Redux Toolkit

Redux 工具集

拉勾教育
— 互联网人实战大学 —

概述

对 Redux 进行的二次封装，用于高效 Redux 开发，使 Redux 的使用变得简单。

```
npm install @reduxjs/toolkit@1.6.0 react-redux@7.2.4
```

文档

快速入门

1. 创建状态切片

对于状态切片，我们可以认为它就是原本 Redux 中的那一个个的小的 Reducer 函数。
在 Redux 中，原本 Reducer 函数和 Action 对象需要分别创建，现在通过状态切片替代，它会返回 Reducer 函数和 Action 对象.

快速入门

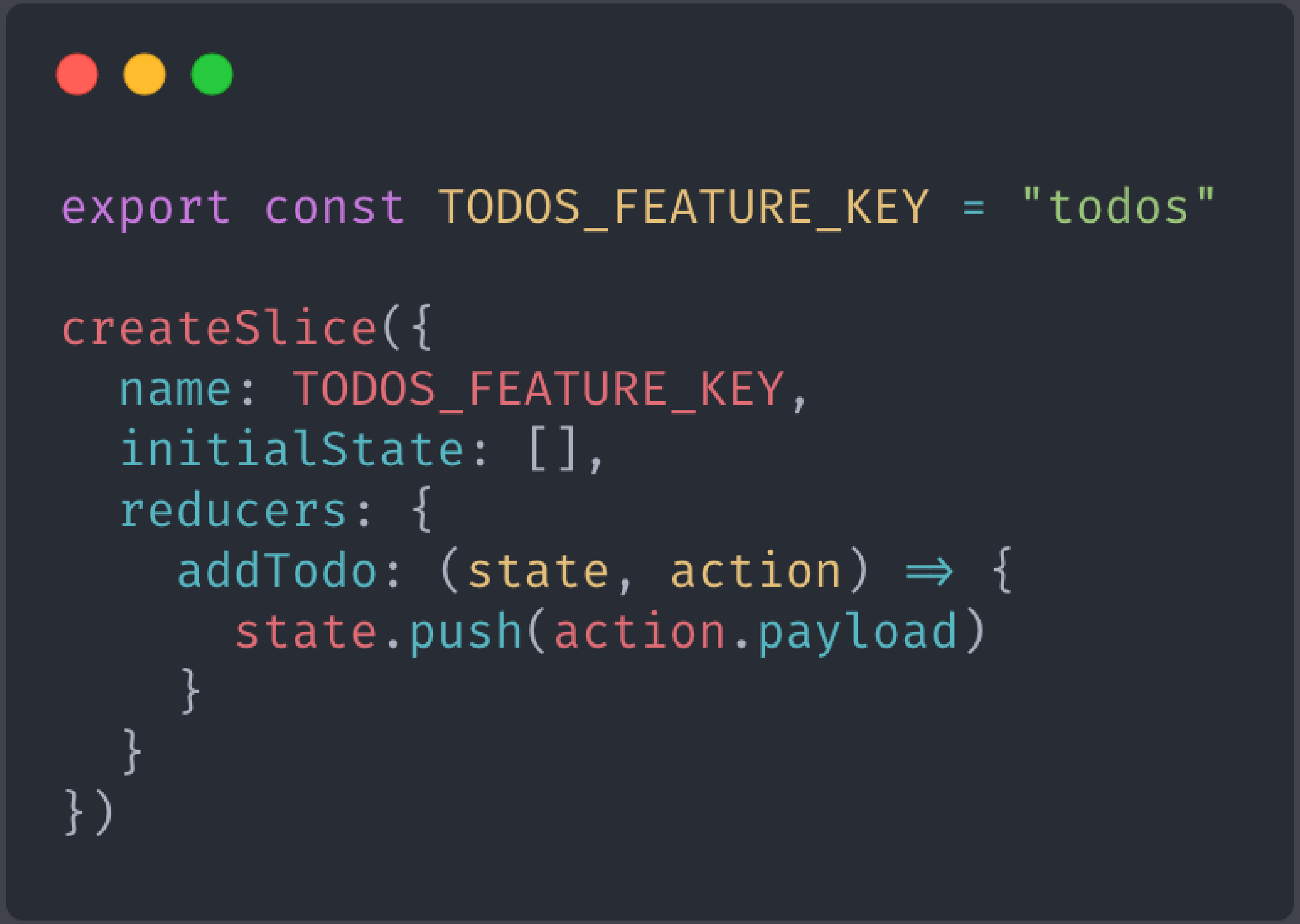
1. 创建状态切片

```
import { createSlice } from "@reduxjs/toolkit"

const { reducer: TodosReducer, actions } = createSlice()
```

