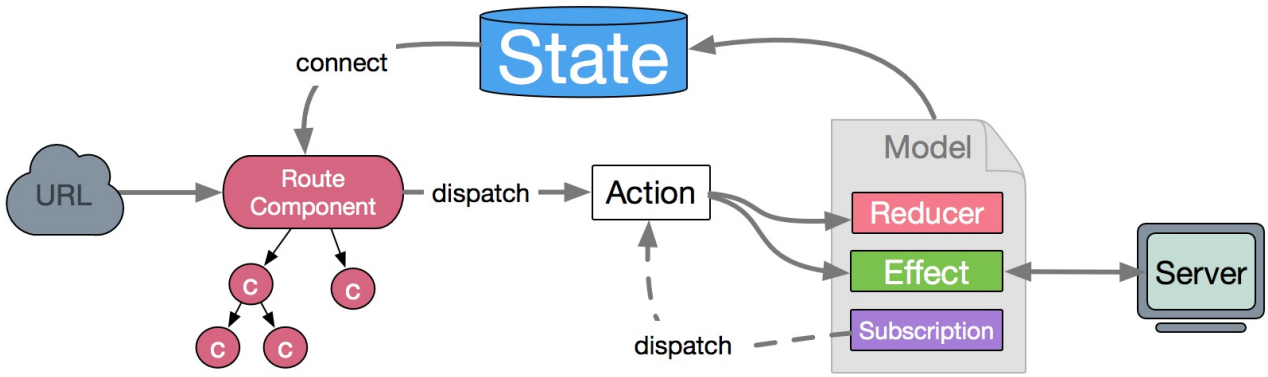
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=322 sentences=1539, words=12769

1.dva介绍

- dva (https://github.com/dvajs/dva)首先是一个基于 redux 和 redux-saga 的数据流方案，然后为了简化开发体验,dva 还额外内置了 react-router 和 fetch,所以也可以理解为一个轻量级的应用框架



1.1 前置知识

- react
- react-router-dom
- redux
- react-redux
- connected-react-router
- history
- dva

2. 初始化项目

```
create-react-app zhufeng-dva-source-typescript --typescript
cd  zhufeng-dva-source-typescript
cnpm install dva@2.6.0-beta.20 redux react-redux react-router-dom connected-react-router history --save
npm start
```

3. 基本计数器

3.1 使用

3.1.1 index.tsx

```
import React from 'react';
import dva, { connect } from './dva';
const app = dva();
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    }
});
function Counter(props) {
    return (
        <div>
            <p>{props.number}p>
            <button onClick={() => props.dispatch({ type: "counter/add" })}>+button>
        div>
    )
}
const ConnectedCounter = connect(
    (state) => state.counter
)(Counter);
app.router(() => <ConnectedCounter />);
app.start('#root');
```

3.2 实现

3.2.1 dva\index.tsx

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers } from 'redux';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
export { connect };

export default function () {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = {};
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }
    function router(router: Router) {
        app._router = router;
    }

    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
        let store = createStore(rootReducer);
        ReactDOM.render(<Provider store={store}>{app._router()}Provider>, document.querySelector(root));
        function createReducer() {
          return combineReducers(initialReducers);
        }
    }

    return app;
}

function getReducer(model) {
    let { reducers, state: defaultState } = model;
    let reducer = (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
    return reducer;
}
export * from './typings';
```

3.2.2 prefixNamespace.tsx

src\dva\prefixNamespace.tsx

```
import { NAMESPACE_SEP } from './constants';
function prefix(obj, namespace) {
    return Object.keys(obj).reduce((memo, key) => {
        const newKey = `${namespace}${NAMESPACE_SEP}${key}`;
        memo[newKey] = obj[key];
        return memo;
    }, {});
}
export default function prefixNamespace(model) {
    if (model.reducers)
        model.reducers = prefix(model.reducers, model.namespace);
    return model;
}
```

3.2.3 constants.tsx

src\dva\constants.tsx

```
export const NAMESPACE_SEP = '/';
```

3.2.4 typings.tsx

src\dva\typings.tsx

```ts
import { Dispatch, Reducer, AnyAction, MiddlewareAPI, StoreEnhancer } from 'redux';
import { History } from 'history';
export interface ReducersMapObject {
    [key: string]: Reducer;
}
export interface ReducerEnhancer {
    (reducer: Reducer): void,
}
export interface EffectsCommandMap {
    put: (action: A) => any,
    call: Function,
    select: Function,
    take: Function,
    cancel: Function,
    [key: string]: any,
}
export type EffectType = 'takeEvery' | 'takeLatest' | 'watcher' | 'throttle';
export type EffectWithType = [Effect, { type: EffectType }];
export type Effect = (action: AnyAction, effects: EffectsCommandMap) => void;
export type ReducersMapObjectWithEnhancer = [ReducersMapObject, ReducerEnhancer];
export interface EffectsMapObject {
    [key: string]: Effect | EffectWithType,
}
export interface SubscriptionsMapObject {
    [key: string]: Subscription,
}
export interface SubscriptionAPI {
    history: History,
    dispatch: Dispatch,
}
export type Subscription = (api: SubscriptionAPI, done: Function) => void;
export interface Model {
    namespace: string,
    state?: any,
    reducers?: ReducersMapObject | ReducersMapObjectWithEnhancer,
    effects?: EffectsMapObject,
    subscriptions?: SubscriptionsMapObject,
}
export interface RouterAPI {
    history: History,
    app: DvaInstance,
}
export interface Router {
    (api?: RouterAPI): JSX.Element | Object,
}
export interface onActionFunc {
    (api: MiddlewareAPI): void,
}
export interface Hooks {
    onError?: (e: Error, dispatch: Dispatch) => void,
    onAction?: onActionFunc | onActionFunc[],
    onStateChange?: () => void,
    onReducer?: ReducerEnhancer,
    onEffect?: () => void,
    onHmr?: () => void,
    extraReducers?: ReducersMapObject,
    extraEnhancers?: StoreEnhancer[],
}
export interface DvaInstance extends Record {
    //use: (hooks: Hooks) => void,
    model: (model: Model) => void,
    //unmodel: (namespace: string) => void,
    router: (router: Router) => void,
    start: (selector?: HTMLElement | string) => any,
}
```

4. 支持effects

4.1 使用

4.1.1 index.tsx

```
import React from 'react';
import dva, { connect } from './dva';
const app = dva();
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
+    effects: {
+        *asyncAdd(action, { call, put }) {
+            yield call(delay, 1000);
+            yield put({ type: 'counter/add' });
+        }
+    }
});
function Counter(props) {
    return (

            {props.number}
             props.dispatch({ type: "counter/add" })}>加1
+             props.dispatch({ type: "counter/asyncAdd" })}>异步+

    )
}
const ConnectedCounter = connect(
    (state) => state.counter
)(Counter);
app.router(() => );
app.start('#root');

+function delay(ms) {
+    return new Promise((resolve) => {
+        setTimeout(function () {
+            resolve();
+        }, ms);
+    });
+}
```

4.2 实现

4.2.1 dva\index.tsx

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
+import { createStore, combineReducers, applyMiddleware } from 'redux';
+import createSagaMiddleware from 'redux-saga';
+import * as sagaEffects from 'redux-saga/effects';
+import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
export { connect };

export default function () {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = {};
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }
    function router(router: Router) {
        app._router = router;
    }

    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
+        const sagas = getSagas(app);
+        const sagaMiddleware = createSagaMiddleware();
+        let store = createStore(rootReducer, applyMiddleware(sagaMiddleware));
+        sagas.forEach(saga => sagaMiddleware.run(saga));
+        ReactDOM.render({app._router()}, document.querySelector(root));
        function createReducer() {
           return combineReducers(initialReducers);
        }
    }
+    function getSagas(app) {
+        let sagas: Array = [];
+        for (const model of app._models) {
+            sagas.push(getSaga(model.effects, model));
+        }
+        return sagas;
+    }

    return app;
}

+function getSaga(effects, model) {
+    return function* () {
+        for (const key in effects) {
+            const watcher = getWatcher(key, model.effects[key], model);
+            yield sagaEffects.fork(watcher);
+        }
+    };
+}
+function getWatcher(key, effect, model) {
+    return function* () {
+        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
+            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
+        });
+    };
+}
+function prefixType(type, model) {
+    if (type.indexOf('/') === -1) {
+        return `${model.namespace}${NAMESPACE_SEP}${type}`;
+    }
+    return type;
+}
function getReducer(model) {
    let { reducers, state: defaultState } = model;
    let reducer = (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
    return reducer;
}
```

