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

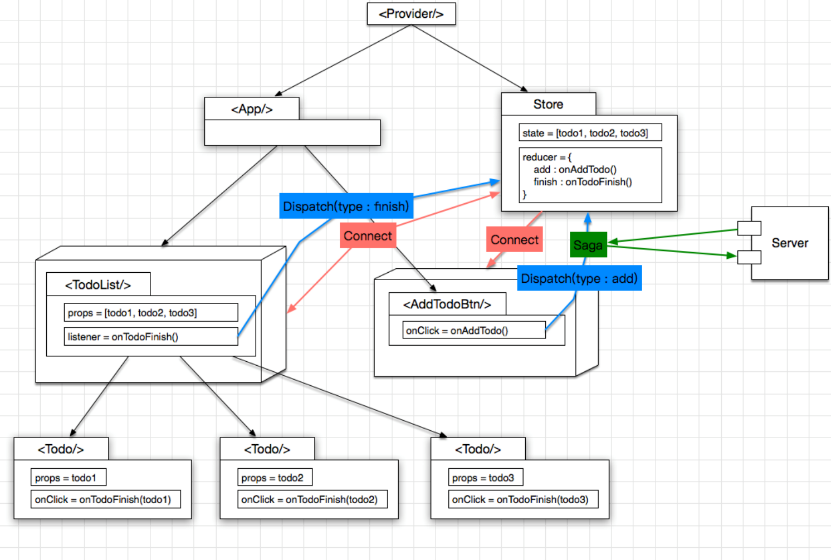
官网

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>

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

6.subscription是啥？

7. 动态加载model

有不少业务场景下，我们可能会定义出很多个model，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。

我们也可以利用这个特性来做model的动态加载。见例子，这样，在视图切换到这个路由的时候，对应的model就会被加载。同理，也可以做model的动态移除，不过，一般情况下是不需要移除的。

8. 使用model共享全局信息

从业务场景来说，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。

注意，如果当前应用中加载了不止一个model，在其中一个的effect里面做select操作，是可以获取另外一个中的state的

9. model的复用

model在state中是直接以namespace作key存放的，实际上只有一份实例

10. 动态扩展model

在上一节中，我们提到可以把model进行分类，以实现在若干视图中的共享，但业务需求是比较多变的，很可能我们又会遇到这种情况：

注意到dva中的每个model，实际上都是普通的JavaScript对象，包含

namespace

state

reducers

effects

subscriptions

注意这个的用法：



11. 长流程的业务逻辑

在一个effect中，可以使用多个put来分别调用reducer来更新状态。

在effect中可能会存在多个异步的服务调用，比如说，要调用一次服务端的验证，成功之后再去提交数据，这时候，在一个effect中就会存在多个call操作了。

yield select是啥

12. 使用take操作进行事件监听

在redux-saga中，提供了take和takeLatest这两个操作，dva是redux-saga的封装，也是可以使用这种操作的。

看不懂：



所以，我们也可以在dva中使用take操作来监听action

13．多任务调度

上一节我们提到的是多个任务的串行执行方式，这是业务中最常见的多任务执行方式，只需逐个yield call就可以了。

任务的并行执行

如果想要让任务并行执行，可以通过下面这种方式

1. 比如dashboard，其中各组件之间业务关联较小，就可以用这种方式去分别加载数据
2. 如果多个任务之间存在竞争关系，可以通过下面这种方式

14．跨model的通信

没看...

