

目录

[第一部分 产品框架 1](#_Toc480535743)

[1.1 总体概述 1](#_Toc480535744)

[1.2 后端技术 1](#_Toc480535745)

[1.2.1 后端概述 1](#_Toc480535746)

[1.2.2 主要技术框架的优势 4](#_Toc480535747)

[1.3 前端技术 7](#_Toc480535748)

[1.3.1 前端概述 7](#_Toc480535749)

[1.3.2 安卓前端技术策略 8](#_Toc480535750)

[1.3.3 React-native跨平台开发 10](#_Toc480535751)

[第二部分 算法设计 13](#_Toc480535752)

[2.1 特色算法与技术 13](#_Toc480535753)

[2.2 数据存储 14](#_Toc480535754)

[2.3 技术难点与解决方法 14](#_Toc480535755)

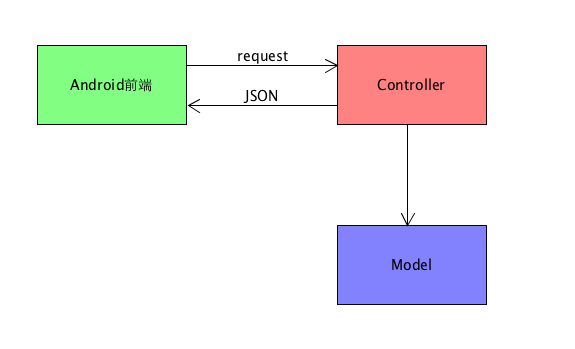
# 第一部分 产品框架

## 1.1 总体概述

为了后续迭代拓展开发以及针对不同的终端开发定制版本的简便，uKnow开发过程采用了**前后端分离**的开发模式。

不同于传统的MVC模式，前后端分离实现的是前端发送请求，后端接受该请求并返回JSON数据给前端，最后前端利用数据进行界面渲染。采取前后端分离使得在开发过程中能够实现前后端**松耦合**，开发效率高。

前后端分离模式下前后端交互如下：



## 1.2 后端技术

### 1.2.1 后端概述

在前后端分离模式下，前端关注界面展现，后端关注业务逻辑，分工明确，职责清晰，要实现这种方式，不得不提**REST**服务。

REST，即Representational State Transfer，表述性状态转移。REST是使用URL来访问资源的方式的Web服务。它是一个“**无状态**”的架构模式，因为在任何时候都可以由客户端发出请求到服务端，最终返回自己想要的数据，当前请求不会受到上次请求的影响。也就是说，服务端将内部资源发布REST服务，客户端通过URL来访问这些资源。REST是**SOA**面向服务思想的一种实现，它提供了包括GET、POST、PUT、DELETE、HEAD、OPTIONS等请求方式。

在uKnow后端开发过程中，我们也设计了符合**RESTful**风格的后台API，使得前端开发人员不必关注接口的具体实现，而只需注意其逻辑使用。总的接口设计如下：

|  |  |
| --- | --- |
| 推荐页面 | /api/questions/recommend |
| 点击问题后进入的问题详情 | /api/questions/:question\_id |
| 点击头像跳转到提问页面 | /api/users/:user\_id/introduction |
| 收听 | /api/questions/:question\_id/listenings |
| 评价 | /api/questions/:question\_id/comments |
| 搜索感兴趣的问题 | /api/questions/find |
| 找人模块页面 | /api/users/:user\_id/follows  /api/users/:user\_id/recommendations |
| 关注功能 | /api/users/:user\_id/follows |
| 找人功能 | /api/users/find |
| 提问(创建问题) | /api/questions |
| “我的”模块 | /api/users/:user\_id |
| 修改信息 | /api/users/:user\_id |
| 回答问题 | /api/questions/:question\_id/answer |
| 登录后完善信息 | /api/users/:user\_id/perfect |
| 微信登录 | /api/wxlogin |
| token登录 | /api/tklogin |