4.2.2 prefixNamespace.tsx

src\dva\prefixNamespace.tsx

```
import { NAMESPACE_SEP } from './constants';
function prefix(obj, namespace) {
    return Object.keys(obj).reduce((memo, key) => {
        const newKey = `${namespace}${NAMESPACE_SEP}${key}`;
        memo[newKey] = obj[key];
        return memo;
    }, {});
}
export default function prefixNamespace(model) {
    if (model.reducers)
        model.reducers = prefix(model.reducers, model.namespace);
+    if (model.effects) {
+        model.effects = prefix(model.effects, model.namespace);
+    }
    return model;
}
```

5. 支持路由

5.1 使用

5.1.1 index.tsx

```
import React from 'react';
import dva, { connect } from './dva';
+import { Router, Route } from './dva/router';
const app = dva();
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        }
    }
});
function Counter(props) {
    return (

            {props.number}
             props.dispatch({ type: "counter/add" })}>加1
             props.dispatch({ type: "counter/asyncAdd" })}>异步+

    )
}
const ConnectedCounter = connect(
    (state) => state.counter
)(Counter);
const Home = () => Home
+app.router((api: any) => (
+
+        <>
+
+
+        </>
+
+));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

5.2 实现

5.2.1 dva\index.tsx

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
+import { createHashHistory } from 'history';
+let history = createHashHistory();
export { connect };

export default function () {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = {};
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }
    function router(router: Router) {
        app._router = router;
    }

    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        let store = createStore(rootReducer, applyMiddleware(sagaMiddleware));
        sagas.forEach(saga => sagaMiddleware.run(saga));
+        ReactDOM.render({app._router({ history })}, document.querySelector(root))
        function createReducer() {
           return combineReducers(initialReducers);
        }
    }
    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model));
        }
        return sagas;
    }

    return app;
}

function getSaga(effects, model) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
    let { reducers, state: defaultState } = model;
    let reducer = (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
    return reducer;
}
export * from './typings';
```

5.2.2 router.tsx

src\dva\router.tsx

```
export * from 'react-router-dom';
```

6. 跳转路径

6.1 使用

6.1.1 index.tsx

```
import React from 'react';
import dva, { connect } from './dva';
+import { Router, Route, routerRedux } from './dva/router';
+import { ConnectedRouter } from 'connected-react-router';
const app = dva();
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
+        *goto({ to }, { put }) {
+            yield put(routerRedux.push(to));
+        }
    }
});
function Counter(props) {
    return (

            {props.number}
             props.dispatch({ type: "counter/add" })}>加1
             props.dispatch({ type: "counter/asyncAdd" })}>异步+
+             props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/

    )
}
const ConnectedCounter = connect(
    (state) => state.counter
)(Counter);
const Home = () => Home
app.router((api: any) => (
+
        <>

        </>
+
));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

6.2 实现

6.2.1 dva\index.tsx

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
+import { routerMiddleware, connectRouter, ConnectedRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

export default function () {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
+    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }
    function router(router: Router) {
        app._router = router;
    }

    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
+        let store = createStore(rootReducer, applyMiddleware(routerMiddleware(history), sagaMiddleware));
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root));
        function createReducer() {
           return combineReducers(initialReducers);
        }
    }
    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model));
        }
        return sagas;
    }

    return app;
}

function getSaga(effects, model) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
    let { reducers, state: defaultState } = model;
    let reducer = (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
    return reducer;
}
export * from './typings';
```

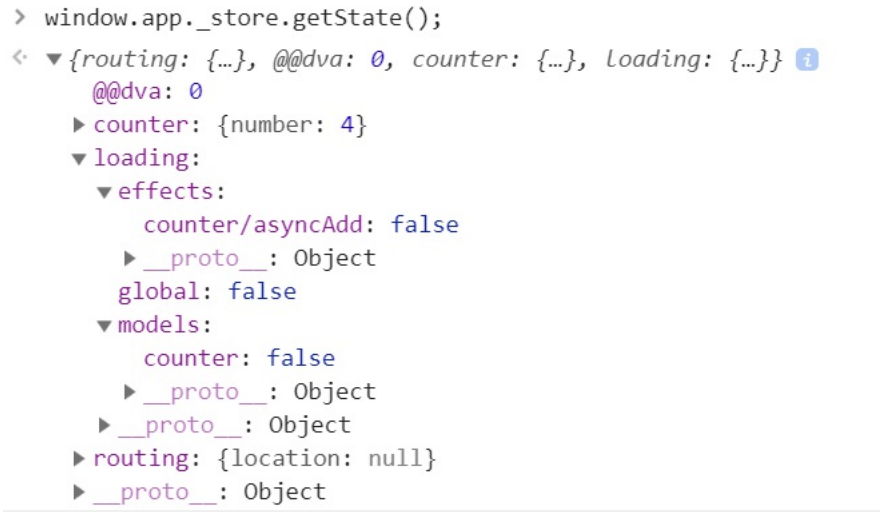
6.2.2 src\dva\router.tsx

src\dva\router.tsx

```
import * as routerRedux from 'connected-react-router';
export * from 'react-router-dom';
export {
    routerRedux
}
```

7. dva-loading

- Auto loading data binding plugin for dva. You don't need to write showLoading and hideLoading any more.
- dva-loading (https://github.com/dvajs/dva/tree/master/packages/dva-loading)



7.1 使用

7.1.1 index.tsx

```
import React from 'react';
import dva, { connect } from './dva';
import { Router, Route, routerRedux } from './dva/router';
import { ConnectedRouter } from 'connected-react-router';
+import createLoading from './dva-loading';
const app = dva();
+app.use(createLoading());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    }
});
function Counter(props) {
    return (

+             {props.loading.models.counter ? 'loading' : props.counter.number}
              props.dispatch({ type: "counter/add" })}>加1
              props.dispatch({ type: "counter/asyncAdd" })}>异步+
              props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/

    )
}
const ConnectedCounter = connect(
+    (state) => state
)(Counter);
const Home = () => Home
app.router((api: any) => (
+
        <>

        </>
+
));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

7.2 实现

7.2.1 dva\index.tsx

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
+import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter, ConnectedRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

+export default function (opts = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }
    function router(router: Router) {
        app._router = router;
    }
+    const plugin = new Plugin();
+    plugin.use(filterHooks(opts));
+    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        let store = createStore(rootReducer, applyMiddleware(routerMiddleware(history), sagaMiddleware));
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root));
        function createReducer() {
+        const extraReducers = plugin.get('extraReducers');
+        return combineReducers({
+            ...initialReducers,
+            ...extraReducers
+        });
       }
    }
    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
+            sagas.push(getSaga(model.effects, model, plugin.get('onEffect')));
        }
        return sagas;
    }

    return app;
}

+function getSaga(effects, model, onEffect) {
    return function* () {
        for (const key in effects) {
+            const watcher = getWatcher(key, model.effects[key], model, onEffect);
            yield sagaEffects.fork(watcher);
        }
    };
}
+function getWatcher(key, effect, model, onEffect) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
+            if (onEffect) {
+                for (const fn of onEffect) {
+                    effect = fn(effect, sagaEffects, model, key);
+                }
+            }
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
    let { reducers, state: defaultState } = model;
    let reducer = (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
    return reducer;
}
export * from './typings';
```

