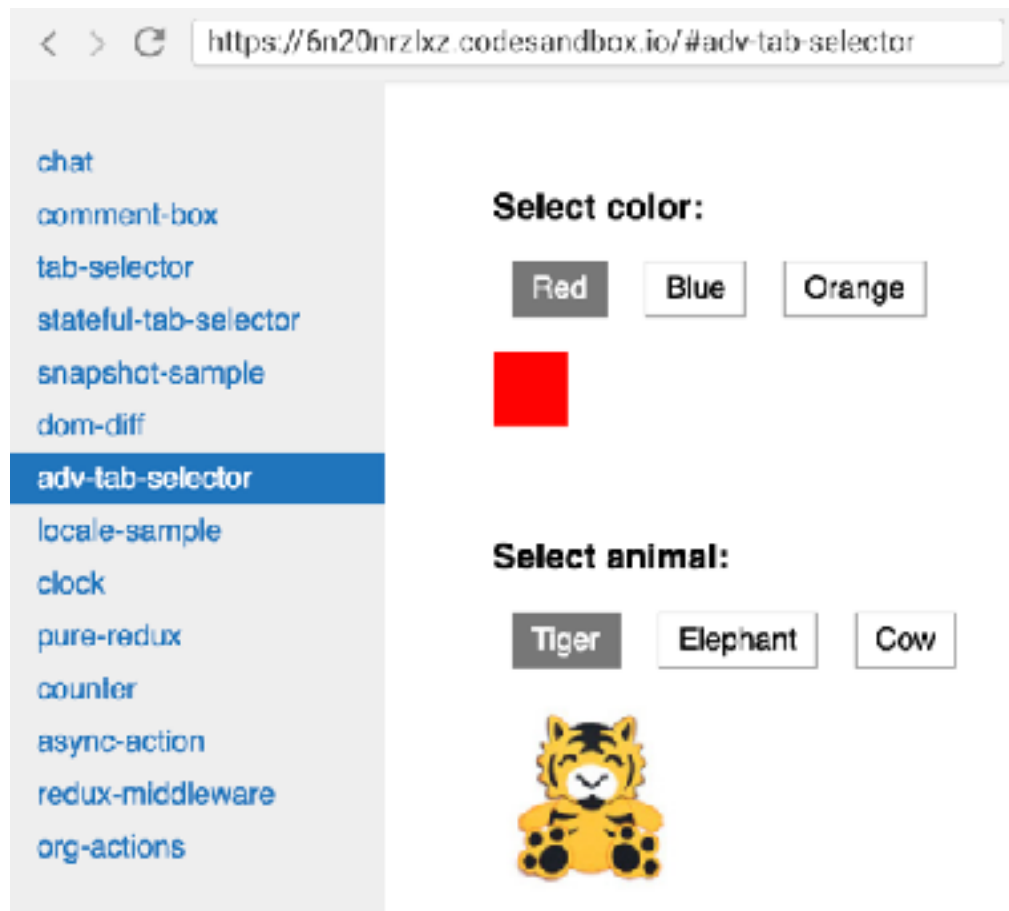


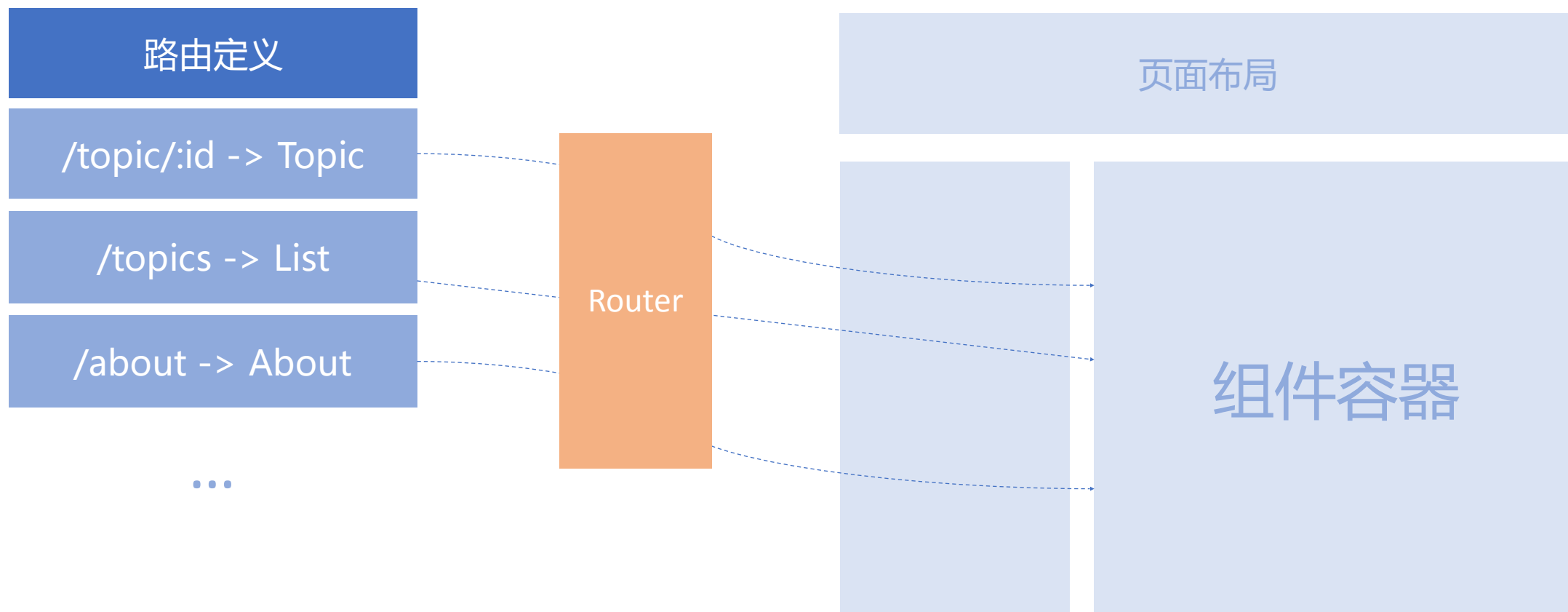
# React Router (1): 路由不只是页面切换，更是代码组织方式

# 为什么需要路由

1. 单页应用需要进行页面切换
2. 通过 URL 可以定位到页面
3. 更有语义的组织资源



# 路由实现的基本架构



# React Router 的实现

```
<Router>
  <div>
    <ul id="menu">
      <li><Link to="/home">Home</Link></li>
      <li><Link to="/hello">Hello</Link></li>
      <li><Link to="/about">About</Link></li>
    </ul>

    <div id="page-container">
      <Route path="/home" component={Home} />
      <Route path="/hello" component={Hello} />
      <Route path="/about" component={About} />
    </div>
  </div>
</Router>
```

# React Router 的特性

1. 声明式路由定义

2. 动态路由

```
const App = () => (  
  <div>  
    <nav>  
      <Link to="/dashboard">Dashboard</Link>  
    </nav>  
    <div>  
      <Route path="/dashboard" component={Dashboard}/>  
    </div>  
  </div>  
)
```

# 三种路由实现方式

1. URL 路径

2. hash 路由

3. 内存路由

# 基于路由配置进行资源组织

1. 实现业务逻辑的松耦合
2. 易于扩展，重构和维护
3. 路由层面实现 Lazy Load

# React Router API

1. `<Link>` : 普通链接，不会触发浏览器刷新
2. `<NavLink>` : 类似 Link 但是会添加当前选中状态
3. `<Prompt>` : 满足条件时提示用户是否离开当前页面
4. `<Redirect>` : 重定向当前页面，例如登录判断
5. `<Route>` : 路由配置的核心标记，路径匹配时显示对应组件
6. `<Switch>` : 只显示第一个匹配的路由



# <Link>：普通链接，不会触发浏览器刷新

```
import { Link } from 'react-router-dom'  
  
<Link to="/about">About</Link>
```

<NavLink> : 类似 Link 但是会添加当前选中状态

```
<NavLink  
  to="/faq"  
  activeClassName="selected"  
>FAQs</NavLink>
```

<Prompt> : 满足条件时提示用户是否离开当前页面

```
import { Prompt } from 'react-router'

<Prompt
  when={formIsHalfFilledOut}
  message="Are you sure you want to leave?"
/>
```

## <Redirect>：重定向当前页面，例如登录判断

```
import { Route, Redirect } from 'react-router'

<Route exact path="/" render={() => (
  loggedIn ? (
    <Redirect to="/dashboard"/>
  ) : (
    <PublicHomePage/>
  )
)} />
```

# <Route>：路径匹配时显示对应组件

```
import { BrowserRouter as Router, Route } from 'react-router-dom'

<Router>
  <div>
    <Route exact path="/" component={Home}/>
    <Route path="/news" component={NewsFeed}/>
  </div>
</Router>
```

# <Switch> : 只显示第一个匹配的路由

```
import { Switch, Route } from 'react-router'

<Switch>
  <Route exact path="/" component={Home}/>
  <Route path="/about" component={About}/>
  <Route path="/:user" component={User}/>
  <Route component={NoMatch}/>
</Switch>
```

# 小结

1. 前端路由是什么
2. React Router 如何实现路由
3. 基于路由思考资源的组织
4. React Router 核心 API

## React Router (2): 参数定义，嵌套路由的使用场景



# 通过 URL 传递参数

1. 如何通过 URL 传递参数：`<Route path= "/topic/:id" ... />`
2. 如何获取参数：`this.props.match.params`
3. <https://github.com/pillarjs/path-to-regexp>