官网

知识地图

1. 注意两种语法的描述，尤其是注意这个this在es5里的用法，还有bind

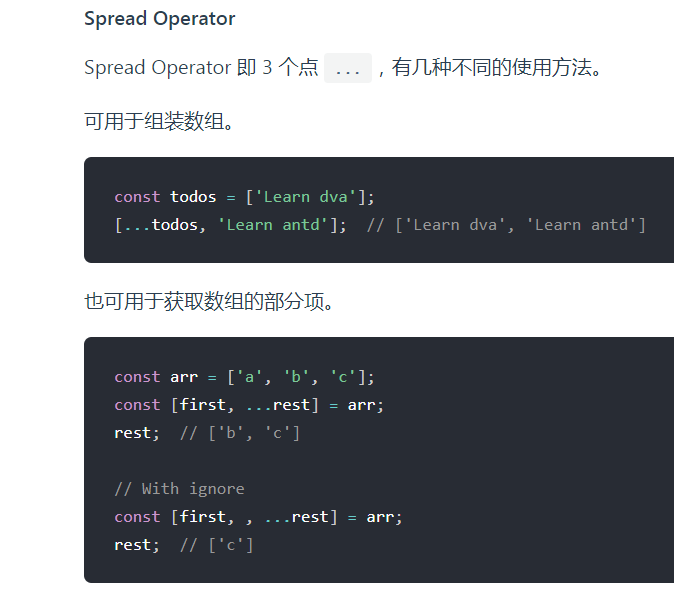


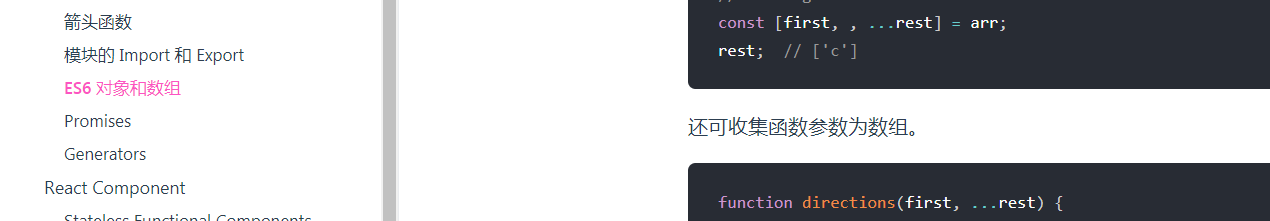
1. 啥意思



1. 会变成这个样子的吗？



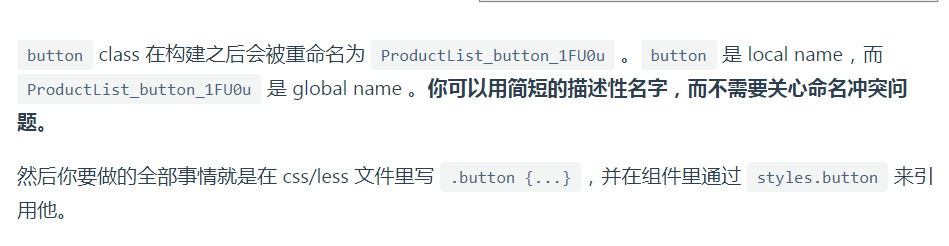
1. 
2. 这一节要好好看，是es6的语法



1. dva 的 effects 是通过 generator 组织的。Generator 返回的是迭代器，通过 yield 关键字实现暂停功能。这是一个典型的 dva effect，通过 yield 把异步逻辑通过同步的方式组织起来。
2. 所以请尽量声明 propTypes 对 props 进行校验，以减少不必要的问题



1. css modules



10.可以条件判断来选择不同的classname



1. 这个啥意思



1. reducers

建议最多一层嵌套，以保持 state 的扁平化，深层嵌套会让 reducer 很难写和难以维护。

1. Effects

effect可以看作是redux-saga的任务单元

Effect

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。

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。至于为什么我们这么纠结于 纯函数，如果你想了解更多可以阅读Mostly adequate guide to FP，或者它的中文译本JS函数式编程指南。

# put

用于触发 action 。

call

用于调用异步逻辑，支持 promise 。

select

用于从 state 里获取数据。

13. 错误处理

# 全局错误处理

dva 里，effects 和 subscriptions 的抛错全部会走 onError hook，所以可以在 onError 里统一处理错误。

然后 effects 里的抛错和 reject 的 promise 就都会被捕获到了

本地错误处理

如果需要对某些 effects 的错误进行特殊处理，需要在 effect 内部加 try catch 。

14.异步请求

这里是fetch的用法





15. Subscription

subscriptions 是订阅，用于订阅一个数据源，然后根据需要 dispatch 相应的 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。格式为 ({ dispatch, history }) => unsubscribe

