Abandonando redux

Como gerenciador de estado

Henrique Sosa

Software Engineer at QuintoAndar

@henriquesosa

Por que não gostar do redux?

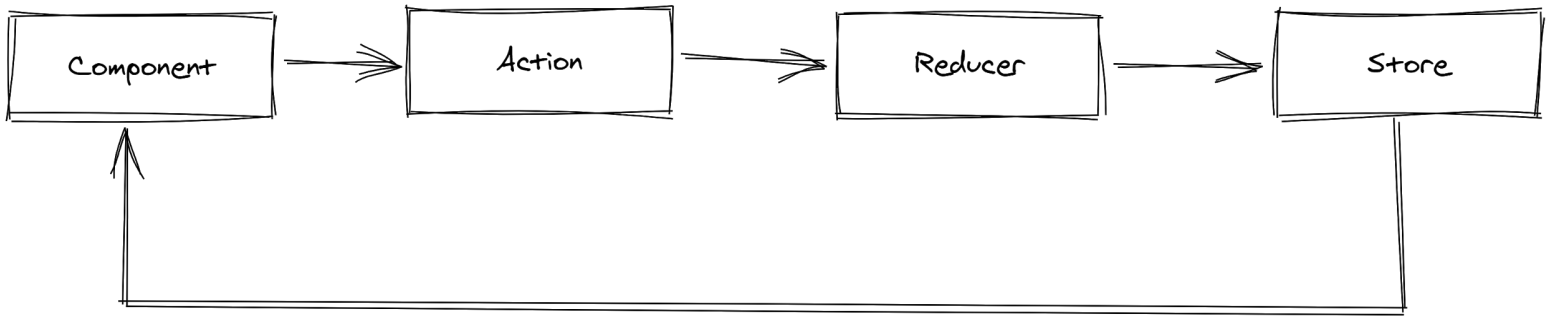
Eu não tenho nada contra o redux, muito pelo contrário. Mas o exercício de não pensar sobre nos faz imaginar novas possibilidades.

Mas vamos um pouco falar sobre redux

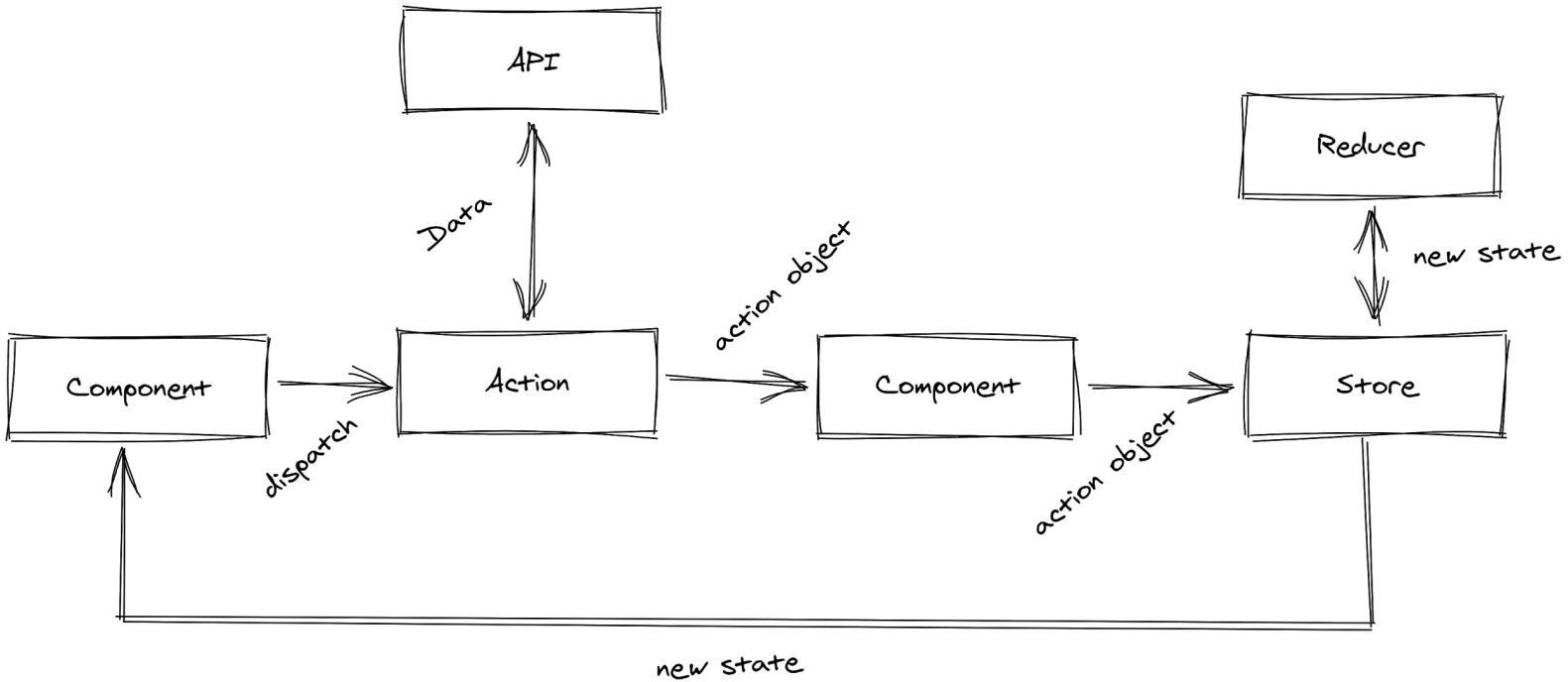
Em primeiro lugar, sou um grande fã do trabalho do Dan Abramov

