UPYUN OPEN TALK

NO.7 2015.05.23 @成都

iOS开发与前端技术

一拍一产品背后的故事

前端开发过程中的点滴感受

张克军

@kejunz

我的简历

张克军
@kejunz

重构时代	2002. 网页工程师	CSS
	2003. 网页工程师	W3C
	2004. 网页工程师	SEO
Ajax时代	2005. 前端工程师	
	2006. 前端工程师	jQuery
	2007. 前端工程师	YUI
	2008. 前端工程师	Dojo…
	2009. 前端工程师	WPO
工具泛滥的时代	2010. 前端工程师	AMD CMD
	2011. 前端工程师	Backbone Ember Angular …
	2012. 前端工程师	TJ's Tools Node npm
		Browserify Webpack Grunt Gulp …
		React
移动时代	2013. 前端工程师	HTML 5 CSS 3 全栈
	2014. 前端工程师	SVG ES6
X 时代	2015. 前端工程师 / 产品经理 / 设计师 / 创业者 …	

一拍一（ypy.douban.com）是豆瓣旗下女性摄影服务平台，
致力于为女性提供安全、高品质、有爱的摄影服务体验。

2015, 开始找回拍照的意义...

一拍一第一位下单用户: Emma / 摄影师: 大狮视觉

今天谈论的是...

1. 挑战是什么?
2. 源文件和目标文件
3. 我所理解的React
4. 比较项目中用到几种数据处理方式

挑战是
开发中要解决好的关键问题:

1. 控制复杂度
2. 适应变化
3. 文件组织
4. 组件复用
5. 数据处理

基于React的组件架构解决 1 / 2 / 4

多子系统的文件组织和构建针对 1 / 2 / 3

数据处理模式针对 5

维护源文件

static_source

SCSS编译

React (JSX) 编译

支持Node包加载

支持ES6

发布“目标文件”

static

CSS

JS

Bower管理的外部库文件
(构建时copy出编译后的文件)

仅仅是为了克服浏览器的缺陷和标准的滞后

为什么不在发布时编译源文件：时间长，编译工具及版本复杂

```
├── apitests
├── app.py
├── app.yaml
├── cron_handler.py
├── databases
├── dev.yaml
├── libs
├── Makefile
├── model
├── pip-req.txt
├── static
├── static_source
├── templates
├── tests
├── tools
└── view
```

链接到源文件目录下的build目录

```
├── css
│   ├── admin
│   ├── app -> ../../static_source/build/css
│   ├── backyard
│   ├── ext -> ../../static_source/build/ext/css
│   ├── order
│   ├── promotion
│   │   └── app -> ../../../static_source/build/promotion/css
│   ├── story
│   └── weixin_app -> ../../static_source/build/weixin_app/css
├── js
│   ├── admin
│   ├── app -> ../../static_source/build/js
│   ├── backyard
│   ├── ext -> ../../static_source/build/ext/js
│   ├── promotion
│   │   └── app -> ../../../static_source/build/promotion/js
│   └── weixin_app -> ../../static_source/build/weixin_app/js
├── fonts
└── pics
```

- apitests
- app.py
- app.yaml
- cron_handler.py
- databases
- dev.yaml
- libs
- Makefile
- model
- pip-req.txt
- static
- **static_source**
  - **css**
    - admin
    - backyard
    - docs
    - ionicons
    - mods
    - order
    - platform
    - promotion
    - weixin_app
      - mods
      - pages
  - **js**
    - admin
    - backyard
    - mods
    - promotion
    - weixin_app
      - actions
      - dispatcher
      - mods
      - pages
      - stores
  - **bower_components**
  - **node_modules**
- templates
- tests
- tools
- view

- **build**
  - **css**
    - admin
    - backyard
    - docs
    - order
  - **ext**
    - css
    - js
  - **js**
    - admin
    - backyard
  - promotion
    - css
    - js
  - weixin_app
  - css
  - js