每个接口都详细说明了接口功能、接口地址、接口参数、接口返回值以及接口请求方法，具体如下：

|  |  |
| --- | --- |
| **接口功能** | **修改信息** |
| 接口地址 | /api/users/:user\_id |
| 接口参数 | {  “username”: “用户名”,  “status”: “用户身份”,  “description”: “用户描述”,  “school”: “学校”,  “major”: “专业”,  “grade”: “年级”,  “avatar”: 图片表单数据  } |
| 接口返回值 | {  “status”: 200(为OK)/500(为ERROR)  “data”: {},  “errmsg”: “status=500时返回错误信息”  }  注：  当status=200，没有errmsg信息 |
| 请求方法 | PATCH |

根据上面的后端接口文档，后端开发只需要实现所有接口即可，不必关注接口何时调用以及怎么调用，开发效率明显提高。

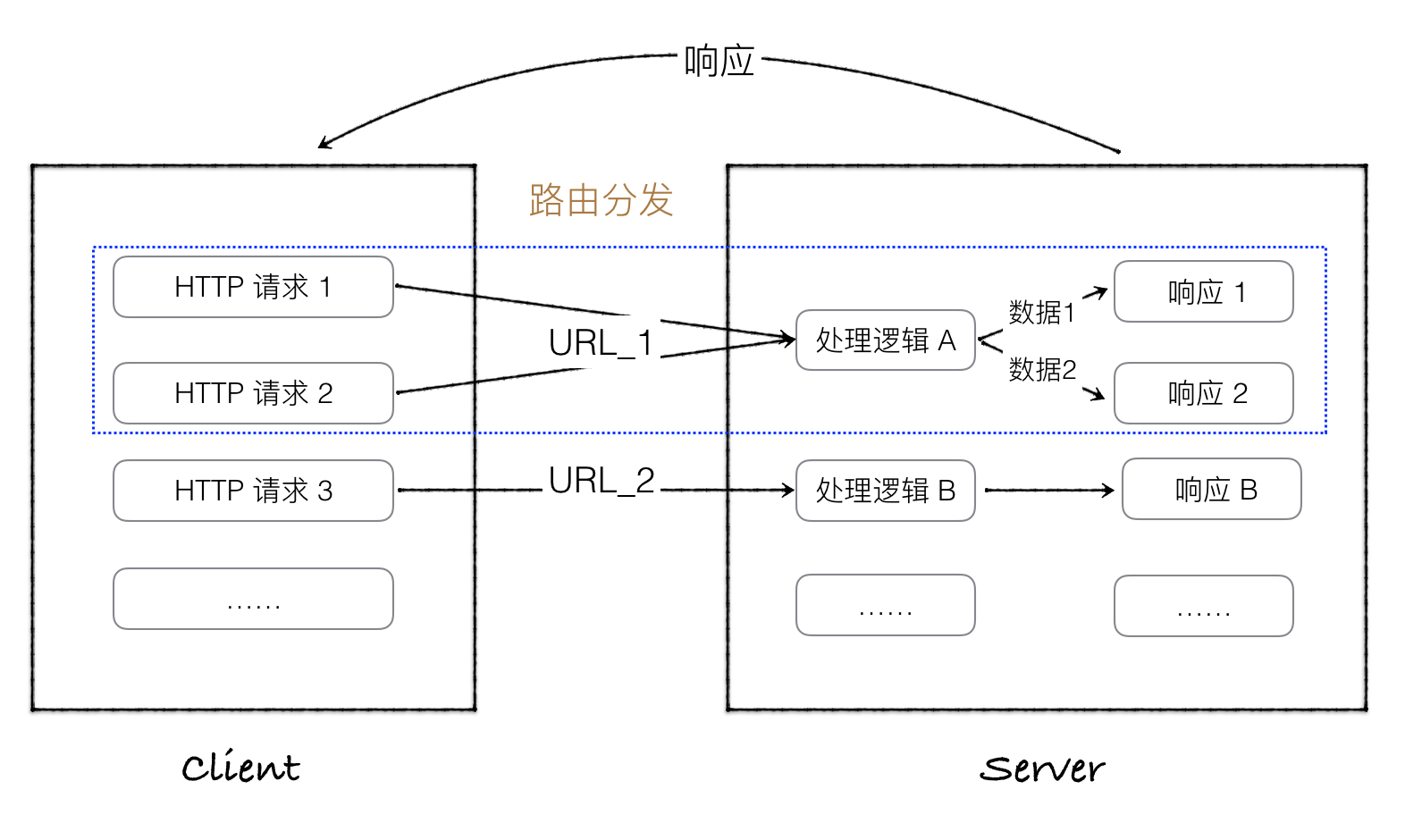
而后端的实现，uKnow总体是采用python的轻量级**Flask**框架编写，同时使用**MySQL**作为数据库，性能较高的**Gunicorn**、**Nginx**服务器处理相应请求。总体技术框架如下：

