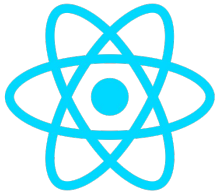


Gerenciamento de Estado no React

Pedro Pimenta
Dez 2019

Introdução

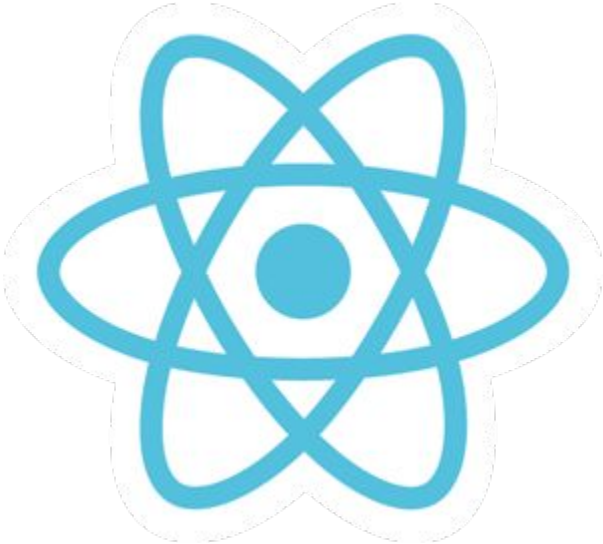
Sumário

- Resumo React
- Fluxo Unidirecional
  - Componentes de classe
  - Componentes funcionais
- Context API
- Redux
- UseReducer()

React

Resumo

- Biblioteca JS feita pelo Facebook
- Feito para *single-page applications*
- Rápido, escalável e simples
- Reutilizável

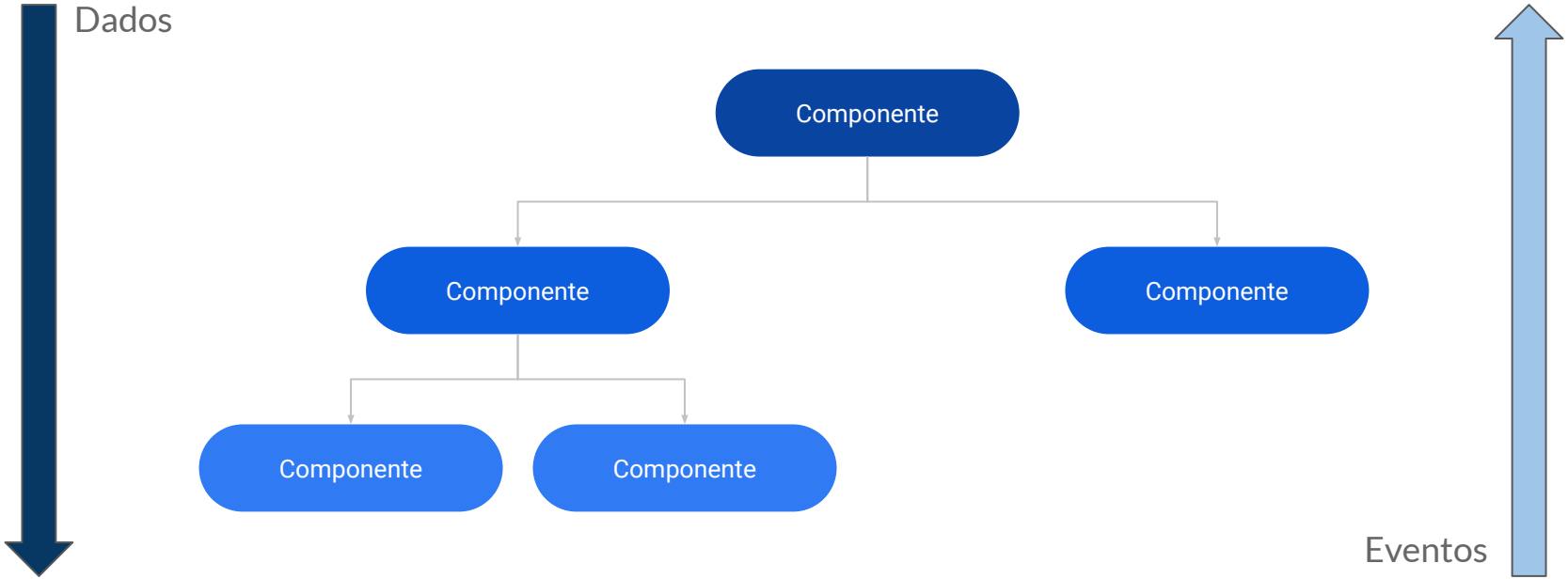


Fluxo de Dados Unidirecional Top-Bottom

Fluxo de Dados Nativo

- Padrão do React
- Uma e apenas uma maneira de passar dados para outras partes da aplicação
- Cada componente da aplicação tem seu estado
- O estado de cada componente é mudado por eventos
- Quando o estado é mudado, o componente é atualizado

