Dva

<https://dvajs.com/guide/concepts.html#state>

2. 大概总结

框架: dva是个框架，集成了redux、redux-saga、react-router-redux、react-router

快速初始化: 可以快速实现项目的初始化，不需要繁琐地配置

简化开发：将initState、saga、reducer集成到一个model里面统一管理，避免文件散落在各个文件里面，便于快速查找与开发

简洁的API：整个项目中只有dva、app.model、app.router、app.use、app.start几个API

无缝对接：跟react的生态没有冲突，例如可以直接使用redux devtool工具

动态机制：app.start以后，仍然可以注册model，灵活性较高

再说说个人觉得不太爽的地方吧:

namespace不统一: dva中的action.type格式是namespace/XXX，且在model中不需要添加前缀namespace，但是在组件中dispatch，却需要添加prefix

action问题：action会散落在两个地方，一个是saga里面，另外一个是component dispatch的时候，当然这个问题在使用redux-saga的时候就会存在，只是dva仍然没有很好地统一起来。

3. 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

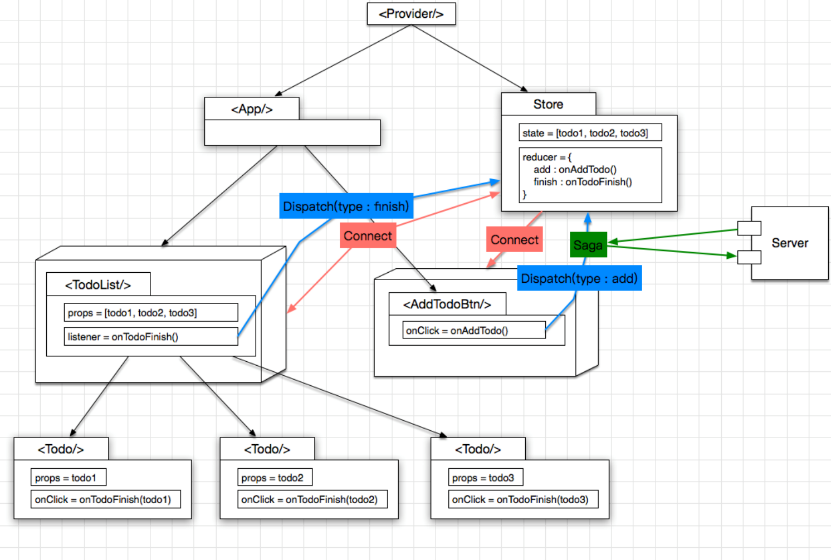
官网

1. 不懂

注册modal，Modal外,namespace



1. dva 为了控制副作用的操作，底层引入了redux-sagas做异步流程控制，由于采用了generator的相关概念，所以将异步转成同步写法，从而将effects转为纯函数
2. Subscription
3. Subscriptions 是一种从 源 获取数据的方法，它来自于 elm。
4. 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。
5. React 只负责页面渲染, 而不负责页面逻辑, 页面逻辑可以从中单独抽取出来, 变成 store
6. 同时又订阅了 store 的状态变化, 一旦状态有变, 被 connect 的组件也随之刷新
7. 使用 dispatch 往 store 发送 action 的这个过程是可以被拦截的, 自然而然地就可以在这里增加各种 Middleware, 实现各种自定义功能, eg: logging



1. Dva 是基于 React + Redux + Saga 的最佳实践沉淀

把 store 及 saga 统一为一个 model 的概念, 写在一个 js 文件里面

增加了一个 Subscriptions, 用于收集其他来源的 action, eg: 键盘操作

model 写法很简约, 类似于 DSL 或者 RoR, coding 快得飞起✈️

1. 但并不需要在应用启动的时候就全部加载，比较典型的是各类管理控制台。如果每个功能页面是通过路由切换，互相之间没有关系的话，通常会使用webpack的require.ensure来做代码模块的懒加载。
2. 有不少场景是可以做全局model的，比如说，我们在路由之间前进后退，model可以用于在路由间共享数据，比较典型的，像列表页和详情页的互相跳转，就可以用同一份model去共享它们的数据。
3. <https://github.com/sorrycc/blog/issues/18>