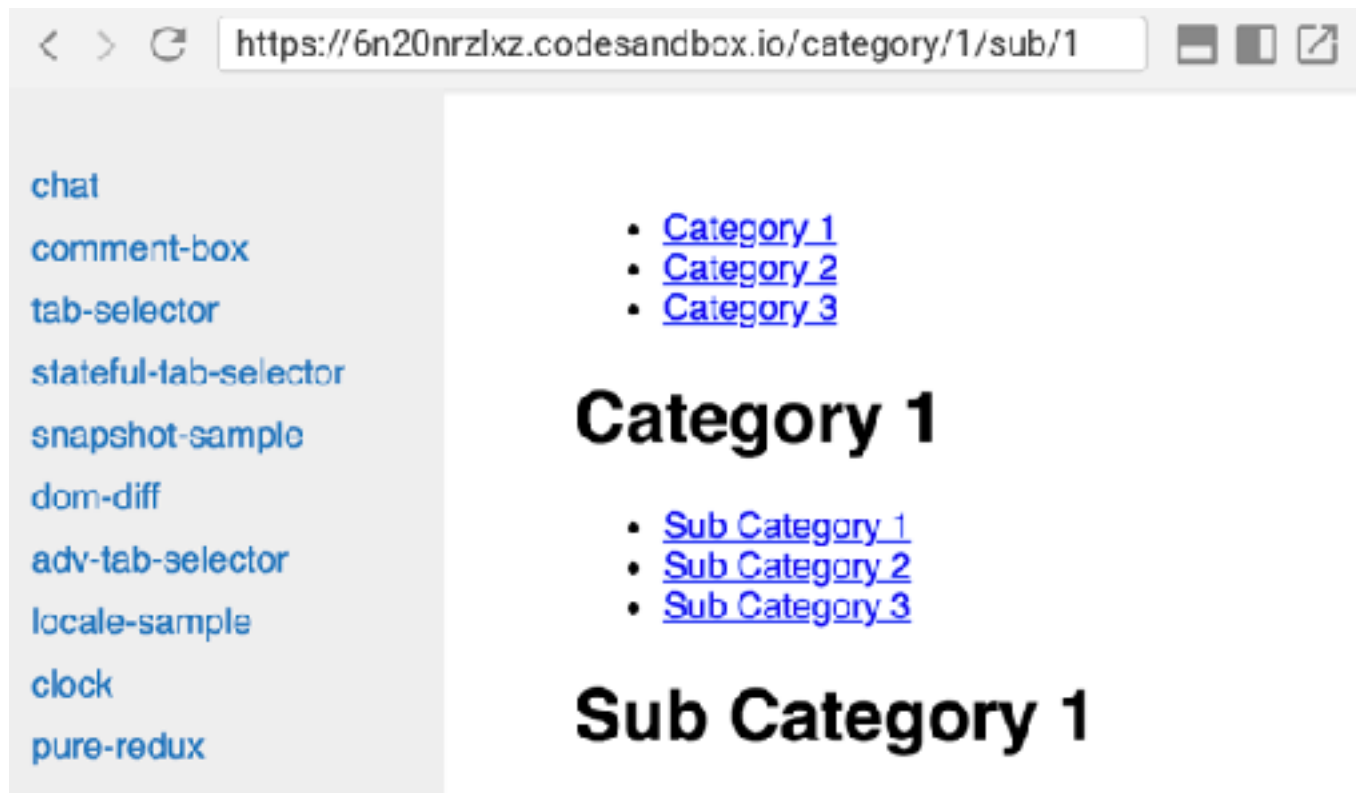
# 何时需要 URL 参数

## 页面状态尽量通过 URL 参数定义



# 嵌套路由

1. 每个 React 组件都可以是路由容器
2. React Router 的声明式语法可以方便的定义嵌套路由



# 小结

1. 路由参数的传递
2. 嵌套路由的实现

UI 组件库对比和介绍：Ant.Design, Material UI, Semantic UI



🔍 在 ant.design 中搜索

# Ant Design

一个服务于企业级产品的设计体系，基于『确定』和『自然』的设计价值观和模块化的解决方案，让设计者专注于更好的用户体验。

开始使用

## 设计语言



30231

## Ant Design

## 介绍

### 设计价值观

### 实践案例

原則

亲密性

对开

### 对比

電氣

直截了当

## 简化交互

足不出户

提供遮情

### 巧用过放

### 即时反应

足不出户

能在这个页面解决的问题，就不要去其它页面解决，因为任何页面刷新和跳转都会引起变化盲视（Change Blindness），导致用户心流（Flow）被打断。频繁的页面刷新和跳转，就像在看戏时，演员说完一行台词就安排一次谢幕一样。

**变换盲视 (Change Blindness)**：指视觉场景中的某些变化并未被观察者注意到的心理现象。产生这种现象的原因包括场景中的障碍物、限制运动、地点的变化，或者是缺乏注意力等。——摘自《维基百科》

心流 (Flow)：也有别号以化境 (Zone) 表示。亦有人翻译为并融状态。定义是一种将个人精神力完全投注在某种活动上的感觉；心流产生时同时会有高度的兴奋及充实感。——摘自《维基百科》

### 覆盖层



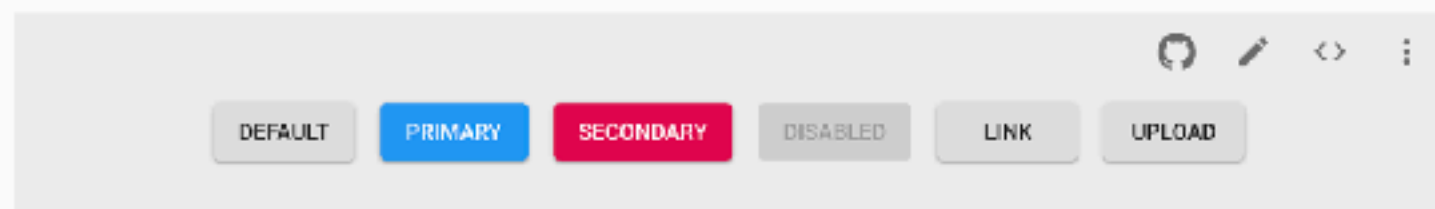


# MATERIAL-UI

React components that implement  
Google's Material Design.

## Contained Buttons

**Contained buttons** are high-emphasis, distinguished by their use of elevation and fill. They contain actions that are primary to your app.

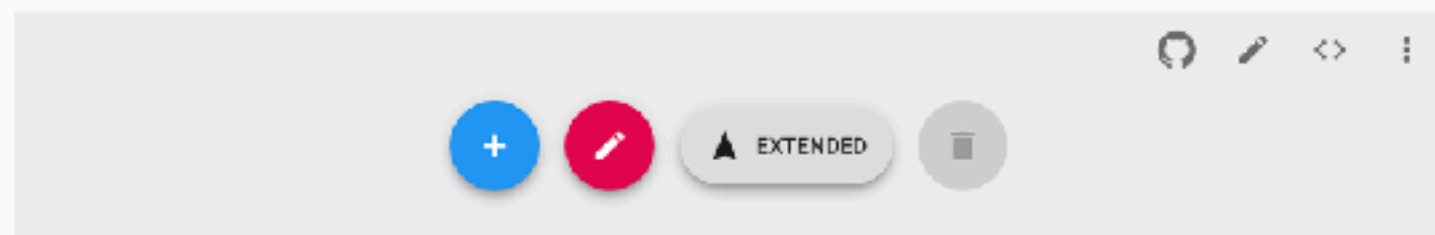


## Floating Action Buttons

A **floating action button** (FAB) performs the primary, or most common, action on a screen. It appears in front of all screen content, typically as a circular shape with an icon in its center. FABs come in three types: regular, mini, and extended.

Only use a FAB if it is the most suitable way to present a screen's primary action.

Only one floating action button is recommended per screen to represent the most common action.






Menu

 Tweet

 Star

41,864

 English

2.3.2

# Semantic UI

User Interface is the language of the web

Get Started

New in 2.3



# 选择 UI 库的考虑因素

1. 组件库是否齐全
2. 样式风格是否符合业务需求
3. API 设计是否便捷和灵活
4. 技术支持是否完善
5. 开发是否活跃

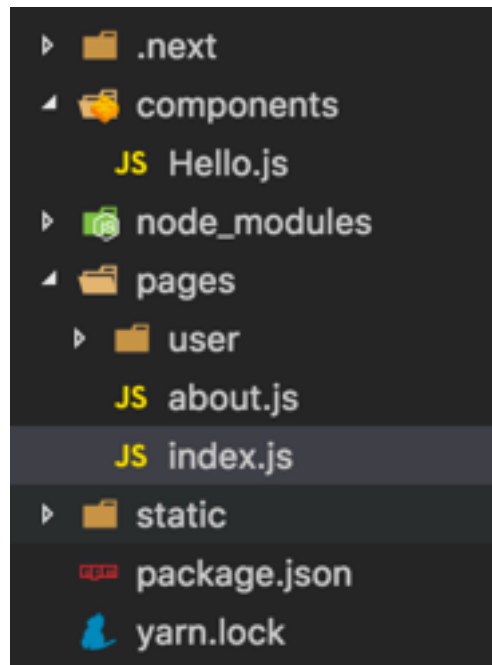
DEMO

# 使用 Next.js 创建 React 同构应用

# 什么是同构应用



# 创建页面



1. 页面就是 pages 目录下的一个组件
2. static 目录映射静态文件
3. page 具有特殊静态方法 `getInitialProps`

# 在页面中使用其它 React 组件

1. 页面也是标准的 node 模块，可使用其它 React 组件
2. 页面会针对性打包，仅包含其引入的组件

# 使用 Link 实现同构路由

```
import Link from 'next/link'

export default () =>
  <div>
    Click{' '}
    <Link href="/about">
      <a>here</a>
    </Link>{' '}
    to read more
  </div>
```

1. 使用 “next/link” 定义链接
2. 点击链接时页面不会刷新
3. 使用 prefetch 预加载目标资源
4. 使用 replace 属性替换 URL

# 动态加载页面

```
import dynamic from 'next/dynamic'

const DynamicComponentWithCustomLoading = dynamic(
  import('../components/hello2'),
  {
    loading: () => <p>...</p>
  }
)

export default () =>
  <div>
    <Header />
    <DynamicComponentWithCustomLoading />
    <p>HOME PAGE is here!</p>
  </div>
```



DEMO

# 小结

1. 同构应用的概念
2. Next.js 的基本用法

# 使用 Jest, Enzyme 等工具进行单元测试

# React 让前端单元测试变得容易

1. React 应用很少需要访问浏览器 API
2. 虚拟 DOM 可以在 NodeJS 环境运行和测试
3. Redux 隔离了状态管理，纯数据层单元测试

# 单元测试涉及的工具

1. Jest: Facebook 开源的 JS 单元测试框架
2. JS DOM: 浏览器环境的 NodeJS 模拟
3. Enzyme : React 组件渲染和测试
- 4.nock: 模拟 HTTP 请求
- 5.sinon: 函数模拟和调用跟踪
- 6.istanbul: 单元测试覆盖率

# Jest



## Delightful JavaScript Testing

```
1  const add = require('./add');
2  describe('add', () => {
3    it('should add two numbers', () => {
4      expect(add(1, 2)).toBe(3);
5    });
6  });
```

```
PASS ./add-test.js
  add
    ✓ should add two numbers (6ms)
```

```
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.903s
Ran all test suites.
>
```



# jsdom

```
const JSDOM = require('jsdom').JSDOM;

global.window = new JSDOM('<!DOCTYPE html><div id="react-root"></div>').window;
global.document = window.document;
global.navigator = window.navigator;
global.HTMLInputElement = window.HTMLInputElement;
```



# Enzyme

```
import React from 'react';
import { shallow } from 'enzyme';
import { DefaultPage } from 'src/features/home/DefaultPage';

describe('home/DefaultPage', () => {
  it('renders node with correct class name', () => {
    const pageProps = {
      home: {},
      actions: {},
    };
    const renderedComponent = shallow(
      <DefaultPage {...pageProps} />
    );

    expect(
      renderedComponent.find('.home-default-page').getElement()
    ).to.exist;
  });
});
```