Diagrama do Fluxo de Dados



Vantagens

- Menor chance de erros
- Mais fácil de debugar *
- Mais controle sobre os dados

Usabilidade

Declarar estado inicial:

```
constructor(props) {
  super(props);
  this.state = {
    state1: 'Default State'
  };
}
```

Uso do setState() para mudar estado:

```
handleChangeState = () => {
  this.setState({ state1: 'Changed State' });
};
```

Hooks

- Nova adição do React 16.8
- Permitem que você use o state e outros recursos do React sem escrever uma classe
- Função useState()

Usabilidade: State no Hooks

Declarar estado inicial (valor inicial passado nos parâmetros) e função de alteração de estado:

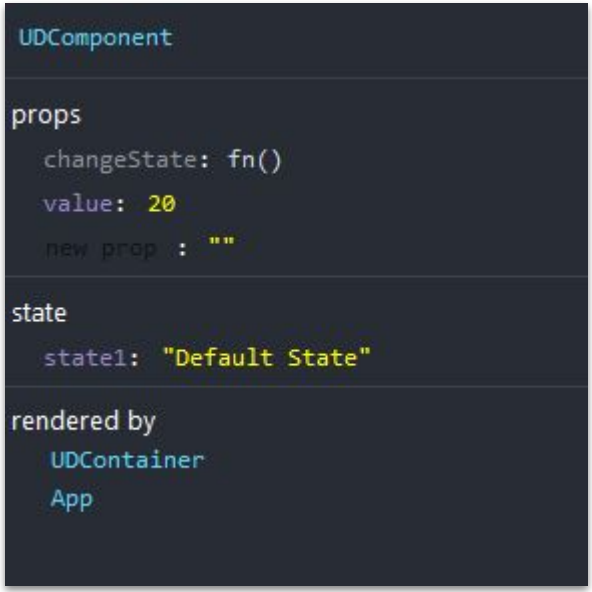
```
const [state1, setState1] = useState(false);
```

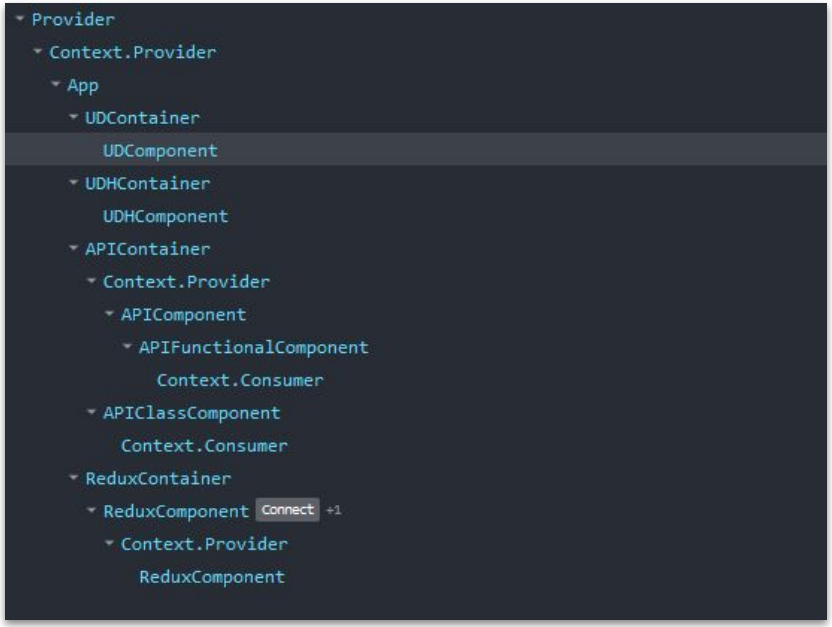
Alterando o estado:

```
setState1(!state1);
```

