Welcome to Slidev

Press Space for next page →

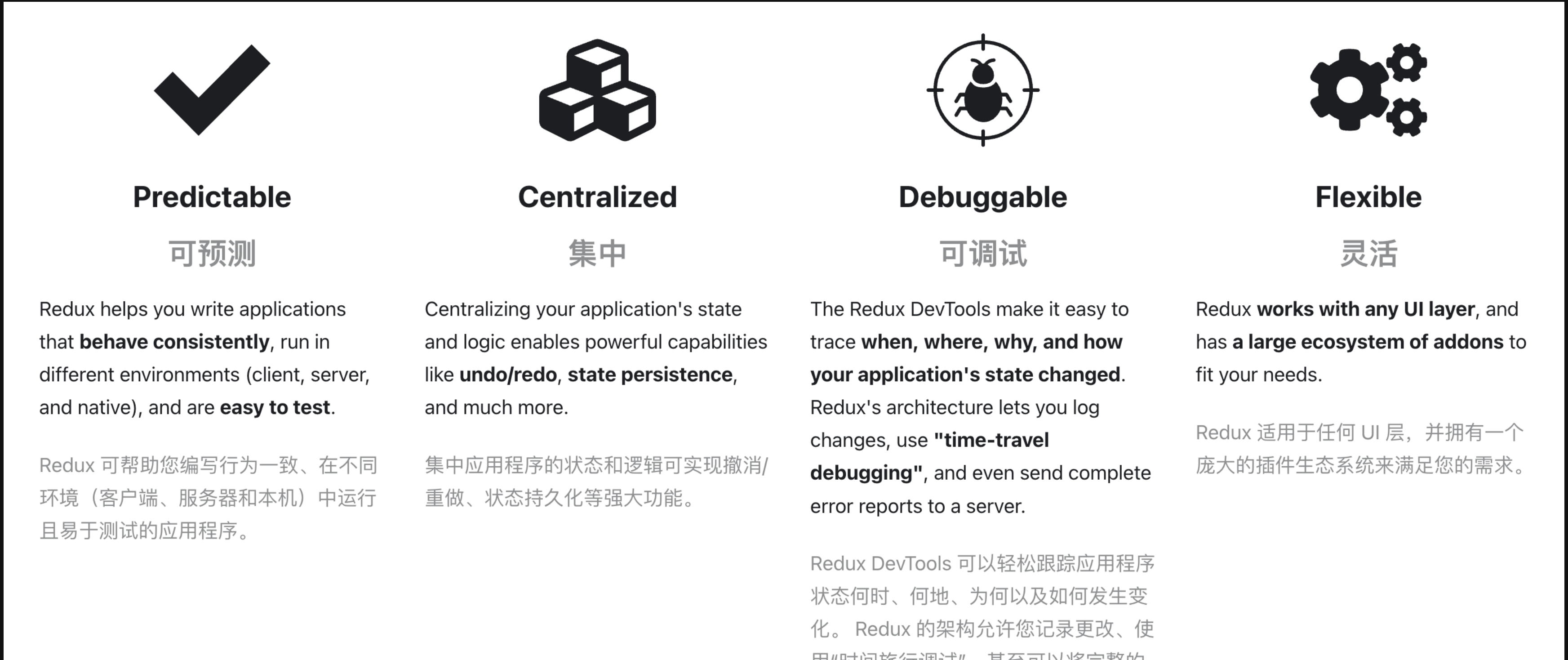
大纲

- 回顾 Redux
  - Redux API
- Redux Toolkit 和 DvaJs 分别是什么
  - Redux Toolkit
  - DvaJs
- Redux Toolkit 和 DvaJs 的对比
  - Action Creator
  - Reducer
  - Effect
  - Selector
  - Model
- 没有用好的能力
- 二者怎么选?

回顾 Redux

Redux 是什么

A Predictable State Container for JS Apps



Predictable

可预测

Redux helps you write applications that **behave consistently**, run in different environments (client, server, and native), and are **easy to test**.