子系统分开构建：

```
var gulp = require('gulp');
var requireDir = require('require-dir');
requireDir('./gulpfiles', { recurse: true });

gulp.task('all-js',        ['mainsite-js', 'admin-js', 'backyard-js', 'wx-js', 'promo-js', 'copy']);
gulp.task('all-css',       ['mainsite-css', 'admin-css', 'backyard-css', 'wx-css', 'promo-css'
gulp.task('admin',       ['admin-js', 'admin-css', 'copy']);
gulp.task('backyard', ['backyard-js', 'backyard-css', 'copy']);
gulp.task('wx',            ['wx-js', 'wx-css', 'copy']);
gulp.task('promotion',['promo-js', 'promo-css', 'copy']);
gulp.task('mainsite',    ['mainsite-js', 'mainsite-css', 'copy']);
gulp.task('default',      ['mainsite', 'admin', 'backyard', 'wx', 'promotion']);
```

```
[23:31:20] Finished 'wx-js' after 1 min
[23:31:20] Starting 'wx'...
[23:31:20] Finished 'wx' after 8.36 µs
[23:31:23] Bundling order_list.js
[23:31:32] Bundling album_detail.js
[23:31:39] Bundling order_modify.js
[23:31:48] Bundling home.js
[23:31:56] Bundling pger_index.js
[23:32:05] Bundling order_detail.js
[23:32:10] Bundling pger_services.js
[23:32:19] Bundling order_list.js
[23:32:24] Bundling settlements.js
[23:32:28] Bundling topic_editor.js
[23:32:30] Bundling services.js
[23:32:40] Bundling topic_list.js
[23:32:44] Finished 'backyard-js' after 2.4 min
[23:32:44] Starting 'backyard'...
[23:32:44] Finished 'backyard' after 88 µs
[23:32:46] Bundling trade_list.js
[23:32:48] Finished 'admin-js' after 2.45 min
[23:32:48] Starting 'admin'...
[23:32:48] Finished 'admin' after 15 µs
```

watchify

require-dir

功能分散

需求各异

规模大

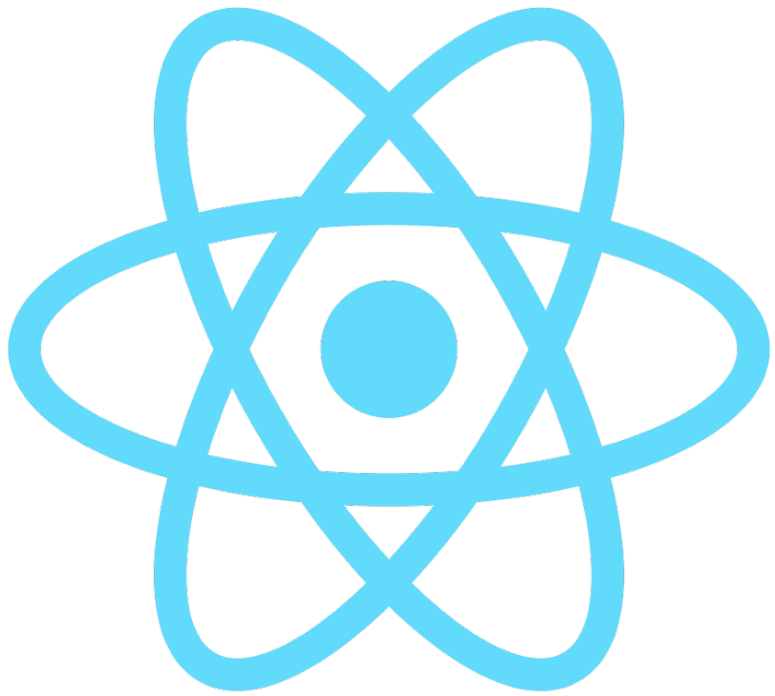
变化快

什么架构能Hold住这种情况?

有些方案(MV*)只适合做好局部

前端工程师长期被OOP和MVC洗脑

FP(函数式编程)和组件架构其实更适合前端开发



React

facebook.github.io/react, 2013年12月发布

“React is intended to help developers build large applications that use data that changes over time. Its goal is to be **simple, declarative** and **composable**. ”

特点:

1. 一个开发UI 组件的框架。（非MV*）
2. 虚拟DOM
   - 合成事件(Synthetic Event)
   - 大批量修改UI，性能高
   - 可以在任何JS Runtime下执行。超越浏览器，Server-Side渲染, React Native …
3. 单向响应式数据流
   - UI响应状态变化自动更新
4. 符合函数式编程思想 （我加的）
   - 描述式、可组合、无副作用

用React创建组件:

1. 组件生命周期

2. 属性和状态

3. JSX语法风格（忘掉模板，全部是组件对象）

Mounting

getDefaultProps()

