用Vue做个仿豆瓣项目

中级前端培训

目录

- 培训流程
- 搭脚手架
- 跨域问题，不使用jsonp
- 目录结构划分
- 组件的组织方式
- 如何组织路由
- Vuex的最佳实践
- Vuex与表单
- 分层管理的概念：api与其他公共库的管理问题
- 示例douban项目有哪些跟培训内容不一致的?

培训流程

豆瓣示例项目： https://github.com/jeneser/douban

- 培训前先了解仿豆瓣示例项目，并仿着做一个，要求达到类似的呈现效果，但不要照抄或者只是浏览源码，完成首页、活动详情页、搜索、登陆；
- 培训过程中结合豆瓣项目讲解培训内容，并当场实操，把项目改造成符合培训内容的要求；
- 培训结束后一周内，继续完成仿豆瓣项目，达到跟示例项目接近的呈现效果，然后上传到github以供审查；

搭建脚手架

- 初始化:vue init webpack vue-douban
- 脚手架开发环境要求：
  - ES6（已支持）
  - eslint,采用standard要求（支持配置）
  - git commit时要求自动执行lint(需要安装pre-commit，并修改package.json)
  - vue文件要求支持scss(需要安装sass-loader,node-sass)
  - 区分开发环境与生产环境（已支持）
  - 生产环境下应该自动清除调试数据，即禁止console.log(需要改配置)
  - 图片开启自动压缩(imagemin-webpack-plugin)
- 练习
  - git commit确认pre-commit生效;
  - 确认生产环境下的console.log被禁止输出
  - 图片开启自动压缩

搭建脚手架

- 脚手架框架要求：
  - 使用normalize.css归一化样式;
  - 使用element ui做为基础样式库;
  - 基于element ui的主题库，生成符合自己主题色的样式库；
  - CSS要求符合BEM规范;
  - 使用lodash作为工具库；
  - 使用vue-lazyload对所有图片懒加载;
  - 符合最佳实践的vuex(后续讨论）；
  - 符合最佳实践的vue-router（后续讨论）；
- 练习：
  - 豆瓣项目修改element的主题色为#00B600
  - 豆瓣标题颜色引用element的变量文件$—color-primary;

跨域问题，不用jsonp

- 1.在config/index.js的dev下

```
diff --git a/config/index.js b/config/index.js
index 926ab36..fdc2567 100644
--- a/config/index.js
+++ b/config/index.js
@@ -40,7 +40,16 @@module.exports = {
     //https://vue-loader.vuejs.org/en/options.html#cachebusting
     cacheBusting: true,

-    cssSourceMap: true
+    cssSourceMap: true,
+    proxyTable: {
+        '/api': {
+            target: 'https://api.douban.com',
+            changeOrigin: true,
+            pathRewrite: {
+                '^/api': '/v2'
+            }
+        }
+    }
   },
```

- 2.https://api.douban.com/v2/event/list->/api/event/list

目录结构划分(理解透彻)

```
src/
├── App.vue
├── assets
|   ├── element-variables.scss # 主题的变量，为全局可引用
|   ├── logo.png
├── components
|   ├── Activity.vue
|   ├── Banner.vue
|   ├── Header.vue
|   ├── Loading.vue
|   └── SubNav.vue
├── layouts # 模板文件
|   ├── Default.vue
|   └── Plain.vue
├── lib # 见分层的讨论
|   └── api.js # api必须集中封装
├── main.js
├── pages
|   ├── ActivityDetail.vue
|   ├── Book.vue
|   ├── Broadcast.vue
|   ├── Group.vue
|   ├── Home.vue
|   ├── Login.vue
|   └── Movie.vue
├── router
|   └── index.js
└── store # vuex 必须分模块
    ├── actions.js # 外部与vuex的所有支持的操作都必须在actions中规定
    ├── index.js
    └── modules
        ├── activities.js
        ├── books.js
        ├── broadcasts.js
        ├── groups.js
        └── movies.js
```