异步数据初始化

比如：当用户进入 /users 页面时，触发 action users/fetch 加载用户数据。

path-to-regexp Package

如果 url 规则比较复杂，比如 /users/:userId/search，那么匹配和 userId 的获取都会比较麻烦。这时推用 path-to-regexp 简化这部分逻辑。

16. Router

# Route Components

Route Components 是指 ./src/routes/ 目录下的文件，他们是 ./src/router.js 里匹配的 Component。

为啥有dispatch



还有这个：



还有这个：



19. Redux Middleware

比如要添加 redux-logger 中间件：



官网-仔细看

指南

1. dispatch

connect Model 的组件通过 props 可以访问到 dispatch，

dispatch可以调用 Model 中的 Reducer 或者 Effects，调用这些方法的方式如面：

注意：type是这种形式的，这里的user表示namespace, 而这里的add 是reducers里的一个方法



1. 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。reducers 是对当前 model 的 state 对象进行作用，需要注意的是 Reducer 必须是纯函数
2. 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，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
3. Subscriptions 是一种从 源 获取数据的方法，它来自于 elm

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

////////////不太懂怎么用的

1. 路由

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

不会用，需要看



(二)使用 Dva 开发复杂 SPA

1. 动态加载model

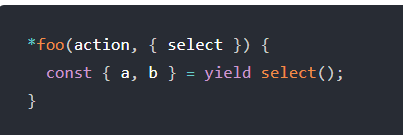
如果每个功能页面是通过路由切换，互相之间没有关系的话，通常会使用webpack的require.ensure来做代码模块的懒加载

不太会用，遇到的话，要仔细看官网

1. 使用model共享全局信息

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。

注意，如果当前应用中加载了不止一个model，在其中一个的effect里面做select操作，是可以获取另外一个中的state的



3.model的复用

4．动态扩展model

注意到dva中的每个model，实际上都是普通的JavaScript对象，包含

namespace

state

reducers

effects

subscriptions

注意这里有两级，model结构中的state，reducers，effects，subscriptions都是对象结构

可以借助dva社区的dva-model-extend库

1. 长流程的业务逻辑

在一个effect中，可以使用多个put来分别调用reducer来更新状态

在effect中可能会存在多个异步的服务调用，在一个effect中可能存在多个call操作了

1. 使用take操作进行事件监听

在redux-saga中，提供了take和takeLatest这两个操作，dva是redux-saga的封装，也是可以使用这种操作的。即在dva中使用take操作来监听action

// take(pattern) pattern是一个字符串，action 会在 action.type === pattern 时被匹配

1. 多任务调度

上一节我们提到的是多个任务的串行执行方式，这是业务中最常见的多任务执行方式，只需逐个yield call就可以了

任务的并行执行

如果想要让任务并行执行，可以通过下面这种方式，把多个要并行执行的东西放在一个数组里，就可以并行执行

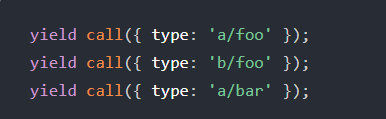
任务的竞争

如果多个任务之间存在竞争关系，可以通过下面这种方式：

1. 跨model的通信

比如一个流程贯穿多个mode这样的场景

在dva中，可以用namespace去指定接受action的model，所以可以通过类似这样的方式去组合



Dva源码解析

1.