快速入门

1. 创建状态切片



```
export const TODOS_FEATURE_KEY = "todos"

createSlice({
  name: TODOS_FEATURE_KEY,
  initialState: [],
  reducers: {
    addTodo: (state, action) => {
      state.push(action.payload)
    }
  }
})
```

快速入门

1. 创建状态切片

```
const { reducer: TodosReducer, actions } = createSlice()

export const { addTodo } = actions
export default TodosReducer
```

快速入门

2. 创建 Store

```
import { configureStore } from "@reduxjs/toolkit"
import TodosReducer, { TODOS_FEATURE_KEY } from "./todos"

export default configureStore({
  reducer: {
    [TODOS_FEATURE_KEY]: TodosReducer
  },
  devTools: process.env.NODE_ENV !== "production"
})
```

快速入门

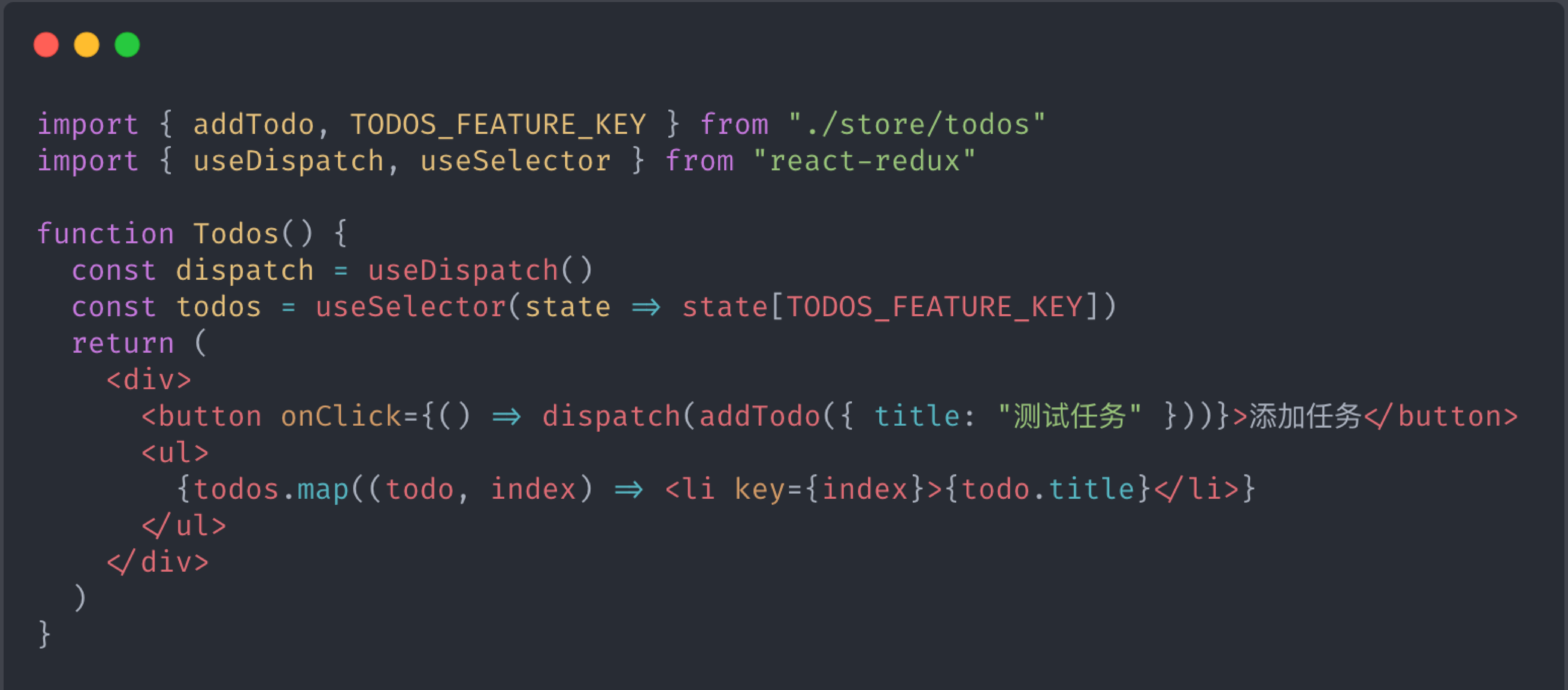
3. 配置 Provider

```
import ReactDOM from "react-dom"
import App from "./App"
import { Provider } from "react-redux"
import store from "./store"

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById("root")
)
```