7.2.2 dva-loading.tsx

src\dva-loading.tsx

```
const SHOW = '@@DVA_LOADING/SHOW';
const HIDE = '@@DVA_LOADING/HIDE';
const NAMESPACE = 'loading';
function createLoading(opts: any = {}) {
    const initialState = {
        global: false,
        models: {},
        effects: {},
    };

    const extraReducers = {
        [NAMESPACE](state = initialState, { type, payload }) {
            const { namespace, actionType } = payload || {};
            let ret;
            switch (type) {
                case SHOW:
                    ret = {
                        ...state,
                        global: true,
                        models: { ...state.models, [namespace]: true },
                        effects: { ...state.effects, [actionType]: true },
                    };
                    break;
                case HIDE: {
                    const effects = { ...state.effects, [actionType]: false };
                    const models = {
                        ...state.models,
                        [namespace]: Object.keys(effects).some(actionType => {
                            const _namespace = actionType.split('/')[0];
                            if (_namespace !== namespace) return false;
                            return effects[actionType];
                        }),
                    };
                    const global = Object.keys(models).some(namespace => {
                        return models[namespace];
                    });
                    ret = {
                        ...state,
                        global,
                        models,
                        effects,
                    };
                    break;
                }
                default:
                    ret = state;
                    break;
            }
            return ret;
        },
    };

    function onEffect(effect, { put }, model, actionType) {
        const { namespace } = model;
        return function* (...args) {
            yield put({ type: SHOW, payload: { namespace, actionType } });
            try {
                yield effect(...args);
            } finally {
                yield put({ type: HIDE, payload: { namespace, actionType } });
            }
        };
    }

    return {
        extraReducers,
        onEffect,
    };
}

export default createLoading;
```

7.2.3 plugin.tsx

src\dva\plugin.tsx

```
const hooks = [
    'onEffect',
    'extraReducers'
];
export function filterHooks(obj) {
    return Object.keys(obj).reduce((memo, key) => {
        if (hooks.indexOf(key) > -1) {
            memo[key] = obj[key];
        }
        return memo;
    }, {});
}
export default class Plugin {
    hooks: any
    constructor() {
        this.hooks = hooks.reduce((memo, key) => {
            memo[key] = [];
            return memo;
        }, {});
    }
    use(plugin) {
        const { hooks } = this;
        for (const key in plugin) {
            hooks[key].push(plugin[key]);
        }
    }
    get(key) {
        const { hooks } = this;
        if (key === 'extraReducers') {
            return getExtraReducers(hooks[key]);
        } else {
            return hooks[key];
        }

    }
}
function getExtraReducers(hook) {
    let ret = {};
    for (const reducerObj of hook) {
        ret = { ...ret, ...reducerObj };
    }
    return ret;
}
```

8. dynamic

- dva/dynamic是解决组件动态加载问题的方法。*8.1 使用#****8.1.1 index.tsx #

```
import React from 'react';
import dva, { connect } from './dva';
+import { Router, Route, routerRedux, Link } from './dva/router';
import { ConnectedRouter } from 'connected-react-router';
import createLoading from './dva-loading';
+import dynamic from './dva/dynamic';
const app = dva();
app.use(createLoading());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    }
});
function Counter(props) {
    return (

             {props.loading.models.counter ? 'loading' : props.counter.number}
             props.dispatch({ type: "counter/add" })}>加1
             props.dispatch({ type: "counter/asyncAdd" })}>异步+
             props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/

    )
}
const ConnectedCounter = connect(
    (state) => state
)(Counter);
+const Home = () => Home;
+const UserPageComponent = (dynamic as any)({
+    app,
+    models: () => [
+        import('./models/users'),
+    ],
+    component: () => import('./routes/UserPage'),
+});
app.router((api: any) => (

        <>
+
+               Home
+               counter
+               users
+


+
        </>

));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

** 8.2 实现 #**** 8.2.1 dva\index.tsx #**

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter, ConnectedRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

export default function (opts = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }
```

```
    function router(router: Router) {
        app._router = router;
    }
    const plugin = new Plugin();
    plugin.use(filterHooks(opts));
    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        let store = createStore(rootReducer, applyMiddleware(routerMiddleware(history), sagaMiddleware));
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root))
+        app.model = injectModel.bind(app);
+        app._getSaga = getSaga;
+        function injectModel(m) {
+            m = model(m);
+            initialReducers[m.namespace] = getReducer(m);
+            store.replaceReducer(createReducer());
+            if (m.effects) {
+                sagaMiddleware.run(app._getSaga(m.effects, m));
+            }
+            if (m.subscriptions) {
+                runSubscription(m.subscriptions, m, app);
+            }
+        }
+        function runSubscription(subscriptions, model, app) {
+            for (const key in subscriptions) {
+                subscriptions[key](
+                    {
+                        dispatch: prefixedDispatch(store.dispatch, model),
+                        history: app._history,
+                    }
+                );
+            }
+        }
+        function prefixedDispatch(dispatch, model) {
+            return action => {
+                const { type } = action;
+                return dispatch({ ...action, type: prefixType(type, model) });
+            };
+        }
        function createReducer() {
            const extraReducers = plugin.get('extraReducers');
            return combineReducers({
                ...initialReducers,
                ...extraReducers
            });
        }
+    }

    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model, plugin.get('onEffect')));
        }
        return sagas;
    }

    return app;
}

function getSaga(effects, model, onEffect) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model, onEffect);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model, onEffect) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            if (onEffect) {
                for (const fn of onEffect) {
                    effect = fn(effect, sagaEffects, model, key);
                }
            }
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
    let { reducers = {}, state: defaultState } = model;
+    return (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
}
export * from './typings';
```

** 8.2.2 dynamic.tsx #**

src\dva\dynamic.tsx

```
import React, { Component, JSXElementConstructor } from 'react';
let defaultLoadingComponent = () => <div>loadingdiv>;
export default function dynamic(config) {
    const { app, models, component } = config;
    return class DynamicComponent extends Component<any, any> {
        LoadingComponent: any = null
        constructor(props) {
            super(props);
            this.LoadingComponent = config.LoadingComponent || defaultLoadingComponent;
            let AsyncComponent: JSXElementConstructor | null = null;
            this.state = { AsyncComponent };
        }
        componentDidMount() {
            Promise.all([Promise.all(models()), component()]).then(([models, component]) => {
                let resolveModels = models.map((model) => (model as any).default);
                resolveModels.forEach(model => {
                    app.model(model);
                });
                let AsyncComponent = component.default || component;
                this.setState({ AsyncComponent });
            });
        }

        render() {
            const { AsyncComponent } = this.state;
            const { LoadingComponent } = this;
            if (AsyncComponent)
                return <AsyncComponent {...this.props} />;
            return <LoadingComponent {...this.props} />;
        }
    };
}
```

** 8.2.3 src\models\users.tsx #**

src\models\users.tsx

```
function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
export default {
    namespace: 'users',
    state: { list: [{ id: 1, name: '珠峰' }, { id: 2, name: '架构' }] },
    reducers: {
        add(state, action) {
            return { list: [...state.list, { id: Date.now(), name: action.payload }] };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'add', payload: '学院' });
        }
    }
}
```

** 8.2.4 routes\UserPage.tsx #**

src\routes\UserPage.tsx

```
import React from 'react';
import { connect } from "../dva";

function UserPage(props) {
    let list = props.list || [];
    return (
        <>
            <button onClick={() => props.dispatch({ type: 'users/asyncAdd' })}>+button>
            <ul>
                {
                    list.map(user => (
                        <li key={user.id}>{user.name}li>
                    ))
                }
            ul>
        </>
    )
}
export default connect(state => state.users)(UserPage);
```

