Vue.js实践

如何使用Vue2.0开发富交互式WEB应用

| 我是谁

钟恒

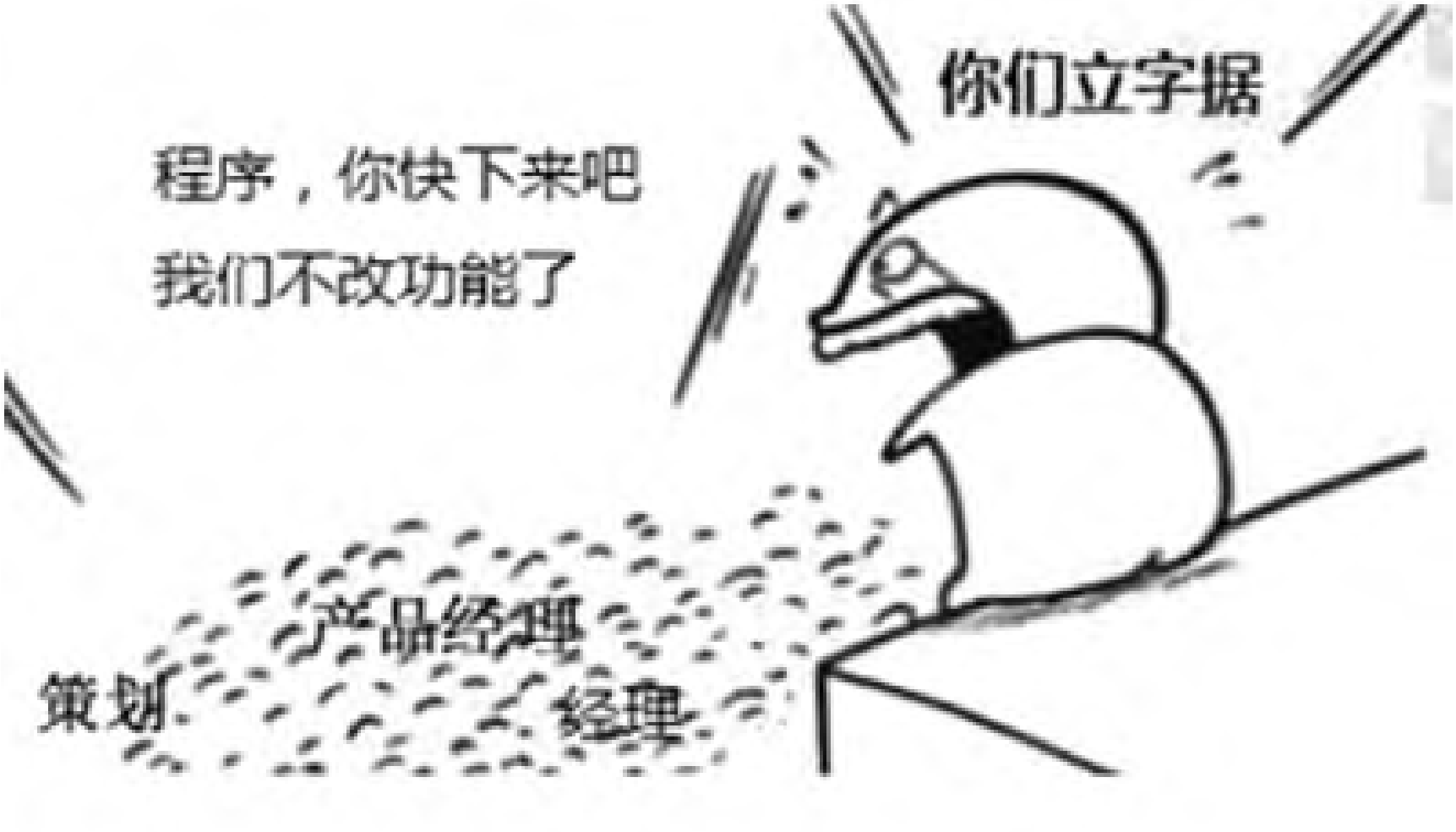
360奇舞团前端工程师

声享开发者

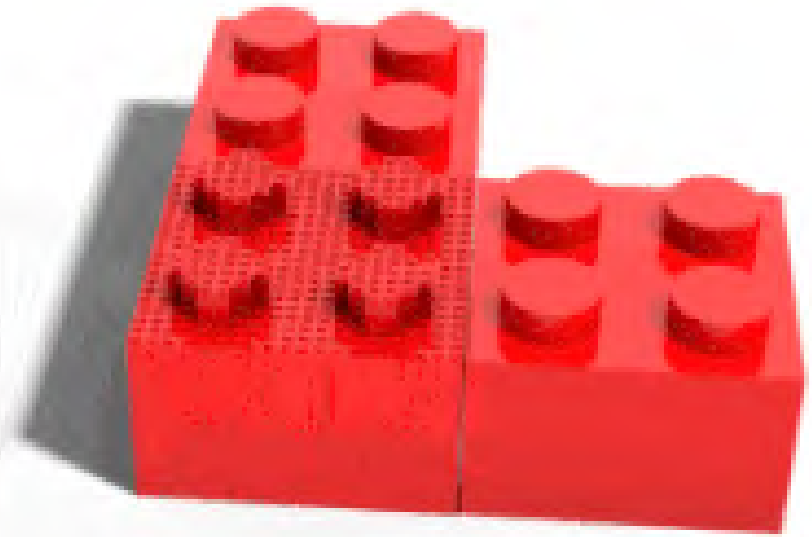
提纲

1. 架构
2. 开发
3. ~~填坑~~优化

架构难点



怎么办



架构需求

复用性高

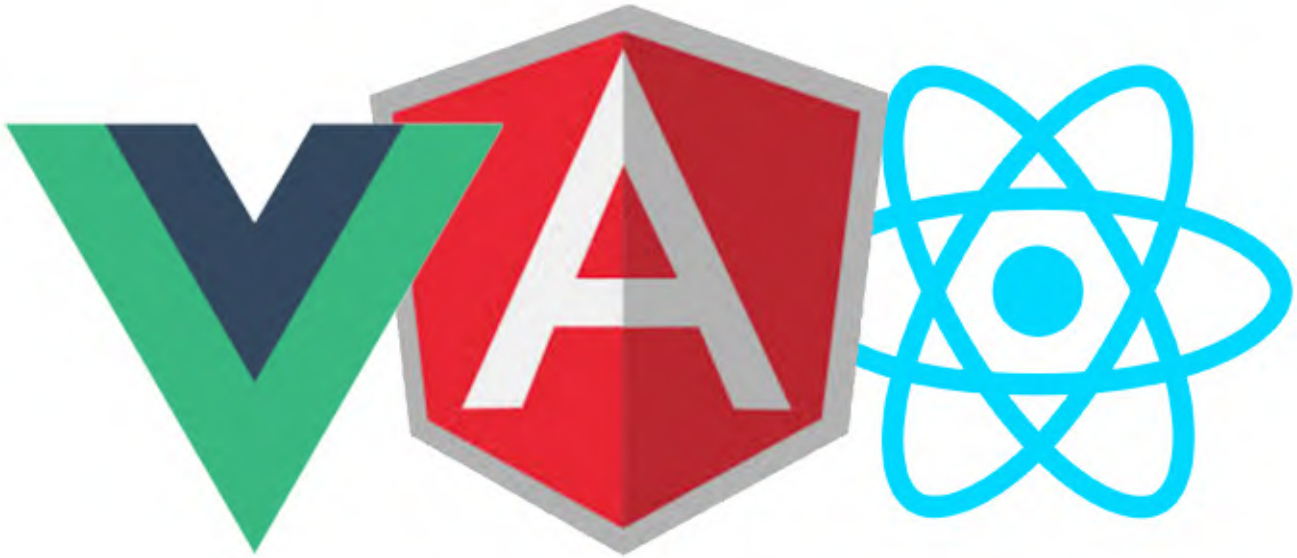
易于维护

易于变更

团队合作

前端界现在怎么组件化呢?

框架时代



框架对我们的开发方式有什么改变?