Redux 可帮助您编写行为一致、在不同环境（客户端、服务器和本机）中运行且易于测试的应用程序。

Centralized

集中

Centralizing your application's state and logic enables powerful capabilities like **undo/redo**, **state persistence**, and much more.

集中应用程序的状态和逻辑可实现撤消/重做、状态持久化等强大功能。

Debuggable

可调试

The Redux DevTools make it easy to trace **when, where, why, and how your application's state changed**. Redux's architecture lets you log changes, use **"time-travel debugging"**, and even send complete error reports to a server.

Redux DevTools 可以轻松跟踪应用程序状态何时、何地、为何以及如何发生变化。 Redux 的架构允许您记录更改、使用"时间旅行调试"，甚至可以将完整的

Flexible

灵活

Redux **works with any UI layer**, and has **a large ecosystem of addons** to fit your needs.

Redux 适用于任何 UI 层，并拥有一个庞大的插件生态系统来满足您的需求。

Redux 核心概念

Actions

```javascript
const addTodoAction = {
  type: "todos/todoAdded",
  payload: "Buy milk",
};

const addTodo = (todo) => {
  return {
    type: "todos/todoAdded",
    payload: todo,
  };
};
```

Reducers

```
const initialState = { value: 0 };

function counterReducer(state = initialState, action) {
  // Check to see if the reducer cares about this action
  if (action.type === "counter/incremented") {
    // If so, make a copy of `state`
    return {
      ...state,
      // and update the copy with the new value
      value: state.value + 1,
    };
  }
  // otherwise return the existing state unchanged
  return state;
}
```

Store

```
import { applyMiddleware, createStore } from "redux";
import { composeWithDevTools } from "redux-devtools-extension";
import thunkMiddleware from "redux-thunk";

import monitorReducersEnhancer from "./enhancers/monitorReducers";
import loggerMiddleware from "./middleware/logger";
import rootReducer from "./reducers";

export default function configureStore(preloadedState) {
  const middlewares = [loggerMiddleware, thunkMiddleware];
  const middlewareEnhancer = applyMiddleware(...middlewares);

  const enhancers = [middlewareEnhancer, monitorReducersEnhancer];
  const composedEnhancers = composeWithDevTools(...enhancers);

  const store = createStore(rootReducer, preloadedState, composedEnhancers);

  if (process.env.NODE_ENV_ !== "production" && module.hot) {
    module.hot.accept("./reducers", () => store.replaceReducer(rootReducer));
  }
  return store;
}
```