|  |  |
| --- | --- |
| **技术** | **功能** |
| Flask | 整个后端项目的框架，实现controller以及业务功能 |
| MySQL | 开源的关系型数据库 |
| Nginx | 作为负载均衡服务器和反向代理服务器，做静态资源缓存 |
| Gunicorn | 管理多进程的服务器 |
| 结巴分词 | 对用户输入进行分词，为推荐和搜索功能服务 |
| 微信登录 | 第三方登录，实现快速登录 |

### 1.2.2 主要技术框架的优势

·**Flask**

Flask是一个用python编写的轻量级的Web应用框架，配置使用非常简单，其处理请求过程如下：

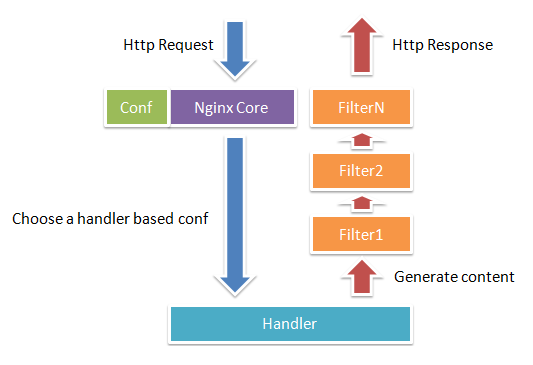


Flask使用route()装饰器把一个函数绑定到对应的URL上，通过路由分发将客户端的请求交由不同的函数处理，最后返回数据或者利用模板引擎渲染页面。

作为一个微框架，Flask非常简洁优雅，语法简单，部署很方便。Flask支持用扩展来给应用添加功能，众多的扩展提供了数据库集成、表单验证、上传处理、各种各样的开放认证技术等功能。Flask的路由使用一个简单但有力的模式语法，映射URLs到代码，使后端实现起来极其简单。Flask通过Blueprint来提供模块化，自己对项目结构划分成不同的模块进行组织。Flask的配置很灵活，有多种方法配置，不同环境的配置也非常方便。

·**Nginx**

Nginx是一个高性能的HTTP和反向代理服务器，是一款面向性能设计的HTTP服务器，其处理请求过程如下：



Nginx服务器的核心是Nginx Core，其通过接收客户端请求，然后根据配置分发到相应处理器，最后通过多层过滤器将结果返回。

后端使用Nginx作为负载均衡服务器和反向代理服务器，做静态文件缓存，占有内存少，并发能力强。Nginx非常轻量级而且占用更少内存及资源，同时支持高并发。高并发下Nginx依然能保持低资源低消耗高性能处理请求，因为其采取的的方式是异步非阻塞。而且，Nginx高度模块化，设计编写模块相对简单，文件配置也较为简便。