## Shallow Rendering

```
import { shallow } from 'enzyme';

const wrapper = shallow(<MyComponent />);
// ...
```

## Full Rendering

```
import { mount } from 'enzyme';

const wrapper = mount(<MyComponent />);
// ...
```

## Static Rendering

```
import { render } from 'enzyme';

const wrapper = render(<MyComponent />);
// ...
```



# Nock

```
it('handles fetchRedditReactjsList failure', () => {
  nock('http://www.reddit.com/')
    .get('/r/reactjs.json')
    .reply(500, null);
  const store = mockStore({ redditReactjsList: [] });

  return store.dispatch(fetchRedditReactjsList())
    .catch(() => {
      const actions = store.getActions();
      expect(actions[0]).to.have.property('type', HOME_FETCH_REDDIT_REACTJS_LIST_BEGIN);
      expect(actions[1]).to.have.property('type', HOME_FETCH_REDDIT_REACTJS_LIST_FAILURE);
      expect(actions[1]).to.have.nested.property('data.error').that.exist;
    });
});
```

# Sinon

```
it('counter actions are called when buttons clicked', () => {
  const pageProps = {
    home: {},
    actions: {
      counterPlusOne: sinon.spy(),
      counterMinusOne: sinon.spy(),
      resetCounter: sinon.spy(),
      fetchRedditReactjsList: sinon.spy(),
    },
  };
  const renderedComponent = shallow(
    <DefaultPage {...pageProps} />
  );
  renderedComponent.find('.btn-plus-one').simulate('click');
  renderedComponent.find('.btn-minus-one').simulate('click');
  renderedComponent.find('.btn-reset-counter').simulate('click');
  renderedComponent.find('.btn-fetch-reddit').simulate('click');
  expect(pageProps.actions.counterPlusOne).to.have.property('callCount', 1);
  expect(pageProps.actions.counterMinusOne).to.have.property('callCount', 1);
  expect(pageProps.actions.resetCounter).to.have.property('callCount', 1);
  expect(pageProps.actions.fetchRedditReactjsList).to.have.property('callCount', 1);
});
```

# Istanbul

```
$ npm install -g istanbul
$ cd /path/to/your/source/root
$ istanbul cover test.js
```

```
function meaningOfLife() {
  return 42;
}

function meaningOfLife() {
  __cov_ECTNDGoq6USQiIaViK8Qyw.f['1']++;
  __cov_ECTNDGoq6USQiIaViK8Qyw.s['2']++;
  return 42;
}
```

18 passing (61ms)

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
all files	100	100	100	100	
src	100	100	100	100	
index.js	100	100	100	100	
src/reducer.js	100	100	100	100	
routeConfig.js	100	100	100	100	
src/components	100	100	100	100	
PageNotFound.js	100	100	100	100	
HomeNav.js	100	100	100	100	
index.js	100	100	100	100	
src/container	100	100	100	100	
App.js	100	100	100	100	
src/features/home	100	100	100	100	
DefaultPage.js	100	100	100	100	
Hello.js	100	100	100	100	
TestPage1.js	100	100	100	100	
TestPage2.js	100	100	100	100	
actions.js	100	100	100	100	
constants.js	100	100	100	100	
index.js	100	100	100	100	
reducer.js	100	100	100	100	
route.js	100	100	100	100	