快速入门

4. 在组件中触发 Action，获取状态



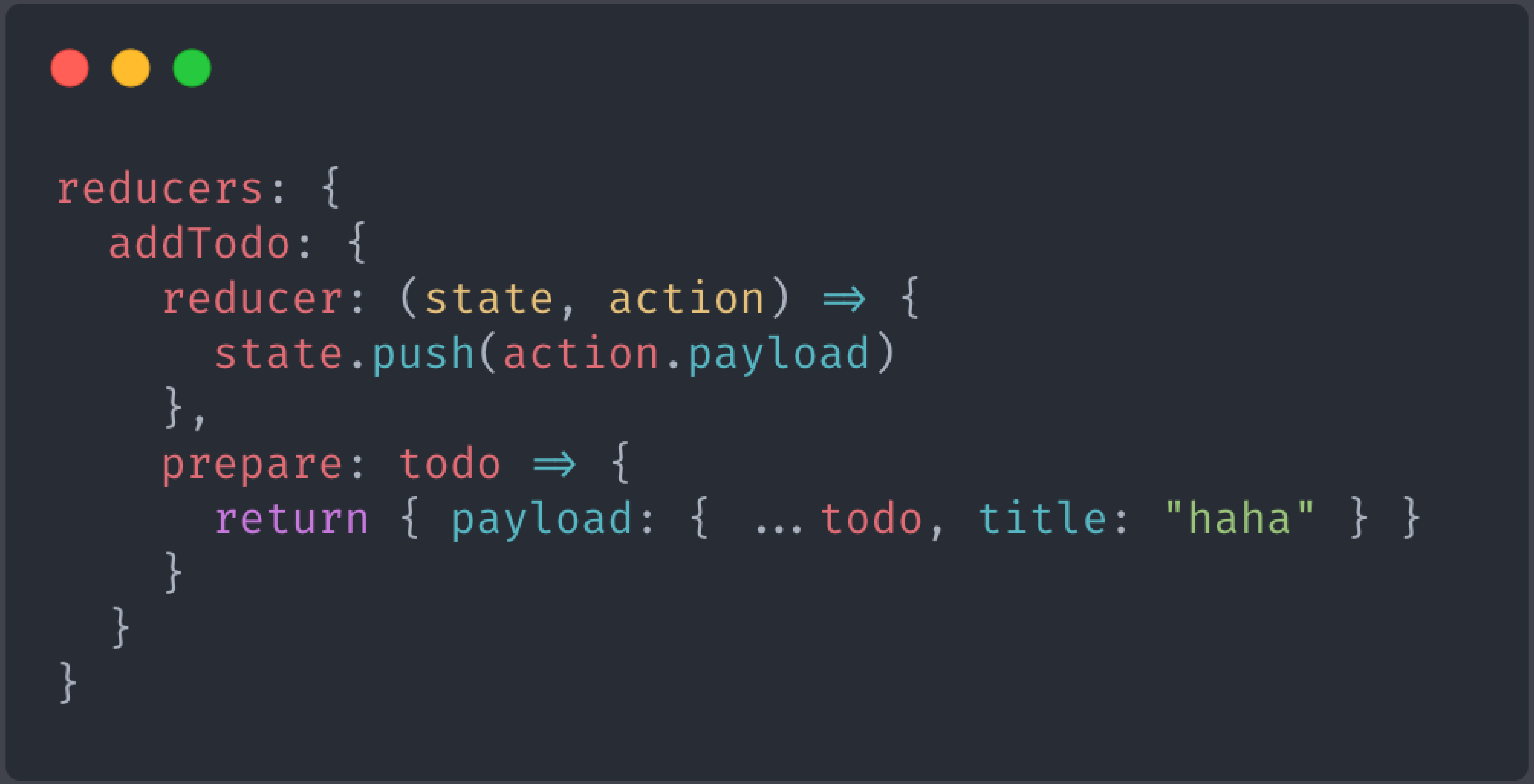
```jsx
import { addTodo, TODOS_FEATURE_KEY } from "./store/todos"
import { useDispatch, useSelector } from "react-redux"

function Todos() {
  const dispatch = useDispatch()
  const todos = useSelector(state => state[TODOS_FEATURE_KEY])
  return (
    <div>
      <button onClick={() => dispatch(addTodo({ title: "测试任务" }))}>添加任务</button>
      <ul>
        {todos.map((todo, index) => <li key={index}>{todo.title}</li>)}
      </ul>
    </div>
  )
}
```