Ferramentas de Auxílio e *Debugging*

- Extensão: React Dev Tools





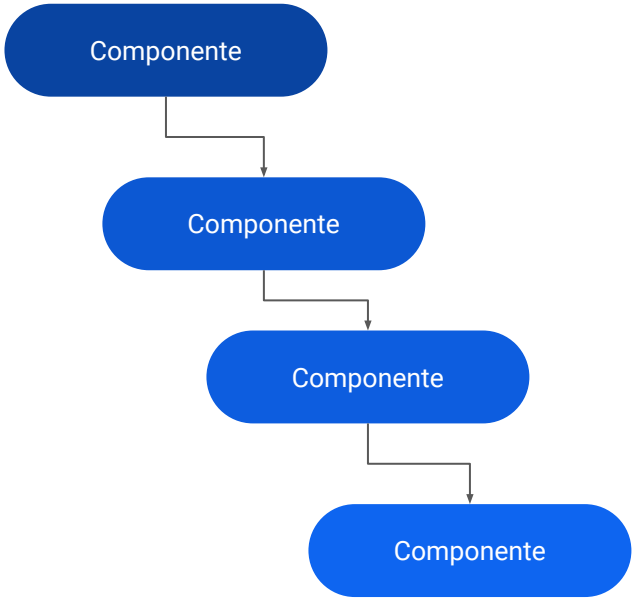
Context API

Context API

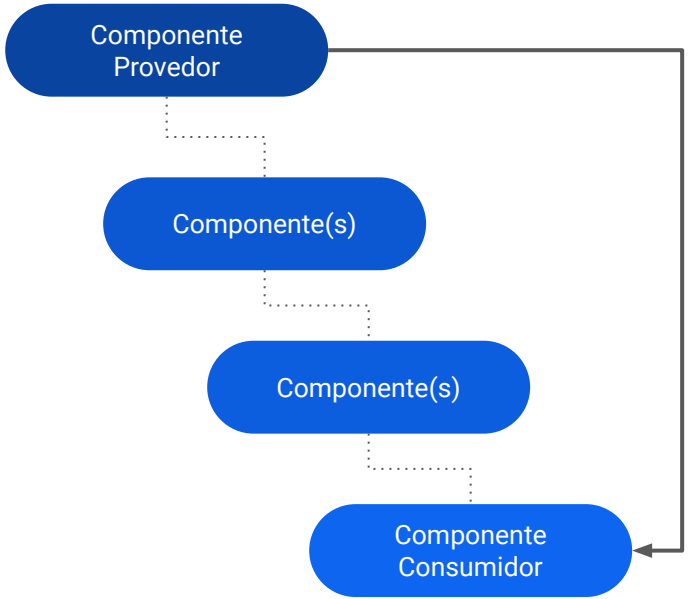
- Nova adição do React 16.3
- Maneira de passar dados sem *prop drilling*
- Recomendada para dados “globais”
  - Tema, autenticação, preferências de usuário
- Organizada em *contexts*, *providers* e *consumers*

Diagrama do Fluxo de Dados na Context API

Fluxo Normal



Context API



Vantagens

- Evita o *prop drilling*
- Simplifica acesso a dados que precisam ser acessados de várias partes da aplicação

Usabilidade

Criar contexto (com ou sem valores *default* por parâmetro):

```
const ContextExample = React.createContext({
  textValue: 'Default',
  numberValue: 0,
});
```

Criar *provider* e passar prop *value* contendo os dados que serão acessados pelos *consumers*:

```
<ContextExample.Provider
  value={{
    textExample,
    numberExample,
    changeText,
    increaseNumber
  }}
>
  <APIComponent />
</ContextExample.Provider>
```

Usabilidade

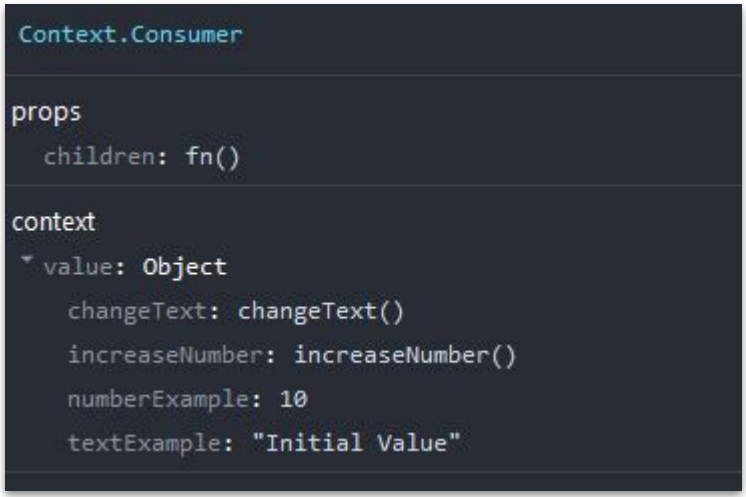
Acessar pelo *consumer* os valores passados no *provider*:

```
<ContextExample.Consumer>
  {value => (
    <>
      <div>Context Text Value: {value.textExample}</div>
      <Button onClick={() => value.changeText()} label="Change Context Text Value" />

      <div>Context Number Value: {value.numberExample}</div>
      <Button onClick={() => value.increaseNumber()} label="Increase Context Number Value"/>
    </>
  )}
</ContextExample.Consumer>
```