PFUN5:85344A rekit:32:6563

DEMO

# 小结

1. React 让单元测试变得更容易
2. 单元测试所涉及的工具和技术

常用开发调试工具：ESLint, Prettier, React DevTool, Redux  
DevTool



1. 使用 `.eslintrc` 进行规则的配置
2. 使用 `airbnb` 的 JavaScript 代码风格

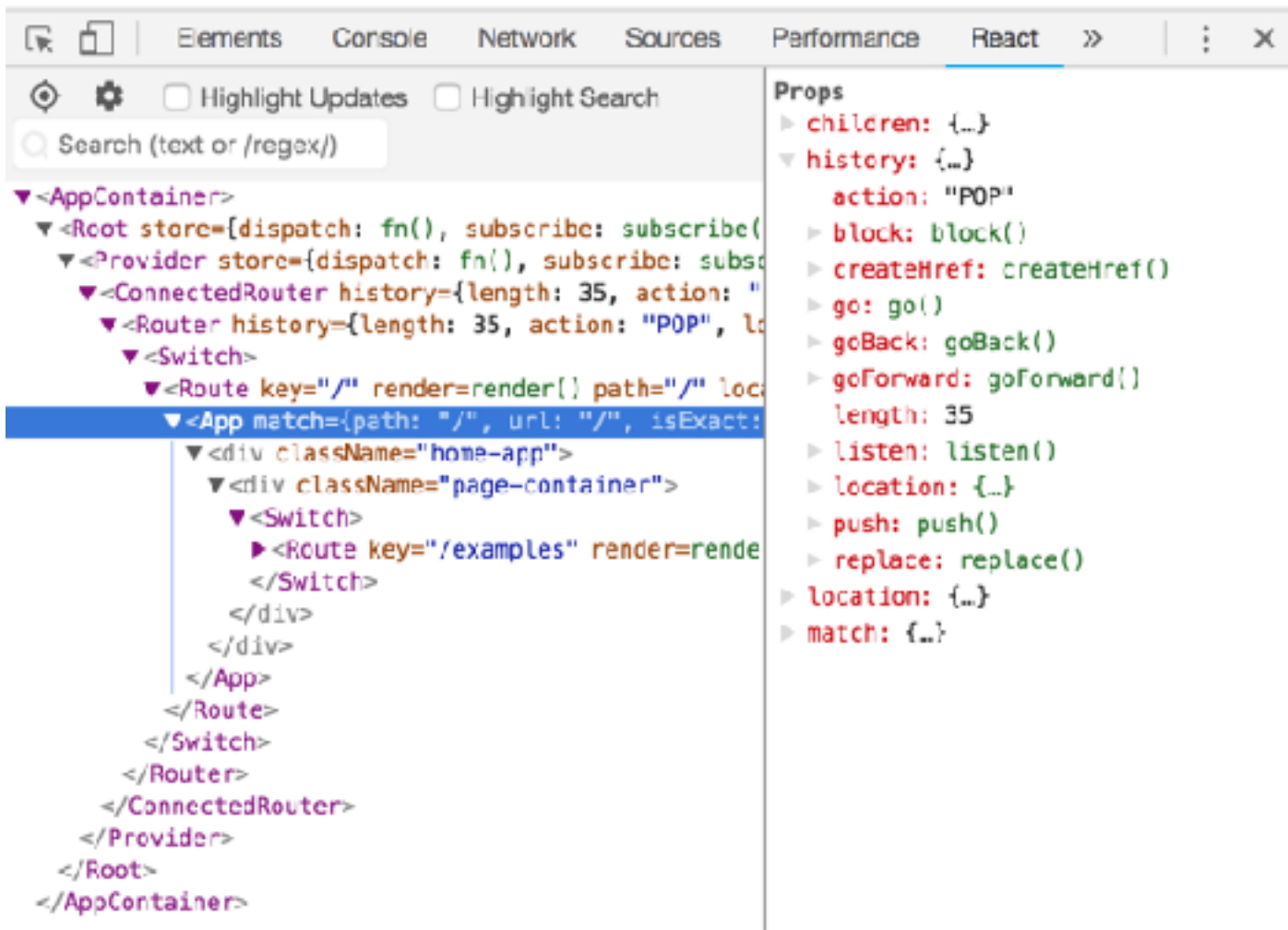


# Prettier

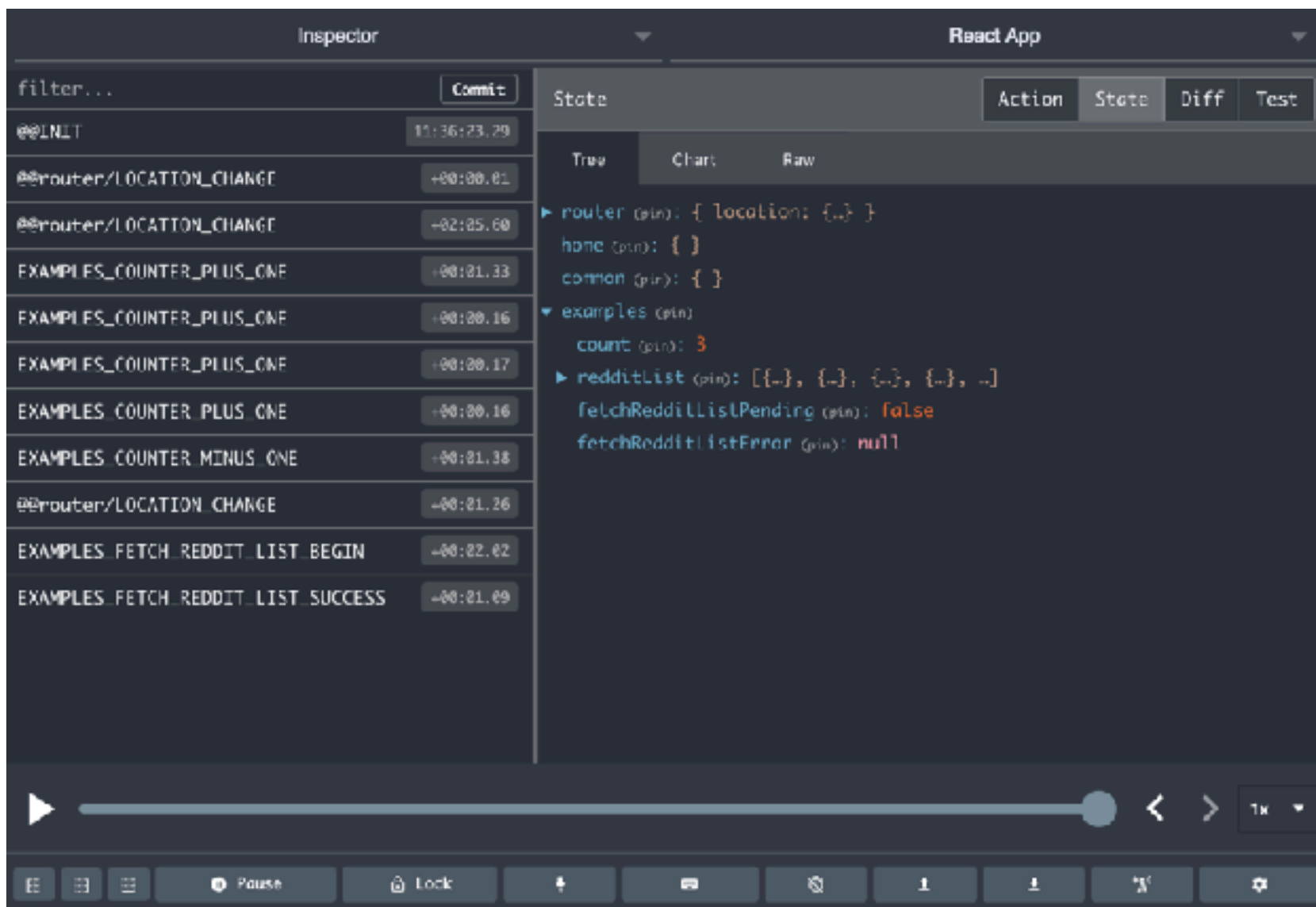
1. 代码格式化的神器
2. 保证更容易写出风格一致的代码



# React Dev Tools



# Redux Dev Tools

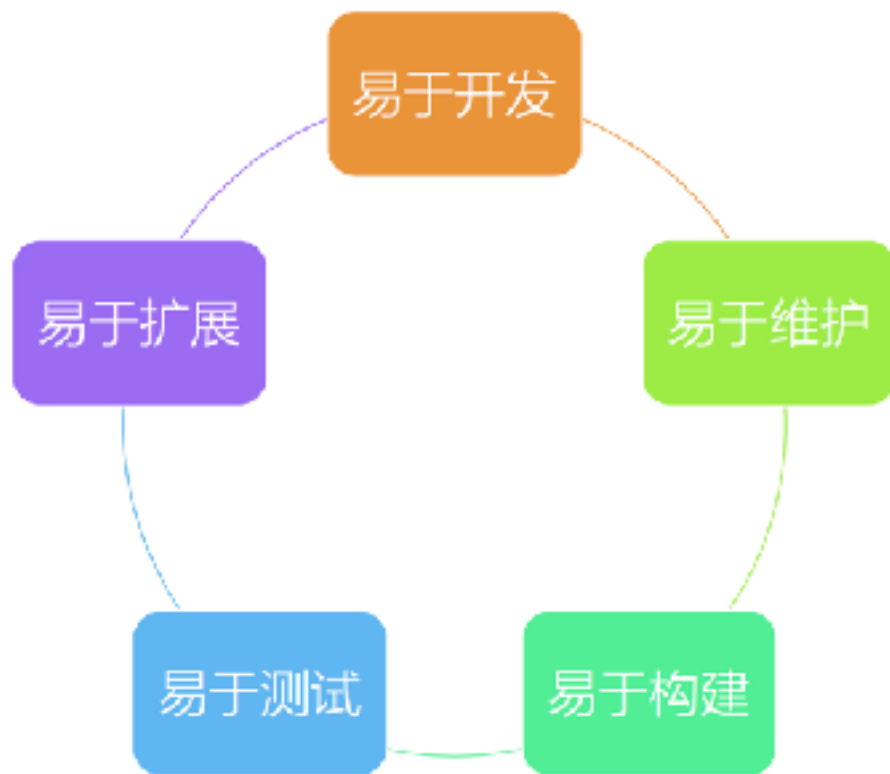


DEMO

前端项目的理想架构：

可维护，可扩展，可测试，易开发，易构建

# 理想架构



# 易于开发

1. 开发工具是否完善
2. 生态圈是否繁荣
3. 社区是否活跃

# 易于扩展

1. 增加新功能是否容易
2. 新功能是否会显著增加系统复杂度

# 易于维护

1. 代码是否容易理解
2. 文档是否健全



# 易于测试

1. 功能的分层是否清晰
2. 副作用少
3. 尽量使用纯函数

# 易于构建

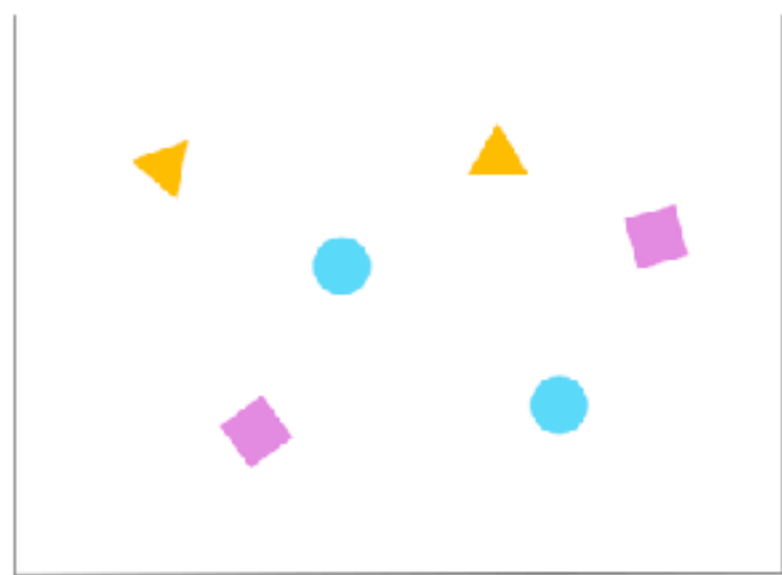
1. 使用通用技术和架构
2. 构建工具的选择

拆分复杂度（1）：

按领域模型（feature）组织代码，降低耦合度

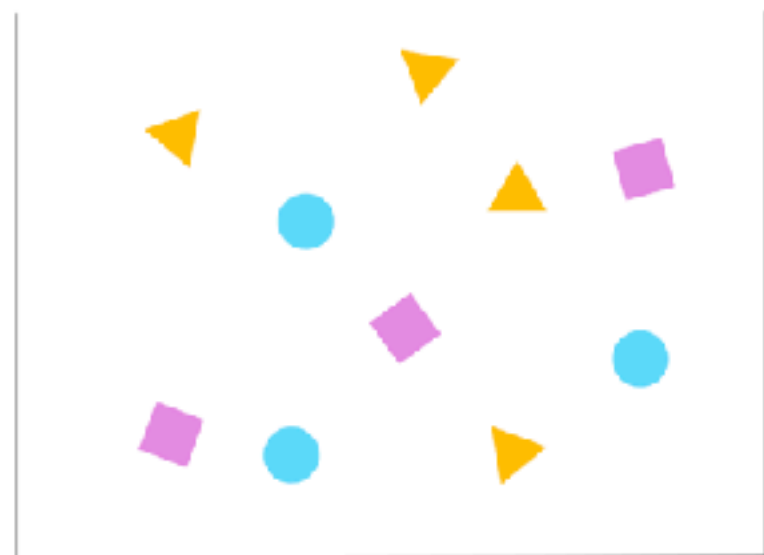
# 项目初期：规模小，模块关系清晰

- ▲ Reducer
- Action
- Component

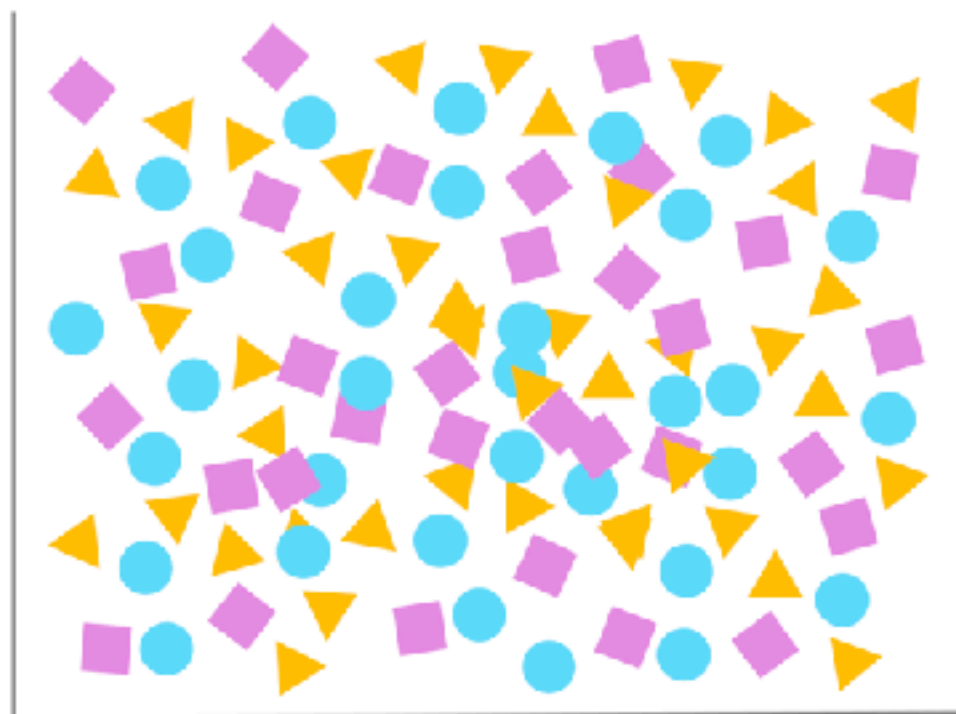


Application

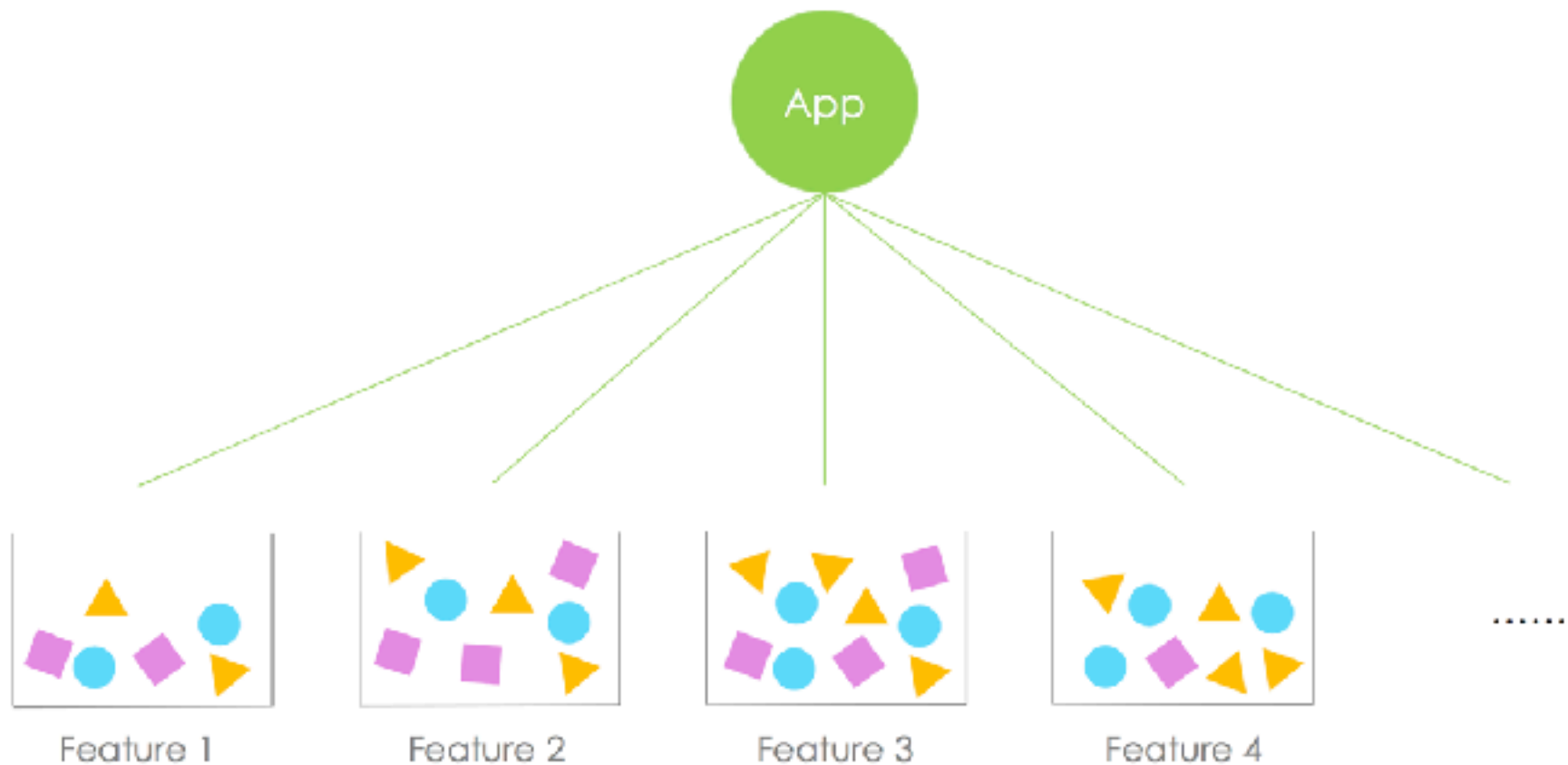
项目逐渐复杂，添加了更多组件和其他元素



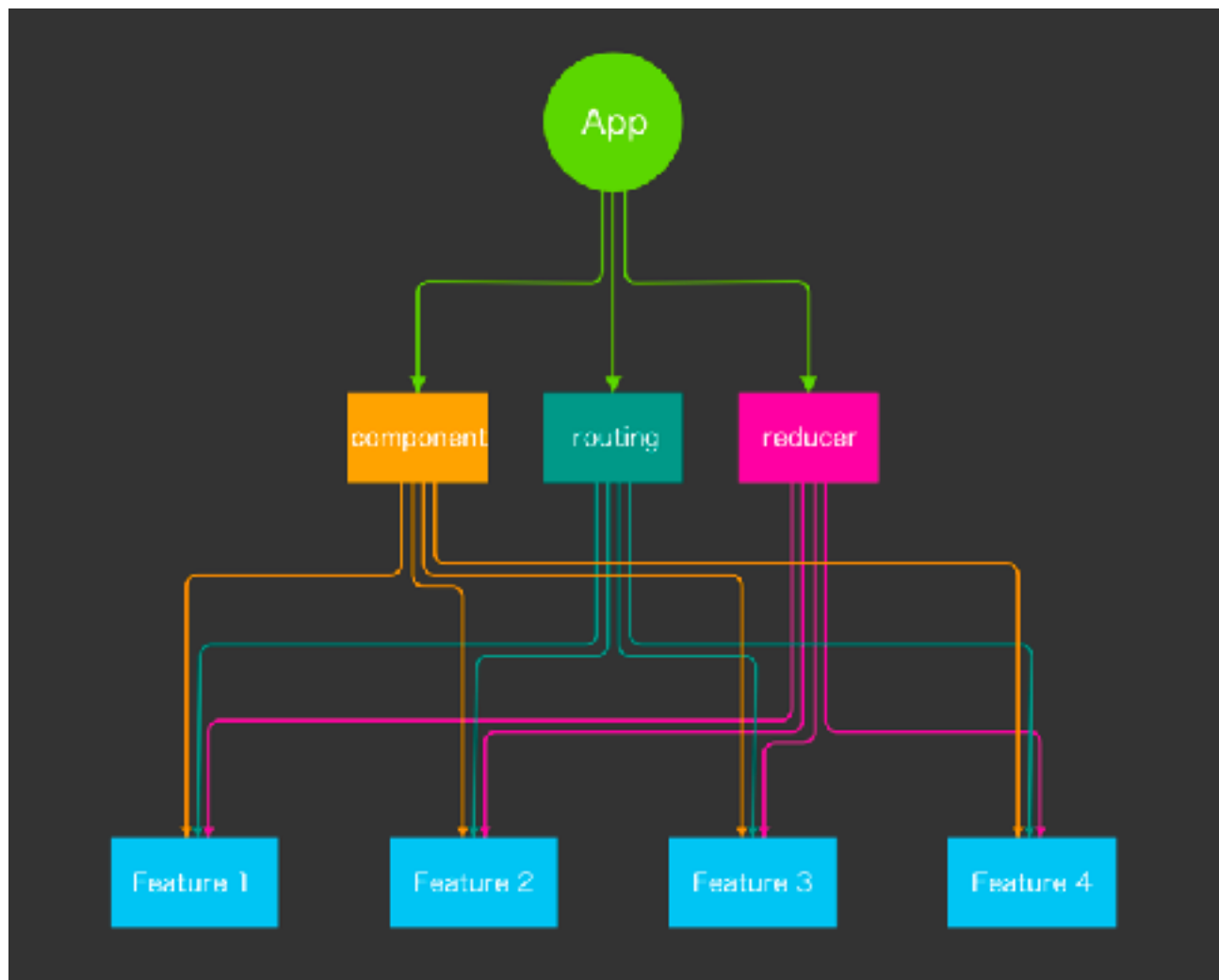
项目收尾：文件结构，模块依赖错综复杂



# 将业务逻辑拆分成高内聚松耦合的模块



# 通过 React 技术栈实现





# 小结

1. 大型前端应用需要拆分复杂度
2. 使用 React 技术栈如何实现

拆分复杂度（2）：

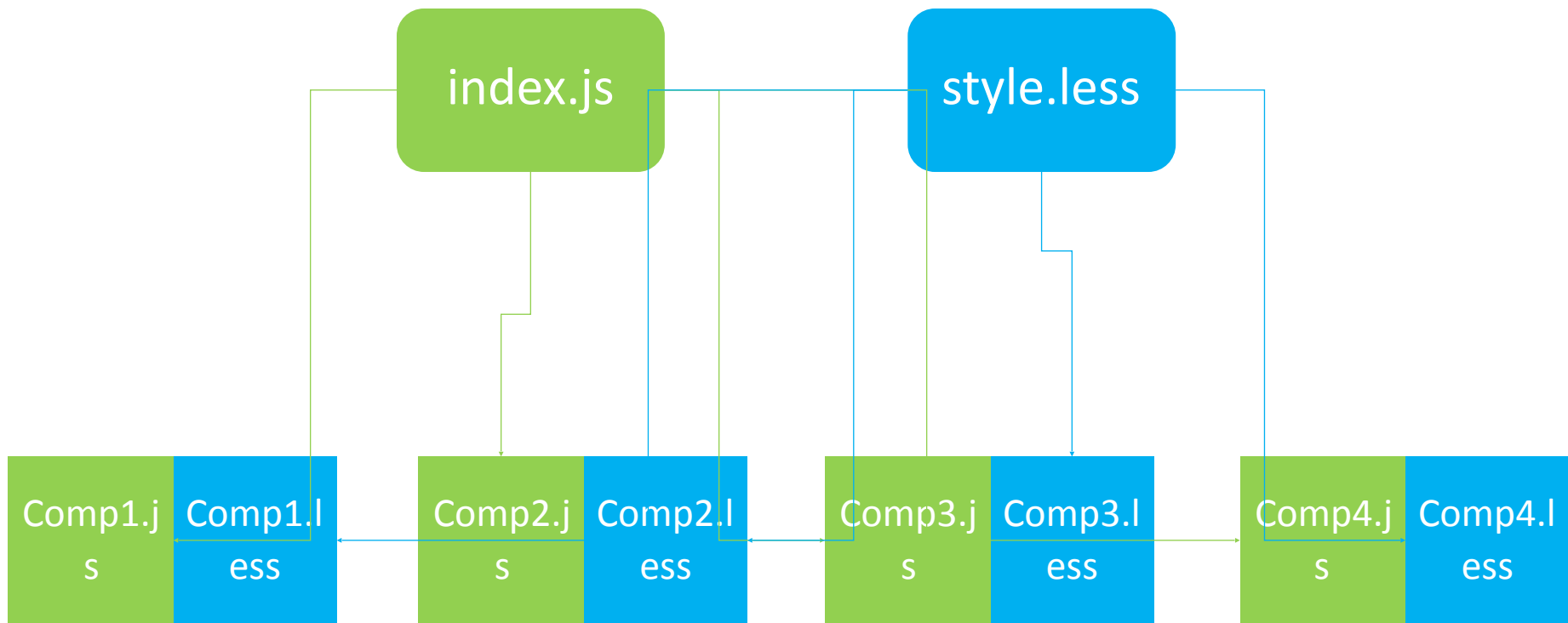
如何组织 component, action 和 reducer

# 文件夹结构

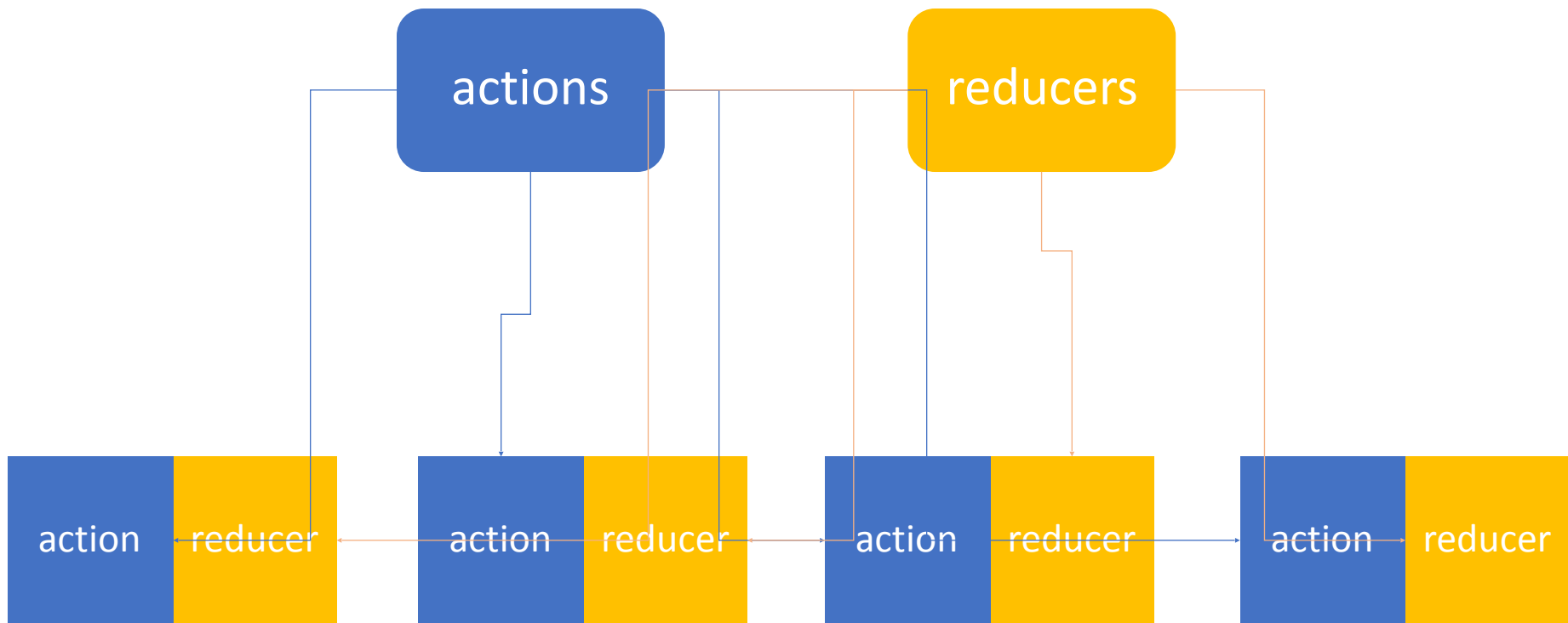
- 按 feature 组织源文件
- 组件和样式文件同一级
- Redux 单独文件夹
- 单元测试保持同样目录结构放在 tests 文件夹

```
src
├── common
│   ├── configStore.js
│   ├── history.js
│   ├── rootReducer.js
│   └── routeConfig.js
├── features
│   ├── common
│   └── examples
│       └── redux
│           ├── actions.js
│           ├── constants.js
│           ├── counterMinusOne.js
│           ├── counterPlusOne.js
│           ├── counterReset.js
│           ├── fetchRedditList.js
│           ├── initialState.js
│           ├── reducer.js
│           ├── CounterPage.js
│           ├── CounterPage.less
│           ├── index.js
│           ├── Layout.js
│           ├── Layout.less
│           ├── RedditListPage.js
│           └── RedditListPage.less
```

# 组件和样式



# 组织 Action 和 Reducer



# counterPlusOne.js

```
import { EXAMPLES_COUNTER_PLUS_ONE } from './constants';

export function counterPlusOne() {