数据绑定


```
new Vue({
  template: '<input type="number" v-model="num">',
  data: {
    num: 0
  }
})
```

数据绑定完美无缺？ ~~易于维护~~

UI结构划分

问题

组件数目过多

单组件过重

不便于更改

~~复用性高~~

~~易于维护~~

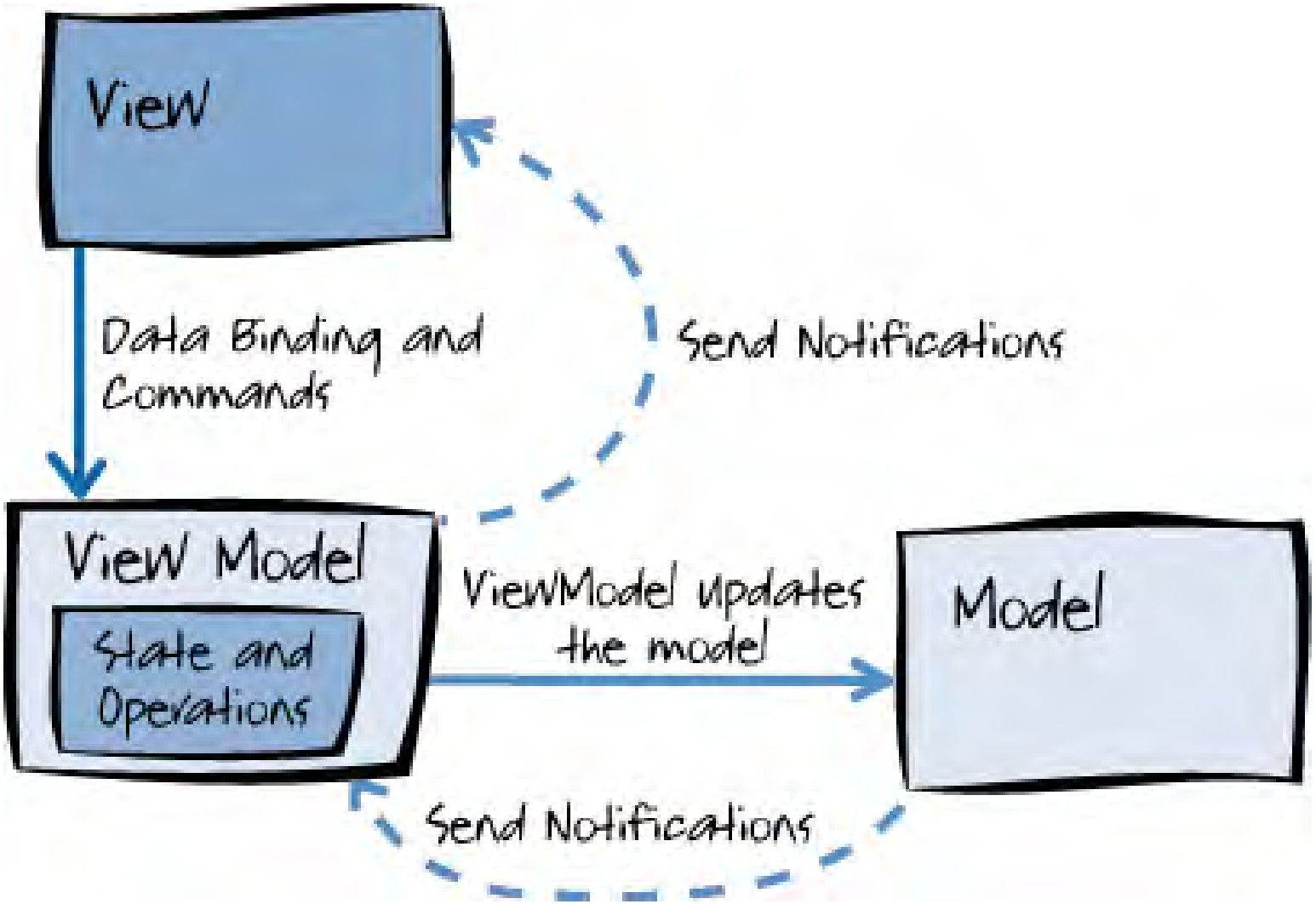
~~易于变更~~

~~团队合作~~

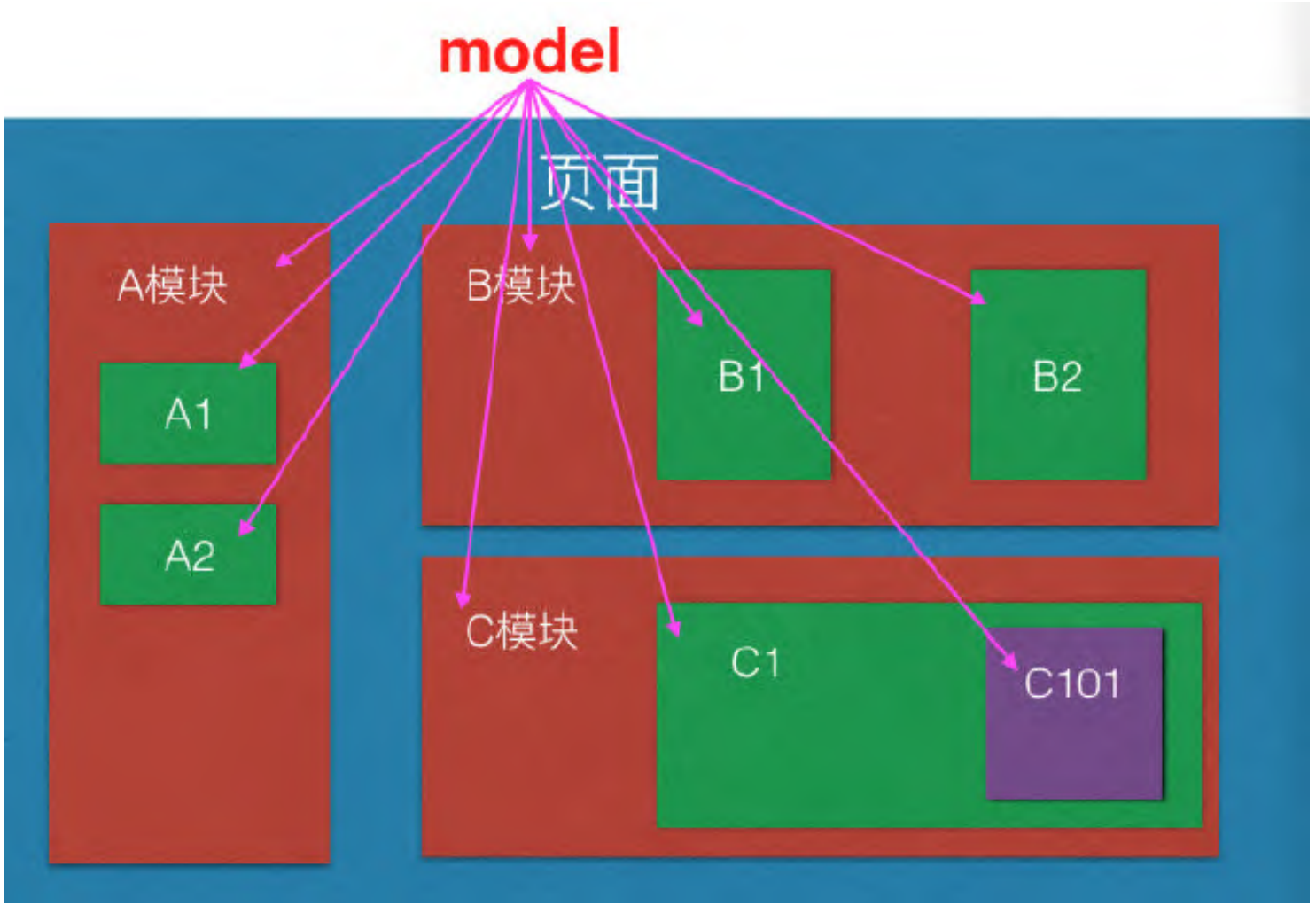
复杂的软件必须有清晰合理的架构，

否则无法开发和维护。

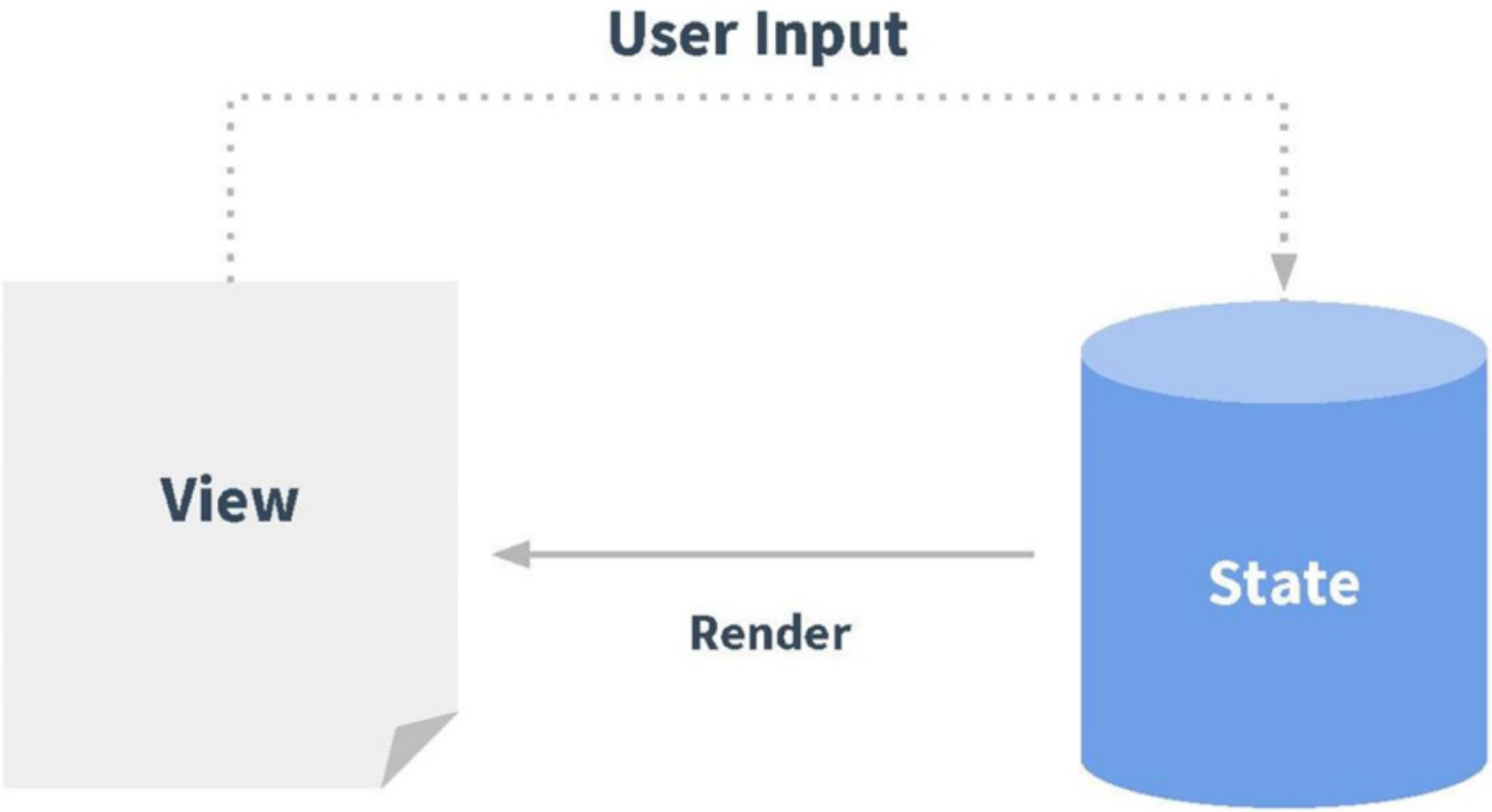
| 什么是MVVM


View
Data Binding and Commands
Send Notifications
View Model
State and Operations
ViewModel updates the model
Model
Send Notifications