9. onAction中间件

```
cnpm i redux-logger -S
```

** 9.1 使用 #**** 9.1.1 index.tsx #**

```
import React from 'react';
import dva, { connect } from './dva';
import { Router, Route, routerRedux, Link } from './dva/router';
import createLoading from './dva-loading';
import dynamic from './dva/dynamic';
+import { createLogger } from './redux-logger';

const app = dva();
+app.use({ onAction: createLogger() });
app.use(createLoading());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    }
});
function Counter(props) {
    return (

             {props.loading.models.counter ? 'loading' : props.counter.number}
             props.dispatch({ type: "counter/add" })}>加1
             props.dispatch({ type: "counter/asyncAdd" })}>异步+
             props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/

    )
}
const ConnectedCounter = connect(
    (state) => state
)(Counter);
const Home = () => Home;
const UserPageComponent = (dynamic as any)({
    app,
    models: () => [
        import('./models/users'),
    ],
    component: () => import('./routes/UserPage'),
});
app.router((api: any) => (

        <>

                Home
                counter
                users

        </>

));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

** 9.2 实现 #**** 9.2.1 dva\index.tsx #**

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

export default function (opts = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }
```

```
    function router(router: Router) {
        app._router = router;
    }
    const plugin = new Plugin();
    plugin.use(filterHooks(opts));
    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
+        const extraMiddlewares = plugin.get('onAction');
+        let store = createStore(rootReducer, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware));
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root))
        app.model = injectModel.bind(app);
        app._getSaga = getSaga;
        function injectModel(m) {
            m = model(m);
            initialReducers[m.namespace] = getReducer(m);
            store.replaceReducer(createReducer());
            if (m.effects) {
                sagaMiddleware.run(app._getSaga(m.effects, m));
            }
            if (m.subscriptions) {
                runSubscription(m.subscriptions, m, app);
            }
        }
        function runSubscription(subscriptions, model, app) {
            for (const key in subscriptions) {
                subscriptions[key](
                    {
                        dispatch: prefixedDispatch(store.dispatch, model),
                        history: app._history,
                    }
                );
            }
        }
        function prefixedDispatch(dispatch, model) {
            return action => {
                const { type } = action;
                return dispatch({ ...action, type: prefixType(type, model) });
            };
        }
        function createReducer() {
            const extraReducers = plugin.get('extraReducers');
            return combineReducers({
                ...initialReducers,
                ...extraReducers
            });
        }
    }

    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model, plugin.get('onEffect')));
        }
        return sagas;
    }

    return app;
}

function getSaga(effects, model, onEffect) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model, onEffect);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model, onEffect) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            if (onEffect) {
                for (const fn of onEffect) {
                    effect = fn(effect, sagaEffects, model, key);
                }
            }
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
    let { reducers = {}, state: defaultState } = model;
    return (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
}
export * from './typings';
```

** 9.2.2 src\dva\plugin.tsx #**

src\dva\plugin.tsx

```
const hooks = [
    'onEffect',
    'extraReducers',
+    'onAction'
];
export function filterHooks(obj) {
    return Object.keys(obj).reduce((memo, key) => {
        if (hooks.indexOf(key) > -1) {
            memo[key] = obj[key];
        }
        return memo;
    }, {});
}
export default class Plugin {
    hooks: any
    constructor() {
        this.hooks = hooks.reduce((memo, key) => {
            memo[key] = [];
            return memo;
        }, {});
    }
    use(plugin) {
        const { hooks } = this;
        for (const key in plugin) {
            hooks[key].push(plugin[key]);
        }
    }
    get(key) {
        const { hooks } = this;
        if (key
            return getExtraReducers(hooks[key]);
        } else {
            return hooks[key];
        }

    }
}
function getExtraReducers(hook) {
    let ret = {};
    for (const reducerObj of hook) {
        ret = { ...ret, ...reducerObj };
    }
    return ret;
}
```

** 9.2.3 src\redux-logger.tsx #**

src\redux-logger.tsx

```

const colors = {
    title: 'inherit',
    prevState: '#9E9E9E',
    action: '#03A9F4',
    nextState: '#4CAF50'
}
const logger = ({ dispatch, getState }) => next => (action) => {
    let prevState = getState();
    let startedTime = Date.now();
    let returnedValue = next(action);
    let took = Date.now() - startedTime;
    let nextState = getState();
    const headerCSS: any[] = [];
    headerCSS.push('color: gray; font-weight: lighter;');
    headerCSS.push(`color: ${colors.title};`);
    headerCSS.push('color: gray; font-weight: lighter;');
    headerCSS.push('color: gray; font-weight: lighter;');
    const title = titleFormatter(action, startedTime, took);

    console.groupCollapsed(`%c ${title}`, ...headerCSS);
    console.log('%c prev state', `color: ${colors.prevState}; font-weight: bold`, prevState);
    console.log('%c action    ', `color: ${colors.action}; font-weight: bold`, action);
    console.log('%c next state', `color: ${colors.nextState}; font-weight: bold`, nextState);
    console.groupEnd();
    return returnedValue;
}
function titleFormatter(action, time, took) {
    const parts = ['action'];
    parts.push(`%c${String(action.type)}`);
    parts.push(`%c@ ${new Date(time).toLocaleTimeString()}`);
    parts.push(`%c(in ${took.toFixed(2)} ms)`);
    return parts.join(' ');
}

function createLogger() {
    return logger;
}
export {
    createLogger
}
```

10. stateChange

** 10.1 使用 #**** 10.1.1 index.tsx #**

```
import React from 'react';
import dva, { connect } from './dva';
import { Router, Route, routerRedux, Link } from './dva/router';
import createLoading from './dva-loading';
import dynamic from './dva/dynamic';
import { createLogger } from './redux-logger';

const app = dva({
+    initialState: localStorage.getItem('state') ? JSON.parse(localStorage.getItem('state')!) : {},
+    onStateChange: function () {
+        localStorage.setItem('state', JSON.stringify(arguments[0]));
+    }
});
app.use({ onAction: createLogger() });
app.use(createLoading());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    }
});
function Counter(props) {
    return (

             {props.loading.models.counter ? 'loading' : props.counter.number}
             props.dispatch({ type: "counter/add" })}>加1
             props.dispatch({ type: "counter/asyncAdd" })}>异步+
             props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/

    )
}
const ConnectedCounter = connect(
    (state) => state
)(Counter);
const Home = () => Home;
const UserPageComponent = (dynamic as any)({
    app,
    models: () => [
        import('./models/users'),
    ],
    component: () => import('./routes/UserPage'),
});
app.router((api: any) => (

        <>

                Home
                counter
                users

        </>

));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

** 10.2 实现 #**** 10.2.1 dva\index.tsx #**

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

export default function (opts: any = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
```

```
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }

    function router(router: Router) {
        app._router = router;
    }
    const plugin = new Plugin();
    plugin.use(filterHooks(opts));
    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        const extraMiddlewares = plugin.get('onAction');
+        let store = createStore(rootReducer, opts.initialState, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware));
+        const listeners = plugin.get('onStateChange');
+        for (const listener of listeners) {
+            store.subscribe(() => {
+                listener(store.getState());
+            });
+        }
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root))
        app.model = injectModel.bind(app);
        app._getSaga = getSaga;
        function injectModel(m) {
            m = model(m);
            initialReducers[m.namespace] = getReducer(m);
            store.replaceReducer(createReducer());
            if (m.effects) {
                sagaMiddleware.run(app._getSaga(m.effects, m));
            }
            if (m.subscriptions) {
                runSubscription(m.subscriptions, m, app);
            }
        }
        function runSubscription(subscriptions, model, app) {
            for (const key in subscriptions) {
                subscriptions[key](
                    {
                        dispatch: prefixedDispatch(store.dispatch, model),
                        history: app._history,
                    }
                );
            }
        }
        function prefixedDispatch(dispatch, model) {
            return action => {
                const { type } = action;
                return dispatch({ ...action, type: prefixType(type, model) });
            };
        }
        function createReducer() {
          const extraReducers = plugin.get('extraReducers');
          return combineReducers({
            ...initialReducers,
            ...extraReducers
          });
       }
    }

    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model, plugin.get('onEffect')));
        }
        return sagas;
    }

    return app;
}

function getSaga(effects, model, onEffect) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model, onEffect);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model, onEffect) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            if (onEffect) {
                for (const fn of onEffect) {
                    effect = fn(effect, sagaEffects, model, key);
                }
            }
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
```

```
    let { reducers = {}, state: defaultState } = model;
    return (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
}
export * from './typings';
```

** 10.2.2 src\dva\plugin.tsx #**

src\dva\plugin.tsx

```
const hooks = [
    'onEffect',
    'extraReducers',
    'onAction',
+    'onStateChange'
];
export function filterHooks(obj) {
    return Object.keys(obj).reduce((memo, key) => {
        if (hooks.indexOf(key) > -1) {
            memo[key] = obj[key];
        }
        return memo;
    }, {});
}
export default class Plugin {
    hooks: any
    constructor() {
        this.hooks = hooks.reduce((memo, key) => {
            memo[key] = [];
            return memo;
        }, {});
    }
    use(plugin) {
        const { hooks } = this;
        for (const key in plugin) {
            hooks[key].push(plugin[key]);
        }
    }
    get(key) {
        const { hooks } = this;
        if (key
            return getExtraReducers(hooks[key]);
        } else {
            return hooks[key];
        }

    }
}
function getExtraReducers(hook) {
    let ret = {};
    for (const reducerObj of hook) {
        ret = { ...ret, ...reducerObj };
    }
    return ret;
}
```

