**Walkthrough(攻略)**

## Browser Extension(浏览器扩展)

如果你不想安装redux devtools集成到我们的项目代码中，我们可以使用[Redux DevTools Extension](https://github.com/zalmoxisus/redux-devtools-extension)扩展工具在chrome浏览或者是火狐浏览器上安装。这个浏览器扩展工具提供了许多流行的监视，非常容易配置来过滤actions，并且它还不需要install任何的包。

## Manual Integration(手动整合)

如果你想要完全控制devtools在哪里显示，或者开发自定义的监控，你可能会有手动配置的需求。虽然要多了一些步骤，但是你将完全控制监视器和它们的配置项。

### Installation(安装)

npm install --save-dev redux-devtools

你可能同时也想安装一些监视器:

npm install --save-dev redux-devtools-log-monitor

npm install --save-dev redux-devtools-dock-monitor

### Usage(使用)

#### Create a DevTools Component（创建一个DevlTools组件）

在你的项目中，通过一个monitor元素来创建一个DevTools组件，我们可以看一看下面例子中我们的monitor将由[LogMonitor](https://github.com/gaearon/redux-devtools-log-monitor) 合并 [DockMonitor](https://github.com/gaearon/redux-devtools-dock-monitor)组成：

##### containers/DevTools.js

import React from 'react';

// 从redux-devtools中引入

import { createDevTools } from 'redux-devtools';

// Monitors是单独的包,我们也可以自己定义一个

import LogMonitor from 'redux-devtools-log-monitor';

import DockMonitor from 'redux-devtools-dock-monitor';

// createDevTools 通过一个监视器和产生器来创建一个DevTools component

const DevTools = createDevTools(

// Monitors 是一个个单独的props.

// 查找这些包的github地址来学习了解他们的props.

// 这里,我们把LogMonitor放在DockMonitor里.

// 注意: DockMonitor默认是可见的.

<DockMonitor toggleVisibilityKey='ctrl-h'

changePositionKey='ctrl-q'

defaultIsVisible={true}>

<LogMonitor theme='tomorrow' />

</DockMonitor>

);

export default DevTools;

注意你现在可以直接使用 LogMonitor 安放你应用的UI内而不需要嵌在 DockMonitor里。

// 如果你的确不想使用dockingUI, 直接使用 <LogMonitor>

const DevTools = createDevTools(

<LogMonitor theme='solarized' />

);

#### Use DevTools.instrument() Store Enhancer(Store增强版)

 我们通过createDevTools()创建的DevTools组件有一个特殊的静态方法，叫做instrument().它返回一个store的增强版[store enhancer](http://redux.js.org/docs/Glossary.html" \l "store-enhancer) ，我们需要在开发环境中使用。

store enhancer是一个函数，提高了createStore()的行为。你可以把store enhancer作为最后一个参数传递给 createStore()。你可能已经使用了另一个store enhancer—[applyMiddleware()](http://redux.js.org/docs/api/applyMiddleware.html)。不像applyMiddleware()，你需要关心的只有如何使用DevTools.instrument()在开发环境中，不在生产环境中。

最简单的办法是使用 compose()函数来联系几个store enhancers。比如这样：compose(applyMiddleware(m1, m2, m3), DevTools.instrument())。

你可以给它添加额外的参数：DevTools.instrument({ maxAge: 50, shouldCatchErrors: true })。可以看[redux-devtools-instrument's API](https://github.com/zalmoxisus/redux-devtools-instrument" \l "api) 获取更多的细节。

你应该添加DevTools.instrument()在applyMiddleware后面在你的compose函数参内，这是很关键的。这是因为applyMiddleware 可能是异步的，但是 DevTools.instrument() 期望所有的action都是一个普通的对象而不是异步中间件比如 redux-promis或者redux-thunk所解释的。所以确保中间件 applyMiddleware放在compose参数的第一位，DevTools.instrument()紧随其后。

##### store/configureStore.js

import { createStore, applyMiddleware, compose } from 'redux';

import rootReducer from '../reducers';

import DevTools from '../containers/DevTools';

const enhancer = compose(

// 你想在开发环境使用的Middleware:

applyMiddleware(d1, d2, d3),

// 这是必需的! 使用你选择的附带monitors的Redux DevTools

DevTools.instrument()

);

export default function configureStore(initialState) {

// 注意: 只有Redux >= 3.1.0 才支持enhancer作为第三个参数.