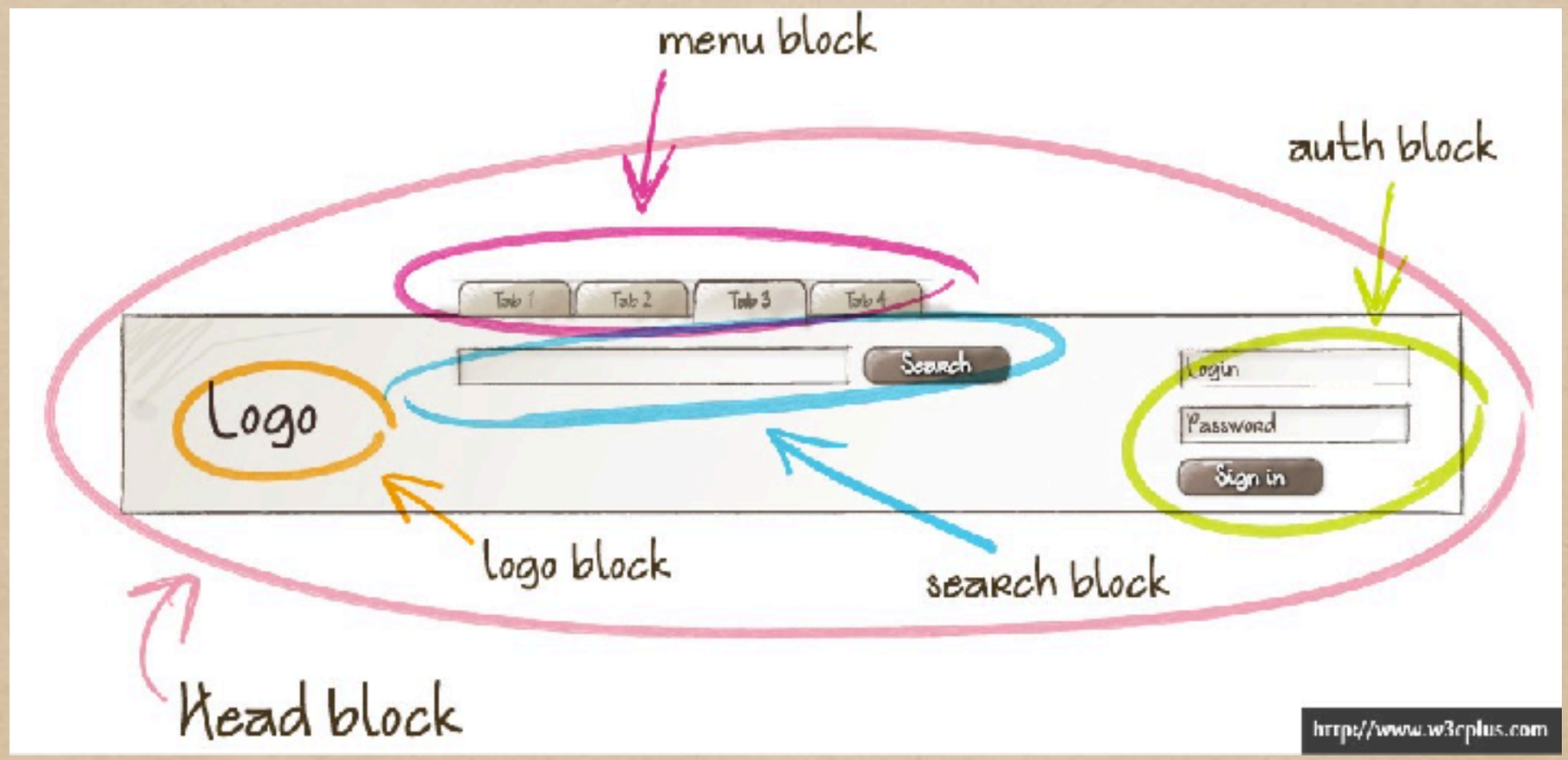
组件的组织方式

- 按照基础组件库(element ui), components, layouts, pages 的方式去划分；
- 组件的核心概念:生命周期、数据驱动、无副作用。
- 组件与指令的区分：有无副作用?
- 为什么我们反复强调重视组件化？因为跟设计概念体现了一一致性（结合下页的图文理解）
- 组件化的理论与实践上的冲突

组件副作用的含义：是否可以作为一个独立节点插入到DOM树中且不影响其他DOM节点，还是作为其他节点的一部分或者影响了其他节点?