Ferramentas de Auxílio e *Debugging*

- Extensão: React Dev Tools



Redux

Redux

- Biblioteca JS para gerenciar o estado da sua aplicação
- Muito usada com React
- Estado global
- Baseado em *Stores*, *Actions* e *Reducers*

Diagrama de Funcionamento do Redux

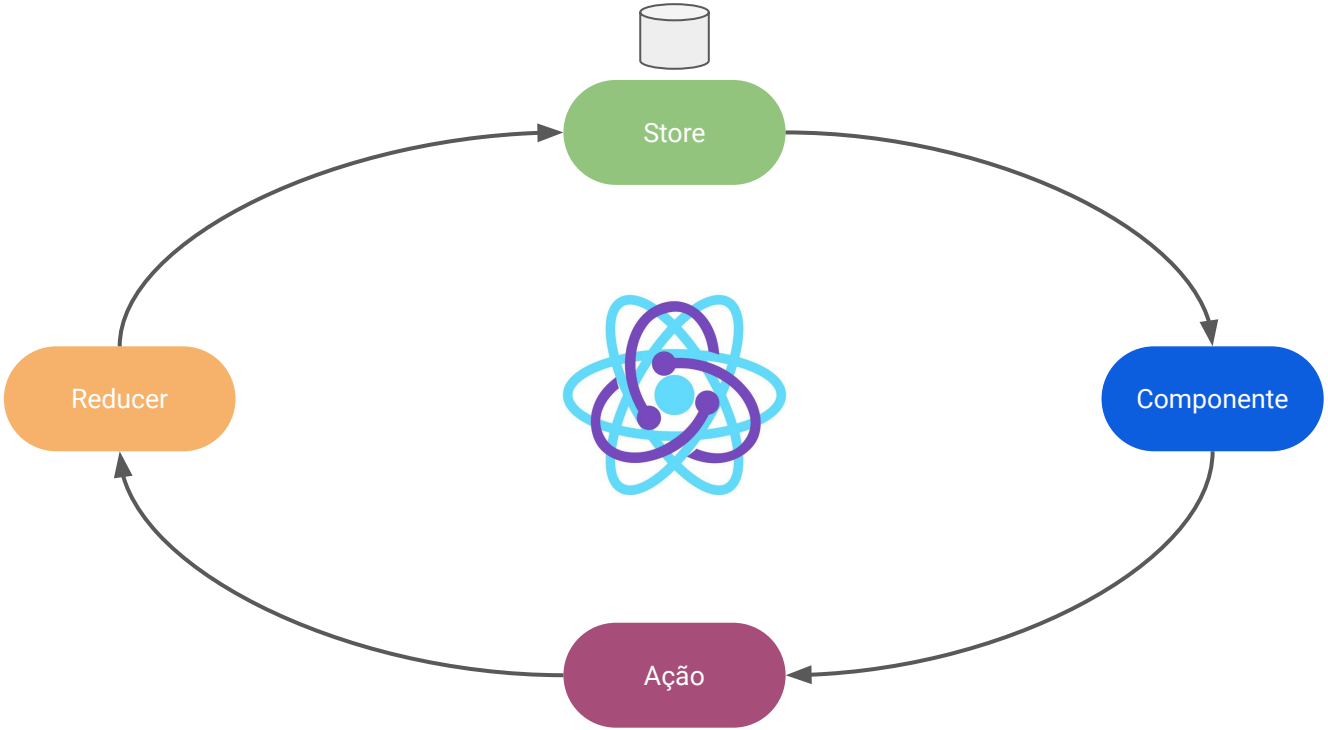
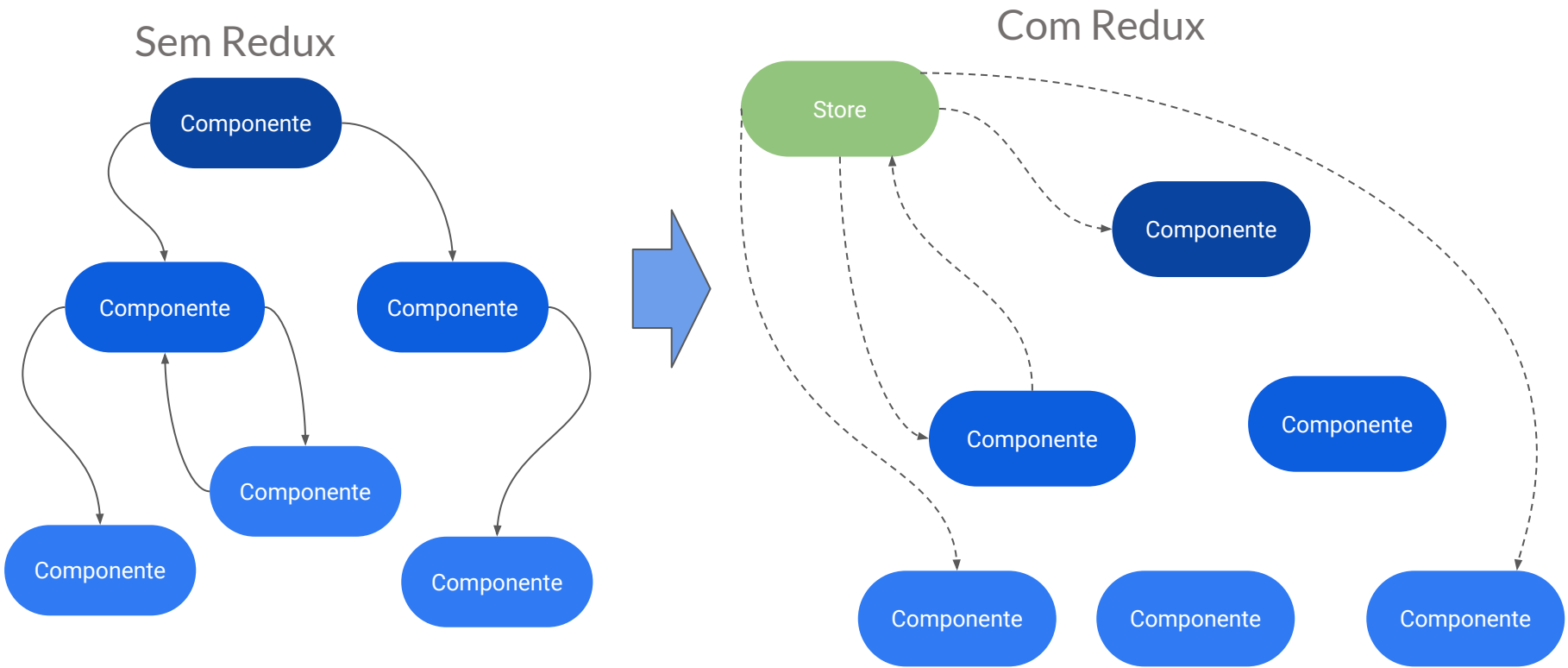