从 'dva' 依赖中引入 dva ：import dva from 'dva';

通过函数生成一个 app 对象：const app = dva();

加载插件：app.use({});

注入 model：app.model(require('./models/example'));

添加路由：app.router(require('./routes/indexAnother'));

启动：app.start('#root');

在这 6 步当中，dva 完成了 使用 React 解决 view 层、redux 管理 model、saga 解决异步的主要功能。

事实上在我查阅资料以及回忆用过的脚手架时，发现目前端框架之所以被称为“框架”也就是解决了这些事情。前端工程师至今所做的事情都是在\*\*分离动态的 data 和静态的 view \*\*，只不过侧重点和实现方式也不同。

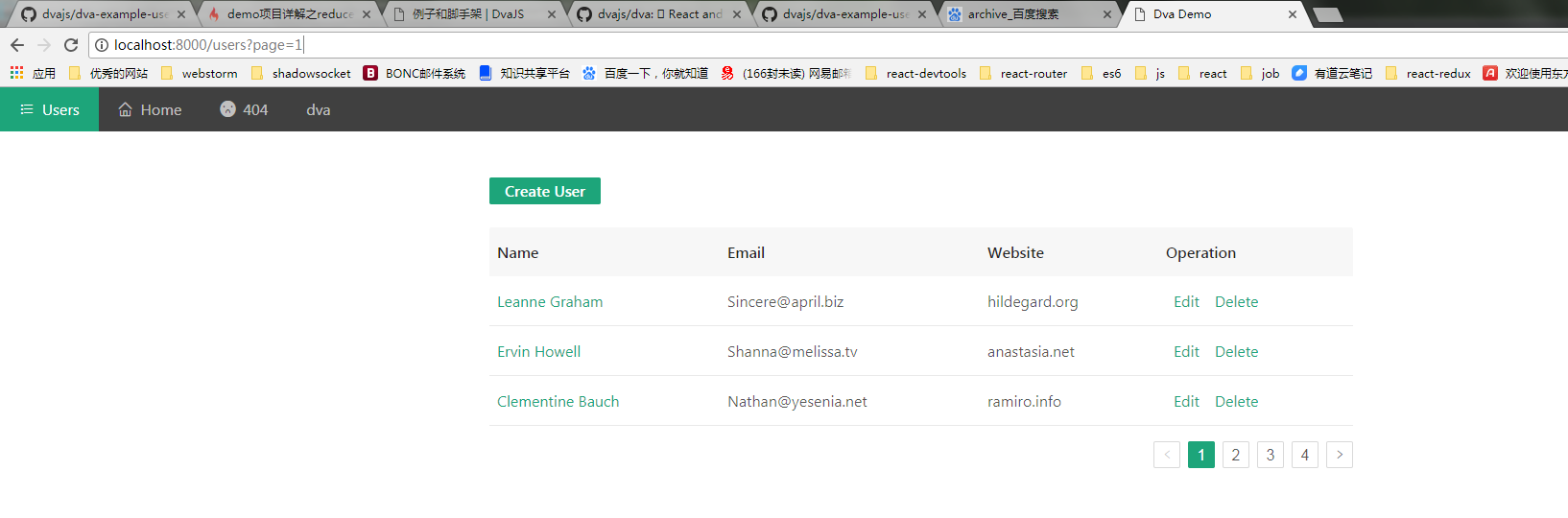
至今为止出了这么多框架，但是前端 MVX 的思想一直都没有改变。

1. export、import可以有多个，export default仅有一个。default后面不能跟const/let之类的关键字好像
2. <https://www.jianshu.com/p/c7b3b9c98d04>
3. <https://www.jianshu.com/p/69f13e9123d9>
4. 在dva里， 组件间通过传递参数来通信

