复杂 WEB 应用的状态管理

挖了个大坑

我是谁？我在说什么？

应用规模增大，逻辑愈发复杂
展示逻辑与数据逻辑分离
单独维护的数据逻辑

状态管理

单页应用的数据流方案探索

https://github.com/xufei/blog/issues/47

知乎大 V，前端网红

重磅力作

有这样一个问题

从前没人讨论状态管理的时候

我们的应用是没有状态的吗?

应用的状态在组件树的每一个节点里

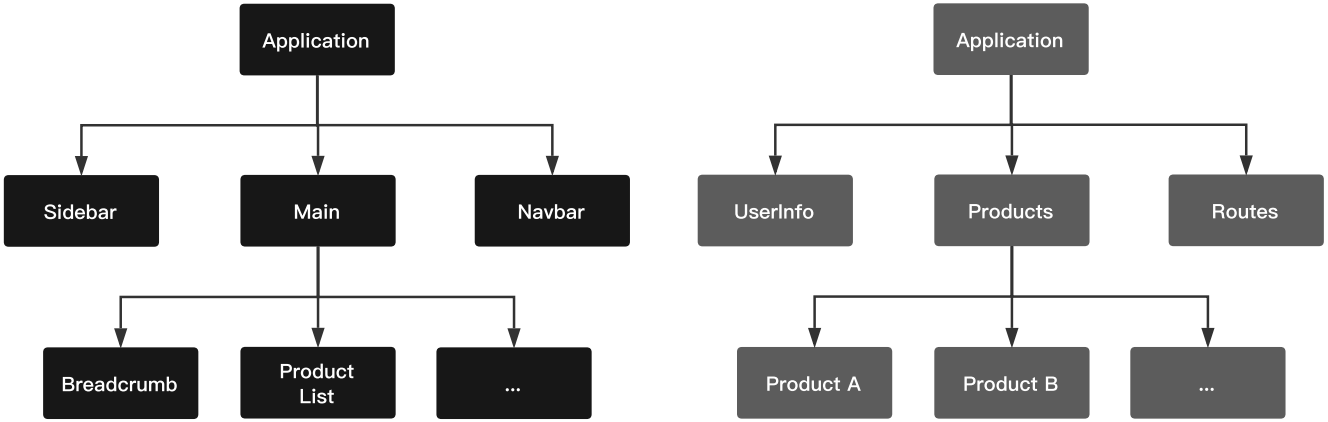
在 **React** 的 **Component** 里

在 **Angular** 的 **Controller** 里

在 **Vue** 的 **Component** 里

还有一些流落在 **service** 里，甚至 **utils** 里

Local State，也就是 State in Component 不好吗？


Application
Sidebar
Main
Navbar
Breadcrumb
Product List
...
Application
UserInfo
Products
Routes
Product A
Product B
...