·**MySQL**

MySQL是一个开源的关系型数据库管理系统。MySQL支持多种存储引擎，提供用于管理、检查、优化数据库操作的管理工具，同时提供多语言支持，支持多线程，充分利用了CPU资源。MySQL支持大型的数据库，可以方便地支持上千万条记录的数据库，其拥有一个非常快速而且稳定的基于线程的内存分配系统，可以持续使用而不必担心其稳定性。

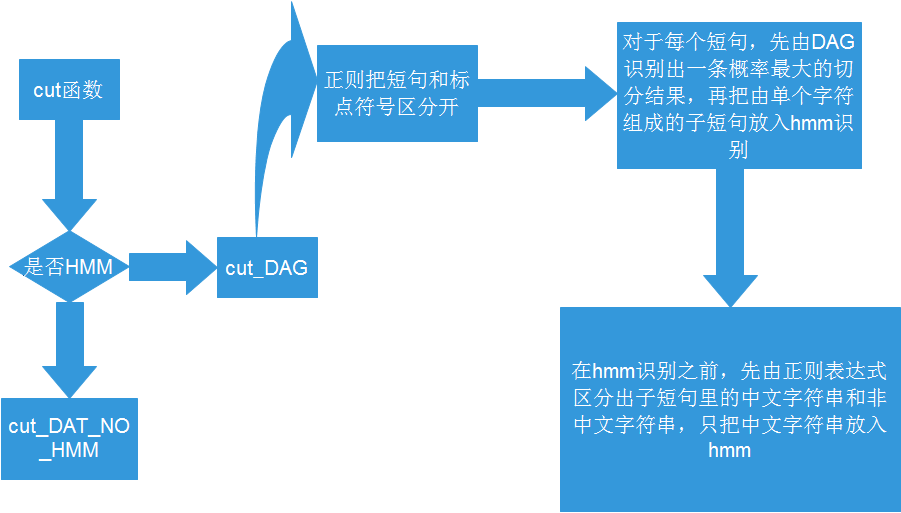
·**Gunicorn**

Gunicorn是一个高性能的Python WSGI的HTTP服务器，与各种Web框架兼容，配置非常简单，轻量级的资源消耗，以及相当迅速。它的特点是与各个web结合紧密，部署特别方便。Gunicorn 一般用来管理多个进程，自动重启挂掉的进程，防止服务器长时间停止服务，还可以动态调整worker的数量，请求多的时候增加worker的数量，请求少的时候减少。

·**结巴分词**

结巴分词是Python中文分词工具。

结巴分词流程图如下：



结巴分词基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图 (DAG)，然后采用了动态规划查找最大概率路径, 找出基于词频的最大切分组合。

结巴分词支持三种分词模式：

① 精确模式，试图将句子最精确地切开，适合文本分析；

② 全模式，把句子中所有的可以成词的词语都扫描出来, 速度非常快，但是不能解决歧义；

③ 搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

本项目的推荐和搜索功能都借助了结巴分词工具。

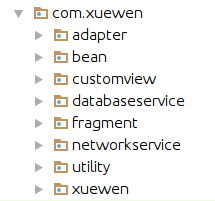
## 1.3 前端技术

### 1.3.1 前端概述

目前前端主要开发安卓版本，Android APP的开发采用Android原生开发技术开发，语言为**Java**，安卓版本我们目前也做了很多优化工作，比如核心功能的完善，app缓存，版本更新等等。为了ios平台用户同步使用产品，我们团队同时进行了跨平台的**react-native**的技术进行探索，并且成功打造了react-native的初步版本。