↓

getInitialState()

↓

componentWillMount

↓

render

↓

componentDidMount

Updating

setProps

componentWillReceiveProps

↓

setState

shouldComponentUpdate

true | false ✕

componentWillUpdate

↓

render

↓

componentDidUpdate

Unmounting

componentWillUnmount

把组件调用看做函数调用：

this.**props** - 组件属性。从外部传入，相当于函数的参数

this.**state** - 组件内部状态。状态：UI状态、数据。状态变化，重新执行render

setState / setProps - 实例修改状态 / 属性的接口。

```
……
render: function() {
    return (
        <ul>
        { this.state.items.map( e => <li>{ e }</li>) }
        </ul>
    )
}
// 更新
this.state.items.push(5);
this.setState({
  items: this.state.items
});

// 排序
this.setState({
  items: this.state.items.sort()
});
```

HTML:

```
<ul class="list"></ul>
```

JS:

```
var list = $('.list');
items.forEach(e => {
  list.append('<li>' + e + '</li>');
});

// 更新
list.append('<li>5</li>');


// 排序
list.html('');
items.sort().forEach(e => {
  list.append('<li>' + e + '</li>');
});
```

需求变了，UI结构调整，比如ul-li 要改成用 div 实现了。描述式和可预测性在复杂应用中作用非常显著。

JSX， 是一种类XQuery的语法(http://www.w3.org/TR/xquery-30/)。需要经过react-tools编译。

XQuery

```
<ul>
{
    for $x in doc("books.xml")/bookstore/book/title
    order by $x
    return <li>{ data($x) }</li>
}
</ul>

输出结果:
<ul>
    <li>Everyday Italian</li>
    <li>Harry Potter</li>
    <li>Learning XML</li>
    <li>XQuery Kick Start</li>
</ul>
```

Scala原生支持类XQuery的语法:

```
val marshalled =
  <music>
  { artists.map { artist =>
    <artist name={artist.name}>
    { artist.albums.map { album =>
      <album title={album.title}>
      { album.songs.map(song => <song title={song.title} length={song.length}/>) }
      <description>{album.description}</description>
      </album>
    }}
    </artist>
  }}
</music>
```

不要和模板联系起来。React里没有模板。

传统javascript模板：

```
var headline = ‘<h1>Welcome, {name}</h1>’;
```

React：

```
var headline = <h1>Welcome, {name}</h1>;
```

```
var headline = React.DOM.h1({}, ‘Welcome’, name);
```

组件就是函数

如：add(add(1,2), 3) // 把1,2,3换成组件

函数式编程（FP）

“In computer science, functional programming is a pro
gramming paradigm that treats computation as the ev
aluation of mathematical functions and avoids state a
nd mutable data.”

“函数式编程是一种编程范式，它将运算看做是数学中函数的计算，并且避免了状态以及可变数据"

函数式编程（FP）

1. 函数是一等公民，函数和其它数据类型一样。高阶函数
2. 只用表达式，不用语句
3. 无副作用。所有调用返回新值，不修改外部变量
4. 引用透明。不依赖外部变量，只依赖输入参数，参数相同，返回结果相同

函数式编程（FP）

1. 复用性强，可组合（很硬）
2. 语义性强，描述式（很硬）
3. 便于单元测试
4. 适合并发编程

作品信息

彻底删除

订单号: [redacted]

浏览量: 1234

标题: 儿童摄影 [编辑]

标签: 儿童 清新 [编辑]

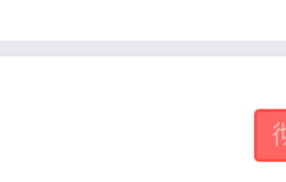
描述: 儿童 [编辑]

是否展示: 是 否

照片列表

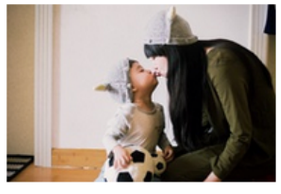
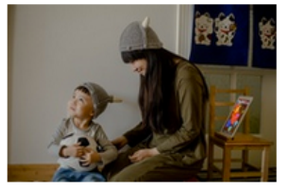
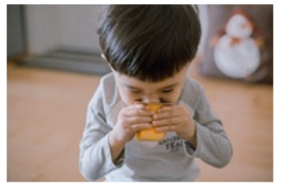
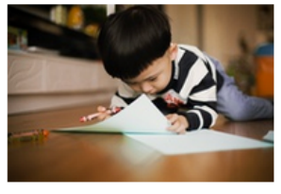
确保已征求照片中人的同意公开其照片