11. onReducer

- redux-undo (https://github.com/omnidan/redux-undo)利用高阶reducer来增强reducer的例子,它主要作用是使任意reducer变成可以执行撤销和重做的全新reducer

** 11.1 使用 #**** 11.1.1 index.tsx #**

```
import React from 'react';
import dva, { connect } from './dva';
import { Router, Route, routerRedux, Link } from './dva/router';
import createLoading from './dva-loading';
import dynamic from './dva/dynamic';
import { createLogger } from './redux-logger';
+import undoable, { ActionCreators } from './redux-undo';

const app = dva({
    initialState: localStorage.getItem('state') ? JSON.parse(localStorage.getItem('state')!) : undefined,
    onStateChange: function () {
        localStorage.setItem('state', JSON.stringify(arguments[0]));
    },
+    onReducer: reducer => {
+        let undoReducer = undoable(reducer);
+        return (state, action) => {
+            let newState: any = undoReducer(state, action);
+            return { ...newState, router: newState.present.router };
+        }
+    }
});
app.use({ onAction: createLogger() });
app.use(createLoading());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    }
});
function Counter(props) {
    return (

             {props.loading.models.counter ? 'loading' : props.counter.number}
             props.dispatch({ type: "counter/add" })}>加1
             props.dispatch({ type: "counter/asyncAdd" })}>异步+
             props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/
+             props.dispatch(ActionCreators.undo())}>undo
+             props.dispatch(ActionCreators.redo())}>redo

    )
}
const ConnectedCounter = connect(
+    (state) => state.present
)(Counter);
const Home = () => Home;
const UserPageComponent = (dynamic as any)({
    app,
    models: () => [
        import('./models/users'),
    ],
    component: () => import('./routes/UserPage'),
});
app.router((api: any) => (

        <>

                Home
                counter
                users

        </>

));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

** 11.2 实现 #**** 11.2.1 dva\index.tsx #**

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };
```

```
export default function (opts: any = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }

    function router(router: Router) {
        app._router = router;
    }
    const plugin = new Plugin();
    plugin.use(filterHooks(opts));
    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
+        const reducerEnhancer = plugin.get('onReducer');
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        const extraMiddlewares = plugin.get('onAction');
        let store = createStore(rootReducer, opts.initialState, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware));
        const listeners = plugin.get('onStateChange');
        for (const listener of listeners) {
            store.subscribe(() => {
                listener(store.getState());
            });
        }
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root))
        app.model = injectModel.bind(app);
        app._getSaga = getSaga;
        function injectModel(m) {
            m = model(m);
            initialReducers[m.namespace] = getReducer(m);
            store.replaceReducer(createReducer());
            if (m.effects) {
                sagaMiddleware.run(app._getSaga(m.effects, m));
            }
            if (m.subscriptions) {
                runSubscription(m.subscriptions, m, app);
            }
        }
        function runSubscription(subscriptions, model, app) {
            for (const key in subscriptions) {
                subscriptions[key](
                    {
                        dispatch: prefixedDispatch(store.dispatch, model),
                        history: app._history,
                    }
                );
            }
        }
        function prefixedDispatch(dispatch, model) {
            return action => {
                const { type } = action;
                return dispatch({ ...action, type: prefixType(type, model) });
            };
        }
        function createReducer() {
            const extraReducers = plugin.get('extraReducers');
+            return reducerEnhancer(combineReducers({
                ...initialReducers,
                ...extraReducers
            }));
        }
    }

    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model, plugin.get('onEffect')));
        }
        return sagas;
    }

    return app;
}

function getSaga(effects, model, onEffect) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model, onEffect);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model, onEffect) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            if (onEffect) {
                for (const fn of onEffect) {
                    effect = fn(effect, sagaEffects, model, key);
                }
            }
```

```
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
    let { reducers = {}, state: defaultState } = model;
    return (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
}
export * from './typings';
```

** 11.2.2 src\dva\plugin.tsx #**

src\dva\plugin.tsx

```
const hooks = [
    'onEffect',
    'extraReducers',
    'onAction',
    'onStateChange',
+    'onReducer'
];
export function filterHooks(obj) {
    return Object.keys(obj).reduce((memo, key) => {
        if (hooks.indexOf(key) > -1) {
            memo[key] = obj[key];
        }
        return memo;
    }, {});
}
export default class Plugin {
    hooks: any
    constructor() {
        this.hooks = hooks.reduce((memo, key) => {
            memo[key] = [];
            return memo;
        }, {});
    }
    use(plugin) {
        const { hooks } = this;
        for (const key in plugin) {
            hooks[key].push(plugin[key]);
        }
    }
    get(key) {
        const { hooks } = this;
        if (key
            return getExtraReducers(hooks[key]);
+        } else if (key === 'onReducer') {
+            return getOnReducer(hooks[key]);
+        } else {
            return hooks[key];
        }

    }
}
+function getOnReducer(hook) {
+    return function (reducer) {
+        for (const reducerEnhancer of hook) {
+            reducer = reducerEnhancer(reducer);
+        }
+        return reducer;
+    };
+}
function getExtraReducers(hook) {
    let ret = {};
    for (const reducerObj of hook) {
        ret = { ...ret, ...reducerObj };
    }
    return ret;
}
```

** 11.2.3 src\routes\UserPage.tsx #**

src\routes\UserPage.tsx

```
import React from 'react';
import { connect } from "../dva";

function UserPage(props) {
    let list = props.list || [];
    return (
        <>
             props.dispatch({ type: 'users/asyncAdd' })}>+

                {
                    list.map(user => (
                        {user.name}
                    ))
                }

        </>
    )
}
+export default connect(state => state.present.users)(UserPage);
```

** 11.2.4 src\redux-undo.tsx #**

src\redux-undo.tsx

```
const UNDO_TYPE = "@@redux-unto/UNDO";
const REDO_TYPE = "@@redux-unto/REDO";

export const ActionCreators = {
    undo() {
        return { type: UNDO_TYPE }
    },
    redo() {
        return { type: REDO_TYPE };
    }
}
export default function undoable(reducer) {
    let undoType = UNDO_TYPE, redoType = REDO_TYPE;
    const initialState = {
        past: [],
        future: [],
        present: reducer(undefined, {})
    }
    return function (state = initialState, action) {
        const { past, present, future } = state;
        switch (action.type) {
            case undoType:
                const previous = past[past.length - 1];
                const newPast = past.slice(0, past.length - 1);
                return {
                    past: newPast,
                    present: previous,
                    future: [present, ...future]
                }
            case redoType:
                const next = future[0];
                const newFuture = future.slice(1);
                return {
                    past: [...past, present],
                    present: next,
                    future: newFuture
                }
            default:
                const newPresent = reducer(present, action);
                return {
                    past: [...past, present],
                    present: newPresent,
                    future: []
                }
        }
    }
}
```

12. extraEnhancers

- redux-persist (https://github.com/rt2zz/redux-persist)会将redux的store中的数据缓存到浏览器的localStorage中