Mas a maioria de seus exemplos e de outros comunidade afora, são de aplicações muito simples 😛

// A história que é contada



// A realidade


API
Data
Reducer
new state
Component
Action
action object
Component
Store
dispatch
action object
new state

Toda inscrição tem seu preço

- Mutabilidade
- Propagação
- Falta de responsabilidade
- Reducer é um condicional que pode ser custoso

E não começamos a falar sobre validações, apis, caching offline e mais.

Temos um aprendizado longo

- Reselect
- Redux
- Redux-pack
- React-redux
- Redux-thunk
- Redux-saga
- Immutable
- ...

Apenas para fazer um fetch e mostrar algo na tela

Hooks

Definição de hooks

Lançado na versão **16.8** do React.

Hook é uma maneira simplificada de acessar o ciclo de vida e o estado de um componente sem precisar escrever uma classe para isso.

React já implementa varios deles.

useState
useMemo
useEffect
useCallback

// Carregamento da API do Google de uma maneira simplificada.

```
class MyComponent extends Component {
 state = {
   googleReady: false,
 }

 componentDidMount() {
   this.loadGoogleApi();
 }

 componentDidUpdate() {
   if (this.state.googleReady) {
     this.props.fetchHouses(selectedCity, this.googleAPI);
   }
 }

 loadGoogleApi = () => {
   const googleApiKey = getGoogleApiKey();
   return loadGoogleApi(googleApiKey, this.props.intl.language, GOOGLE_LIBRARIES)
     .then((googleApi) => {
       this.googleAPI = googleApi;
       this.setState({ googleReady: true });
     }).catch((e) => {
       Raven.captureException(`Error on googleApi.loadAPI() ${e}`, { googleApiKey, GOOGLE_LIBRARIES });
     });
 }

 render() {
   return (
     <ChildComponent houses={this.props.houses} />
   )
 }
}
```

// Usando a mesma API com hooks

```
const MyComponent = () => {
 const [ googleApi ] = useGoogleAPI(); // middleware hook
 const [ houses, housesActions ] = useHouses(); // state hook
 const { fetchHouses } = housesActions; // usando uma action de um state hook

 useEffect(() => { // React hook
   fetchHouses({ googleApi });
 }, [googleApi]);

 return (
   <ChildComponent houses={houses} />
 )
}
```

E agora?

Regras definidas

- Componentes não podem conter nenhuma regra de validação ou lidar com dados diretamente
- APIs devem ser sempre normalizadas
- O estado deve ser presente em qualquer contexto
- O codigo deve ser **escalavel**, **limpo** e **mantenivel**.

Estado descentralizado

Cada **hook** é responsável pelo seu próprio contexto.

- useAppBar.js
- useInspection.js
- useReview.js
- useSettings.js

useGlobal.js

Hooks são diretamente conectados ao estado de vida de um componente. Por isso precisamos persistir os dados de alguma maneira.

State hooks

```
import useGlobal from '../utils/useGlobal';

const initialState = {
 showLogo: true,
 ...
};

export const actions = {
 toggleLogo: (store, show) => {
   store.setState({ showLogo: show });
 },
...
};

const useAppBar = useGlobal(initialState, actions);

export default useAppBar;
```

Usando um state hook

```
const [appBar, appBarActions] = useAppBar();
```

O projeto

- Pages
  - ExitInspectionReview.js
  - ...
- State
  - Api
    - Main.js
    - ...
  - Hooks
    - useAppBar.js
    - ...
  - Utils
    - useGlobal.js
    - ...

Pontos negativos

Falta de debug

Como esta implementação não é baseada numa linha do tempo tal como o redux, é preciso implementar manualmente logs e debuggers.

Redux mindset

De certa forma o redux criou um padrão onde e como as coisas devem ficar.

Suporte ao redux

A comunidade gastou tempo suficiente para amadurecer bibliotecas e ferramentas para facilitar nosso dia-a-dia.

Pontos positivos

Diminuição na curva de aprendizado.

- Reselect
- Redux
- Redux-pack
- React-redux
- Redux-thunk
- Redux-saga
- Immutable
- ...

Apenas para fazer um fetch e mostrar algo na tela.

Não há desperdício de memória

Quando o redux roda o bootstrap, a função **configStore()** faz com que todos os estados pré-definidos sejam instanciados.

Testes

A partir do momento que tornamos tudo um objeto de funções, podemos testar unitariamente de maneira fácil.

É **só** React.

Obrigado!