添加/移除照片







React.render(<**EditAlbumBasicInfo** data={ albumInfoData } />, $('#album-basic-info')[0]);

React.render(<**PhotoList** data={ albumPhotoData } />, $('#album-photos')[0]);

可以很直观的看到UI全貌
描述式带来的可预性的好处
可维护性很强

```
<div>
  <div className="photo-list">
  <ul>
  {
    data.photos && data.photos.length ? data.photos.map( (photo, i) =>
            <PhotoListItem key={i} data={ photo } />
          : <li className="blank">(尚未添加照片)</li>
  }
  </ul>
  </div>
  {this.state.currentPhoto  && <PhotoPreview data={ this.state.currentPhoto } /> }
</div>
```

复杂的部分被抽象成组件
只关心传入属性和返回结果
复用性很强

组件是引用透明的。不受外部影响，属性相同，返回总是相同的。（完全做到还依赖不可变数据对象）

高阶组件，组件作为属性传给组件，组成一个新组件

EditAlbumBasicInfo({ data: albumInfoData }, mountNode)

PhotoList({ data: albumPhotoData }, mountNode)

```
React.DOM.div({},
  React.DOM.div({ className: ‘photo-list’},
    React.DOM.ul({}, data.photos.map( (photo, i) => { PhotoListItem({…}) }),
    this.state.currentPhoto && PhotoPreview({ data: this.state.currentPhoto })
  )
)
```

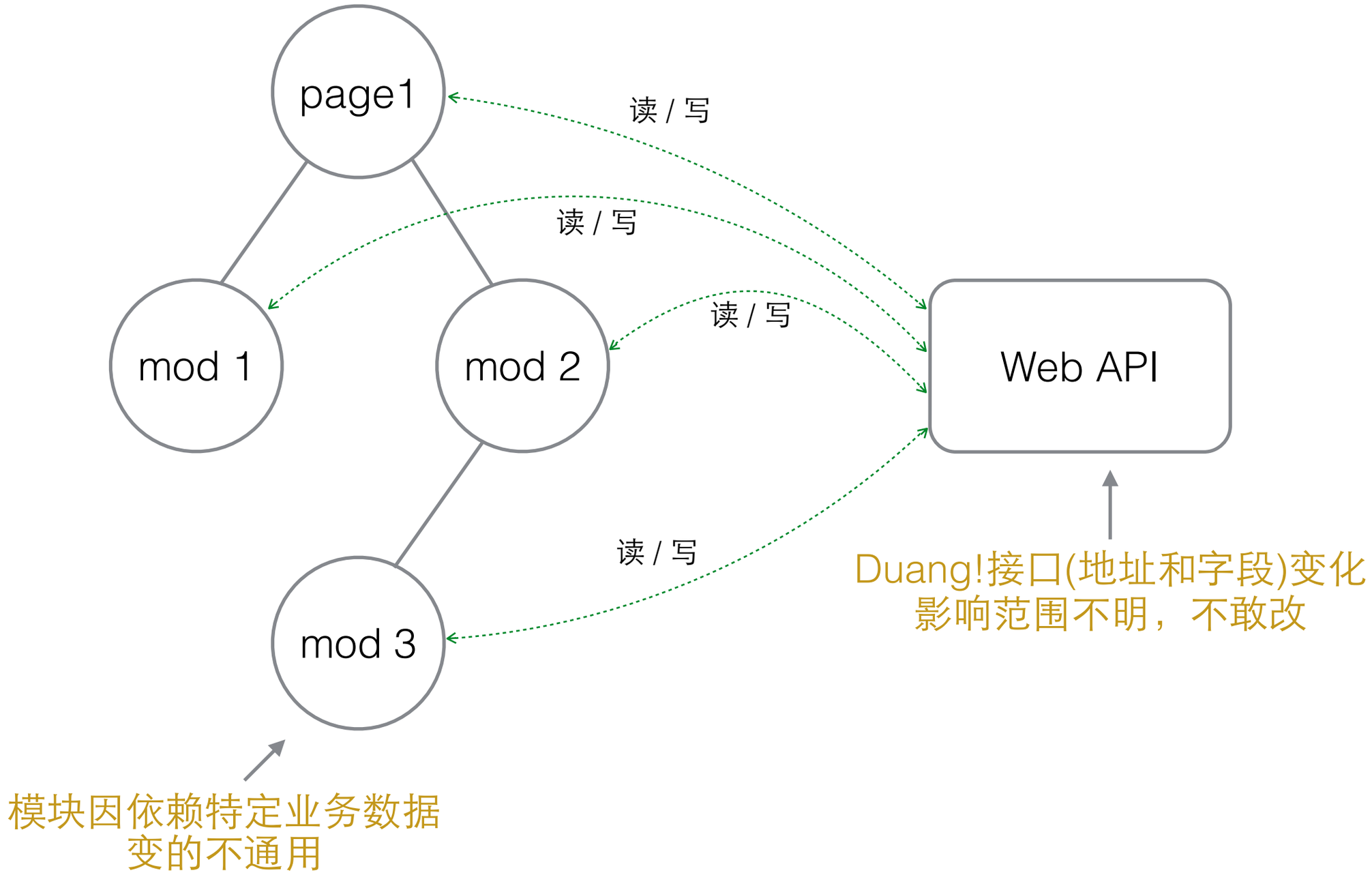
如同高阶函数

```
function( {}, function( {}, function( {}, function(…), function(…) ) ) )  // 返回一个新组件
```