**整体的安卓构架：**

进行大模块的分层处理如下：



|  |  |
| --- | --- |
| **模块** | **内容** |
| adapter | 处理数据的渲染和数据到界面的绑定 |
| bean | 从网络中获取的数据的最小单位。每个bean既是网络访问中用都到的实体，也是持久化用到的实体 |
| customview | 自定义View，如音频播放控件 |
| databaseservice | 封装了数据库操作 |
| fragment | 动态加载的界面碎片 |
| networkservice | 封装了网络访问操作 |
| utility | 是一些我们需要用到的工具，如校验，封装的语音助手等。 |
| xuewen | 主要是 activity逻辑的编写，集中处理业务逻辑。 |

为了产品更加稳定同时减少开发量，前端在实现时我们也使用了很多的**第三方库**，具体如下：

|  |  |
| --- | --- |
| **名称** | **功能** |
| retrofit | 用于网络访问 |
| butterknife | 界面元素绑定，注解方式提替代；  findviewbyid的使用，使得代码更加的精简 |
| universal-image-loader | 用于加载图片 |
| gson | 解析json |
| ormlite-android | 数据持久化 |
| 讯飞sdk | 语音识别。用于语音的录制和转成文本分析，为后面的搜索系统做前期的准备 |
| prettytime | 做时间日期格式化 |
| material design | 使用组件CoordinatorLayout做整体界面的上拉收缩效果 |
| SwipeRefreshLayout | 实现网络访问延迟时的等待提示和手动刷新 |
| TextWatcher | 重载了TextWatcher对文本的字数进行了限制 |
| BitmapDecoder | 封装BitmapDecoder用于上传图片时的下采样 |

### 1.3.2 安卓前端技术策略

**·网络访问**：

使用retrofit进行网络访问，注解的语法使得编码规范可维护，逻辑清晰。如：

这就完成了一个接口定义。

编程思想上，retrofit减少解耦，降低耦合，让接口开发灵活，不同api之间互相不干扰。代码风格上，retrofit使用注解方式，代码简洁，易懂，易上手。设计思想上，retrofit采用建造者模式，开发构建简便。

**·自动登录：**

不同于web，Android手机属于用户个人设备，往往不需要在每次打开时都需要输入用户名密码进行登录。一般来说，一个APP需要让用户感觉到只输入过一次用户名密码。这就需要所谓的“自动登录”。

自动登录在用户以往登录过，但session超时等原因造成未登录的时候，APP携带用户的验证信息，自动登录，与后台建立会话。

为了更好的用户体验，消除掉那仅剩一次的输入用户名密码过程，我们采取了微信登录的策略。用户授权微信登录以后，APP将授权码作为凭据发送至服务器。服务器生成一个token作为替代用户名密码的登录凭据回传给客户端。往后，APP只需要携带这个token，就能够自动登录。

安全性方面，token存放在程序私有数据空间，一般情况下（手机没有被root）不可被访问，保证了用户登录的安全性。

**·图片加载：**

采用universal-image-loader（UIL），主要有以下优点：

1. 多线程下载图片，图片可以来源于网络，文件系统，项目文件夹assets中以及drawable中等；

2. 支持随意的配置ImageLoader，例如线程池，内存缓存策略，硬盘缓存策略，图片显示选项以及其他的一些配置；

3. 支持图片的内存缓存，文件系统缓存或者SD卡缓存；

4. 支持图片下载过程的监听；

5. 根据控件(ImageView)的大小对Bitmap进行裁剪，减少Bitmap占用过多的内存；

6. 较好的控制图片的加载过程，例如暂停图片加载，重新开始加载图片，一般使用在ListView,GridView中，滑动过程中暂停加载图片，停止滑动的时候去加载图片；