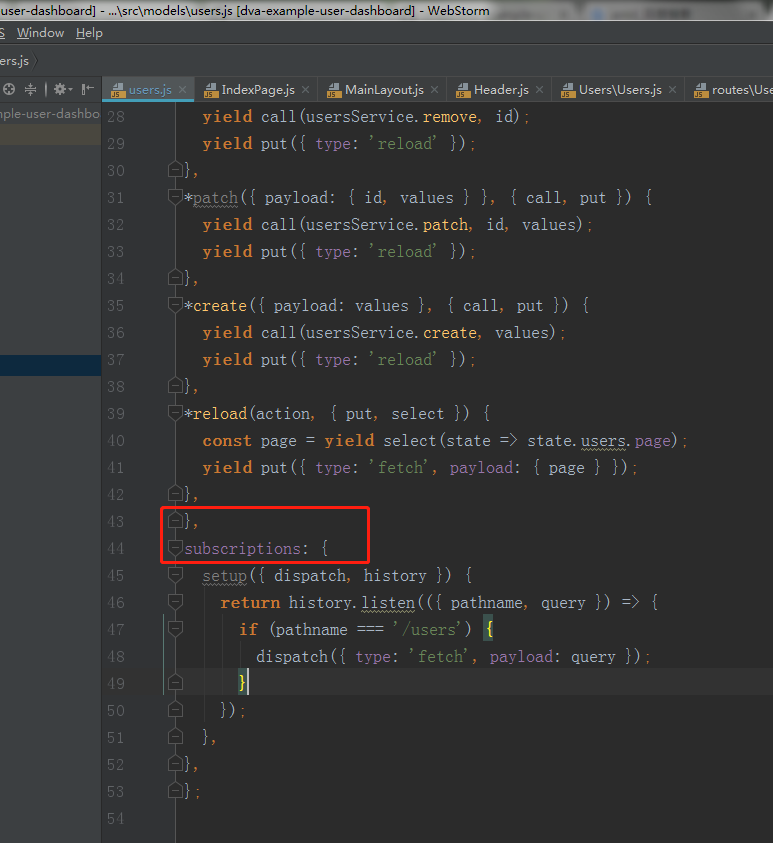
model和组件的通信就是connect之后，在组件里能通过model的namespace来访问到

dva官方例子

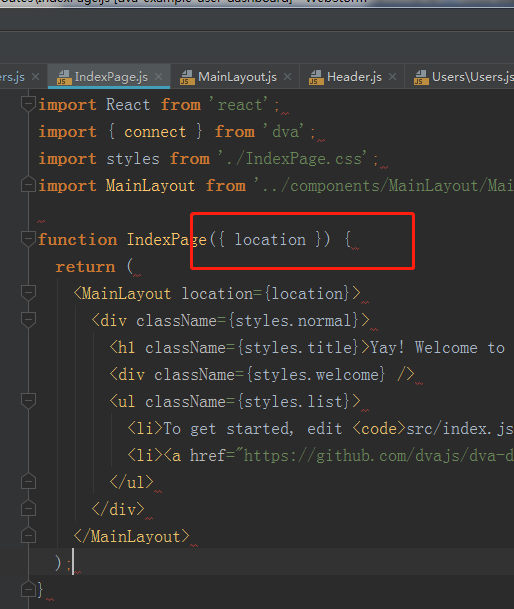
1.



