目录

[dva-cli 2](#_Toc496435897)

[Install 2](#_Toc496435898)

[new 2](#_Toc496435899)

[dva init [options] 2](#_Toc496435900)

[dva generate (g) 2](#_Toc496435901)

[输出文件 3](#_Toc496435902)

[dva 3](#_Toc496435903)

[dva/mobile 3](#_Toc496435904)

[dva/router 3](#_Toc496435905)

[dva/fetch 3](#_Toc496435906)

[dva/saga 4](#_Toc496435907)

[dva API 4](#_Toc496435908)

[app = dva(opts) 4](#_Toc496435909)

[app.use(hooks) 5](#_Toc496435910)

[hooks 包含： 5](#_Toc496435911)

[onError(fn, dispatch) 5](#_Toc496435912)

[onAction(fn | fn[]) 6](#_Toc496435913)

[onStateChange(fn) 6](#_Toc496435914)

[onReducer(fn) 6](#_Toc496435915)

[onEffect(fn) 7](#_Toc496435916)

[onHmr(fn) 7](#_Toc496435917)

[extraReducers 7](#_Toc496435918)

[extraEnhancers 7](#_Toc496435919)

[app.model(model) 7](#_Toc496435920)

[app.unmodel(namespace) 8](#_Toc496435921)

[app.router(({ history, app } => RouterConfig) 8](#_Toc496435922)

[app.start(selector?) 8](#_Toc496435923)

[Model 9](#_Toc496435924)

[namespace 10](#_Toc496435925)

[state 10](#_Toc496435926)

[reducers 11](#_Toc496435927)

[effects 11](#_Toc496435928)

[subscriptions 12](#_Toc496435929)

[roadhogrc 12](#_Toc496435930)

[cli 12](#_Toc496435931)

[安装 12](#_Toc496435932)

[使用 12](#_Toc496435933)

[本地开发 12](#_Toc496435934)

[打包发布 13](#_Toc496435935)

[测试 13](#_Toc496435936)

[HMR (热替换) 13](#_Toc496435937)

[Mock 13](#_Toc496435938)

[配置 14](#_Toc496435939)

[默认配置 14](#_Toc496435940)

# dva-cli

## Install

$ npm install dva-cli -g

## new

$ dva new myapp

**options**

* --demo -- Generate a dead simple project for quick prototype
* --no-install -- Disable npm install after files created

## dva init [options]

Create app in current directory. It's options is the same as dva new.

## dva generate (g)

Generate route, model and component.

**例子：**

$ dva g route product-list

$ dva g model products

$ dva g component Editor

$ dva g component Users/UserModal

$ dva g component Header --no-css

# 输出文件

## dva

默认输出文件。

## dva/mobile

dva/mobile 是无 router 版的 dva，可用于多页应用，react-native 等。叫 dva/mobile 有点欠考虑，可能在下个大版本中改名。

## dva/router

默认输出 [react-router@2.x](https://github.com/ReactTraining/react-router/tree/v2.8.1) 接口， [react-router-redux](https://github.com/reactjs/react-router-redux) 的接口通过属性 routerRedux 输出。

比如：

import { Router, Route, routerRedux } from 'dva/router';

## dva/fetch

异步请求库，输出 [isomorphic-fetch](https://github.com/matthew-andrews/isomorphic-fetch) 的接口。不和 dva 强绑定，可以选择任意的请求库。

## dva/saga

输出 [redux-saga](https://github.com/yelouafi/redux-saga) 的接口，主要用于用例的编写。（用例中需要用到 effects）

# dva API

## app = dva(opts)

创建应用，返回 dva 实例。(注：dva 支持多实例)

opts 包含：

* **history**：指定给路由用的 history，默认是 hashHistory
* **initialState**：指定初始数据，优先级高于 model 中的 state，默认是 {}

如果要配置 history 为 browserHistory，可以这样：

import { browserHistory } from 'dva/router';

const app = dva({

history: browserHistory,

});

另外，出于易用性的考虑，opts 里也可以配所有的 [hooks](https://github.com/dvajs/dva/blob/master/docs/API_zh-CN.md#appusehooks) ，下面包含全部的可配属性：

const app = dva({

history,

initialState,

onError,

onAction,

onStateChange,

onReducer,

onEffect,

onHmr,

extraReducers,

extraEnhancers,

});

## app.use(hooks)

配置 hooks 或者注册插件。（插件最终返回的是 hooks ）

比如注册 [dva-loading](https://github.com/dvajs/dva-loading) 插件的例子：

import createLoading from 'dva-loading';

...

app.use(createLoading(opts));