组件的组织方式

- BEM与组件的概念



如何组织路由

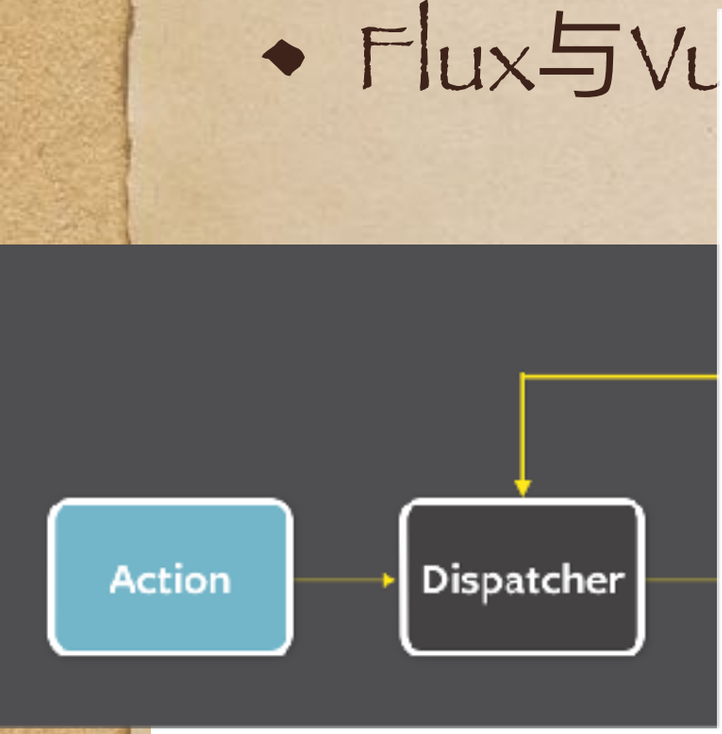
- 基于模板的路由组织

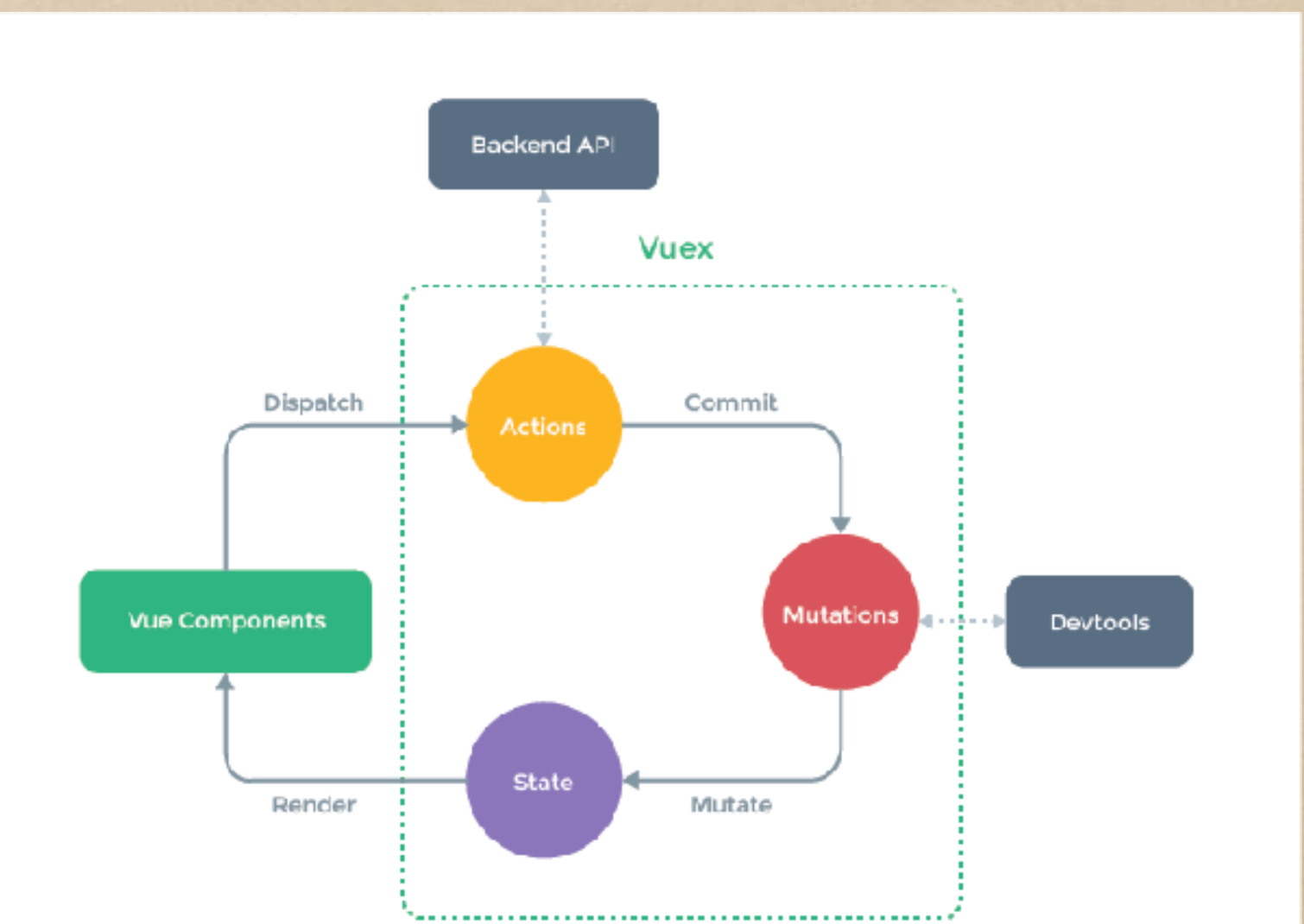
```
import Default from ‘@/layouts/Default’
import Login from ‘@/pages/Login'
export default new Router({
  mode: 'history',
  routes: [
    {
      path: '/',
      component: Default, // Default为默认模板文件
      children: [
        {
          path: '',
          name: 'Home',
          component: Home
        },
        {
          path: 'books',
          name: 'Book',
          component: Book
        },
        {
          path: 'activities/:id',
          name: 'ActivityDetail',
          component: ActivityDetail
        }
      ]
    },
    {
      path: '/login',
      name: 'Login',
      component: Login // 登陆页面未使用模板
    }
  ]
})
```