  return {
    type: EXAMPLES_COUNTER_PLUS_ONE,
  };
}

export function reducer(state, action) {
  switch (action.type) {
    case EXAMPLES_COUNTER_PLUS_ONE:
      return {
        ...state,
        count: state.count + 1,
      };

    default:
      return state;
  }
}
```

# constants.js

```
export const EXAMPLES_COUNTER_PLUS_ONE = 'EXAMPLES_COUNTER_PLUS_ONE';
export const EXAMPLES_COUNTER_MINUS_ONE = 'EXAMPLES_COUNTER_MINUS_ONE';
export const EXAMPLES_COUNTER_RESET = 'EXAMPLES_COUNTER_RESET';
export const EXAMPLES_FETCH_REDDIT_LIST_BEGIN = 'EXAMPLES_FETCH_REDDIT_LIST_BEGIN';
export const EXAMPLES_FETCH_REDDIT_LIST_SUCCESS = 'EXAMPLES_FETCH_REDDIT_LIST_SUCCESS';
export const EXAMPLES_FETCH_REDDIT_LIST_FAILURE = 'EXAMPLES_FETCH_REDDIT_LIST_FAILURE';
export const EXAMPLES_FETCH_REDDIT_LIST_DISMISS_ERROR = 'EXAMPLES_FETCH_REDDIT_LIST_DISMISS_ERROR';
```

# reducer.js

```
import initialState from './initialState';
import { reducer as counterPlusOneReducer } from './counterPlusOne';
import { reducer as counterMinusOneReducer } from './counterMinusOne';
import { reducer as counterResetReducer } from './counterReset';
import { reducer as fetchRedditListReducer } from './fetchRedditList';

const reducers = [
  counterPlusOneReducer,
  counterMinusOneReducer,
  counterResetReducer,
  fetchRedditListReducer,
];

export default function reducer(state = initialState, action) {
  let newState;
  switch (action.type) {
    // Handle cross-topic actions here
    default:
      newState = state;
      break;
  }
  return reducers.reduce((s, r) => r(s, action), newState);
}
```



# actions.js

```
export { counterPlusOne } from './counterPlusOne';  
export { counterMinusOne } from './counterMinusOne';  
export { counterReset } from './counterReset';  
export { fetchRedditList, dismissFetchRedditListError } from './fetchRedditList';
```

# rootReducer.js

```
import { combineReducers } from 'redux';
import { routerReducer } from 'react-router-redux';
import homeReducer from '../features/home/redux/reducer';
import commonReducer from '../features/common/redux/reducer';