** 12.1 使用 #**** 12.1.1 index.tsx #**

```
import React from 'react';
import dva, { connect } from './dva';
import { Router, Route, routerRedux, Link } from './dva/router';
import createLoading from './dva-loading';
import dynamic from './dva/dynamic';
import { createLogger } from './redux-logger';
import undoable, { ActionCreators } from './redux-undo';
+import { persistStore, persistReducer } from './redux-persist';
+import storage from './redux-persist/lib/storage';
+import { PersistGate } from './redux-persist/lib/integration/react';
+const persistConfig = {
+    key: 'root',
+    storage
+}
const app = dva({
+    //initialState: localStorage.getItem('state') ? JSON.parse(localStorage.getItem('state')!) : undefined,
    onStateChange: function () {
+       //localStorage.setItem('state', JSON.stringify(arguments[0]));
+       console.log('currentState', arguments[0]);
    },
    onReducer: reducer => {
        let undoReducer = undoable(reducer);
+       let rootReducer = persistReducer(persistConfig, undoReducer);
       return (state, action) => {
+           let newState: any = rootReducer(state, action);
           return { ...newState, router: newState.present.router };
       }
   },
+   extraEnhancers: [
+       createStore => (...args) => {
+           const store: any = createStore(...args);
+           const persistor = persistStore(store);
+           store.persistor = persistor;
+           return store;
+       }
+   ]
});
app.use({ onAction: createLogger() });
app.use(createLoading());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    }
});
function Counter(props) {
    return (

            {props.loading.models.counter ? 'loading' : props.counter.number}
            props.dispatch({ type: "counter/add" })}>加1
            props.dispatch({ type: "counter/asyncAdd" })}>异步加1
            props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/
            props.dispatch(ActionCreators.undo())}>undo
            props.dispatch(ActionCreators.redo())}>redo

    )
}
const ConnectedCounter = connect(
    (state) => state.present
)(Counter);
const Home = () => Home;
const UserPageComponent = (dynamic as any)({
    app,
    models: () => [
        import('./models/users'),
    ],
    component: () => import('./routes/UserPage'),
});
app.router((api: any) => (

           <>

                   Home
                   counter
                   users

           </>

));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

** 12.2 实现 #**** 12.2.1 dva\index.tsx #**

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
+import { createStore, combineReducers, applyMiddleware, compose } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

export default function (opts: any = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }

    function router(router: Router) {
        app._router = router;
    }
    const plugin = new Plugin();
    plugin.use(filterHooks(opts));
    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model);
        }
        const reducerEnhancer = plugin.get('onReducer');
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        const extraMiddlewares = plugin.get('onAction');
+        const extraEnhancers = plugin.get('extraEnhancers');
+        const enhancers = [...extraEnhancers, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware)];
+        //let store = createStore(rootReducer, opts.initialState, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware));
+        let store = createStore(rootReducer, opts.initialState, compose(...enhancers));
+        app._store = store;
        const listeners = plugin.get('onStateChange');
        for (const listener of listeners) {
            store.subscribe(() => {
                listener(store.getState());
            });
        }
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root))
        app.model = injectModel.bind(app);
        app._getSaga = getSaga;
        function injectModel(m) {
            m = model(m);
            initialReducers[m.namespace] = getReducer(m);
            store.replaceReducer(createReducer());
            if (m.effects) {
                sagaMiddleware.run(app._getSaga(m.effects, m));
            }
            if (m.subscriptions) {
                runSubscription(m.subscriptions, m, app);
            }
        }
        function runSubscription(subscriptions, model, app) {
            for (const key in subscriptions) {
                subscriptions[key](
                    {
                        dispatch: prefixedDispatch(store.dispatch, model),
                        history: app._history,
                    }
                );
            }
        }
        function prefixedDispatch(dispatch, model) {
            return action => {
                const { type } = action;
                return dispatch({ ...action, type: prefixType(type, model) });
            };
        }
        function createReducer() {
            const extraReducers = plugin.get('extraReducers');
            return reducerEnhancer(combineReducers({
                ...initialReducers,
                ...extraReducers
            }));
        }
    }

    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model, plugin.get('onEffect')));
        }
        return sagas;
    }
```

```
    return app;
}

function getSaga(effects, model, onEffect) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model, onEffect);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model, onEffect) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            if (onEffect) {
                for (const fn of onEffect) {
                    effect = fn(effect, sagaEffects, model, key);
                }
            }
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model) {
    let { reducers = {}, state: defaultState } = model;
    return (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
}
export * from './typings';
```

** 12.2.2 plugin.tsx #**

src\dva\plugin.tsx

```
const hooks = [
    'onEffect',
    'extraReducers',
    'onAction',
    'onStateChange',
    'onReducer',
+    "extraEnhancers"
];
export function filterHooks(obj) {
    return Object.keys(obj).reduce((memo, key) => {
        if (hooks.indexOf(key) > -1) {
            memo[key] = obj[key];
        }
        return memo;
    }, {});
}
export default class Plugin {
    hooks: any
    constructor() {
        this.hooks = hooks.reduce((memo, key) => {
            memo[key] = [];
            return memo;
        }, {});
    }
    use(plugin) {
        const { hooks } = this;
        for (const key in plugin) {
+            if (key === 'extraEnhancers') {
+                hooks[key] = plugin[key];
+            } else {
+                hooks[key].push(plugin[key]);
+            }
        }
    }
    get(key) {
        const { hooks } = this;
        if (key
            return getExtraReducers(hooks[key]);
        } else if (key
            return getOnReducer(hooks[key]);
        } else {
            return hooks[key];
        }

    }
}
function getOnReducer(hook) {
    return function (reducer) {
        for (const reducerEnhancer of hook) {
            reducer = reducerEnhancer(reducer);
        }
        return reducer;
    };
}
function getExtraReducers(hook) {
    let ret = {};
    for (const reducerObj of hook) {
        ret = { ...ret, ...reducerObj };
    }
    return ret;
}
```

** 12.2.3 redux-persist\index.tsx #**

src\redux-persist\index.tsx

```
import persistReducer from './persistReducer';
import persistStore from './persistStore';

export {
    persistReducer,
    persistStore
}
export const REHYDRATE = 'REHYDRATE';
```

** 12.2.4 persistReducer.tsx #**

src\redux-persist\persistReducer.tsx

```
export default function (persistConfig, reducer) {
    let isInited = false;
    return (state, action) => {
        switch (action.type) {
            case 'PERSIST_INIT':
                isInited = true;
                let value = persistConfig.storage.getItem('persist:' + persistConfig.key);
                state = value ? JSON.parse(value) : undefined;
                return reducer(state, action);
            default:
                if (isInited) {
                    state = reducer(state, action);
                    persistConfig.storage.setItem('persist:' + persistConfig.key, JSON.stringify(state));
                    return state;
                }
                return reducer(state, action);
        }

    }
}
```

** 12.2.5 persistStore.tsx #**

src\redux-persist\persistStore.tsx

```
export default function (store) {
    let persistor = {
        ...store,
        initState() {
            persistor.dispatch({
                type: 'PERSIST_INIT',
            })
        }
    };
    return persistor;
}
```

** 12.2.6 redux-persist\lib\storage.tsx #**

src\redux-persist\lib\storage.tsx

```
let storage = {
    setItem(key, val) {
        localStorage.setItem(key, val);
    },
    getItem(key) {
        return localStorage.getItem(key);
    }
}
export default storage;
```

** 12.2.7 redux-persist\lib\integration\react.tsx #**

src\redux-persist\lib\integration\react.tsx

```
import React, { Component } from 'react';

class PersistGate extends Component<any> {
    componentDidMount() {
        this.props.persistor.initState();
    }
    render() {
        return this.props.children;
    }
}

export { PersistGate }
```

13. dva-immer

- immer (https://immerjs.github.io/immer/docs/introduction)是 mobx的作者写的一个 immutable 库，核心实现是利用 ES6 的 proxy，几乎以最小的成本实现了 js 的不可变数据结构
- dva-immer (https://github.com/dvajs/dva/tree/master/packages/dva-immer)
    - currentState 被操作对象的最初状态
    - draftState 根据 currentState 生成的草稿状态，它是 currentState 的代理，对 draftState 所做的任何修改都将被记录并用于生成 nextState 。在此过程中，currentState 将不受影响
    - nextState 根据 draftState 生成的最终状态
    - produce 生产 用来生成 nextState的函数

```
cnpm i dva-immer -S
```

```
import produce from "immer"
const baseState = [
    {
        todo: "Learn typescript",
        done: true
    },
    {
        todo: "Try immer",
        done: false
    }
]
const nextState = produce(baseState, draftState => {
    draftState.push({todo: "Tweet about it"})
    draftState[1].done = true
})
```