7. 提供在较慢的网络下对图片进行加载。

**·缓存：**

流量、内存的精打细算是安卓开发常需面对的问题提，UIL提供了很好的缓存策略：

1. UI请求数据，使用唯一的Key值索引Memory Cache中的Bitmap；

2. 内存缓存：缓存搜索，如果能找到Key值对应的Bitmap，则返回数据。否则执行第三步；

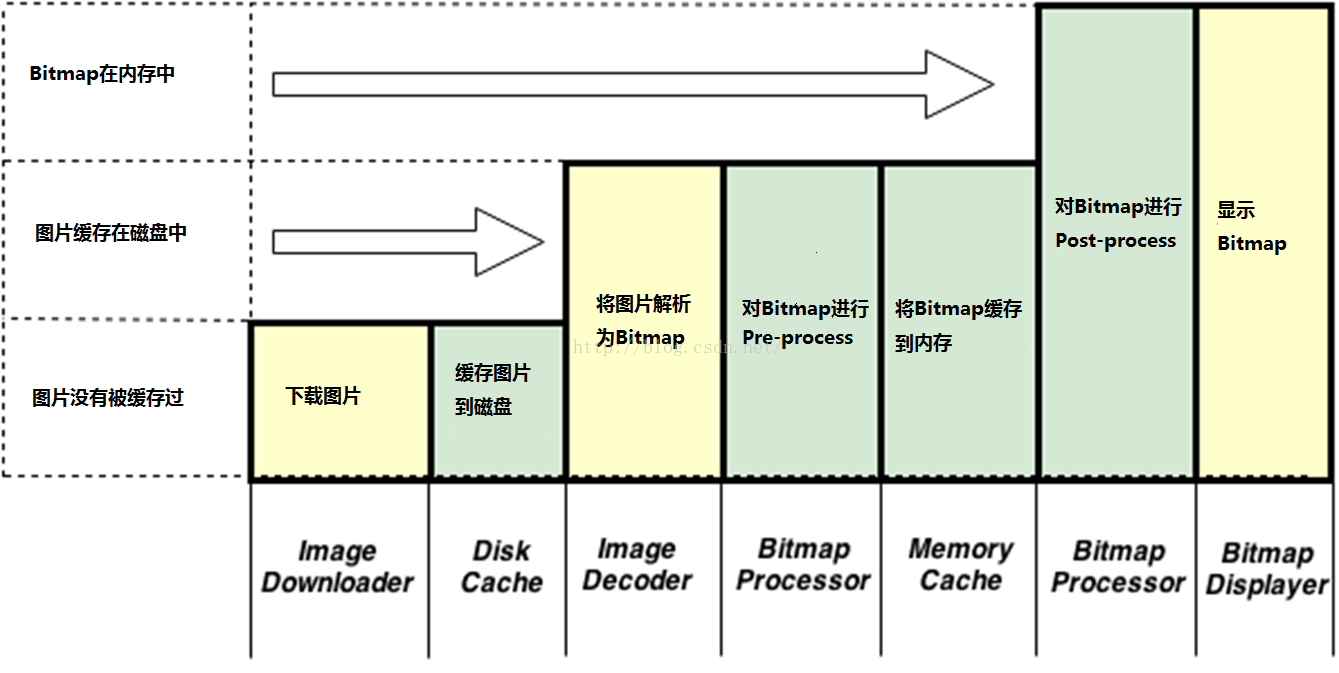
3. 硬盘存储：使用唯一Key值对应的文件名，检索SDCard上的文件；

4. 如果有对应文件，使用BitmapFactory.decode\*方法，解码Bitmap并返回数据，同时将数据写入缓存。如果没有对应文件，执行第五步；

5. 下载图片：启动异步线程，从数据源下载数据(Web)；

6. 若下载成功，将数据同时写入硬盘和缓存，并将Bitmap显示在UI中。

原理如图：

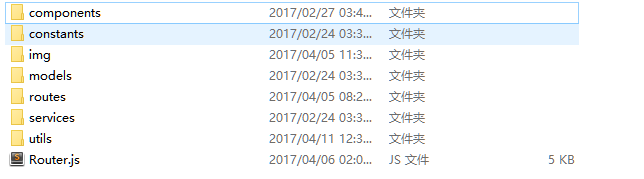
****

### 1.3.3 React-native跨平台开发

**Rn软件构架：**

项目采用的是淘宝的dva构架，底层由redux做整体的app的状态管理，redux-saga做项目的流程控制。

整体的项目目录为：



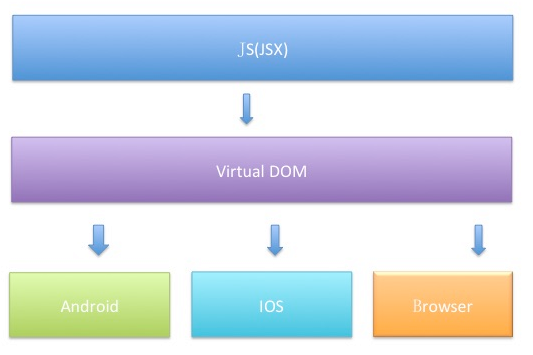
**相关详解:**

|  |  |
| --- | --- |
| **模块** | **内容** |