import examplesReducer from '../features/examples/redux/reducer';

const reducerMap = {
  router: routerReducer,
  home: homeReducer,
  common: commonReducer,
  examples: examplesReducer,
};

export default combineReducers(reducerMap);
```

DEMO

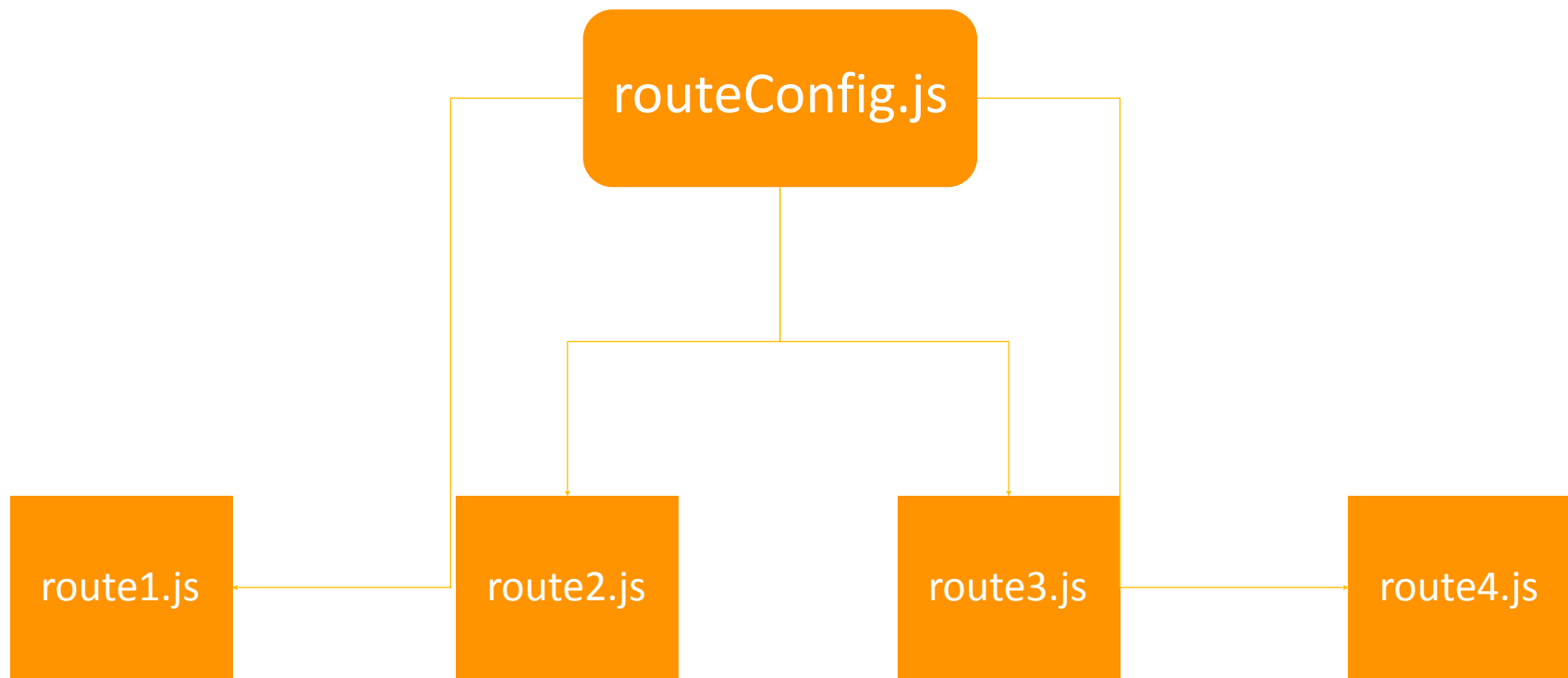
# 小结

1. 按 feature 组织组件，action 和 reducer
2. 使用 root loader 加载 feature 下的各个资源
3. 做到高内聚松耦合

拆分复杂度（3）：

如何组织 React Router 的路由配置

# 在每个 feature 中单独定义自己的路由



# 使用 JSON 定义顶层路由

```
import { WelcomePage, CounterPage, RedditListPage, Layout } from './';

export default {
  path: 'examples',
  name: 'Examples',
  component: Layout,
  childRoutes: [
    { path: '', name: 'Welcome page', component: WelcomePage },
    { path: 'counter', name: 'Counter page', component: CounterPage },
    { path: 'reddit', name: 'Reddit list page', protected: true, component: RedditListPage },
  ],
};
```

# 解析 JSON 路由到 React Router 语法

```
function renderRouteConfigV3(routes, contextPath) {  
  // Resolve route config object in React Router v3.  
  const children = []; // children component list  
  
  const renderRoute = (item, routeContextPath) => {  
    let newContextPath;  
    if (/^\/?/.test(item.path)) {  
      newContextPath = item.path;  
    } else {  
      newContextPath = `${routeContextPath}/${item.path}`;  
    }  
    newContextPath = newContextPath.replace(/\/+/g, '/');  
    if (item.component && item.childRoutes) {  
      const childRoutes = renderRouteConfigV3(item.childRoutes, newContextPath);  
      children.push(  
        <Route  
          key={newContextPath}  
          render={props => <item.component {...props}>{childRoutes}</item.component>}  
          path={newContextPath}  
        />  
      );  
    } else if (item.component) {  
      children.push(<Route key={newContextPath} component={item.component} path={newContextPath} exact />);  
    } else if (item.childRoutes) {  
      item.childRoutes.forEach(r => renderRoute(r, newContextPath));  
    }  
  };  
}
```



DEMO

# 小结

1. 每个 feature 都有自己的专属路由配置
2. 顶层路由使用 JSON 配置更易维护和理解
3. 如何解析 JSON 配置到 React Router 语法