** 13.1 使用 #**** 13.1.1 index.tsx #**

```
import React from 'react';
import dva, { connect } from './dva';
import { Router, Route, routerRedux, Link } from './dva/router';
import createLoading from './dva-loading';
import dynamic from './dva/dynamic';
import { createLogger } from './redux-logger';
import undoable, { ActionCreators } from './redux-undo';
import { persistStore, persistReducer } from './redux-persist';
import storage from './redux-persist/lib/storage';
import { PersistGate } from './redux-persist/lib/integration/react';
+import immer from './dva-immer';
const persistConfig = {
    key: 'root',
    storage
}
const app = dva({
    //initialState: localStorage.getItem('state') ? JSON.parse(localStorage.getItem('state')!) : undefined,
    onStateChange: function () {
        //localStorage.setItem('state', JSON.stringify(arguments[0]));
        console.log('currentState', arguments[0]);
    },
    onReducer: reducer => {
        let undoReducer = undoable(reducer);
        let rootReducer = persistReducer(persistConfig, undoReducer);
        return (state, action) => {
            let newState: any = rootReducer(state, action);
+            return { ...newState, router: newState.present ? newState.present.router : null };
        }
    },
    extraEnhancers: [
        createStore => (...args) => {
            const store: any = createStore(...args);
            const persistor = persistStore(store);
            store.persistor = persistor;
            return store;
        }
    ]
});
app.use({ onAction: createLogger() });
app.use(createLoading());
+app.use(immer());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    }
});
function Counter(props) {
    return (

             {props.loading.models.counter ? 'loading' : props.counter.number}
             props.dispatch({ type: "counter/add" })}>加1
             props.dispatch({ type: "counter/asyncAdd" })}>异步加1
             props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/
             props.dispatch(ActionCreators.undo())}>undo
             props.dispatch(ActionCreators.redo())}>redo

    )
}
const ConnectedCounter = connect(
    (state) => state.present
)(Counter);
const Home = () => Home;
const UserPageComponent = (dynamic as any)({
    app,
    models: () => [
        import('./models/users'),
    ],
    component: () => import('./routes/UserPage'),
});
app.router((api: any) => (

            <>

                    Home
                    counter
                    users

            </>

));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

** 13.2 实现 #**** 13.2.1 dva\index.tsx #**

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware, compose } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

export default function (opts: any = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }

    function router(router: Router) {
        app._router = router;
    }
    const plugin = new Plugin();
    plugin.use(filterHooks(opts));
    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
+           initialReducers[model.namespace] = getReducer(model, plugin._handleActions);
        }
        const reducerEnhancer = plugin.get('onReducer');
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        const extraMiddlewares = plugin.get('onAction');
        const extraEnhancers = plugin.get('extraEnhancers');
        const enhancers = [...extraEnhancers, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware)];
        //let store = createStore(rootReducer, opts.initialState, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware));
        let store = createStore(rootReducer, opts.initialState, compose(...enhancers));
        app._store = store;
        const listeners = plugin.get('onStateChange');
        for (const listener of listeners) {
            store.subscribe(() => {
                listener(store.getState());
            });
        }
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root))
        app.model = injectModel.bind(app);
        app._getSaga = getSaga;
        function injectModel(m) {
            m = model(m);
+           initialReducers[m.namespace] = getReducer(m, plugin._handleActions);
            store.replaceReducer(createReducer());
            if (m.effects) {
                sagaMiddleware.run(app._getSaga(m.effects, m));
            }
            if (m.subscriptions) {
                runSubscription(m.subscriptions, m, app);
            }
        }
        function runSubscription(subscriptions, model, app) {
            for (const key in subscriptions) {
                subscriptions[key](
                    {
                        dispatch: prefixedDispatch(store.dispatch, model),
                        history: app._history,
                    }
                );
            }
        }
        function prefixedDispatch(dispatch, model) {
            return action => {
                const { type } = action;
                return dispatch({ ...action, type: prefixType(type, model) });
            };
        }
        function createReducer() {
            const extraReducers = plugin.get('extraReducers');
            return reducerEnhancer(combineReducers({
                ...initialReducers,
                ...extraReducers
            }));
        }
    }

    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
            sagas.push(getSaga(model.effects, model, plugin.get('onEffect')));
        }
        return sagas;
```

```
    }

    return app;
}

function getSaga(effects, model, onEffect) {
    return function* () {
        for (const key in effects) {
            const watcher = getWatcher(key, model.effects[key], model, onEffect);
            yield sagaEffects.fork(watcher);
        }
    };
}
function getWatcher(key, effect, model, onEffect) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            if (onEffect) {
                for (const fn of onEffect) {
                    effect = fn(effect, sagaEffects, model, key);
                }
            }
            yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model, handleActions) {
    let { reducers = {}, state: defaultState } = model;
+    let reducer = (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
+    if (handleActions) {
+        return handleActions(reducers || {}, defaultState);
+    }
+    return reducer;
}

export * from './typings';
```

** 13.2.2 dva\plugin.tsx #**

src\dva\plugin.tsx

```
const hooks = [
    'onEffect',
    'extraReducers',
    'onAction',
    'onStateChange',
    'onReducer',
    "extraEnhancers",
+    '_handleActions'
];
export function filterHooks(obj) {
    return Object.keys(obj).reduce((memo, key) => {
        if (hooks.indexOf(key) > -1) {
            memo[key] = obj[key];
        }
        return memo;
    }, {});
}
export default class Plugin {
    hooks: any
+    _handleActions: any = null
    constructor() {
        this.hooks = hooks.reduce((memo, key) => {
            memo[key] = [];
            return memo;
        }, {});
    }
    use(plugin) {
        const { hooks } = this;
        for (const key in plugin) {
+            if (key === '_handleActions') {
+                this._handleActions = plugin[key];
            } else if (key
                hooks[key] = plugin[key];
            } else {
                hooks[key].push(plugin[key]);
            }
        }
    }
    get(key) {
        const { hooks } = this;
        if (key
            return getExtraReducers(hooks[key]);
        } else if (key
            return getOnReducer(hooks[key]);
        } else {
            return hooks[key];
        }

    }
}
function getOnReducer(hook) {
    return function (reducer) {
        for (const reducerEnhancer of hook) {
            reducer = reducerEnhancer(reducer);
        }
        return reducer;
    };
}
function getExtraReducers(hook) {
    let ret = {};
    for (const reducerObj of hook) {
        ret = { ...ret, ...reducerObj };
    }
    return ret;
}
```

** 13.2.3 dva-immer.tsx #**

src\dva-immer.tsx

```
import produce from 'immer';
export default function () {
    return {
        _handleActions(handlers, defaultState) {
            return (state = defaultState, action) => {
                const { type } = action;
                const ret = produce(state, draft => {
                    const handler = handlers[type];
                    if (handler) {
                        return handler(draft, action);
                    }
                });
                return ret || {};
            };
        },
    };
}
```

14. onError

** 14.1 使用 #**** 14.1.1 index.tsx #**

```
import React from 'react';
import dva, { connect } from './dva';
import { Router, Route, routerRedux, Link } from './dva/router';
import createLoading from './dva-loading';
import dynamic from './dva/dynamic';
import { createLogger } from './redux-logger';
import undoable, { ActionCreators } from './redux-undo';
import { persistStore, persistReducer } from './redux-persist';
import storage from './redux-persist/lib/storage';
import { PersistGate } from './redux-persist/lib/integration/react';
import immer from './dva-immer';
const persistConfig = {
```

```
    key: 'root',
    storage
}
const app = dva({
    //initialState: localStorage.getItem('state') ? JSON.parse(localStorage.getItem('state')!) : undefined,
    onStateChange: function () {
        //localStorage.setItem('state', JSON.stringify(arguments[0]));
        console.log('currentState', arguments[0]);
    },
    onReducer: reducer => {
        let undoReducer = undoable(reducer);
        let rootReducer = persistReducer(persistConfig, undoReducer);
        return (state, action) => {
            let newState: any = rootReducer(state, action);
            return { ...newState, router: newState.present ? newState.present.router : null };
        }
    },
    extraEnhancers: [
        createStore => (...args) => {
            const store: any = createStore(...args);
            const persistor = persistStore(store);
            store.persistor = persistor;
            return store;
        }
    ],
+    onError(err, dispatch) {
+        console.error('onError', err);
+    }
});
app.use({ onAction: createLogger() });
app.use(createLoading());
app.use(immer());
app.model({
    namespace: 'counter',
    state: { number: 0 },
    reducers: {
        add(state) {
            return { number: state.number + 1 };
        }
    },
    effects: {
        *asyncAdd(action, { call, put }) {
            yield call(delay, 1000);
            yield put({ type: 'counter/add' });
+            throw new Error('asyncAdd');
        },
        *goto({ to }, { put }) {
            yield put(routerRedux.push(to));
        }
    },
+    subscriptions: {
+        changeTitle({ dispatch, history }, done) {
+            console.log('changeTitle');
+            done('出错啦');
+        }
+    }
});
function Counter(props) {
    return (

            {props.loading.models.counter ? 'loading' : props.counter.number}
            props.dispatch({ type: "counter/add" })}>加1
            props.dispatch({ type: "counter/asyncAdd" })}>异步加1
            props.dispatch({ type: "counter/goto", to: '/' })}>跳转到/
            props.dispatch(ActionCreators.undo())}>undo
            props.dispatch(ActionCreators.redo())}>redo

    )
}
const ConnectedCounter = connect(
    (state) => state.present
)(Counter);
const Home = () => Home;
const UserPageComponent = (dynamic as any)({
    app,
    models: () => [
        import('./models/users'),
    ],
    component: () => import('./routes/UserPage'),
});
app.router((api: any) => (

            <>

                    Home
                    counter
                    users

            </>

));
app.start('#root');

function delay(ms) {
    return new Promise((resolve) => {
        setTimeout(function () {
            resolve();
        }, ms);
    });
}
```