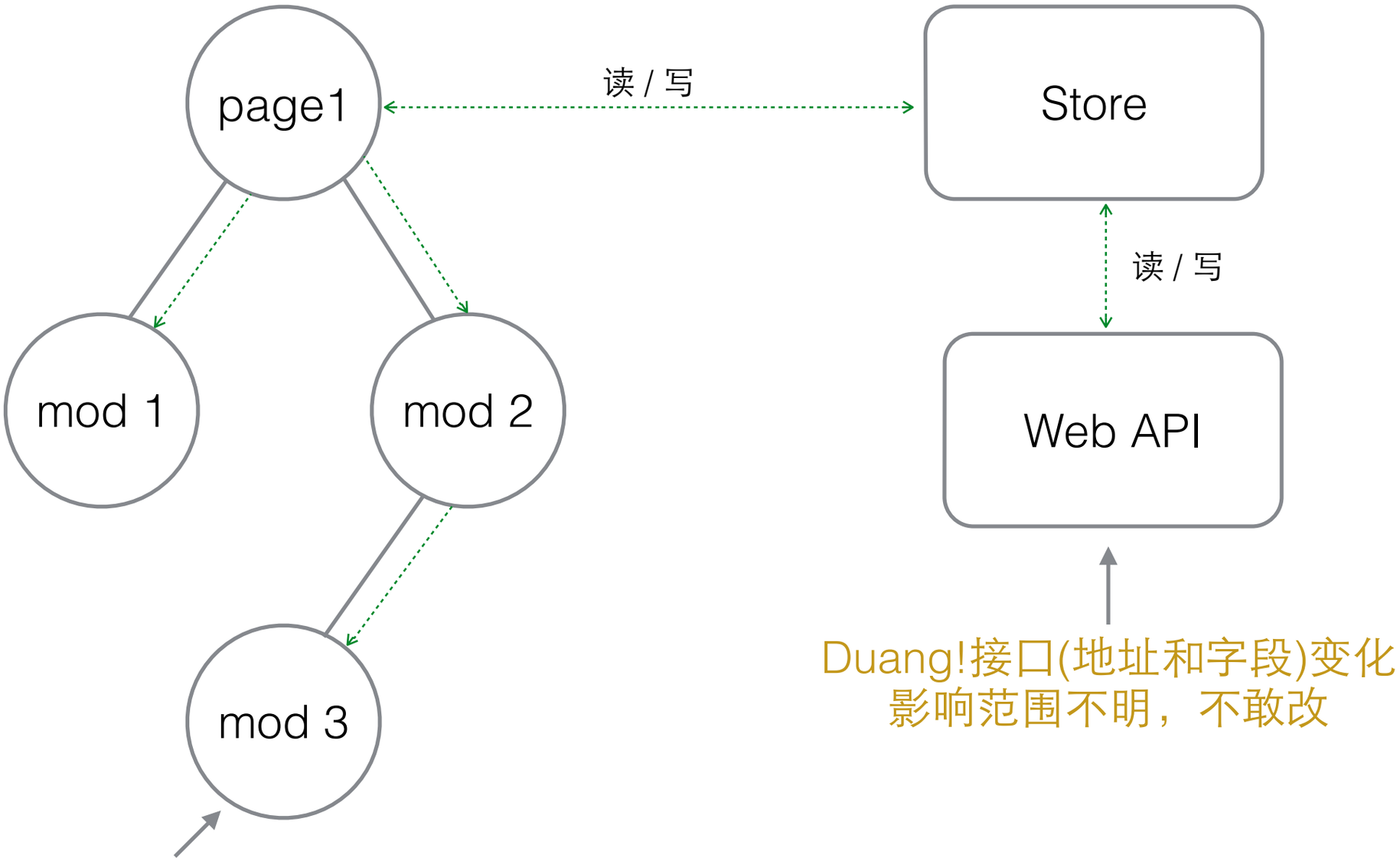
组件设计思路：

道生一，一生二，二生三，三生万物

数据处理

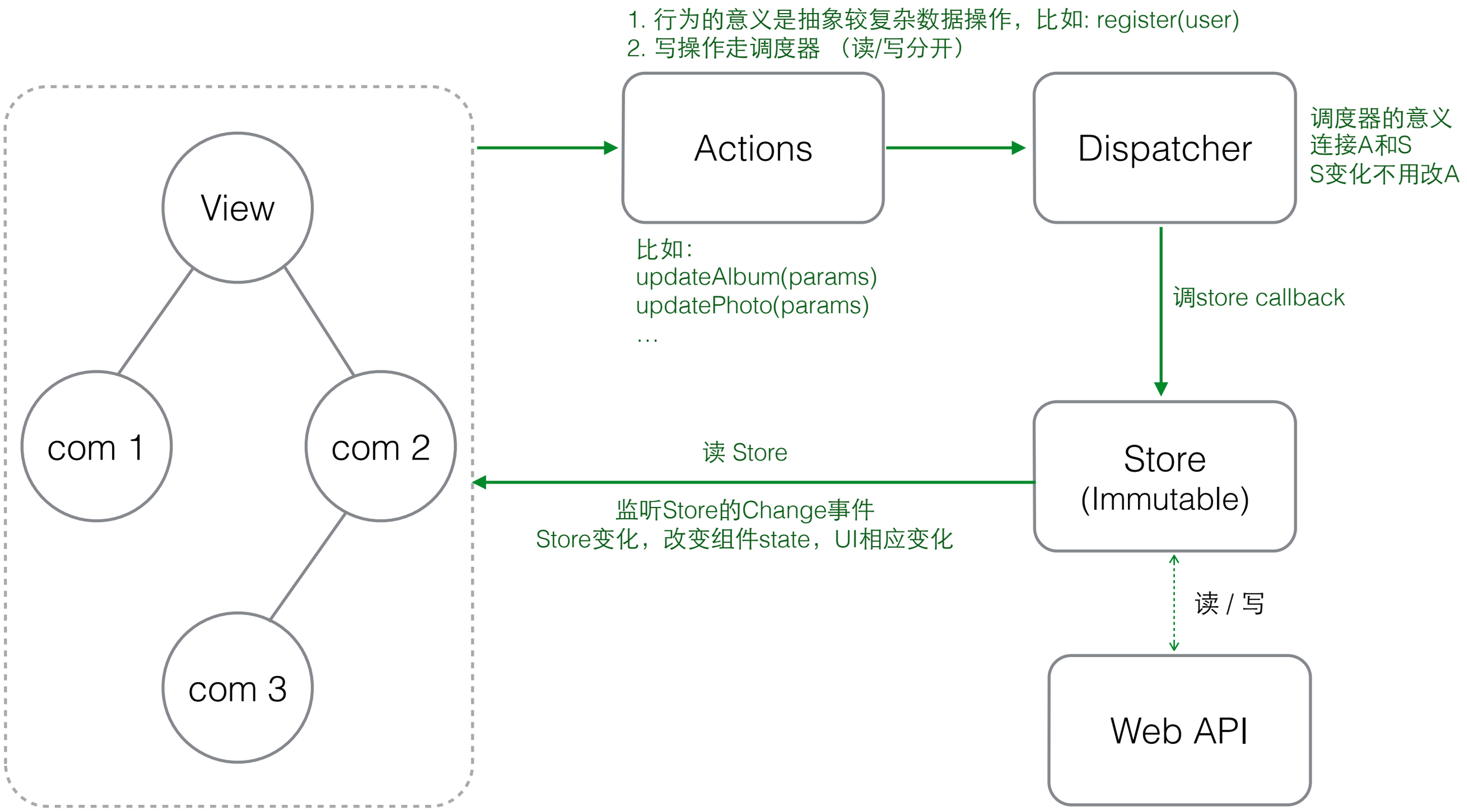

page1
读 / 写
读 / 写
读 / 写
mod 1
mod 2
Web API
读 / 写
mod 3
Duang!接口(地址和字段)变化
影响范围不明，不敢改
模块因依赖特定业务数据
变的不通用