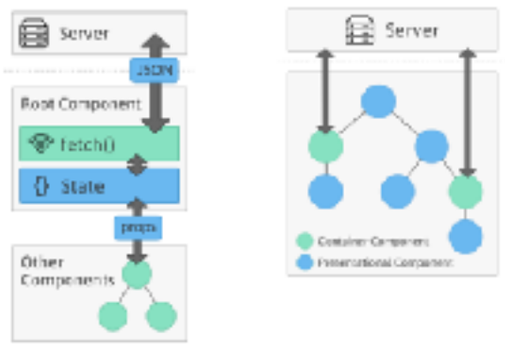
练习:建一个html文件，直接原生JS测试history API实践：前进、后退、替换

Vuex的最佳实践

- Flux与Vu







React异步请求演化阶段

Vuex的最佳实践

- action, mutation的区分
- 异步请求必须在action中处理(对于中小型应用其实过于繁琐，所以允许组件内请求)
- 异步请求的一个action对应三个GET_DATA, mutation(*_PENDING,*_FULFILLED,*_REJECTED)
- dispatch返回值的归一化处理(promise)
  - 只允许dispatch，不允许commit，同时所有的action都必须规定常量，不能直接引用字符串。
- 为什么我们要做这些约束，这就需要通过下一节的软件分层概念解释?