** 14.2 实现 #**** 14.2.1 src\dva\index.tsx #**

src\dva\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, combineReducers, applyMiddleware, compose } from 'redux';
import createSagaMiddleware from 'redux-saga';
import * as sagaEffects from 'redux-saga/effects';
import { NAMESPACE_SEP } from './constants';
import { connect, Provider } from 'react-redux';
import { DvaInstance, Router, Model } from './typings';
import prefixNamespace from './prefixNamespace';
import { createHashHistory } from 'history';
import Plugin, { filterHooks } from './plugin';
import { routerMiddleware, connectRouter } from "connected-react-router";
let history = createHashHistory();
export { connect };

export default function (opts: any = {}) {
    const app: DvaInstance = {
        _models: [],
        model,
        router,
        _router: null,
        start
    }
    const initialReducers = { router: connectRouter(history) };
    function model(model: Model) {
        const prefixedModel = prefixNamespace(model);
        app._models.push(prefixedModel);
        return prefixedModel;
    }

    function router(router: Router) {
        app._router = router;
    }
    const plugin = new Plugin();
    plugin.use(filterHooks(opts));
    app.use = plugin.use.bind(plugin);
    function start(root) {
        for (const model of app._models) {
            initialReducers[model.namespace] = getReducer(model, plugin._handleActions);
        }
        const reducerEnhancer = plugin.get('onReducer');
        let rootReducer = createReducer();
        const sagas = getSagas(app);
        const sagaMiddleware = createSagaMiddleware();
        const extraMiddlewares = plugin.get('onAction');
        const extraEnhancers = plugin.get('extraEnhancers');
        const enhancers = [...extraEnhancers, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware)];
        //let store = createStore(rootReducer, opts.initialState, applyMiddleware(...extraMiddlewares, routerMiddleware(history), sagaMiddleware));
        let store = createStore(rootReducer, opts.initialState, compose(...enhancers));
        app._store = store;
        const listeners = plugin.get('onStateChange');
        for (const listener of listeners) {
            store.subscribe(() => {
                listener(store.getState());
            });
        }
        sagas.forEach(saga => sagaMiddleware.run(saga));
        ReactDOM.render({app._router({ history })}, document.querySelector(root))
        app.model = injectModel.bind(app);
        app._getSaga = getSaga;
+        for (const model of app._models) {
+            if (model.subscriptions) {
+                runSubscription(model.subscriptions, model, app, plugin.get('onError'));
+            }
+        }
        function injectModel(m) {
            m = model(m);
            initialReducers[m.namespace] = getReducer(m, plugin._handleActions);
            store.replaceReducer(createReducer());
            if (m.effects) {
                sagaMiddleware.run(app._getSaga(m.effects, m));
            }
            if (m.subscriptions) {
                runSubscription(m.subscriptions, m, app, plugin.get('onError'));
            }
        }
+        function runSubscription(subscriptions, model, app, onError) {
            for (const key in subscriptions) {
+                let dispatch = prefixedDispatch(store.dispatch, model);
                subscriptions[key](
                    {
+                        dispatch,
                        history: app._history,
+                    }, err => {
+                        for (let fn of onError) {
+                            fn(err, dispatch);
+                        }
+                    }
                );
            }
        }
        function prefixedDispatch(dispatch, model) {
            return action => {
                const { type } = action;
                return dispatch({ ...action, type: prefixType(type, model) });
            };
        }
        function createReducer() {
            const extraReducers = plugin.get('extraReducers');
            return reducerEnhancer(combineReducers({
                ...initialReducers,
                ...extraReducers
            }));
```

```
        }
    }

    function getSagas(app) {
        let sagas: Array = [];
        for (const model of app._models) {
+            sagas.push(getSaga(model.effects, model, plugin.get('onEffect'), plugin.get('onError')));
        }
        return sagas;
    }

    return app;
}

+function getSaga(effects, model, onEffect, onError) {
    return function* () {
        for (const key in effects) {
+            const watcher = getWatcher(key, model.effects[key], model, onEffect, onError);
            yield sagaEffects.fork(watcher);
        }
    };
}
+function getWatcher(key, effect, model, onEffect, onError) {
    return function* () {
        yield sagaEffects.takeEvery(key, function* sagaWithCatch(...args) {
            if (onEffect) {
                for (const fn of onEffect) {
                    effect = fn(effect, sagaEffects, model, key);
                }
            }
+            try {
                yield effect(...args, { ...sagaEffects, put: action => sagaEffects.put({ ...action, type: prefixType(action.type, model) }) });
+            } catch (err) {
+                for (const fn of onError) {
+                    fn(err);
+                }
+            }
        });
    };
}
function prefixType(type, model) {
    if (type.indexOf('/')
        return `${model.namespace}${NAMESPACE_SEP}${type}`;
    }
    return type;
}
function getReducer(model, handleActions) {
    let { reducers = {}, state: defaultState } = model;
    let reducer = (state = defaultState, action) => {
        let reducer = reducers[action.type];
        if (reducer) {
            return reducer(state, action);
        }
        return state;
    }
    if (handleActions) {
        return handleActions(reducers || {}, defaultState);
    }
    return reducer;
}

export * from './typings';
```

** 14.2.2 src\dva\plugin.tsx #**

src\dva\plugin.tsx

```
const hooks = [
    'onEffect',
    'extraReducers',
    'onAction',
    'onStateChange',
    'onReducer',
    "extraEnhancers",
    '_handleActions',
+    "onError"
];
export function filterHooks(obj) {
    return Object.keys(obj).reduce((memo, key) => {
        if (hooks.indexOf(key) > -1) {
            memo[key] = obj[key];
        }
        return memo;
    }, {});
}
export default class Plugin {
    hooks: any
    _handleActions: any = null
    constructor() {
        this.hooks = hooks.reduce((memo, key) => {
            memo[key] = [];
            return memo;
        }, {});
    }
    use(plugin) {
        const { hooks } = this;
        for (const key in plugin) {
            if (key
                this._handleActions = plugin[key];
            } else if (key
                hooks[key] = plugin[key];
            } else {
                hooks[key].push(plugin[key]);
            }
        }
    }
    get(key) {
        const { hooks } = this;
        if (key
            return getExtraReducers(hooks[key]);
        } else if (key
            return getOnReducer(hooks[key]);
        } else {
            return hooks[key];
        }

    }
}
function getOnReducer(hook) {
    return function (reducer) {
        for (const reducerEnhancer of hook) {
            reducer = reducerEnhancer(reducer);
        }
        return reducer;
    };
}
function getExtraReducers(hook) {
    let ret = {};
    for (const reducerObj of hook) {
        ret = { ...ret, ...reducerObj };
    }
    return ret;
}
```

dva-source (https://gitee.com/zhufengpeixun/zhufeng-dva-source-typescript2)