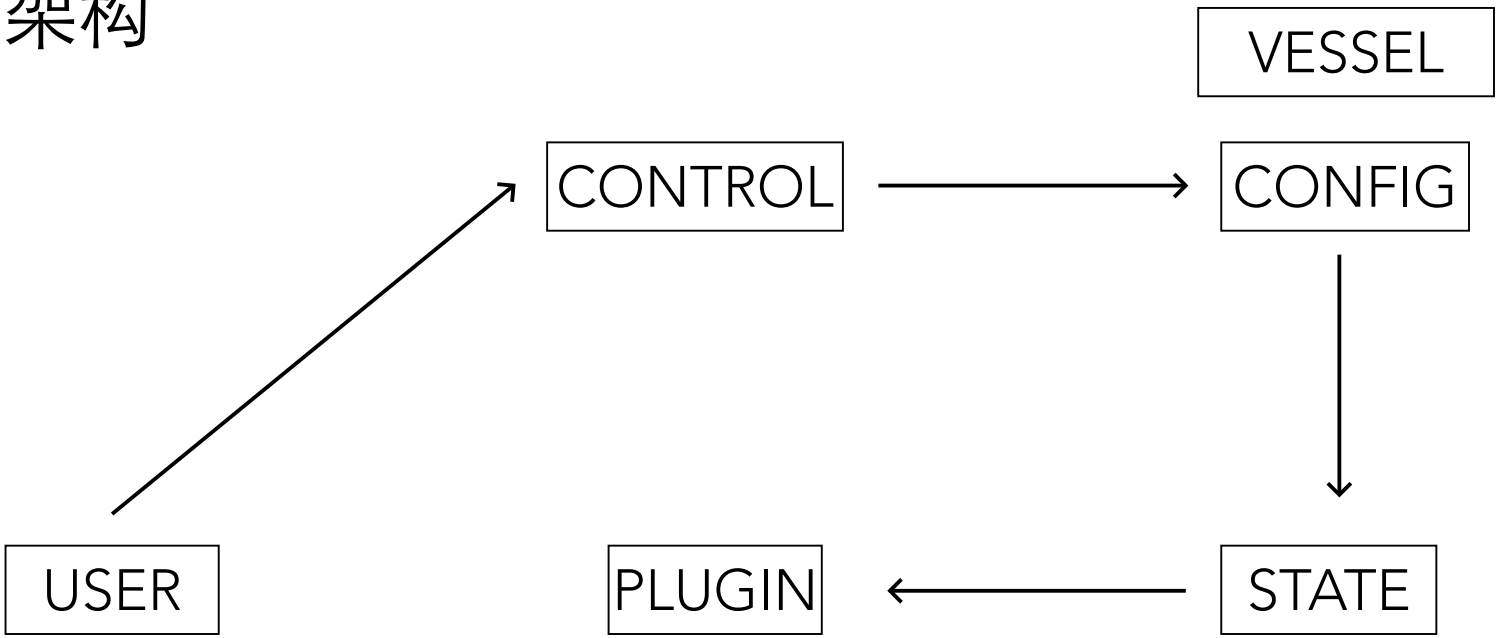
数据驱动



读写分离



架构



业务逻辑 CONFIG → STATE → PLUGIN

交互逻辑 CONTROL 页面布局 VESSEL

展示逻辑 PLUGIN

好处

保留了双向绑定	易于编写交互
数据驱动	易于开发
主业务读写分离	易于维护
业务、交互、展示、布局分离	易于更迭

提纲

1. 架构 ✔

一个MVVM式的组件化架构

2. 开发

组件化的问题

MVVM的问题

3. ~~填坑~~优化

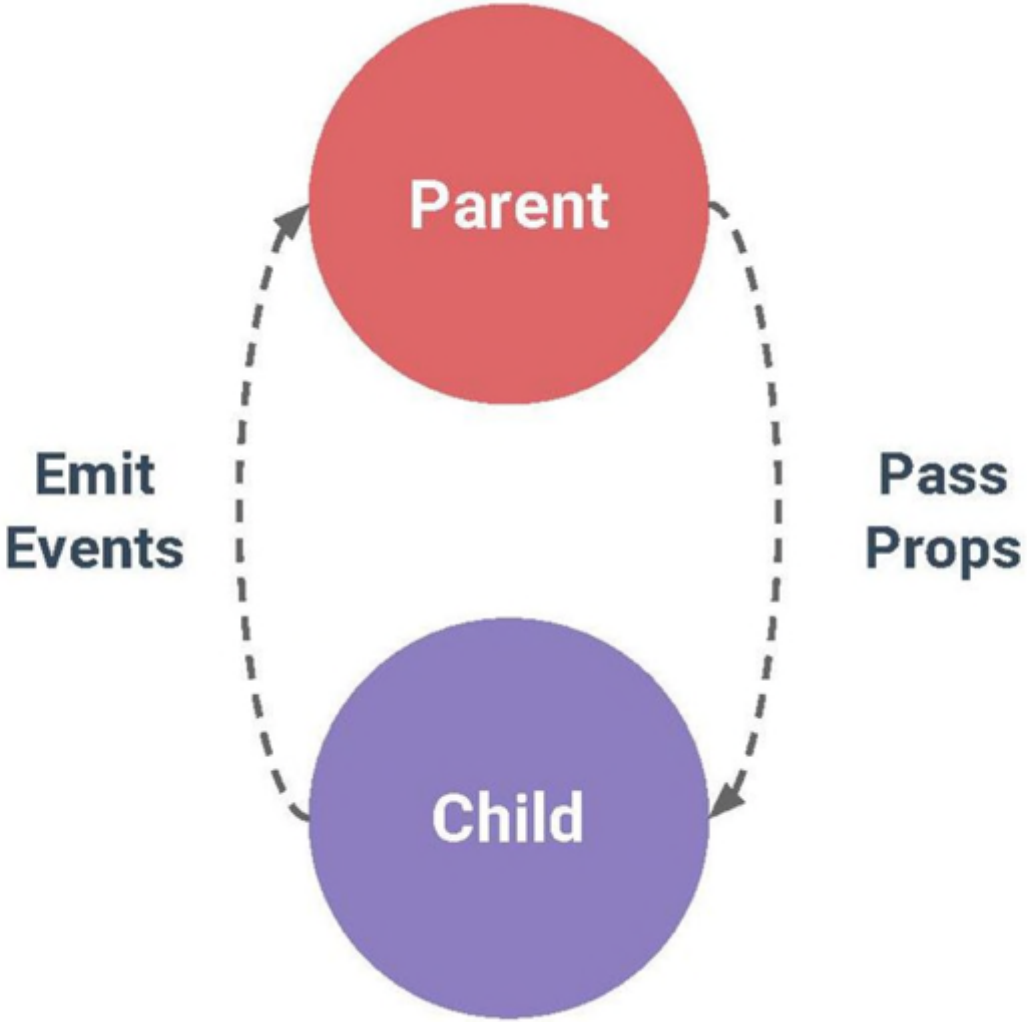
组件化带来的新问题

通信

复用

耦合

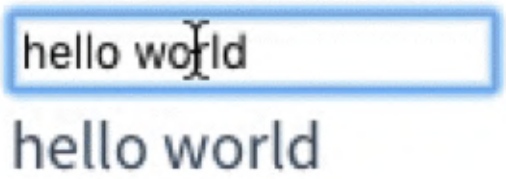
組件通信



| props

```
Vue.component('child', {
  props: ['message'],
  template: '<span>{{ myMessage }}</span>'
})
```

```
<div>
  <input v-model="parentMsg">
  <br>
  <child :message="parentMsg"></child>
</div>
```



HTML属性式传数据太方便了！

| props能双向传数据吗?

Vue.js 1.0

```
<!-- 双向 prop 绑定 -->
<my-component :prop.sync="someThing"></my-component>
```

Vue.js 2.0