Diagrama de Fluxo de Dados no Redux



Vantagens

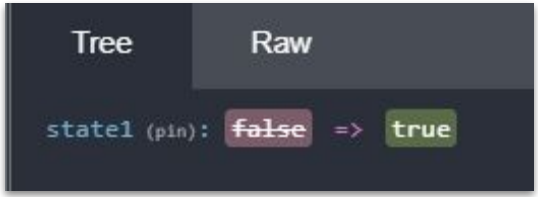
- Remover complexidade
- Aumentar performance
- Aumentar organização
- Aumentar legibilidade
- Centralização de lógica

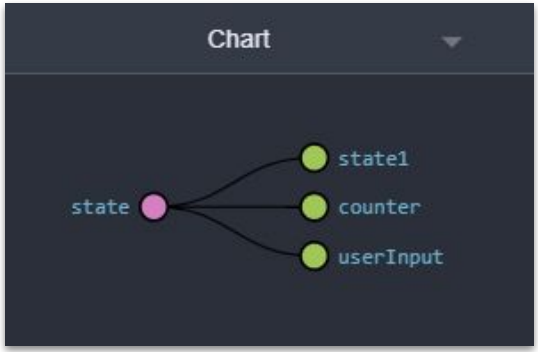
Usabilidade

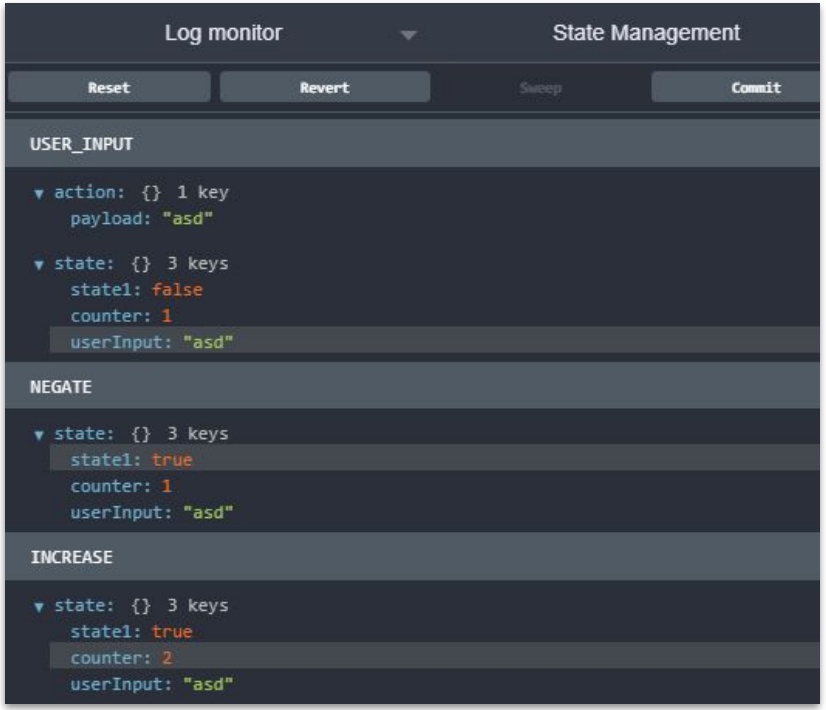
- Store:
  - Action Types
  - Actions
  - Reducers
- Mapear componentes:
  - mapStateToProps()
  - mapDispatchToProps()
- Conectar componentes e store:
  - connect()

Ferramentas de Auxílio e *Debugging*

- Redux DevTools







UseReducer

UseReducer

- Introduzido com o React Hooks
- Criado devido ao sucesso da integração entre React e Redux
- Alternativa para o useState(), para casos mais complexos

Vantagens e desvantagens (comparado ao Redux)

Vantagens:

- Integrado ao React
- Tipo primitivo
- Mais novo

Desvantagens:

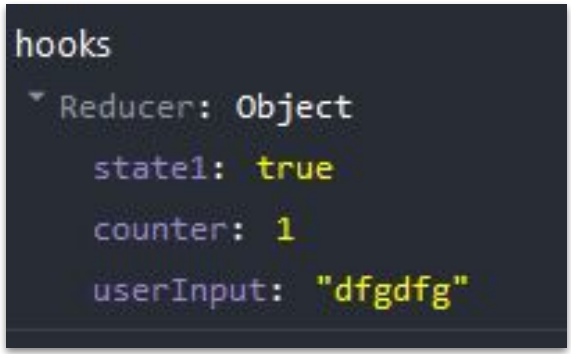
- Menos completo
- Menos ferramentas de auxílio
- Falta *middlewares*

Usabilidade

- Store:
  - Action Types
  - Actions
  - Reducers
- Utilizar a função useReducer, com o *reducer* alvo e o *state* inicial

Ferramentas de Auxílio

- React Dev Tools



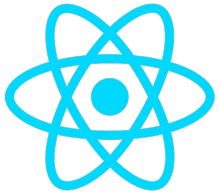
Boas Práticas no Uso de Reducers

- Separar *Reducers* e *Actions*
- Usar *Action Creators*
- Enumerar *Action Types*
- Evitar usar estado local e *hooks* de ciclo de vida *
- Fazer o *reducer* de forma determinística
- Action Types: UPPER_CASE
- Actions: camelCase

Qual técnica escolher?

- Estado do react (useState): casos em que o estado não importa pra outros componentes
- Redux ou useReducer: casos em que o estado deve ser global e muitos o acessam e o alteram de maneira complexa
- ContextAPI: casos em que o estado deve ser acessado por muitos, mas não alterado com frequência
- Faça o que parece menos esquisito!

Obrigado!



Gerenciamento de Estado no React

Pedro Pimenta
Dez 2019