Store改一下就好了




```
<TagInput value={[…]} onSave={ handler } />
```

Flux 一种数据处理模式。单向响应式数据流



Flux 写操作多时，强烈建议用Immutable的Store

js/stores/AppStore.js:

```
var _store = Immutable.fromJS({
  quizs: null,
  userChoices: null,
});
```

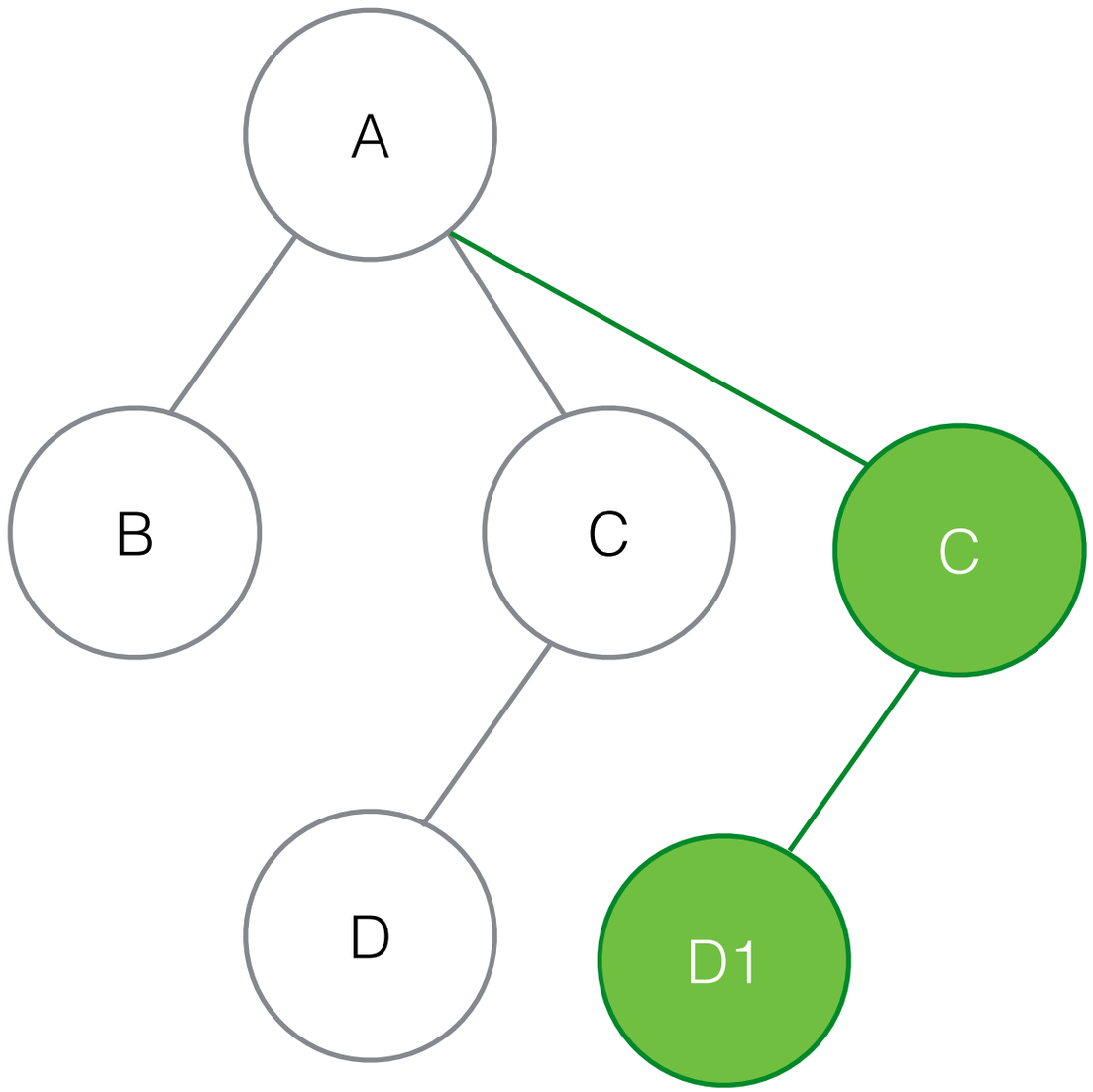
```
StoreApis[Constants.GET_ALL_QUIZS] = function(params={}) {
  if (!params.isforce && _store.get('quizs')) {
    return _store.get('quizs').toJS();
  }
  fetchAllQuizs(params);
};
```