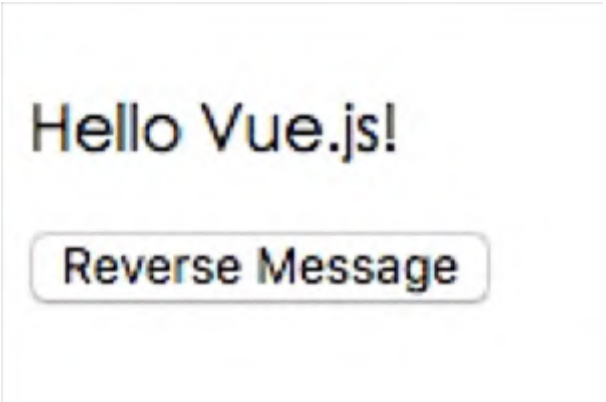
```
props: {
  callback: {
    type: Function,
    default: () => {}
  }
}
```

```
<my-component :callback="setStyles"></my-component>
```

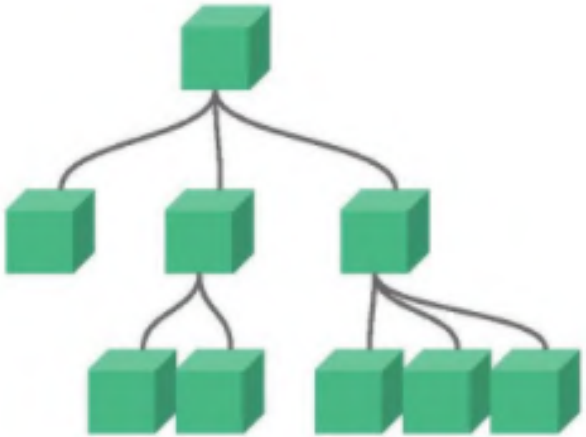
但是每个组件都要预设props很麻烦……

函数调用

```
var app = new Vue({
  el: '#app',
  template: `
<div>
  <child ref="child"></child>
  <button v-on:click="reverse">
     Reverse Message
  </button>
</div>`,
  methods: {
    reverse () {
     this.$refs.child.reverseMessage()
    }
  }
})
```



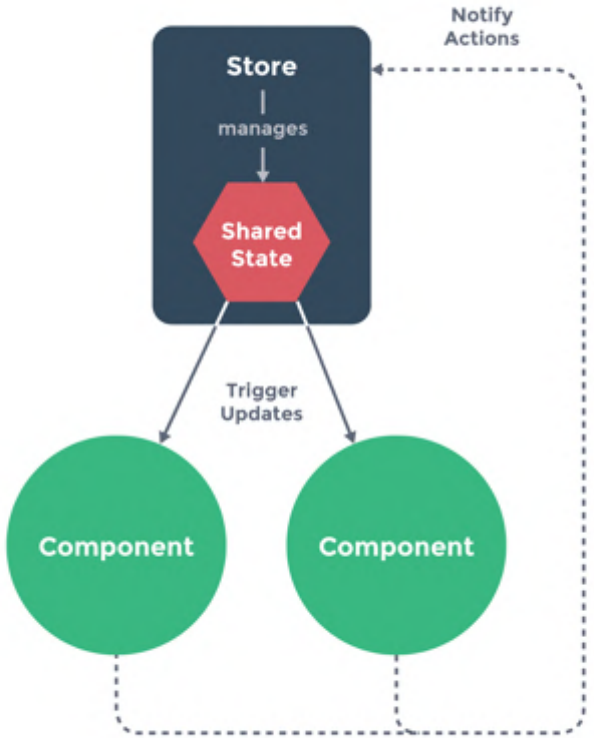
组件树的问题



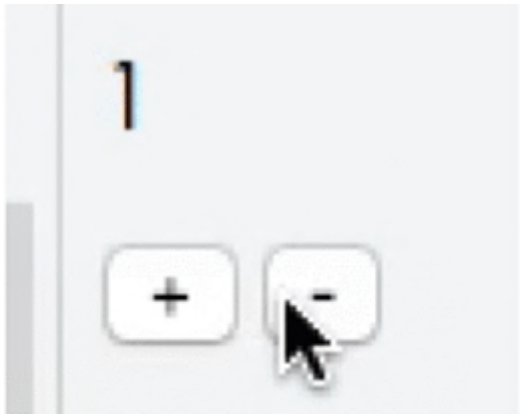
```
this.$parent.$parent.open()
```

需要更清晰扁平的通信方式

利用共享STATE




```
// plugin
<span>{{num}}</span>

// plus
this.num++

//minus
this.num--
```