Vuex的最佳实践

- actions.js的示例文件：

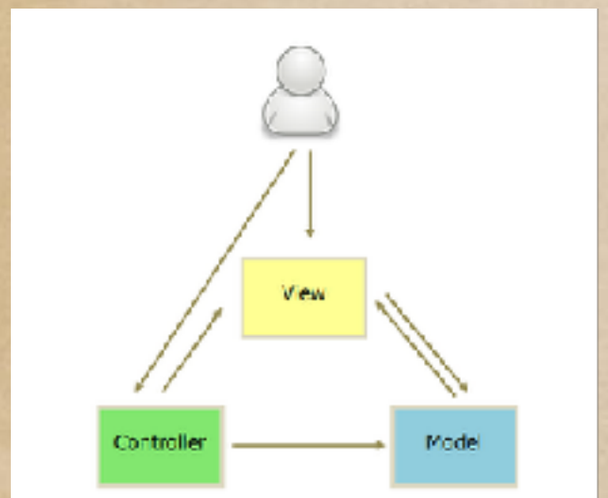
```
export const QUERY_EVENT = 'QUERY_EVENT'
export const GET_EVENT = 'GET_EVENT'

export const createAsyncAction = (type, fn) => async ({commit, state}, payload) => {
  commit({
    type: `${type}_PENDING`,
    params: payload
  })
  try {
    let res = await fn({commit, state}, payload)
    commit({
      type: `${type}_FULFILLED`,
      data: res.data,
      params: payload
    })
    return res.data
  } catch (e) {
    commit({
      type: `${type}_REJECTED`,
      params: payload
    })
    throw e
  }
}
```

Vuex与表单

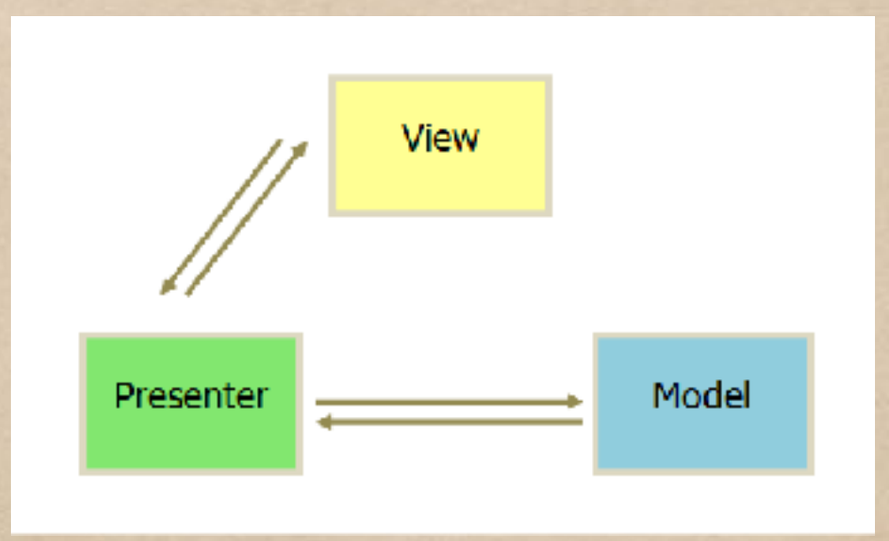
- 单向数据流与双向绑定之间的冲突
- 解决方法：表单仍然使用双向绑定，只从vuex获取初始化数据;所有操作都在组件内部完成，要触发数据改变时将双向绑定的内容通过dispatch传递出去。
- element ui的form问题：没有提供实时判断表单有效