IMMUTABLE

facebook.github.io/immutable-js/

A = Immutable.fromJS({ B:{}, C:{ D: {} } })

A = A.set('C', D1);



List

Stack

Map

OrderedMap

Record

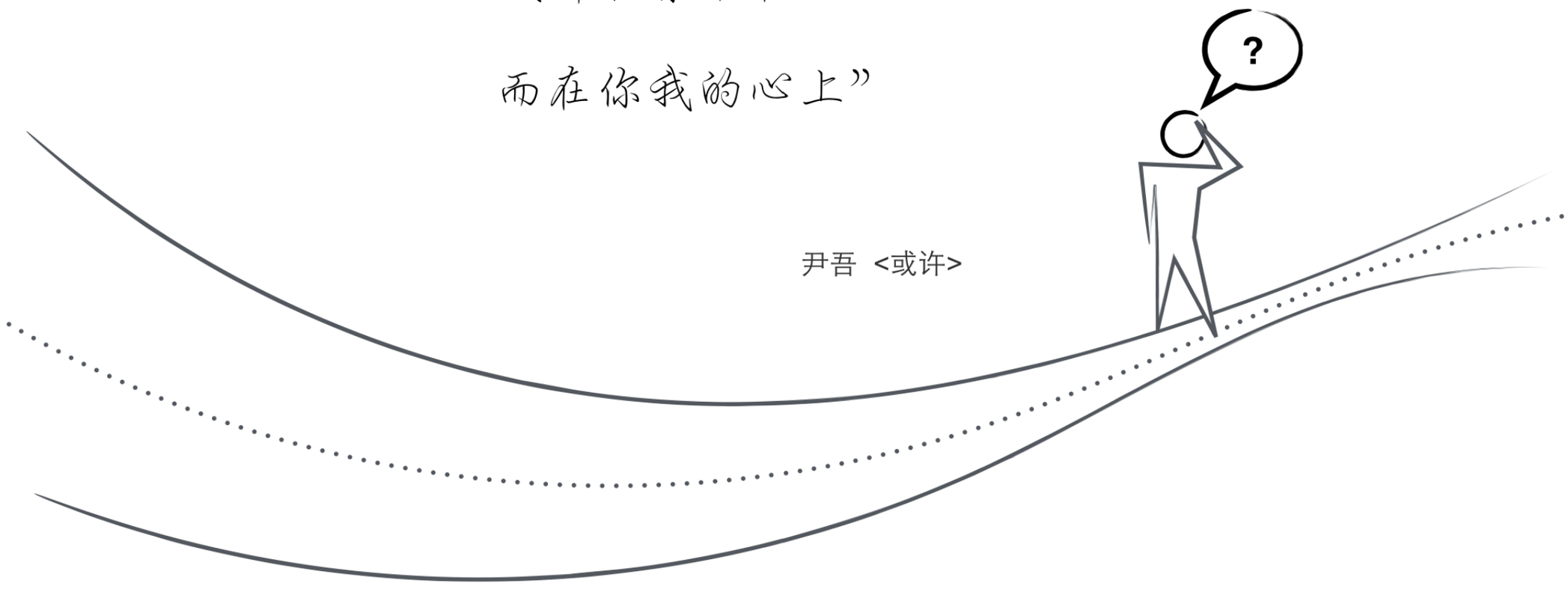
“或许我们追求了一生

仍要从追求本身寻找

或许答案不在远方

而在你我的心上”

尹吾 <或许>



<Thanks! />