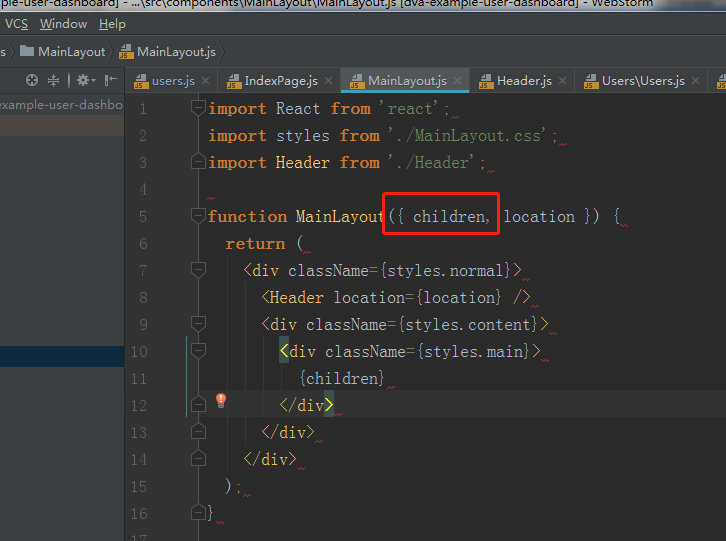
1. 这些没看懂：



一直传下去的location是啥？



Children是啥，看react的官方文档



**Dva的逻辑理解**

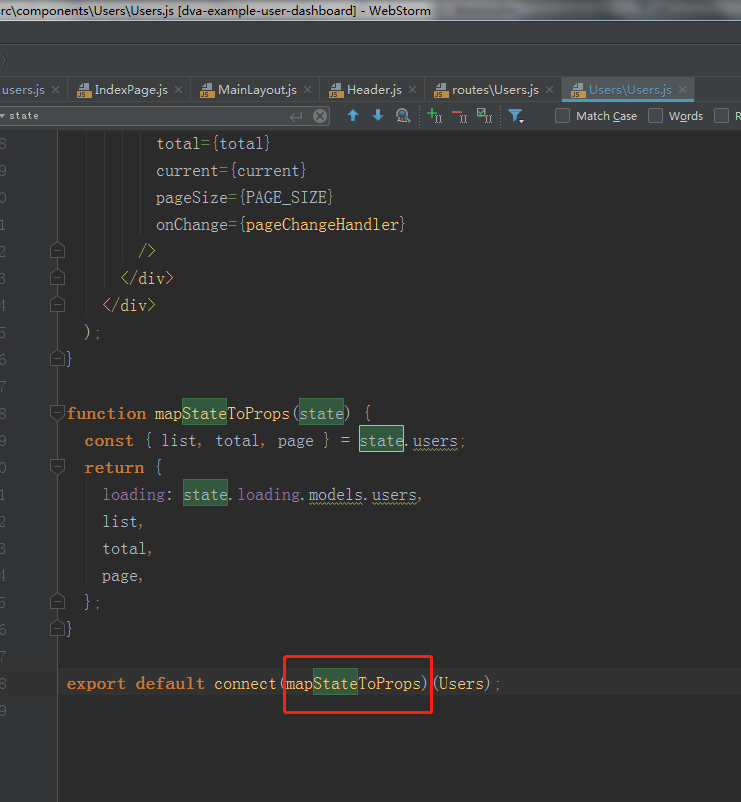
1. connect

<https://blog.csdn.net/samyang1/article/details/80364566>

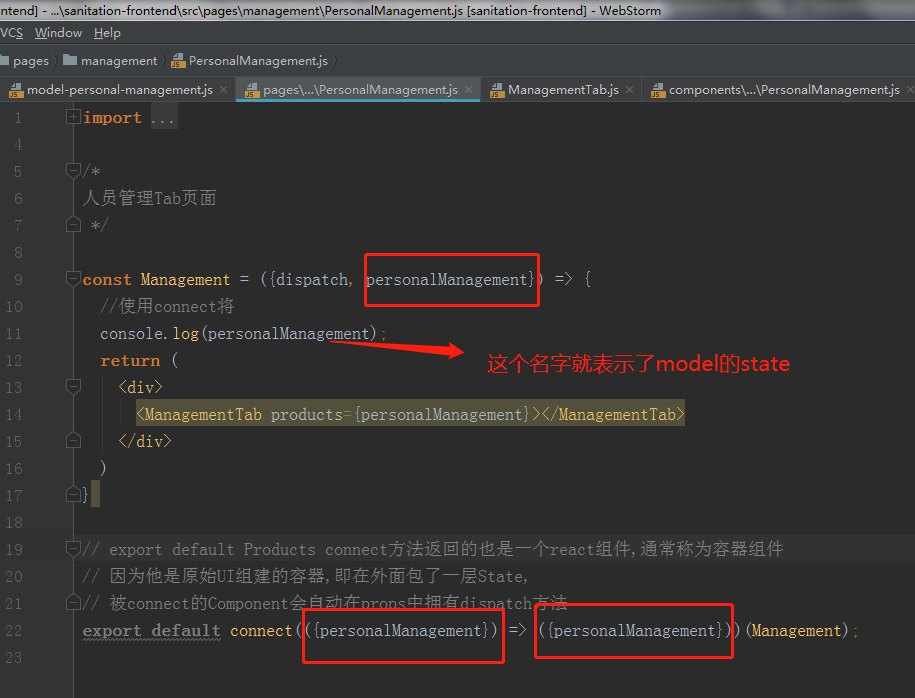
补充：

Connect的第一个参可以是2种情况：

1. 若connect的第一个参传递函数，则要求是：mapStateToProps函数，必须传至少一个参数，这个参数是state，即state必传； 返回的是需要绑定的model：