l 用了共享STATE后

我改的，不好意思……你加个锁吧

这个功能是异步的

我先监听标识位A

我再修改标志位A，通知alpha组件去执行异步功能。

接着修改标志位B，证明标识位A是我修改的。

哦，标识位A被修改了

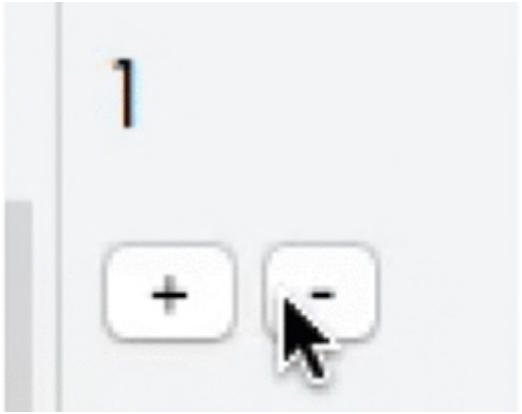
看看标识位B，这是alpha组件的标志

拿到数据咯

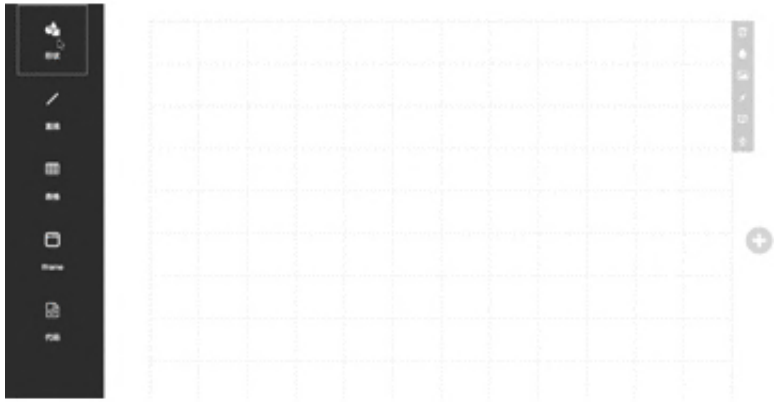
当数据位被多方操作

| Eventbus

```
var bus = new Vue()

// in component A's method
bus.$emit('plus', 1)
bus.$emit('minus', 1)

// in component B's created hook
bus.$on('plus', function (n) {
  this.total += n
})

bus.$on('minus', function (n) {
  this.total -= n
})
```



利用eventbus解决异步问题



```
// 组件内部
bus.$on('open-resource-shape', callback => {
  this.choose().then(callback)
})

// 使用者
bus.$emit('open-resource-shape', shape => {
  this.addShape(shape)
})

// 当然也可以promise啦
getShape () {
  return new Promise((resolve, reject => {
    bus.$emit('open-resource-shape', resolve)
  }))
}

this.getShape()
.then(this.addShape)

// 自然也可以async/await
let shape = await this.getShape()
this.addShape(shape)
```

通信方法选择

方法	场景
props	强耦合的组件间，单纯信息传递
function	强耦合的组件间多种通信方式
state	同步行为、数据量较小、数据位不被共用
eventbus	异步行为、数据量较大、共用的组件

提纲

1. **架构 ✔**
   一个MVVM的组件化架构
2. **开发**
   组件化
      通信✓
      复用
      耦合
   MVVM
3. **~~填坑~~优化**

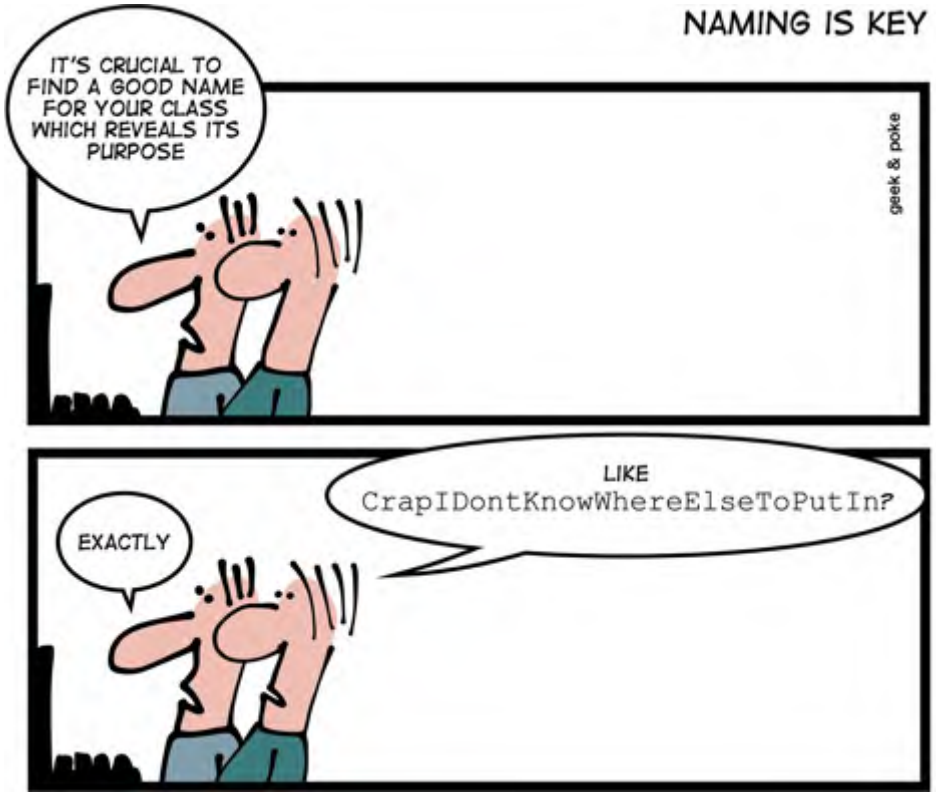
组件复用

冗余

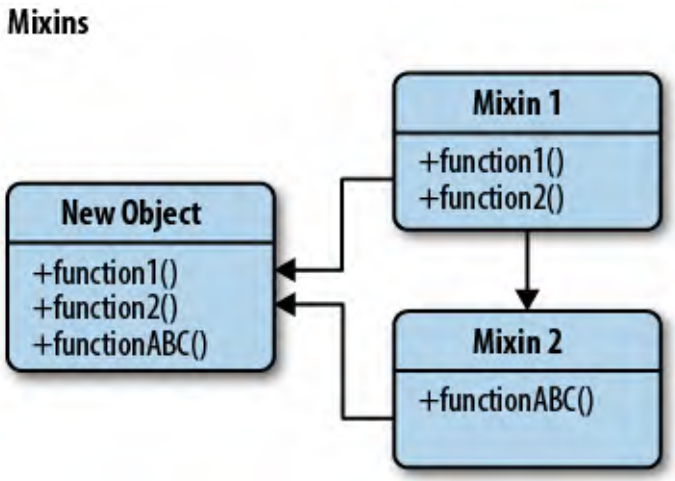
```
if(this.type === 'editing') {
  // some editing code
} else if(this.type === 'preview') {
  // some preview code
} else if(this.type === 'present') {
  // some present code
} else {
  // some base code
}
```

| 包装

```
// plugin-page.vue
<div>
  <slot name="page">
    i am a page
  </slot>
</div>

// present-plugin-page.vue
<div class="PresetPluginPage">
  <plugin-page ref="page">
    <h1 slot="page">
      i am a present page
    </h1>
  </plugin-page>
</div>

//output
<div class="PresetPluginPage">
  <div>
    <h1>
      i am a present page
    </h1>
  </div>
</div>
```



继承




```
// define a mixin object
var myMixin = {
  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}
// define a component that uses this mixin
var Component = Vue.extend({
  mixins: [myMixin]
})
var component = new Component()
// -> "hello from mixin!"
```

复用方法的选择

方法	场景
冗余	必要
包装	依照需求复杂度决定
继承	组件升级

提纲

1. **架构 ✔**
   一个MVVM的组件化架构
2. **开发**
   组件化
      通信 ✓
      复用 ✓
      耦合
   MVVM
3. **填坑优化**

组件耦合

- 单组件修改困难
- 组合新组件困难
- 组件debug困难

解耦的本质就是将变化分离

解耦

```
// wrong
<control-input type="number"></control-input>
// right
<control-number></control-number>
```

组件功能单一

```
// wrong
this.$parent.$parent.$refs['resource-image'].open()
// right
bus.$emit('open-resource-image')
```

采用稳定的接口

```
bindEvents (remove) {
  let method = remove
    ? 'removeEventListener'
    : 'addEventListener'
  window[method]('resize', this.handleResize)
}
```

处理好共享的部分

与服务端解耦

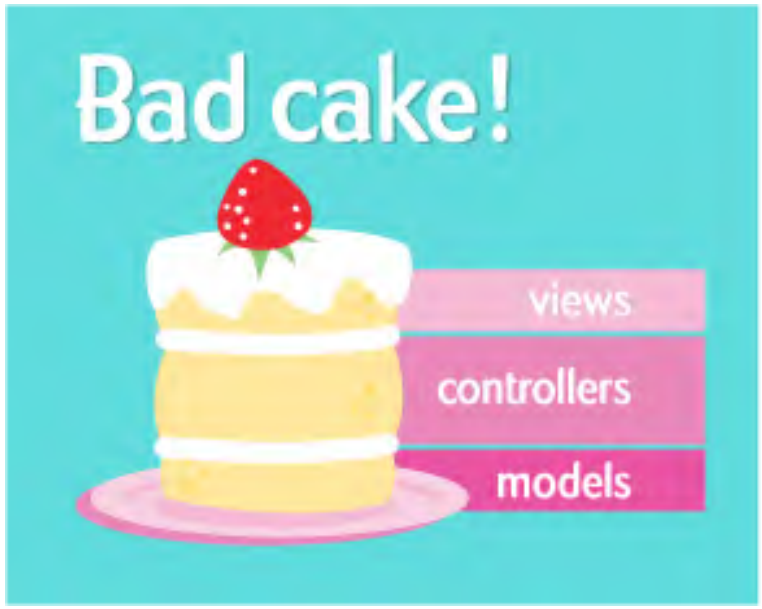
```
this.$http.get('/user/detail')
.then(({body}) => {
  this.user = JSON.parse(body).data
}, err => {
  console.error(err)
})

user.detail().then(detail => this.detail = detail)
```