## hooks 包含：

### onError(fn, dispatch)

effect 执行错误或 subscription 通过 done 主动抛错**时触发，**可用于管理全局出错状态。

注意：subscription 并没有加 try...catch，所以有错误时需通过第二个参数 done 主动抛错。例子：

app.model({

subscriptions: {

setup({ dispatch }, done) {

done(e);

},

},

});

如果我们用 antd，那么最简单的全局错误处理通常会这么做：

import { message } from 'antd';

const app = dva({

onError(e) {

message.error(e.message, /\* duration \*/3);

},

});

### onAction(fn | fn[])

在 action 被 dispatch 时触发，用于注册 redux 中间件。支持函数或函数数组格式。

例如我们要通过 [redux-logger](https://github.com/evgenyrodionov/redux-logger) 打印日志：

import createLogger from 'redux-logger';

const app = dva({

onAction: createLogger(opts),

});

### onStateChange(fn)

state 改变时触发，可用于同步 state 到 localStorage，服务器端等。

### onReducer(fn)

封装 reducer 执行。比如借助 [redux-undo](https://github.com/omnidan/redux-undo) 实现 redo/undo ：

import undoable from 'redux-undo';

const app = dva({

onReducer: reducer => {

return (state, action) => {

const undoOpts = {};

const newState = undoable(reducer, undoOpts)(state, action);

// 由于 dva 同步了 routing 数据，所以需要把这部分还原

return { ...newState, routing: newState.present.routing };

},

},

});

详见 [examples/count-undo](https://github.com/dvajs/dva/blob/master/examples/count-undo/index.js) 。

### onEffect(fn)

封装 effect 执行。比如 [dva-loading](https://github.com/dvajs/dva-loading) 基于此实现了自动处理 loading 状态。

### onHmr(fn)

热替换相关，目前用于 [babel-plugin-dva-hmr](https://github.com/dvajs/babel-plugin-dva-hmr) 。

### extraReducers

指定额外的 reducer，比如 [redux-form](https://github.com/erikras/redux-form) 需要指定额外的 form reducer：

import { reducer as formReducer } from 'redux-form'

const app = dva({

extraReducers: {

form: formReducer,

},

});

### extraEnhancers

指定额外的 [StoreEnhancer](https://github.com/reactjs/redux/blob/master/docs/Glossary.md#store-enhancer) ，比如结合 [redux-persist](https://github.com/rt2zz/redux-persist) 的使用：

import { persistStore, autoRehydrate } from 'redux-persist';

const app = dva({

extraEnhancers: [autoRehydrate()],

});

persistStore(app.\_store);

## app.model(model)

注册 model，详见 [#Model](https://github.com/dvajs/dva/blob/master/docs/API_zh-CN.md#model) 部分。

## app.unmodel(namespace)

取消 model 注册，清理 reducers, effects 和 subscriptions。subscription 如果没有返回 unlisten 函数，使用 app.unmodel 会给予警告⚠️。

## app.router(({ history, app } => RouterConfig)

注册路由表。

通常是这样的：

import { Router, Route } from 'dva/router';

app.router((**{ history }**) => {

return (

**<Router history={history}>**

**<Route path="/" component={App} />**

**<Router>**

);

});

推荐把路由信息抽成一个单独的文件，这样结合 [babel-plugin-dva-hmr](https://github.com/dvajs/babel-plugin-dva-hmr) 可实现路由和组件的热加载，比如：

app.router(require('./router'));

而有些场景可能不使用路由，比如多页应用，所以**也可以传入返回 JSX 元素的函数。**比如：

app.router(() => <App />);

## app.start(selector?)

启动应用。**selector** 可选，如果**没有 selector 参数，会返回一个返回 JSX 元素的函数。**

app.start('**#root**');（对应html中id=“root”）

那么什么时候不加 selector？常见场景有测试、node 端、react-native 和 i18n 国际化支持。

比如通过 react-intl 支持国际化的例子：

import { IntlProvider } from 'react-intl';

...

const App = app.start();

ReactDOM.render(<IntlProvider><App /></IntlProvider>, htmlElement);

# Model

model 是 dva 中最重要的概念。

有多个model,app.model({},{}),每个model独立出来。

export default {namespace:”,}

以下是典型的例子：

app.model({

namespace: 'todo',

state: [],

reducers: {

add(state, { payload: todo }) {

// 保存数据到 state

return [...state, todo];

},

},

effects: {

\*save({ payload: todo }, { put, call }) {

// 调用 saveTodoToServer，成功后触发 `add` action 保存到 state

yield call(saveTodoToServer, todo);

yield put({ type: 'add', payload: todo });

