# DVA学习笔记

## 什么是DVA

1. redux
   是 JavaScript 状态容器，提供可预测化的状态管理
2. redux-saga
   是一个库，致力于在React/Redux应用中简化异步操作
3. react-router
   一个针对React而设计的路由解决方案、可以友好的帮你解决React components 到URl之间的同步映射关系。

## DVA优势

```
redux概念太多，并且 reducer，saga，action 都是分离的（分文件）
```

- 编辑成本高
  需要在 reducer, saga, action 之间来回切换
- 不便于组织业务模型
  比如我们写了一个 userlist 之后，要写一个 productlist，需要复制很多文件。
- saga 书写太复杂
  每监听一个 action 都需要走 fork -> watcher -> worker 的流程
- entry 书写麻烦
  entry 书写比较麻烦

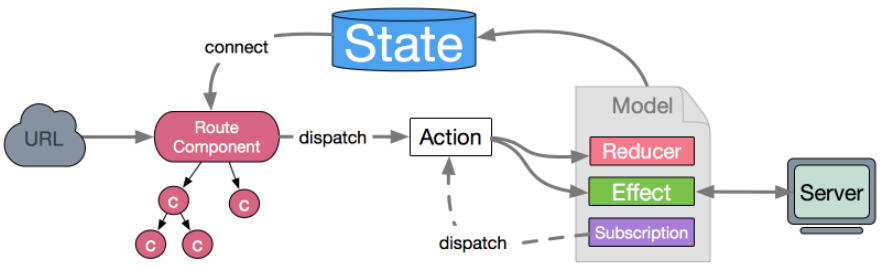
## DVA特征

- 易学易用
  仅有 6 个 api，对 redux 用户尤其友好
- elm 概念
  通过 reducers, effects 和 subscriptions 组织 model
- 支持 mobile 和 react-native
  跨平台

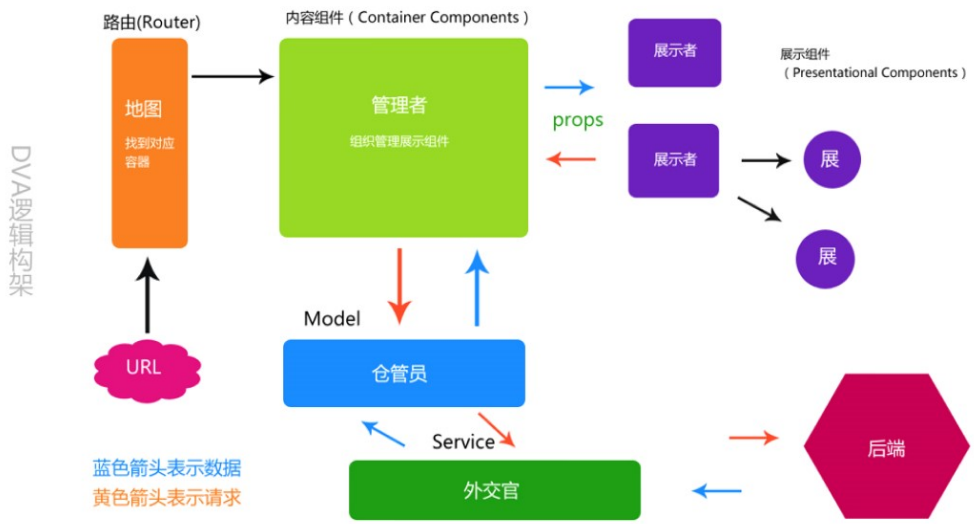
- 支持 HMR
  目前基于 babel-plugin-dva-hmr 支持 components、routes 和 models 的 HMR
- 动态加载 Model 和路由
  按需加载加快访问速度
- 插件机制
  比如 dva-loading 可以自动处理 loading 状态，不用一遍遍地写 showLoading 和 hideLoading
- 完善的语法分析库
  dva-cli 基于此实现了智能创建 model, router 等
- 支持 TypeScript
  通过 d.ts

**DVA相关概念**

1. 数据流向
   数据的改变发生通常是通过用户交互行为或者浏览器行为（如路由跳转等）触发的，当此类行为会改变数据的时候可以通过 dispatch 发起一个 action，如果是同步行为会直接通过 Reducers 改变 State ，如果是异步行为（副作用）会先触发 Effects