// 可以看 https://github.com/rackt/redux/releases/tag/v3.1.0

const store = createStore(rootReducer, initialState, enhancer);

// 热重载reducers (requires Webpack or Browserify HMR to be enabled)

if (module.hot) {

module.hot.accept('../reducers', () =>

store.replaceReducer(require('../reducers')/\*.default if you use Babel 6+ \*/)

);

}

return store;

}

如果你喜欢，你可以添加另一个叫persistState()。它可以让你把控整体的会话（包括全部的dispatch出来的action和监听的状态），只需要一个URL的参数。你可以访问http://localhost:3000/?debug\_session=reproducing\_weird\_bug ，在应用中操作，然后打开http://localhost:3000/?debug\_session=some\_other\_feature ，最后返回到http://localhost:3000/?debug\_session=reproducing\_weird\_bug ，状态将会被清空重置。如果你喜欢它，你可以把它加入到你的项目当中，这也是一个颇具成就兼备灵感的做法。

#### Exclude DevTools from Production Builds(在生产环境中排出DevTools)

最后，我们需要确保我们的DevlTools代码没有被包含进生产环境代码。我们可以使用一些Webpack的[DefinePlugin](https://github.com/webpack/docs/wiki/list-of-plugins" \l "defineplugin)插件，或者是Browserify的[envify](https://github.com/zertosh/loose-envify)。

  秘诀是用过设定process.env.NODE\_ENV 来控制我们的环境设定，比如说我们的redux-devtools 仅在我们的 process.env.NODE\_ENV 为空或者不是’production’运行。

一般来说，如果使用Webpack你需要两个配置文件，一个针对开发环境一个针对生产环境，这是一个列子:

##### webpack.config.prod.js

// ...

plugins: [

new webpack.DefinePlugin({

'process.env.NODE\_ENV': JSON.stringify('production')

})

],

// ...

如果你正在使用Webpack和Bable的ES6模块系统，你应该试着通过判断process.env.NODE\_ENV来决定是否import包含这个DevTools到我们的打包文件中。然而ES6语法禁止这样，导致不会编译成功。恰好，我们可以通过CommonJS规范的require来代替。Babel会将它编译，并在Webpack打包之前决定是否包含。这就是为什么我们要创建一个configureStore.js文件，我们使用configureStore.dev.js 或者 configureStore.prod.js取决于我们的配置。虽然我们为此付出了稍多的维护代价，但好处是，我们可以完全掌控住任何开发依赖不会放进我们的生产代码中，并且我们可以更灵活地使用不用的中间件（比如 崩溃报告、日志记录）在我们的生产环境中。

##### store/configureStore.js

// 使用DefinePlugin (Webpack) 或者 loose-envify (Browserify)

// 把开发和生产环境区分开发打包

if (process.env.NODE\_ENV === 'production') {

module.exports = require('./configureStore.prod');

} else {

module.exports = require('./configureStore.dev');

}

##### store/configureStore.prod.js

import { createStore, applyMiddleware, compose } from 'redux';

import rootReducer from '../reducers';

// 你想在生产环境使用的中间件:

const enhancer = applyMiddleware(p1, p2, p3);

export default function configureStore(initialState) {

// 注意: 只有Redux >= 3.1.0 才支持enhancer作为第三个参数.