| components | 抽象出来的一些react native组件 |
| constants | 全局的一下静态的数据，比如颜色、大小、错误提示灯 |
| Img | 静态图像 |
| models | 这是数据核心，利用页面维度进行数据存储, redux管理整个数据中心 |
| routes | 路由界面 |
| services | 服务器交互的api |
| utils | 一下全局使用的工具函数 |
| Router.js | 页面的根路由 |

**使用的相关技术优势:**

**（1）React**

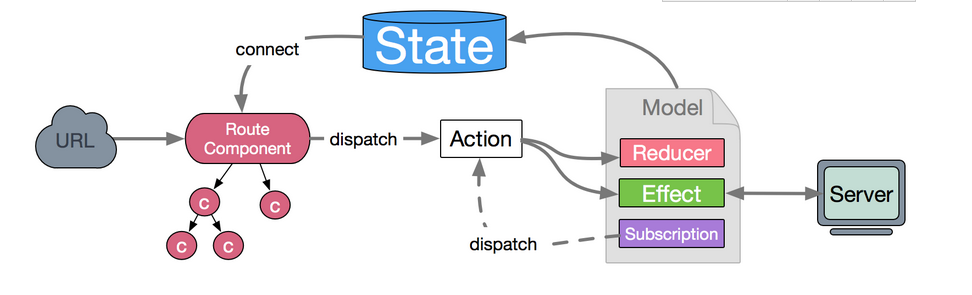
react利用组件化的思想进行应用的开发，避免用户直接对**dom**的操作，利用虚拟dom进行底层的比较操作，然后把不同值直接由react反应到dom上面，从而进行界面的更新，react native是利用react的思想，进行开发跨平台的一个库，它在前面利用js进行逻辑处理，底层调用安卓或者ios的组件，进行页面的渲染。



注: react native本身不提供web版本，可以使用react-web进行转换

**（2）Redux**

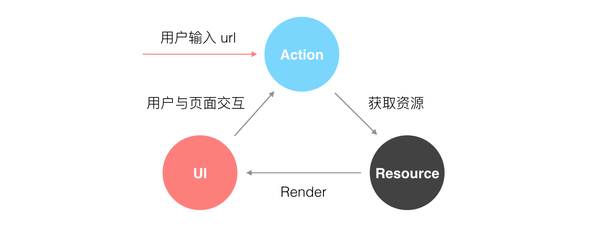
Redux是一个app的状态管理库，可以对整体的应用做一个统一的状态管理，方便不同的页面组件之间进行交互，它还有一层作用是，对页面进行一层简单的内存缓存。淘宝dva是对redux的一层封装，下面是页面的跳转逻辑：