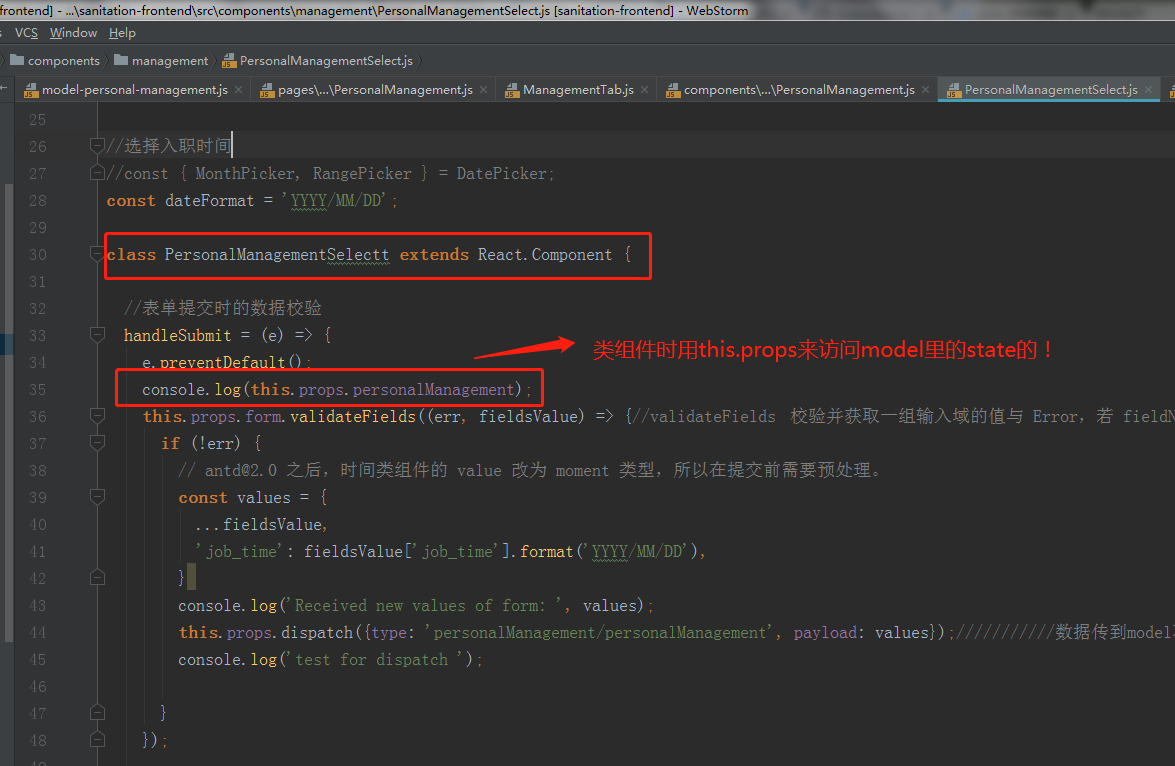


1. 使用es6的箭头函数来绑定，参数必须和model里的namespace同名，且必须返回该参：（注意图中怎么访问model里的state的）



上面是函数组件的访问model的方式，下面是类组件访问model的state的方式：

被 connect 的 类组件 会自动在 props 中拥有 dispatch 方法(见下图)，而若是函数式组件，需要手动传入dispatch和model的state（即model的namespace）(见上图)



1. 关于第一次没有数据时的处理：

通常第一次还没有数据的时候可以在组件的生命周期内部发起dispatch，或者监听路由（subscriptions）当时这个路由的时候发起dispatch 从而更新model。

1. 和model用connect函数连接起来的组件虽然会用到model里的namespace，但不需要import那个model文件
2. 问题：怎么将数据由model传入组件？

答：用connect，见上面的回答