分层管理的概念

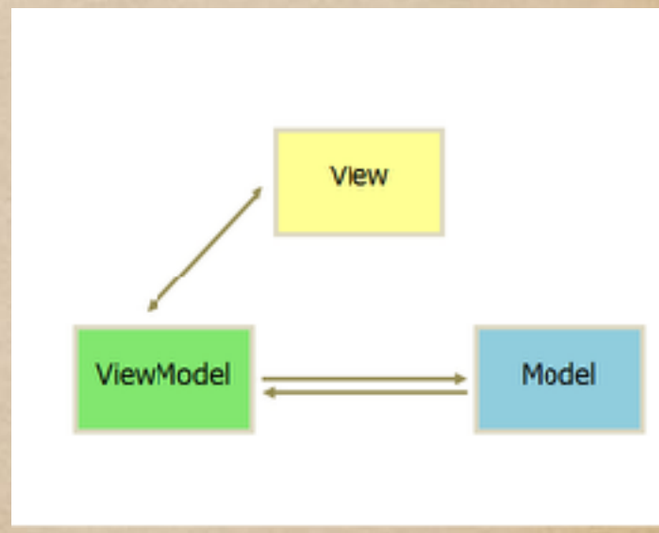
- 复杂的软件必须有清晰合理的架构，否则无法开发和维护。——阮一峰



MVC



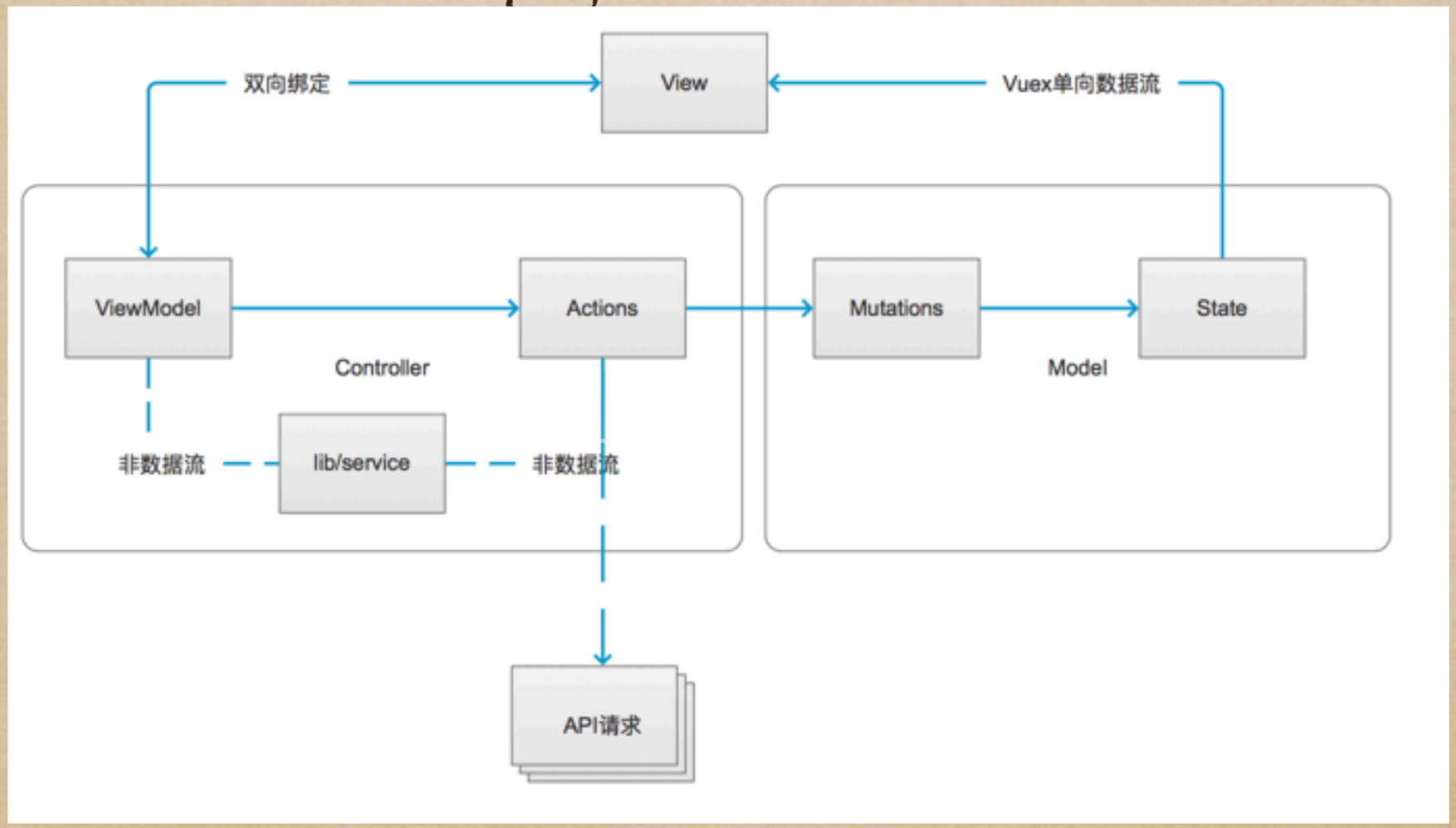
MVP



MVVM

分层管理的概念

- api调用出现理论与实践的冲突问题（目前只要求集中到lib/api.js，不要求全部放在action）



示例douban项目有哪些与培训内容不一致的?

- 跨域问题使用了jsonp
- 目录结构划分不一致，主要体现在组件的组织上
- Vuex的最佳实践上不一致
- 分层管理不一致

THE END!