Dispatch

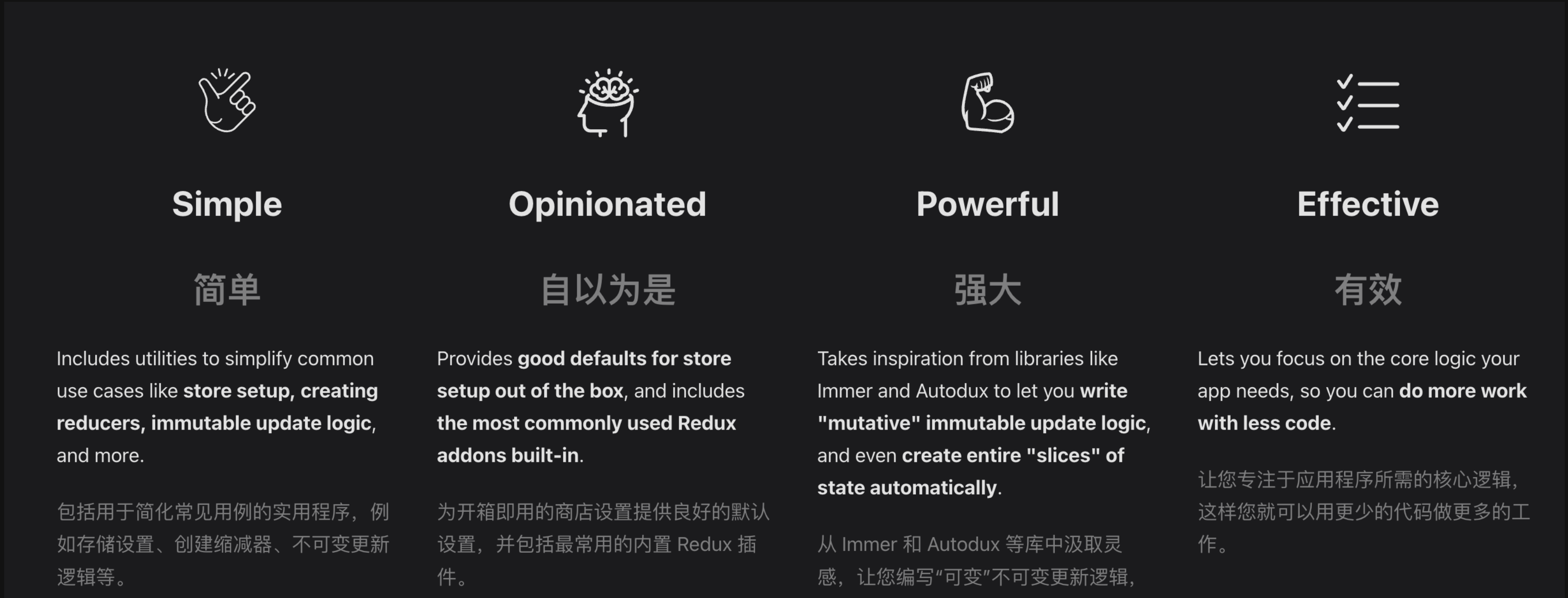
```
import React, { useState } from "react";
import { useDispatch } from "react-redux";

const Header = () => {
  const [text, setText] = useState("");
  const dispatch = useDispatch();

  const handleChange = (e) => setText(e.target.value);

  const handleKeyDown = (e) => {
    const trimmedText = e.target.value.trim();
    // If the user pressed the Enter key:
    if (e.key === "Enter" && trimmedText) {
      // Dispatch the "todo added" action with this text
      dispatch({ type: "todos/todoAdded", payload: trimmedText });
      // And clear out the text input
      setText("");
    }
  };
  return (
    <input
      type="text"
      placeholder="What needs to be done?"
      autoFocus={true}
      value={text}
      onChange={handleChange}
```

Selectors

```
import React from 'react'
import { useSelector } from 'react-redux'

import { availableColors, capitalize } from '../filters/colors'
import { StatusFilters } from '../filters/filtersSlice'

// Omit other footer components

const Footer = () => {
  const todosRemaining = useSelector(state => {
    const uncompletedTodos = state.todos.filter(todo => !todo.completed)
    return uncompletedTodos.length
  })

  const { status, colors } = useSelector(state => state.filters)
```

ToolKit

- "Configuring a Redux store is too complicated"
- "I have to add a lot of packages to get Redux to do anything useful"
- "Redux requires too much boilerplate code"



Simple

简单

Includes utilities to simplify common use cases like **store setup, creating reducers, immutable update logic**, and more.

包括用于简化常见用例的实用程序，例如存储设置、创建缩减器、不可变更新逻辑等。

Opinionated

自以为是

Provides **good defaults for store setup out of the box**, and includes **the most commonly used Redux addons built-in**.

为开箱即用的商店设置提供良好的默认设置，并包括最常用的内置 Redux 插件。

Powerful

强大

Takes inspiration from libraries like Immer and Autodux to let you **write "mutative" immutable update logic**, and even **create entire "slices" of state automatically**.

从 Immer 和 Autodux 等库中汲取灵感，让您编写“可变”不可变更新逻辑，

Effective

有效

Lets you focus on the core logic your app needs, so you can **do more work with less code**.

让您专注于应用程序所需的核心逻辑，这样您就可以用更少的代码做更多的工作。