# 使用 Rekit ( 1 ) : 创建项目 , 代码生成和重构



# Rekit

React 专属 IDE 和工具集

👁 Unwatch ▾

128

★ Unstar

3,209

🍴 Fork

189

# 背景

前端技术功能越来越强大

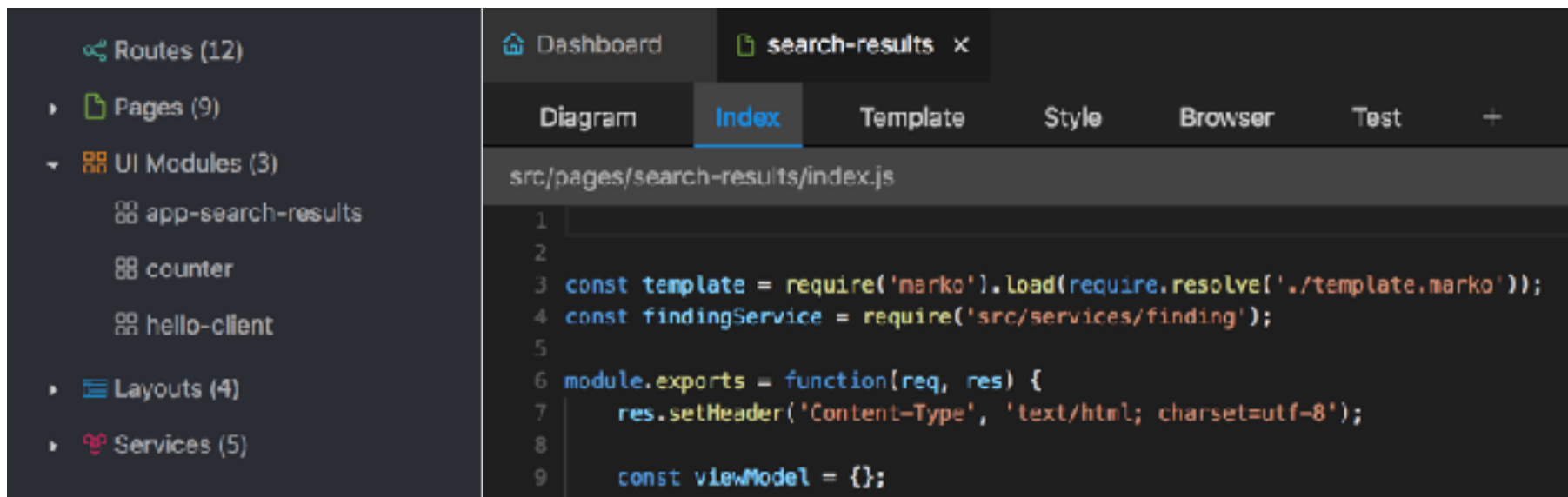


开发变得越来越复杂

- 一个独立功能通常需要多个文件组成
- 代码模板很复杂
- 重构极为困难
- 项目复杂后很难理解和维护

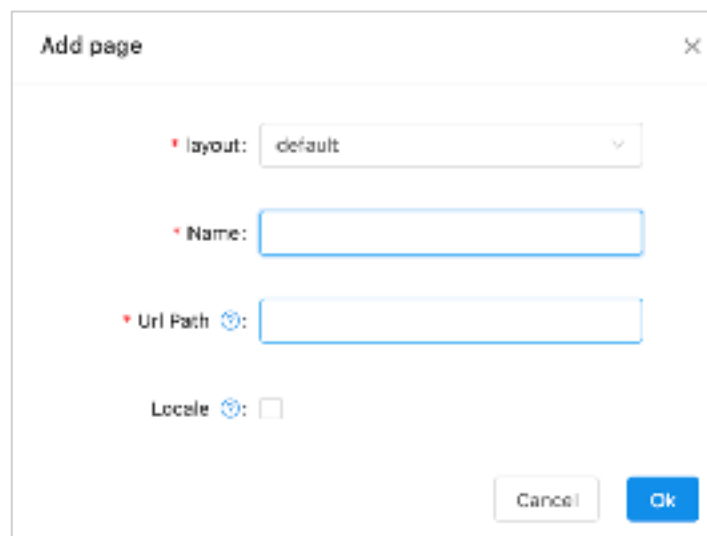
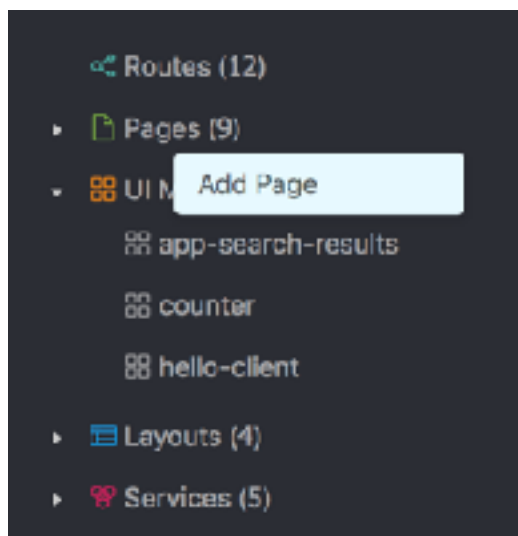
# Rekit: 更好的代码导航

1. 语义化的组织源代码文件
2. 使用子 Tab 来展示项目元素的各个部分
3. 直观的显示和导航某个功能的所有依赖



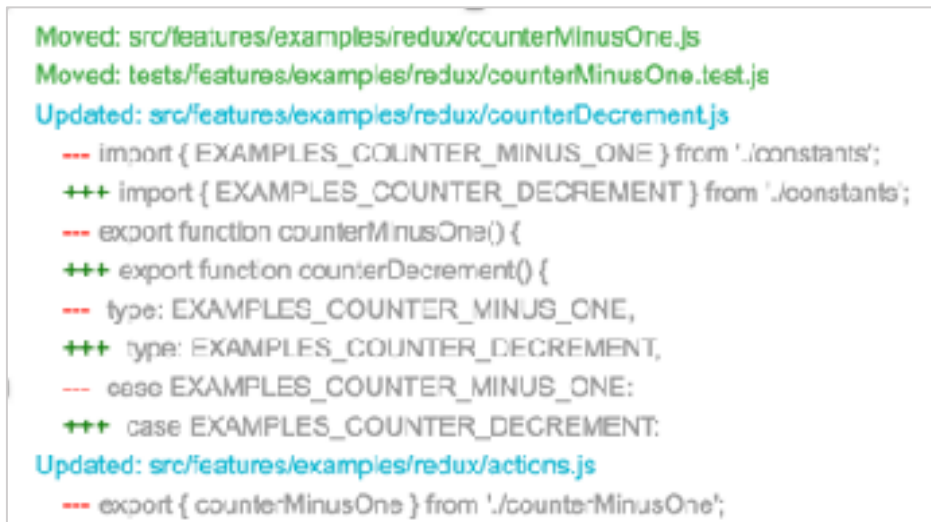
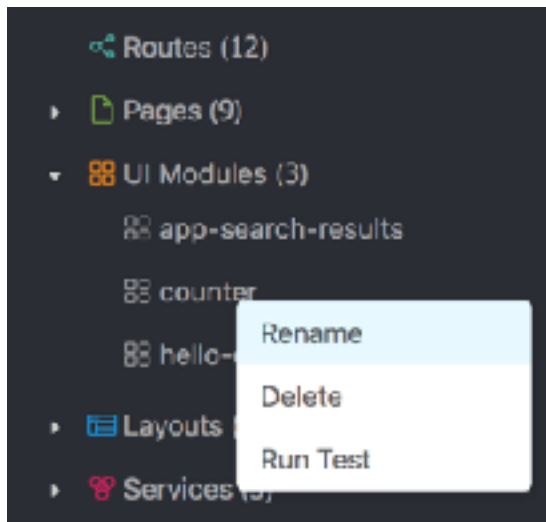
# Rekit: 一键生成项目元素

1. 直观的 UI 用于生成组件，action，reducer 等
2. 模板代码遵循最佳实践
3. 支持命令行方式创建项目元素



# Rekit: 重构非常容易

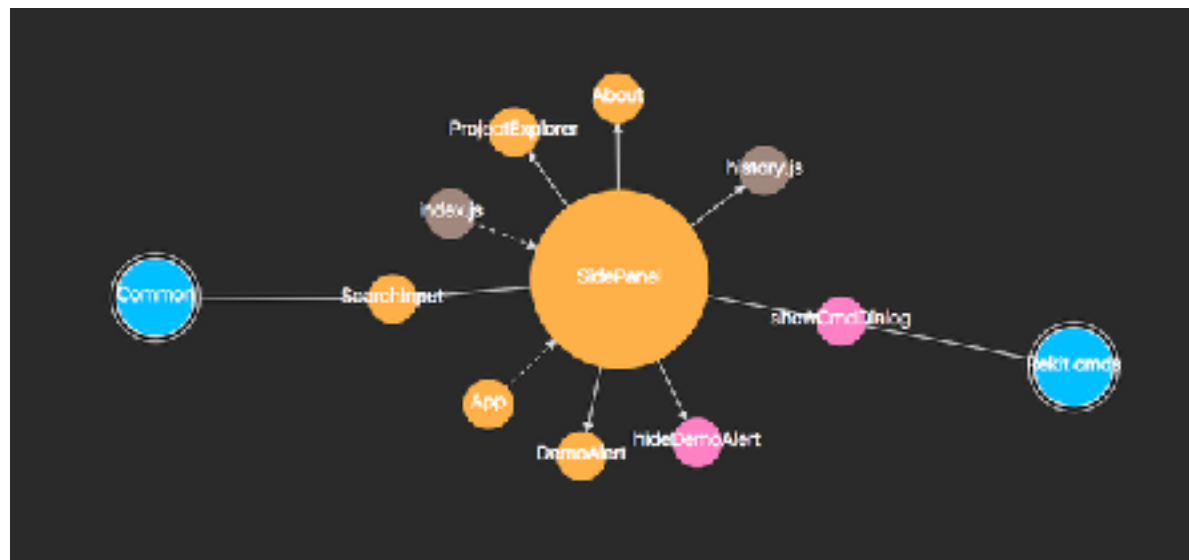
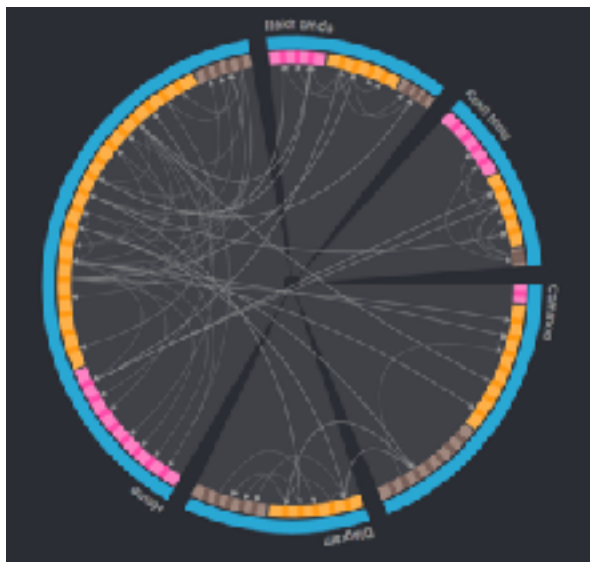
1. 右键菜单重命名或者删除某个项目元素
2. 所有相关代码都会一次性重构从而保证一致性
3. 详细的 log 信息显示重构的完整修改





