIAL_STATE =
```

```
Immutable({
```

```
    currentRoute: 'HomeScreen'
```

```
})
```

```
state.navigation.currentRoute
```

A função **combineReducers()** no arquivo **root** do Redux (**index.js**) cuida da centralização de estados

```
const rootReducer = combineReducers({  
  navigation: require('./NavigationRedux').reducer,  
  user: require('./UserRedux').reducer,  
})
```

0) Conectar o componente ao Redux

- O banco ou estado central encontra-se **fora da hierarquia** de componentes
 - Devemos **conectar** nossos componentes a ele
 - Fazemos isso através da função **connect()**
 - Feito isso, **retornamos** para a árvore de componentes, um componente já conectado ao redux
-
- O **connect** recebe dois parâmetros que vamos discutir mais tarde

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


```
...
```

```
export default connect(...)(MyComponent)
```




E então, o usuário aperta um botão...





Ou talvez um timer desperta a cada minuto
para atualizar alguma informação...



1) A mudança é desencadeada

1. O redux precisa de um **objeto** que represente essa mudança
2. Esse objeto é geralmente chamado de **ação** ou **action**
3. Ele obrigatoriamente possui um **tipo** ou **type**, que facilita identificar uma mudança entre várias

// Actions

```
{type: GET_ACTION_MOVIES}
```

```
{type: LOGOUT}
```

```
{type: LOGIN_SUCCESS, username: 'myUserName', password: 'secret'}
```

```
{type: LOGIN_FAILURE, error: 'message'}
```

```
{type: CADASTRAR, name: 'Aryella Lacerda', cpf: '123-234-456-67'}
```

// Action creators

```
Actions.getActionMovies()
```

```
Actions.logout()
```

```
Actions.loginSuccess('myUserName', 'secret')
```

```
Actions.loginFailure('message')
```

```
Actions.cadastrar('Aryella Lacerda', '123-234-456-67')
```

Quais são os action types?

2) A ação é enviada ao redux

- Nesse momento, você deve chamar uma função do redux com um nome bem intuitivo: o **dispatch()**
 - Quando você **estabelece a conexão** do seu componente com o redux...
-
- 1) Você **cria uma função despachante** que utiliza o dispatch()
 - 2) Você **liga** essa função nova ao props, para facilitar seu acesso

Uma das funções do **connect()** é inserir essas funções despachantes no **props**

```
// Toda conexão com o redux é abstraída dentro do componente
```

```
<Button onPress={this.props.onPress(this.state.username,  
this.state.password)} />
```

```
...
```

```
// Fora do componente ocorre a conexão real
```

```
const mapDispatchToProps = (dispatch) => ({  
  onPress: (username, password) => {  
    dispatch(Actions.cadastrar(username, password))  
  }  
})
```

```
export default connect(..., mapDispatchToProps)(MyComponent)
```

3) Redux recebe a ação e direciona ela

- Imagina-se que você quer alterar o estado de uma maneira específica, **dependendo da ação** que você despachou
- **Então você deve criar uma função que recebe uma ação e altera o estado**
- O redux não faz isso automaticamente para oferecer a você total controle, pois uma ação pode afetar o estado de maneira “inesperada”
- Uma ação `Type.LOGOUT`, por exemplo, não recebe parâmetro, mas mesmo assim altera o estado do usuário:
 - `username = null`
 - `password = null`

```
export const reducer = createReducer(INITIAL_STATE, {  
  [Types.CADASTRAR]: cadastrar,  
  [Types.LOGIN_SUCCESS]: loginSuccess,  
  [Types.LOGIN_FAILURE]: loginFailure,  
  [Types.LOGOUT]: logout,  
})
```

4) O ação chega no reducer apropriado

- Um **reducer** é simplesmente uma função que recebe uma ação (um objeto que representa a mudança de estado) e cria outro estado novinho, alterado como você pediu


```
export const getActionMovies = (state) => state
```

```
export const loginSuccess = (state, {username, password}) => {  
  return state.merge({ username, password })  
}
```

```
export const logout = (state) => {  
  return state.merge({ username: null, password: null })  
}
```



0 reduxsauce



Alguém um dia pensou...

Que dava trabalho demais criar todos os tipos, e todas as ações, manualmente. Aí essa boa alma decidiu facilitar nossas vidas.

// Actions

{type: LOGOUT}

{type: LOGIN_SUCCESS, username: 'myUserName', password: 'secret'}

{type: LOGIN_FAILURE, error: 'message'}

{type: CADASTRAR, name: 'Aryella Lacerda', cpf: '123-234-456-67'}

// Action creators

Actions.logout()

Actions.loginSuccess('myUserName', 'secret')

Actions.loginFailure('message')

Actions.cadastrar('Aryella Lacerda', '123-234-456-67')

```
const { Types, Creators } = createAction({  
  logout: null,  
  loginSuccess: ['username', 'password'],  
  loginFailure: ['message'],  
  cadastrar: ['nome', 'cpf'],  
})
```



E assim, a saga da mudança de estado
encerra-se





Mas a história não acabou



Como você vai acessar o state agora?

- Quando você faz a **conexão** com o redux, você **busca** todas as propriedades do estado que acha interessante
- Essas propriedades do estado central são injetadas no componente com **props** normais


```
// Dentro do componente, o redux é abstraído totalmente
```

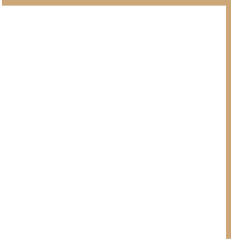
```
<Text>{this.props.username}</Text>
```

```
// Fora do componente, você busca as informações necessárias
```


```
const mapStateToProps = (state) => ({  
  username: state.user.username,  
  error: state.user.error,  
})
```

```
// Exporta o componente conectado
```

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



Agora sim, acabou



Referências

Redux

Reduxsauce