**（3）react-native-router-flux**

无论是传统的后端 MVC 主导的应用，还是在当下最流行的单页面应用中，路由的职责都很重要，但原理并不复杂，即保证视图和 URL 的同步，而视图可以看成是资源的一种表现。当用户在页面中进行操作时，应用会在若干个交互状态中切换，路由则可以记录下某些重要的状态。本应用中，采用了react-native-router-flux为项目的路由系统，它内部自己维护一个类似浏览器的 history对象，记录当前app的路由状态。比如当前的路由栈中有哪几个页面，当前这个界面是否可以返回或者进入其他的界面等等。

url这里面react-native-router-flux中为自定义的key。



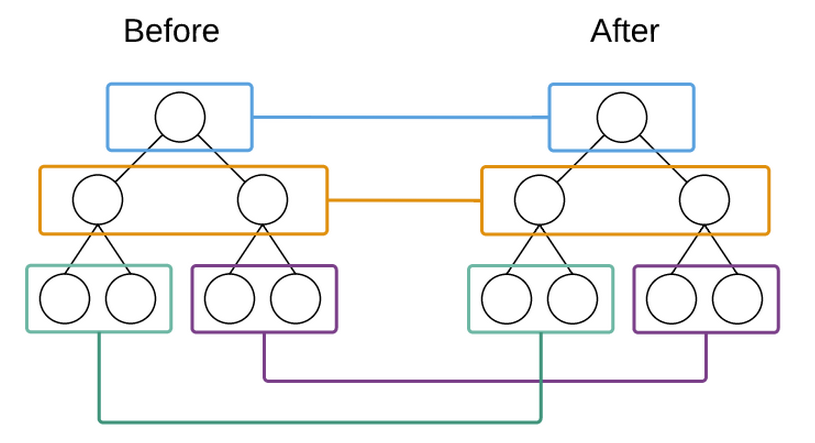
# 第二部分 算法设计

## 2.1 特色算法与技术

**（1）react虚拟dom算法**

为了让用户从dom操作中解脱出来，react使用了虚拟dom进行dom的比较，然后得出不同的部分，再由react进行真实dom的修改操作。

**react的diff算法用在什么地方呢？** 当组件更新的时候，react会创建一个新的虚拟dom树并且会和之前储存的dom树进行比较，这个比较多过程就用到了diff算法，所以组件初始化的时候是用不到的。react提出了一种假设，相同的节点具有类似的结构，而不同的节点具有不同的结构。在这种假设之上进行逐层的比较，如果发现对应的节点是不同的，那就直接删除旧的节点以及它所包含的所有子节点然后替换成新的节点。如果是相同的节点，则只进行属性的更改和更新操作。 而且比较只能是同层比较，算法的复杂度优化到了O(n)。



**（2）搜索推荐算法**

用户录音回答使用了科大讯飞的语言引擎，同时将录音文件转为文本文件，为推荐和搜索提供了更好的数据输入。

问题推荐与用户推荐均采用个性化推荐算法，先利用结巴分词对用户信息进行分词，然后根据分词结果效果最正确最理想的数据。

搜索问题与用户时，同样只用分词工具对用户的输入进行分词，然后根据结果查询数据库，提高搜索的匹配度。

## 2.2 数据存储

普通数据按照各个数据表存储在数据库中，用户头像文件、录音文件则存在服务器静态资源路径下，而数据库存的是文件在服务器相应的路径。

## 2.3 技术难点与解决方法

**（1）推荐和搜索模块的实现**

为了达到更好的推荐与搜索效果，我们并未直接对数据库使用模糊匹配，而是先使用结巴分词对相关信息分词，然后再对分词结果进行查询，这样的效果无疑更加出色。

**（2）前端网络请求等细节处理**

为了达到最好效果，我们在实现时使用了很多的第三方库，具体见上面描述。

**（3）文件大小限制**

修改用户的头像时，从相册中选择的图片有可能过大，上传至服务器时造成较大的流量负担，甚至nginx的配置限制大小的话会断开连接。而对于头像来说不要求图片的质量。所以上传前需要先对大图像做下采样。判断图片大小是否超过最大尺寸，如果超过（无论长宽），将其按比例下采样至最大尺寸。