},

},

subscriptions: {

setup({ history, dispatch }) {

// 监听 history 变化，当进入 `/` 时触发 `load` action

return history.listen(({ pathname }) => {

if (pathname === '/') {

dispatch({ type: 'load' });

}

});

},

},

});

model 包含 5 个属性：

## namespace

model 的命名空间，同时也是他在全局 state 上的属性，只能用字符串，**不支持通过 . 的方式创建多层命名空间。**

## state

初始值，优先级低于传给 dva() 的 opts.initialState。

比如：

const app = dva({

initialState: { count: 1 },

});

app.model({

namespace: 'count',

state: 0,

});

此时，在 app.start() 后 state.count 为 1 。

## reducers

以 key/value 格式定义 reducer。用于处理同步操作，**唯一可以修改 state 的地方**。由 action 触发。

格式为 **(state**, **action**) => newState 或 [(state, action) => newState, enhancer]。

loginSuccess(state, {payload})这样不用加载整个action

详见： <https://github.com/dvajs/dva/blob/master/test/reducers-test.js>

## effects

以 key/value 格式定义 effect。用于处理异步操作和业务逻辑，不直接修改 state。由 action 触发，**可以触发 action，可以和服务器交互，可以获取全局 state 的数据等等。**

格式为 \*(**action**, **effects**) => void 或 [\*(action, effects) => void, { type }]。

login: function \*({payload},{call,put}){ 同样懒加载

type 类型有：

* takeEvery
* takeLatest
* throttle
* watcher

详见：<https://github.com/dvajs/dva/blob/master/test/effects-test.js>

## subscriptions

以 key/value 格式定义 subscription。subscription 是订阅，用于订阅一个数据源，然后根据需要 dispatch 相应的 action。在 app.start() 时被执行，数据源可以是当前的时间、服务器的 websocket 连接、keyboard 输入、geolocation 变化、history 路由变化等等。

格式为 ({ dispatch, history }, done) => unlistenFunction。

注意：如果要使用 app.unmodel()，subscription 必须返回 unlisten 方法，用于取消数据订阅。

# roadhogrc

有个不推荐的方法，.roadhogrc中的配置优先级低于webpack.config.js中的配置，可以在webpack.config.js中按照 webpack 的配置重新配置，覆盖.roadhogrc

## cli

### 安装

$ npm i roadhog -g

### 使用

### 本地开发

$ roadhog server

### 打包发布

$ roadhog build

### 测试

默认会跑 ./test 目录下的所有文件

$ roadhog test

## HMR (热替换)

CSS 在开发模式下会走 style-loader (被内嵌在 JavaScript 文件中)，所以只要保证 JavaScript 的热更新，即可实现 CSS 的热更新。

如果大家使用 [dva](https://github.com/dvajs/dva) ，配上 [babel-plugin-dva-hmr](https://github.com/dvajs/babel-plugin-dva-hmr) 即可实现 routes 和 components 以及相关 CSS 修改的热更新，其他修改会自动刷新页面。

"env": {

"development": {

"extraBabelPlugins": ["dva-hmr"]

}

}

## Mock

roadhog server 支持 mock 功能，类似 [dora-plugin-proxy](https://github.com/dora-js/dora-plugin-proxy)，在 .roadhogrc.mock.js 中进行配置，支持基于 require 动态分析的实时刷新，支持 ES6 语法，以及友好的出错提示。

比如：

export default {

// 支持值为 Object 和 Array

'GET /api/users': { users: [1,2] },

// GET POST 可省略

'/api/users/1': { id: 1 },

// 支持自定义函数，API 参考 express@4

'POST /api/users/create': (req, res) => { res.end('OK'); },

// Forward 到另一个服务器

'GET /assets/\*': 'https://assets.online/',

// Forward 到另一个服务器，并指定子路径

// 请求 /someDir/0.0.50/index.css 会被代理到 https://g.alicdn.com/tb-page/taobao-home, 实际返回 https://g.alicdn.com/tb-page/taobao-home/0.0.50/index.css

'GET /someDir/(.\*)': 'https://g.alicdn.com/tb-page/taobao-home',

};

## 配置

默认配置：

{

"entry": "src/index.js",

"disableCSSModules": false,

"publicPath": "/",

"outputPath": "./dist",

"extraBabelPlugins": [],

"extraPostCSSPlugins": [],

"autoprefixer": null,

"proxy": null,

"externals": null,

"library": null,

"libraryTarget": "var",

"multipage": false,

"define": null,

"env": null,

"theme": null,

}

# 技巧

## effect另一个model

model/apply.js

yield put({

type: 'auth/applySuccess',

payload: {

leave\_left,

},

});

最前面没有/