// 可以看 https://github.com/rackt/redux/releases/tag/v3.1.0

return createStore(rootReducer, initialState, enhancer);

};

##### store/configureStore.dev.js

import { createStore, applyMiddleware, compose } from 'redux';

import { persistState } from 'redux-devtools';

import rootReducer from '../reducers';

import DevTools from '../containers/DevTools';

const enhancer = compose(

//你想在开发环境使用的Middleware:

applyMiddleware(d1, d2, d3),

//这是必需的! 使用你选择的附带monitors的Redux DevTools

DevTools.instrument(),

//参数，让你输入?debug\_session=<key> 在地址栏中持续地debug会话

persistState(getDebugSessionKey())

);

function getDebugSessionKey() {

//你可以编写自定义的逻辑

//默认情况下我们是这样输入在地址栏中的 ?debug\_session=<key>

const matches = window.location.href.match(/[?&]debug\_session=([^&]+)\b/);

return (matches && matches.length > 0)? matches[1] : null;

}

export default function configureStore(initialState) {

// 注意: 只有Redux >= 3.1.0 才支持enhancer作为第三个参数.

// 可以看 <https://github.com/rackt/redux/releases/tag/v3.1.0>

const store = createStore(rootReducer, initialState, enhancer);

// 热重载reducers (requires Webpack or Browserify HMR to be enabled)

if (module.hot) {

module.hot.accept('../reducers', () =>

store.replaceReducer(require('../reducers')/\*.default if you use Babel 6+ \*/)

);

}

return store;

}

#### Render <DevTools> 在你的应用...

最后，引入DevTools 组件到你的页面。

你可以把它放到index.js 里面渲染。

##### index.js

import React from 'react';

import { render } from 'react-dom';

import { Provider } from 'react-redux';

import configureStore from './store/configureStore';

import TodoApp from './containers/TodoApp';

// 不要这样做，你把DevTools带到了生产环境的bundle.

import DevTools from './containers/DevTools';

const store = configureStore();

render(

<Provider store={store}>

<div>

<TodoApp />

<DevTools />

</div>

</Provider>

document.getElementById('app')

);

我们推荐使用一种类似上面configureStore.js的做法。创建一个root.js组件，来渲染我们应用的根节点(通常来说是一<Provider>包含一些组件)。然后根据判断process.env.NODE\_ENV来判断开发和生产版本暴露的js文件。

##### containers/Root.js

if (process.env.NODE\_ENV === 'production') {

module.exports = require('./Root.prod');

} else {

module.exports = require('./Root.dev');

}

##### containers/Root.dev.js

import React, { Component } from 'react';

import { Provider } from 'react-redux';

import TodoApp from './TodoApp';

import DevTools from './DevTools';

export default class Root extends Component {

render() {

const { store } = this.props;

return (

<Provider store={store}>

<div>

<TodoApp />

<DevTools />

</div>

</Provider>

);

}

}

##### containers/Root.prod.js

import React, { Component } from 'react';

import { Provider } from 'react-redux';

import TodoApp from './TodoApp';

export default class Root extends Component {

render() {

const { store } = this.props;

return (

<Provider store={store}>

<TodoApp />

</Provider>

);

}

}

### Gotchas(坑)

**你的reducers必须是一个纯净无副作用的纯函数.** 例如，即使在reducer中使用了一个随机的random的ID来标识某个内容，它也会使其不纯净。这样的操作我们一般放在action的创建中。

**确保DevTools.instrument()和render <DevTools>只在开发环境中使用!** 在生产中使用，它将会导致我们的应用变得非常缓慢，因为actions会源源不断地积累。如上所述，你要根据判断条件来引入它，比如DefinePlugin(Webpack)或者loos-envify(Browserify)来判断不再打包代码时引入它。

这里是一个Redux DevTools处理生产情况下一个极佳的例子：

https://github.com/erikras/react-redux-universal-hot-example/

**DevTools.instrument()store增强器应该被添加到你的中间件的后面，然后被compose函数当作参数包含起来，因为中间件可能是异步的.**否则DevTools将不能观察到通过异步中间件（比如redux-promise或者redux-thunk）的action。

### What Next?(下一步)

现在你可以看到DevTools了，你可能想要学习这些按钮的意思并且使用它们。这一切通常取决于monitor（监视器）。

你可以开始探索关于 [LogMonitor](https://github.com/gaearon/redux-devtools-log-monitor) 和 [DockMonitor](https://github.com/gaearon/redux-devtools-dock-monitor) 的文档，我们默认一起使用它们。我们就可以自定义更多自己要得显示监听内容。