View 与 Model 的逻辑分布不等同

状态数据的拓扑关系依赖界面组件的拓扑关系

越做越没法做

那就全部拿出来

彻底解耦

> 吼啊！
>
> *- Redux*

> **Local state is fine.**

- Redux 作者

所以不仅要搞
Local / Component State,
Global / Shared State
我们也要搞

那么问题来了

状态管理怎么搞

组合与复用逻辑的能力

Single Source Of Truth

调试与测试

组合与复用逻辑的能力

数据可组合

数据转换逻辑可复用

Single Source Of Truth

Single Source Of Truth != Single Store

关键是数据推导的能力

要么只存一份数据，用时计算/推导

要么存在多份数据，但由同一份数据（自动）推导得到

Flux 是怎么被黑的

调试与测试

Command Query Responsibility Segregation, CQRS

Event Sourcing, ES

Action/Message/Event -> Update/Reduce

利用调用栈

限制 & 管理副作用

限制 & 管理副作用

为什么要限制 & 管理

依赖运行环境

影响运行环境

不可重入、不易验证、不易模拟

如何限制 & 管理
禁止私下搞事情
Effect as data
React: View as data
验证描述行为的数据，并不真的执行

```
-- UPDATE

type Msg
  = Input String
  | Send
  | NewMessage String

update : Msg -> Model -> (Model, Cmd Msg)
update msg {input, messages} =
  case msg of
    Input newInput ->
      (Model newInput messages, Cmd.none)

    Send ->
      (Model "" messages, WebSocket.send "ws://echo.websocket.org" input)

    NewMessage str ->
      (Model input (str :: messages), Cmd.none)


-- SUBSCRIPTIONS

subscriptions : Model -> Sub Msg
subscriptions model =
  WebSocket.listen "ws://echo.websocket.org" NewMessage
```

```
import { call } from 'redux-saga/effects'

function* fetchProducts() {
  const products = yield call(Api.fetch, '/products')
  // ...
}
```

```
{
  CALL: {
    fn: Api.fetch,
    args: ['./products']
  }
}
```

```
import { call } from 'redux-saga/effects'
import Api from '...'

const iterator = fetchProducts()

// expects a call instruction
assert.deepEqual(
  iterator.next().value,
  call(Api.fetch, '/products'),
  "fetchProducts should yield an Effect call(Api.fetch, './products')"
)
```

Effect data

谁来产生数据

数据产生到哪

又是谁来消费

谁来产生副作用

Message/Action/Event 触发

redux-thunk, redux-saga, ...

Update/Reducer 触发

redux-loop, elm

不想管，扔出去

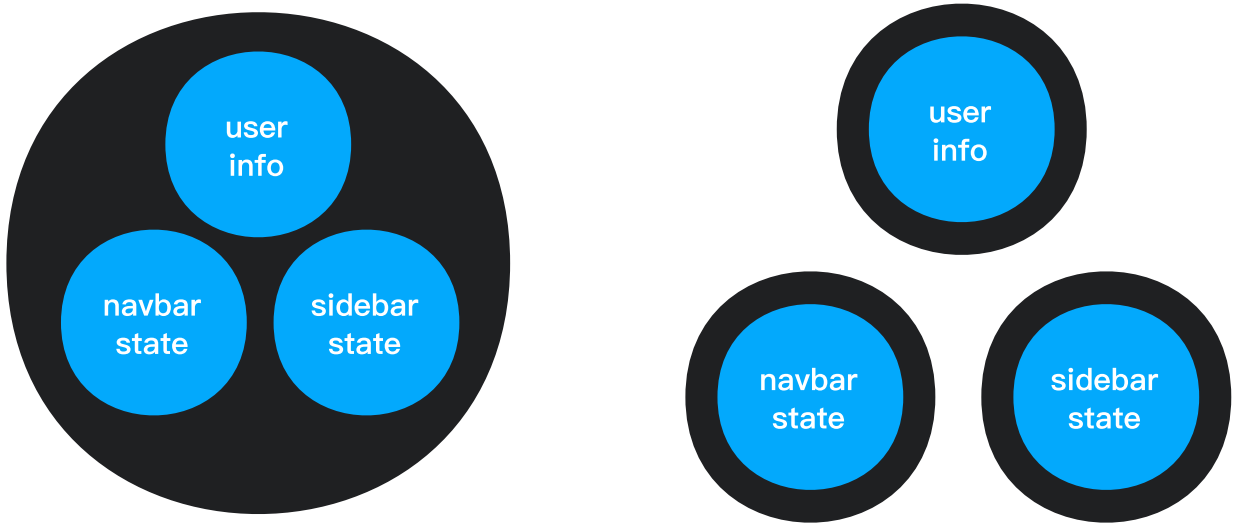
可组合的能力

数据是可组合的
行为是难以组合的

How to create a generic list as a reducer and component enhancer?
https://esdiscuss.org/topic/one-shot-delimited-continuations-with-effect-handlers

还有一些选择要做

Single Store vs Multiple Store


user
info
navbar
state
sidebar
state
user
info
navbar
state
sidebar
state

Single Store

namespace/path/cursor

容易持久化，容易写开发工具

Multiple Store

用起来方便

store 间数据同步

多 store 更新次序

Inject vs Import

替换 store，如后端渲染

静态分析，如定义跳转、类型信息

黑一波 Redux

提供的所有能力都是给予 Global State 的，趋向于贫血组件

Global/Local State 方案不一致

要求 Single Store & Inject

Global Namespace -> Boilerplate

Why not Rx?