2. DVA概念



3. namespace
   model 的命名空间，同时也是他在全局 state 上的属性，只能用字符串，不支持通过 . 的方式创建多层命名空间。
4. State(Models)
   State 表示 Model 的状态数据，通常表现为一个 javascript 对象（当然它可以是任何值）；操作的时候每次都要当作不可变数据（immutable data）来对待，保证每次都是全新对象，没有引用关系，这样才能保证 State 的独立性，便于测试和追踪变化。
5. Action(Models)
   Action 是一个普通 javascript 对象，它是改变 State 的唯一途径。无论是从 UI 事件、网络回调，还是 WebSocket 等数据源所获得的数据，最终都会通过 dispatch 函数调用一个 action，从而改变对应的数据。action 必须带有 type 属性指明具体的行为，其它字段可以自定义，如果要发起一个 action 需要使用 dispatch 函数；需要注意的是 dispatch 是在组件 connect Models以后，通过 props 传入的。
6. dispatch 函数(Models)
   dispatching function 是一个用于触发 action 的函数，action 是改变 State 的唯一途径，但是它只描述了一个行为，而 dipatch 可以看作是触发这个行为的方式，而 Reducer 则是描述如何改变数据的。
   在 dva 中，connect Model 的组件通过 props 可以访问到 dispatch，可以调用 Model 中的 Reducer 或者 Effects.
7. Reducer(Models)
   Reducer（也称为 reducing function）函数接受两个参数：之前已经累积运算的结果和当前要被累积的值，返回的是一个新的累积结果。该函数把一个集合归并成一个单值。
8. Effect(Models)
   Effect 被称为副作用，在我们的应用中，最常见的就是异步操作。它来自于函数编程的概念，之所以叫副作用是因为它使得我们的函数变得不纯，同样的输入不一定获得同样的输出。
9. Subscription(Models)
   Subscriptions 是一种从 源 获取数据的方法，它来自于 elm。
   Subscription 语义是订阅，用于订阅一个数据源，然后根据条件 dispatch 需要的 action。数据源可以是当前的时间、服务器的 websocket 连接、keyboard 输入、geolocation 变化、history 路由变化等等。
10. Router
    这里的路由通常指的是前端路由，由于我们的应用现在通常是单页应

用，所以需要前端代码来控制路由逻辑，通过浏览器提供的
History API 可以监听浏览器url的变化，从而控制路由相关操作。

## API

1. call()
   用于调用异步逻辑，支持Promise
   `yield call(usersService.fetch, { page });`

   其实也可以不用call

   ```
   //与上文等价使用 yield usersService.fetch({ page });
   ```

2. put()
   用于触发action
   `yield put({ type: 'todos/add', payload: 'Learn Dva' });`

3. app = dva(opts)
   创建应用，返回 dva 实例。(注：dva 支持多实例)
   opts 包含：
   history：指定给路由用的 history，默认是 hashHistory
   initialState：指定初始数据，优先级高于 model 中的 state，默认是 {}
4. app.use(hooks)
   配置 hooks 或者注册插件。（插件最终返回的是 hooks ）
   - onError((err, dispatch) => {})
     effect 执行错误或 subscription 通过 done 主动抛错时触发，可用于管理全局出错状态。
   - onAction(fn | fn[])
     在 action 被 dispatch 时触发，用于注册 redux 中间件。支持函数或函数数组格式
   - onStateChange(fn)
     state 改变时触发，可用于同步 state 到 localStorage，服务器端等。
   - onReducer(fn)
     封装 reducer 执行。比如借助 redux-undo 实现 redo/undo
   - onEffect(fn)
     封装 effect 执行。比如 dva-loading 基于此实现了自动处理 loading 状态。
   - onHmr(fn)
     热替换相关，目前用于 babel-plugin-dva-hmr

- extraReducers
  指定额外的 reducer，比如 redux-form 需要指定额外的 form reducer
- extraEnhancers
  指定额外的 StoreEnhancer ，比如结合 redux-persist 的使用

**Model**

- namespace
  model 的命名空间，同时也是他在全局 state 上的属性，只能用字符串，不支持通过 . 的方式创建多
  层命名空间
- state
  初始值，优先级低于传给 dva() 的 opts.initialState
- reducers
  以 key/value 格式定义 reducer。用于处理同步操作，唯一可以修改 state 的地方。由 action 触发。
  格式为 (state, action) => newState 或 [(state, action) => newState, enhancer]
- effects
  以 key/value 格式定义 effect。用于处理异步操作和业务逻辑，不直接修改 state。由 action 触发，
  可以触发 action，可以和服务器交互，可以获取全局 state 的数据等等。
  格式为 *(action, effects) => void 或 [*(action, effects) => void, { type }]
- subscriptions
  以 key/value 格式定义 subscription。subscription 是订阅，用于订阅一个数据源，然后根据需要 dispatch
  相应的 action。在 app.start() 时被执行，数据源可以是当前的时间、服务器的 websocket 连接、keyboard
  输入、geolocation 变化、history 路由变化等等

**UI基本原则**

- 亲密性原则，对内容进行分类
- 对齐原则
- 重复原则
- 对比原则

- 基本原则实战（对视觉效果进行润色）

## 总结

1. 项目问题
   这个demo本身有两个问题：
   - 在good Model中没有使用正确的service，导致无法查询到相应的数据并且无法批量删除good
     解决： 引用并更改为正确的service
   - 在Mock中批量删除没有相应的响应
     解决：在mock good中添加相应的api路由响应，并且在service中单独添加批量删除方法并且导出供Model使用
2. 项目总结
   本次项目对于认识了解并且使用DVA有很大的作用，对于数据流、Model、Service、Effect、Ruducer有了直接认识和体会。虽然是一个模仿的小项目，但是开发过程中问题有：
   - 大错不犯，小错不断
     单词拼写错误，符号错误等一些小错误，开发时时刻注意这个问题，细心专注。
   - 学会调试代码，快速定位问题
     完整的代码不可能一蹴而就，需要在开发中通过调试，打断点、输出等方法。
   - 遵守代码规范，给以后的团队开发做铺垫，适应企业团队开发氛围