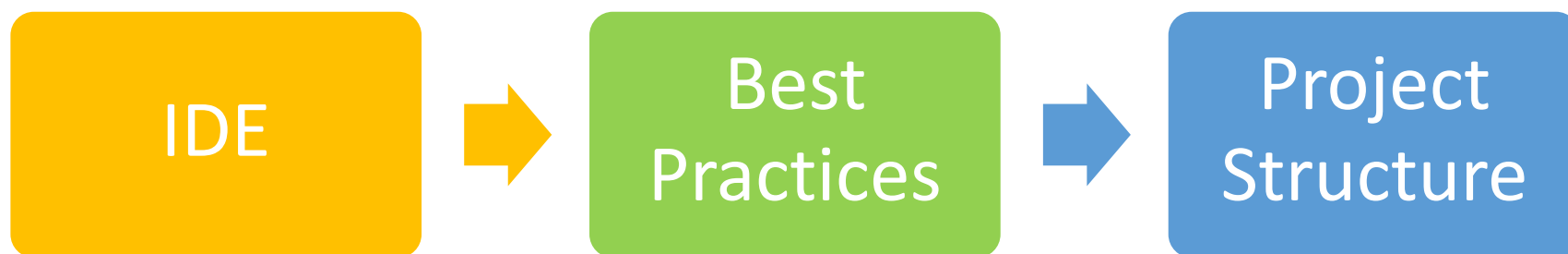
# Rekit: 可视化的项目架构

1. 项目总体架构的可视化图表
2. 项目依赖关系的图表

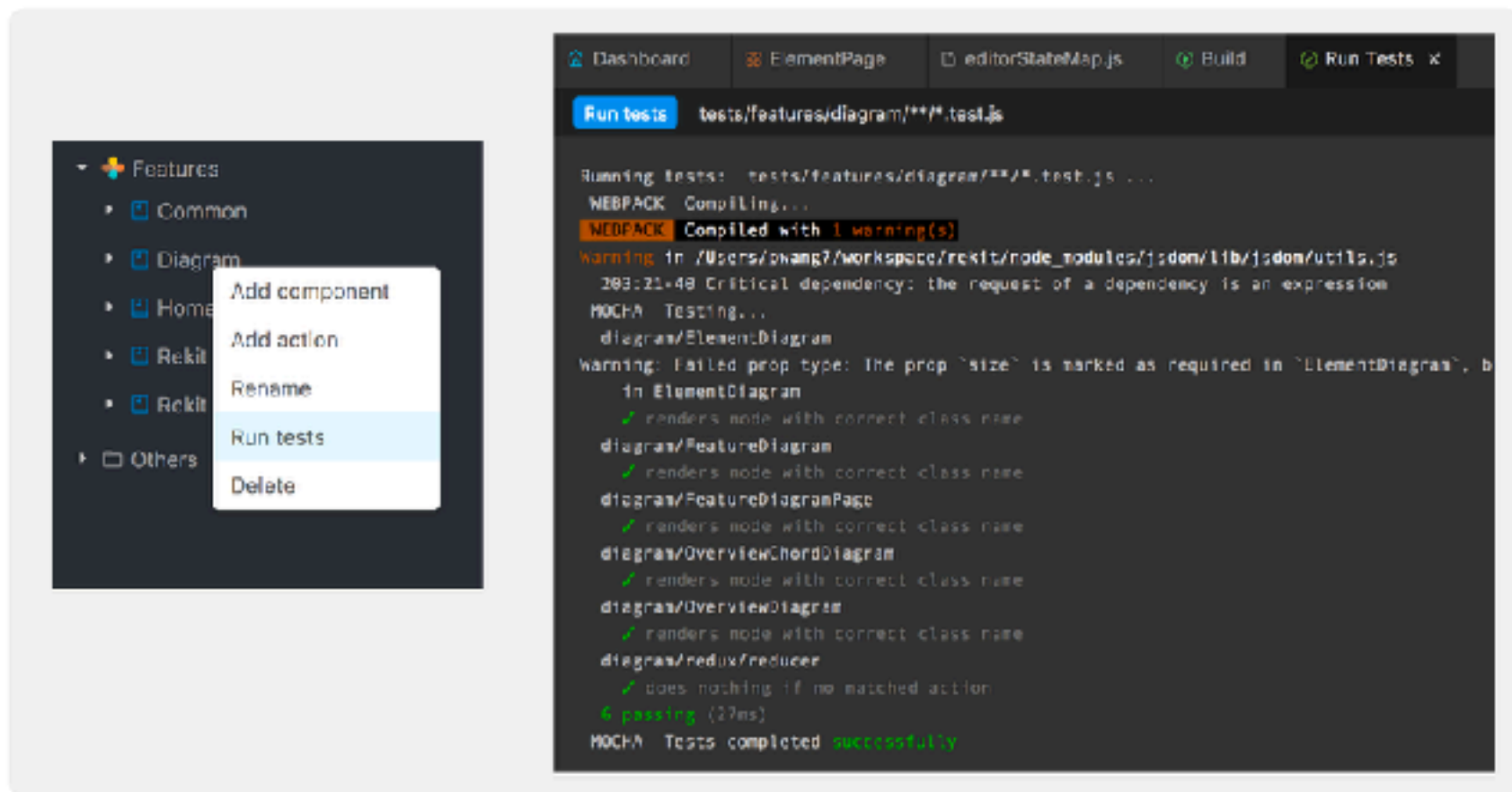


# Rekit 是如何工作的？

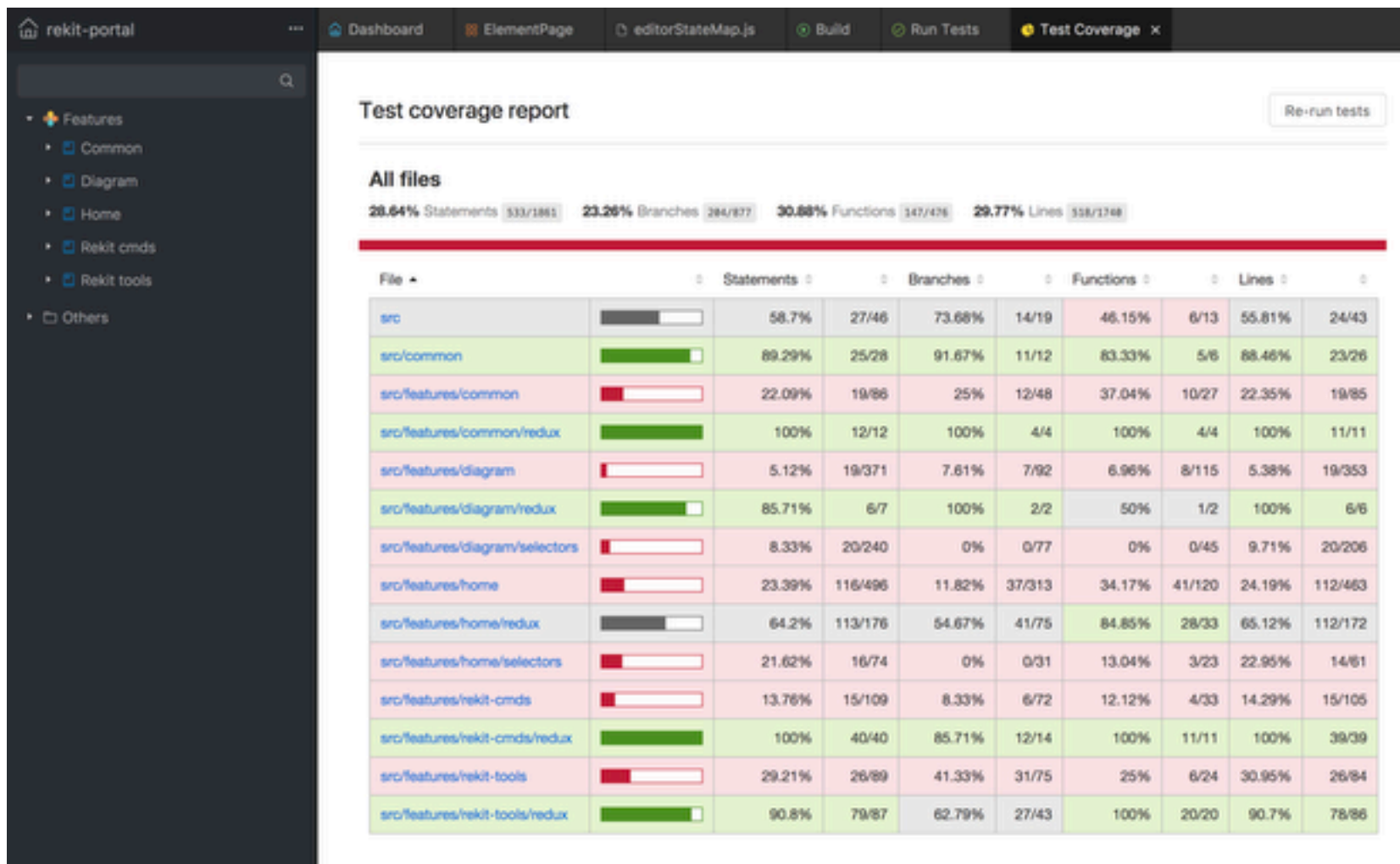
1. 定义了基于 feature 的可扩展文件夹结构
2. 基于最佳实践生成代码和管理项目元素
3. 提供工具和 IDE 确保代码和文件夹结构遵循最佳实践



# 集成单元测试



# 单元测试覆盖率



# 小结

1. 什么是 Rekit
2. 创建 Rekit 项目的背景
3. Rekit Studio 提供的基本功能

## 使用 Rekit ( 2 ) : 遵循最佳实践 , 保持代码一致性

# 遵循最佳实践

1. 以 feature 方式组织代码
2. 拆分组件，action 和 reducer
3. 拆分路由配置

# 通过代码自动生成保持一致性

1. 文件夹结构一致性
2. 文件名一致性
3. 变量名一致性
4. 代码逻辑的一致性



DEMO

# 使用 React Router 管理登录和授权

# 使用 React Router 管理路由授权

1. 实现基础：React Router 的动态路由机制
2. 区分受保护路由和公开路由
3. 访问未授权路由时重定向到登录页面

DEMO

# 实现表单（1）：初始数据，提交和跳转

DEMO

实现表单（2）：错误处理，动态表单元素，内容动态加载

DEMO