An API for asynchronous programming with observable streams

Derivation with time

设想一个场景

在这里我会通过很多的时间

以及欲抑先扬的手段

来黑一波 Rx(JS)

弹性计算 > 云主机 - 实例 > 创建实例

选择地域 | 选择镜像 | 选择存储与网络 | 设置信息

镜像类型

请选择镜像类型。

(•) 公共镜像 ( ) 私有镜像

选择操作系统

不同操作系统对应不同的可用镜像列表。

(•) Debian ( ) CoreOS ( ) CentOS ( ) Windows Server

( ) Ubuntu ( ) OpenSUSE ( ) SUSE ( ) FreeBSD

选择镜像

Debian 8.2 32位 ▲▼

```
imageList: [
  'Debian 8.2 32 位',
  'Debian 8.2 64 位',
  // ...
],

selectedImage: 'Debian 8.2 32 位',
```

```
handleUserSelect(selected) {
  this.selectedImage = selected
},

replaceImageList(imageList) {
  this.imageList = imageList
  this.selectedImage = this.imageList[0]
}
```

```
addSpecialImage(image) {
  this.imageList = [...this.imageList, image]
  this.selectedImage = this.imageList[0]
},

disableInvalidImage(image) {
  this.imageList = this.imageList.filter(
    item => item !== image
  )
  this.selectedImage = this.imageList[0]
},

// ...
```

```
resetImage() {
  this.selectedImage = this.imageList[0]
},

replaceImageList() {
  // ...
  this.resetImage()
},

addSpecialImage() {
  // ...
  this.resetImage()
},

disableInvalidImage() {
  // ...
  this.resetImage()
}
```

借助 redux `subscribe` & reselect

```
const imageListSelector = createSelector(state => { /* get imageList from state */ })

function watchAndReset() {
  let lastImageList = null

  store.subscribe(() => {
    const imageList = imageListSelector(store.getState())
    if (imageList !== lastImageList) {
      store.dispatch({
        type: 'RESET_SELECTED_IMAGE'
      })
    }

    lastImageList = imageList
  })
}
```

借助 mobx `autorun`

```
@action updateSelectedImage(image) {
  this.selectedImage = image
},



watchAndReset() {
  autorun(() => {
    this.updateSelectedImage(this.imageList[0])
  })
}
```

借助 RxJS

```
const selectedImageOnReset$ = imageList$.map(imageList => imageList[0])
const selectedImageOnUserSelect$ = selectChangeEvent$.map(e => e.target.value)
const selectedImage$ = merge(selectedImageOnReset$, selectedImageOnUserSelect$)
```

那就把 时间 / 时序 加进来

```
imageList: {
  value: [
    'Debian 8.2 32 位',
    'Debian 8.2 64 位',
    // ...
  ],
  updatedAt: 1495113029603
}

userSelect: {
  value: 'Debian 8.2 32 位',
  updatedAt: 1495113029301
}
```

```
handleUserSelect(selected) {
  this.userSelect.value = selected
  this.userSelect.updatedAt = Date.now()
},

@computed get selectedImage() {
  return (
    this.imageList.updatedAt > this.userSelect.updateAt
    ? this.imageList.value[0]
    : this.userSelect.value
  )
}
```

所以说呢，

主要是改变思路

Rx 是不那么甜的工具函数包

When to Use Rx(JS)?

Use RxJS for orchestrating asynchronous and event-based computations

Use RxJS to deal with asynchronous sequences of data

Reified / Transparent Reactive Programming

大概就是这样，
Redux 也黑完了
Rx 也黑完了
我们用 MobX

MobX 有什么好的

MobX 只是提供了一个基础的能力

你很难找到角度去黑一个能力

你只能黑我

谢谢！