快速入门

5. Action 预处理

当 Action 被触发后, 可以通过 prepare 方法对 Action 进行预处理, 处理完成后交给 Reducer. prepare 方法必须返回对象.



```
reducers: {
  addTodo: {
    reducer: (state, action) => {
      state.push(action.payload)
    },
    prepare: todo => {
      return { payload: { ...todo, title: "haha" } }
    }
  }
}
```

执行异步操作 (方式一)

1. 创建执行异步操作的 Action 创建函数

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit"
import axios from "axios"

export const loadTodos = createAsyncThunk(
  "todos/loadTodos",
  (payload, thunkAPI) => {
    axios.get(payload) .then(response => thunkAPI.dispatch(setTodos(response.data)))
  }
)
```

执行异步操作 (方式一)

2. 创建接收异步操作结果的 Reducer

```
const { reducer: TodosReducer, actions } = createSlice({
  reducers: {
    setTodos: (state, action) => {
      action.payload.forEach(todo => state.push(todo))
    }
  }
})
```

执行异步操作 (方式一)

3. 在组件中触发 Action

```
function Todos() {
  const dispatch = useDispatch()
  useEffect(() => {
    dispatch(loadTodos("https://jsonplaceholder.typicode.com/todos"))
  }, [])
}
```

执行异步操作 (方式二)

1. 创建执行异步操作的 Action 创建函数

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit"
import axios from "axios"

export const loadTodos = createAsyncThunk("todos/loadTodos", payload => {
  return axios.get(payload).then(response => response.data)
})
```

执行异步操作 (方式二)

2. 创建接收异步操作结果的 Reducer

```
createSlice({
  extraReducers: {
    [loadTodos.fulfilled]: (state, action) => {
      action.payload.forEach(todo => state.push(todo))
    }
  }
})
```

配置中间件

npm i redux-logger

```
import { configureStore, getDefaultMiddleware } from "@reduxjs/toolkit"
import logger from "redux-logger"

export default configureStore({
  middleware: [ ... getDefaultMiddleware(), logger]
})
```

实体适配器

将状态放入实体适配器，实体适配器提供操作状态的各种方法，简化操作。

```
import { createEntityAdapter } from "@reduxjs/toolkit"

const todosAdapter = createEntityAdapter()

todosAdapter.getInitialState()
todosAdapter.addOne(state, action.payload)
todosAdapter.addMany(state, action.payload)
```

实体适配器

在组件中获取状态，展示状态。

```
const todos = useSelector(state => state.todos.entities)

<ul>
  {Object.values(todos).map((todo, index) => (
    <li key={index}>{todo.title}</li>
  ))}
</ul>
```

实体适配器

代码简化，实体操作方法的第一个参数为 state，第二个参数为 action，内部会自动将对数据进行操作。比如对于 addOne 方法，它会自动将 action.payload 添加到 state 中。

```
reducers: {
  addTodo: todosAdapter.addOne
},
extraReducers: {
  [loadTodos.fulfilled]: todosAdapter.addMany
}
```

实体适配器

实体适配器要求每一个实体必须拥有 id 属性作为唯一标识，如果实体中的唯一标识字段不叫做 id，需要使用 selectId 进行声明。

```
const todosAdapter = createEntityAdapter({ selectId: todo ⇒ todo.cid })
```

状态选择器

提供从实体适配器中获取状态的快捷途径。

```
import { createSelector } from "@reduxjs/toolkit"

const { selectAll } = todosAdapter.getSelectors()

export const selectTodosList = createSelector(
  state => state[TODOS_FEATURE_KEY],
  selectAll
)
```

拉勾教育
— 互联网人实战大学 —

状态选择器

在组件中使用状态选择器

```
import { selectTodosList } from "./store/todos"

function Todos() {
 const todos = useSelector(selectTodosList)
}
```

扫码联系老师

技能评估、福利资料、课程优惠

Made with ♥ by LagouFed