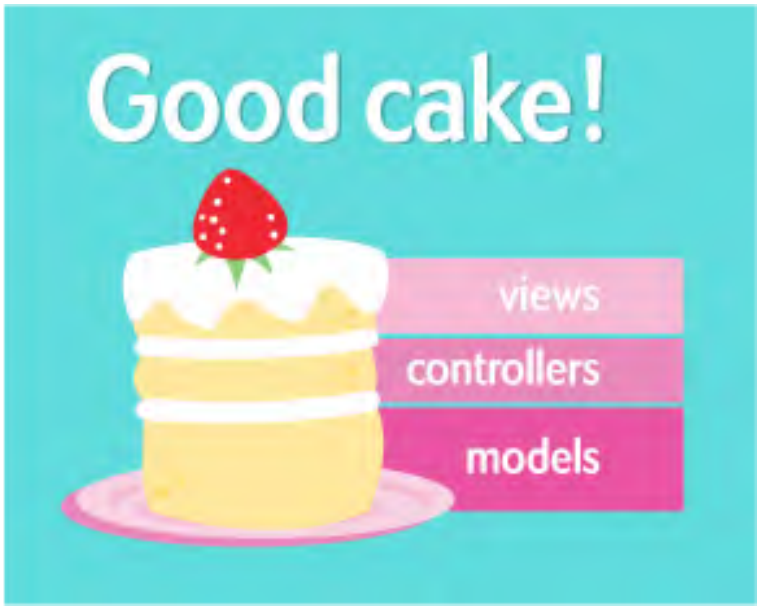
1. 服务端与前端体系不一
2. 同步异步转换
3. 多服务端/跨域的代码
4. 统一的错误处理代码

提纲

1. **架构 ✔**
   一个MVVM的组件化架构
2. **开发**
   组件化 ✓
      通信 ✓
      复用 ✓
      耦合 ✓
   MVVM
3. ~~**填坑**~~**优化**

| MVC的老问题





fat controller ——————————— fat viewModel

| 减肥

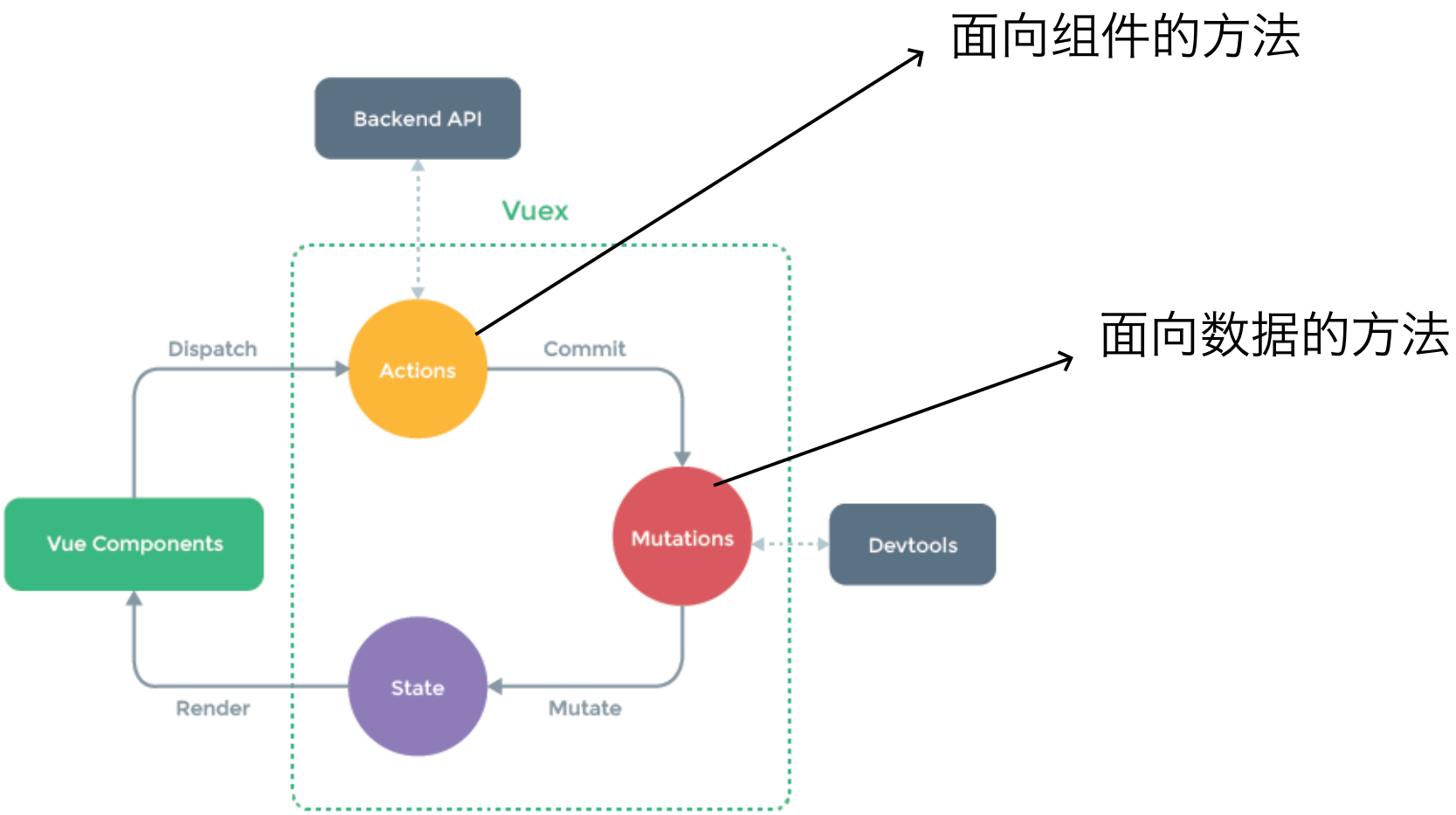
数据存储方式：数组/对象

数据操作方法：增、删、改、查

抽取公用的数据处理部分

隔离变化频繁的controller

利用Vuex优化你的model



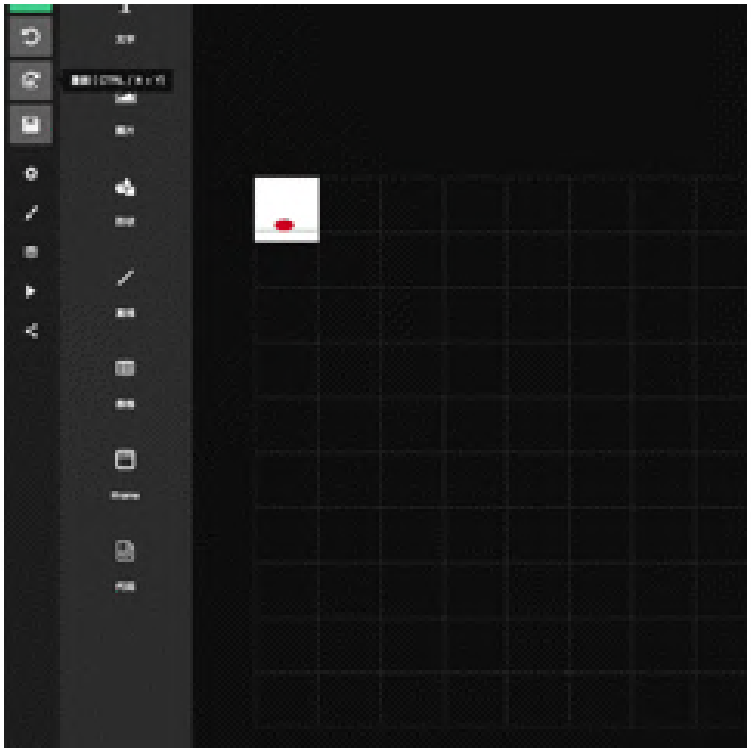
I bug

我们都怕bug

但我们更怕找不到bug

状态机

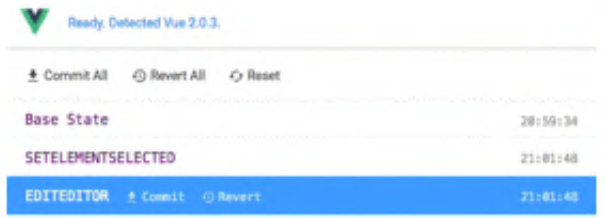
View = Vue(state)



| 打点

$$change = diff(state_m, state_n)$$

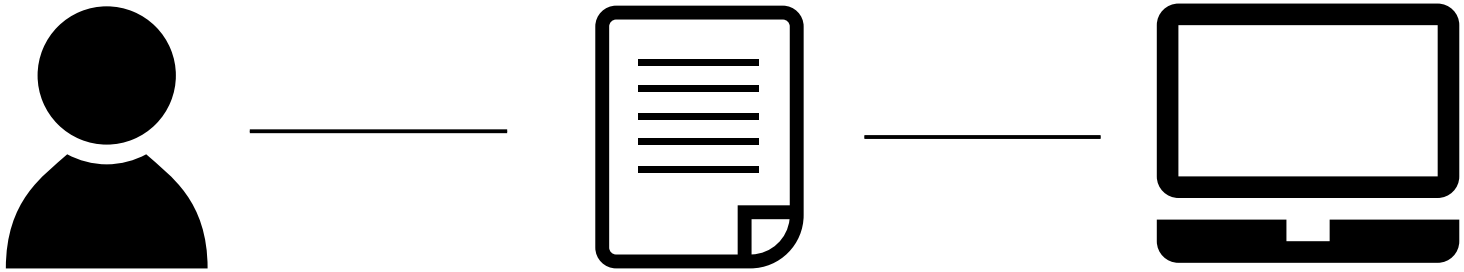
or mutation



提纲

1. **架构 ✔**
   一个MVVM的组件化架构
2. **开发 ✔**
   组件化 ✓
      通信 ✓
      复用 ✓
      耦合 ✓
   MVVM ✓
3. **~~填坑~~优化**

为什么会有坑



开发者　　代码　　浏览器

蜘蛛

读屏器

根据需求选择最适合的开发方式

前端框架的坑

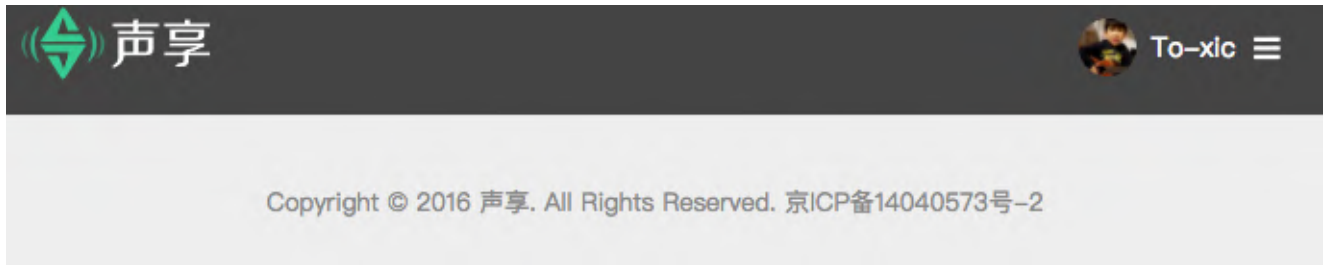
导航	vue-router
首屏体验	lazy-load  Service Worker SSR
SEO	pre-render SSR
老式浏览器	SSR

本质问题

前端JavaScript渲染

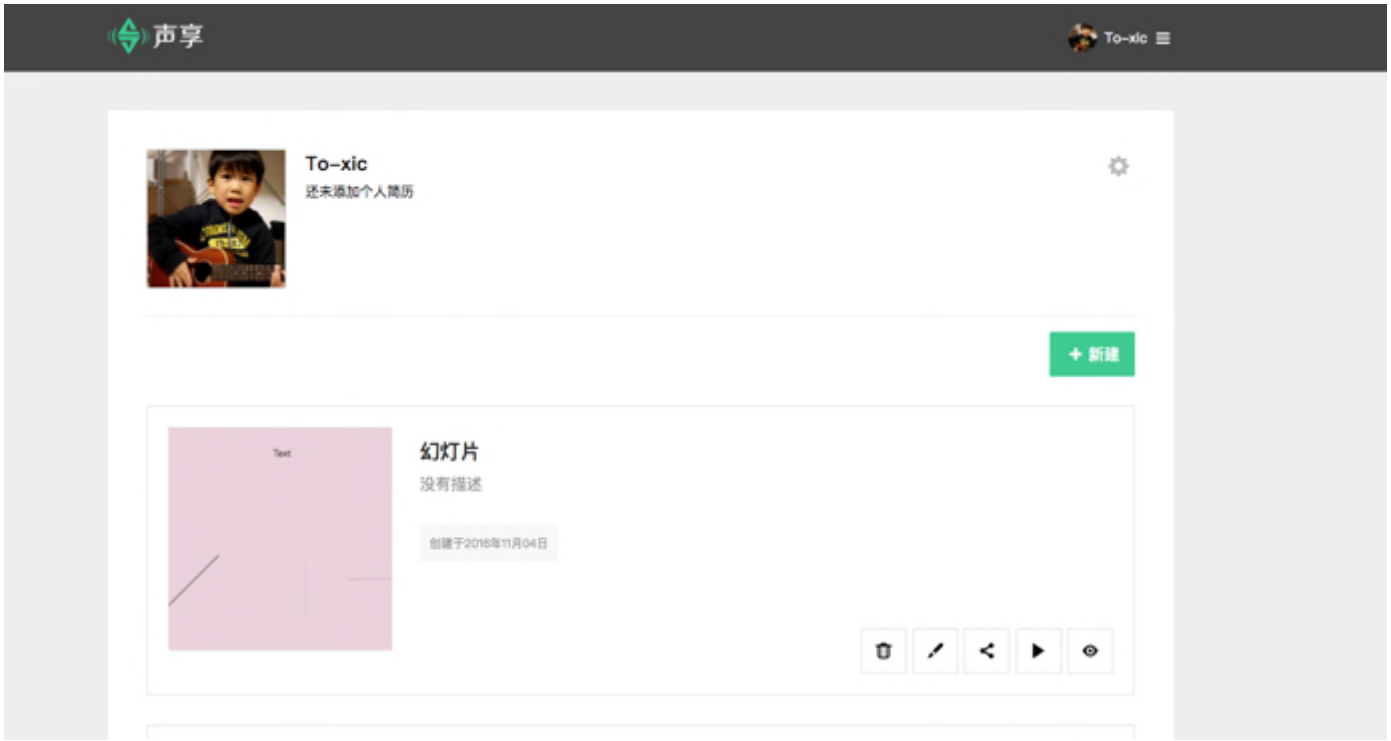
模板 ⟶ 可互动的应用

```
▶ <header class="Header">…</header>
  <mount-vessel></mount-vessel>
▶ <footer class="Footer">…</footer>
```



什么是SSR

模板 —服务端渲染→ 静态HTML —前端渲染→ 可互动应用



| Vue的SSR

```
// the default export should be a function
// which will receive the context of the render call
export default context => {
  return app.preFetch(context).then(() => {
    return app
  })
}
```

```
import path from 'path'
import fs from 'fs'
import * as ssr from 'vue-server-renderer'
import lru from 'lru-cache'

const filePath = path.join(__dirname, '/path/to/your/file')
const code = fs.readFileSync(filePath, 'utf8')
let renderer = ssr.createBundleRenderer(code, {
  cache: lru(1000)
})
renderer.renderToString(options, (err, html) => {
  this.assign({html})
}
```

SSR的性能

```
: it cost 992 to render
```

组件缓存

```
const LRU = require('lru-cache')

const renderer = createRenderer({
  cache: LRU({
    max: 10000
  })
})
```

```
export default {
  name: 'item', // required
  props: ['item'],
  serverCacheKey: props => props.item.id,
  render (h) {
    return h('div', this.item.id)
  }
}
```

自动化首次渲染 ———————— thinkjs的bootstrap执行

SSR对组件的要求

- 前后端均可运行
- 区分静态/动态组件
- 数据彻底分离

提纲

1. **架构 ✔**
   一个MVVM的组件化架构
2. **开发 ✔**
   组件化 ✓
      通信 ✓
      复用 ✓
      耦合 ✓
   MVVM ✓
3. **填坑优化**
   前端框架的坑 ✓
   还能做些什么吗?

WEB应用的痛


未连接到互联网
请试试以下办法：
检查网线、调制解调器和路由器
重新连接到 Wi-Fi 网络
运行网络诊断
ERR_INTERNET_DISCONNECTED

离线化

Page = code(state)

state ——————— localStorage

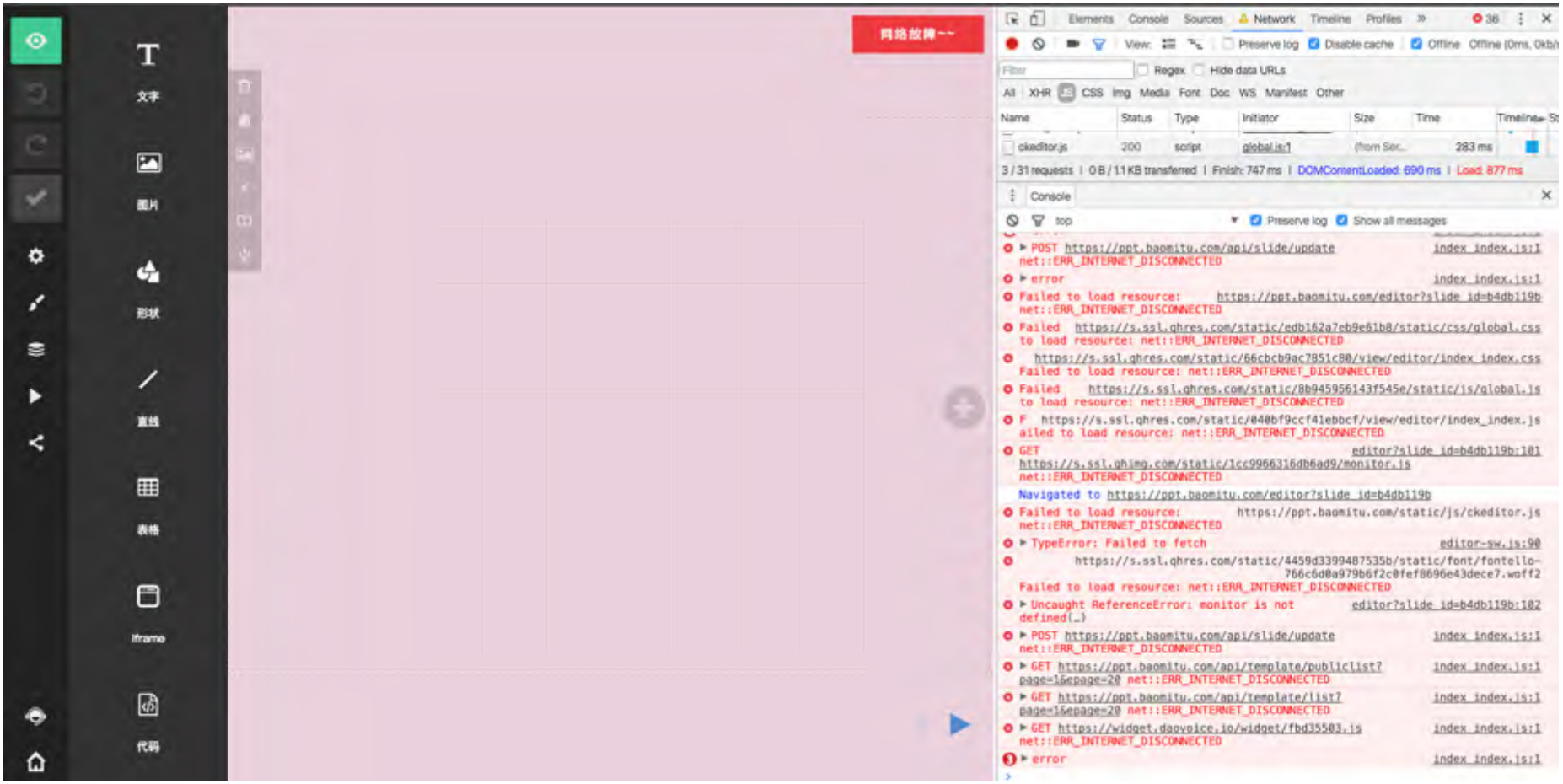
code ——————— Service Worker

离线化

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js')
.then(registration => {
    // success
  }).catch(function (err) {
    // registration failed :(
    console.error(err)
  })
}
```

```
// 监听fetch
self.addEventListener('fetch', function (event)
  let request = event.request
  // 尝试返回线上的数据
  event.respondWith(
    fetch(request)
    .then(function (response) {
      // 若成功，则缓存以备日后使用
      var copy = response.clone()
      caches.open(cacheKeys[1])
      .then(function (cache) {
        cache.put(request, copy)
      })
      return response
    })
    .catch(function () {
      // 若失败则反馈缓存中的部分
      return caches.match(request)
      .then(function (response) {
        return response
      })
    })
  )
}
```

离线化


网络故障~~
文字
图片
形状
直线
表格
iframe
代码
Elements Console Sources Network Timeline Profiles
Preserve log Disable cache Offline
Hide data URLs
All XHR JS CSS Img Media Font Doc WS Manifest Other
Name Status Type Initiator Size Time
ckeditor.js 200 script global.js:1 283 ms
3 / 31 requests | 0 B / 1.1 KB transferred | Finish: 747 ms | DOMContentLoaded: 690 ms | Load: 877 ms
Console
top
Preserve log
Show all messages
POST https://ppt.baomitu.com/api/slide/update net::ERR_INTERNET_DISCONNECTED
error
Failed to load resource: https://ppt.baomitu.com/editor?slide_id=b4db119b net::ERR_INTERNET_DISCONNECTED
Failed https://s.ssl.qhres.com/static/edb162a7eb9e61b8/static/css/global.css to load resource: net::ERR_INTERNET_DISCONNECTED
https://s.ssl.qhres.com/static/66cbcb9ac7851c80/view/editor/index_index.css Failed to load resource: net::ERR_INTERNET_DISCONNECTED
Failed https://s.ssl.qhres.com/static/8b945956143f545e/static/js/global.js to load resource: net::ERR_INTERNET_DISCONNECTED
F https://s.ssl.qhres.com/static/040bf9ccf41ebbcf/view/editor/index_index.js ailed to load resource: net::ERR_INTERNET_DISCONNECTED
GET https://s.ssl.qhimg.com/static/1cc99663l6db6ad9/monitor.js net::ERR_INTERNET_DISCONNECTED
editor?slide_id=b4db119b:101
Navigated to https://ppt.baomitu.com/editor?slide_id=b4db119b
Failed to load resource: https://ppt.baomitu.com/static/js/ckeditor.js net::ERR_INTERNET_DISCONNECTED
TypeError: Failed to fetch
editor-sw.js:90
https://s.ssl.qhres.com/static/4459d3399487535b/static/font/fontello-766c6d0a979b6f2c0fef8696e43dece7.woff2 Failed to load resource: net::ERR_INTERNET_DISCONNECTED
Uncaught ReferenceError: monitor is not defined(…)
editor?slide_id=b4db119b:102
POST https://ppt.baomitu.com/api/slide/update net::ERR_INTERNET_DISCONNECTED
GET https://ppt.baomitu.com/api/template/publiclist?page=1&epage=20 net::ERR_INTERNET_DISCONNECTED
GET https://ppt.baomitu.com/api/template/list?page=1&epage=20 net::ERR_INTERNET_DISCONNECTED
GET https://widget.daovoice.io/widget/fbd35503.js net::ERR_INTERNET_DISCONNECTED
error
index_index.js:1

Q&A

提纲

1. **架构 ✔**
   一个MVVM的组件化架构
2. **开发 ✔**
   组件化 ✓
      通信 ✓
      复用 ✓
      耦合 ✓
   MVVM ✓
3. **~~填坑~~优化**
   前端框架的坑 ✓
   离线化等等