React 全栈项目：硅谷直聘

第 1 章：准备

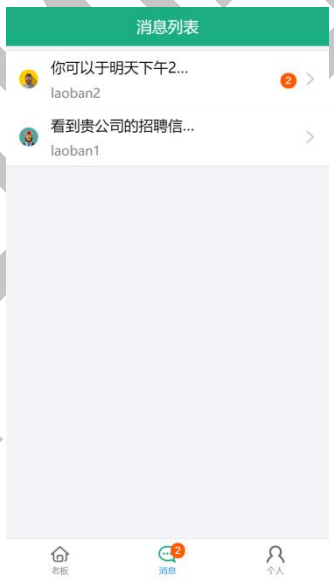
1.1. 项目描述

1) 此项目为一个前后台分离的招聘的 SPA，包括前端应用和后端应用
2) 包括用户注册/登陆，大神/老板列表，实时聊天等模块
3) 前端：使用 React 全家桶+ES6+Webpack 等技术
4) 后端：使用 Node + express + mongodb + socketIO 等技术
5) 采用模块化、组件化、工程化的模式开发

1.2. 项目功能界面

大神主界面

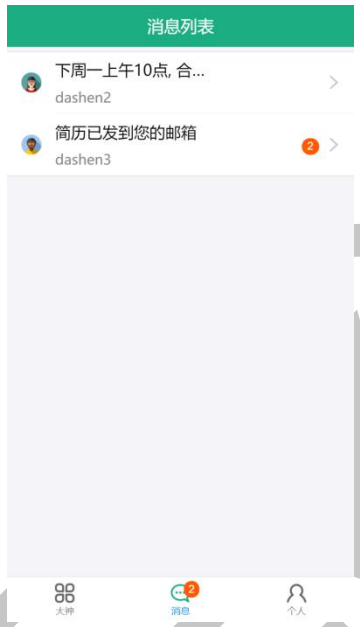
大神消息列表



大神个人中心

laoban 主界面

laoban 消息列表



laoban 个人中心

聊天界面

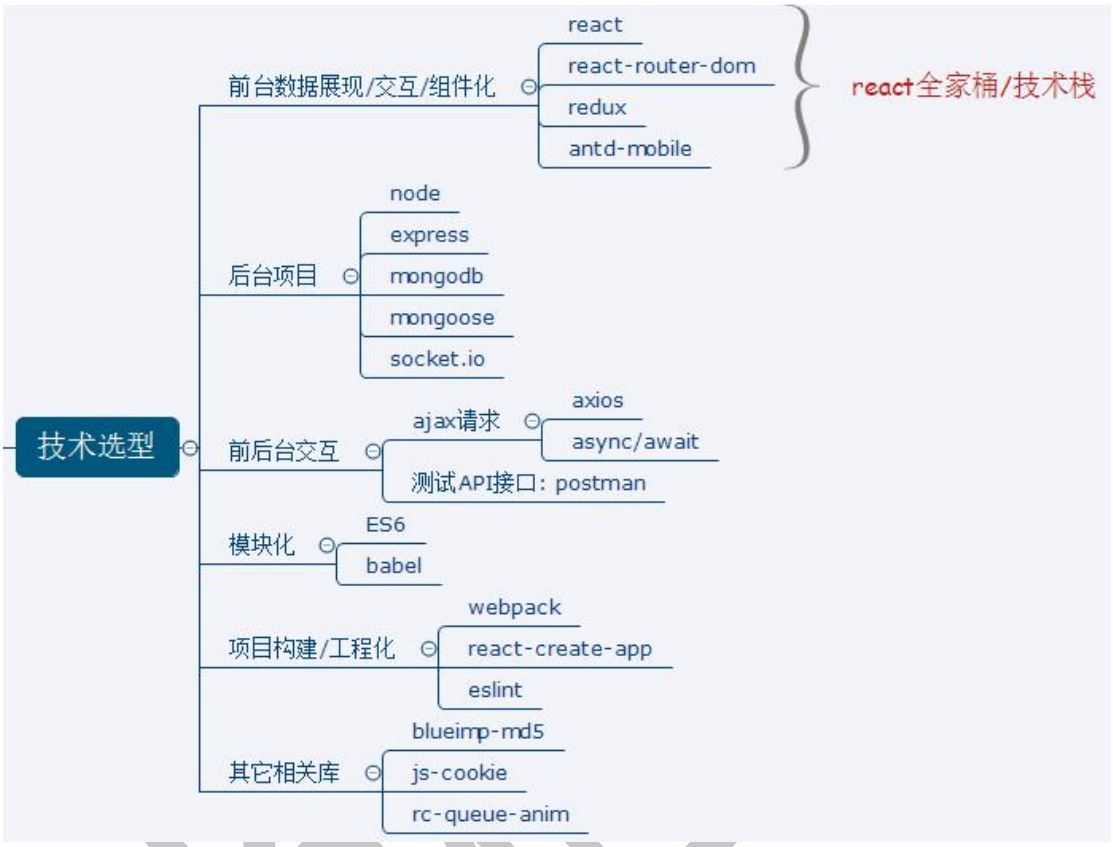
登陆界面

注册界面

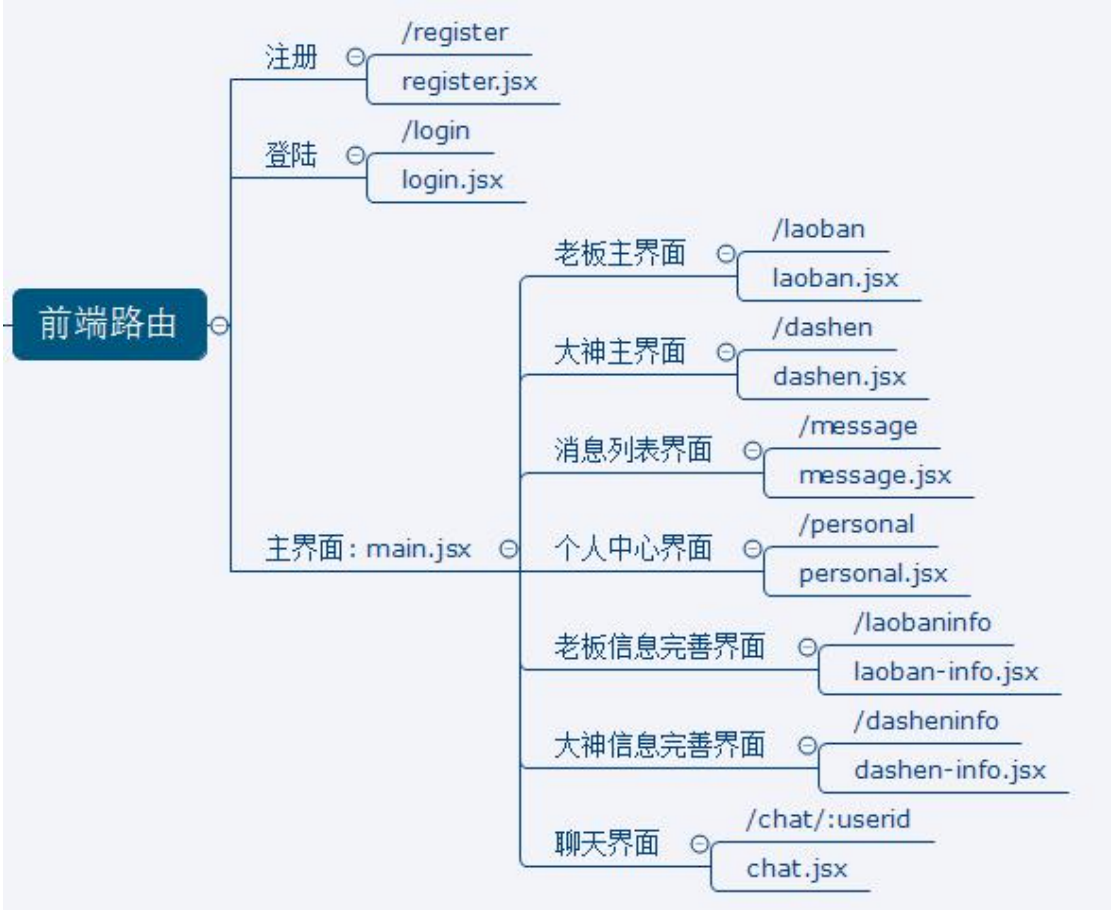
老板信息完善界面

大神信息完善界面

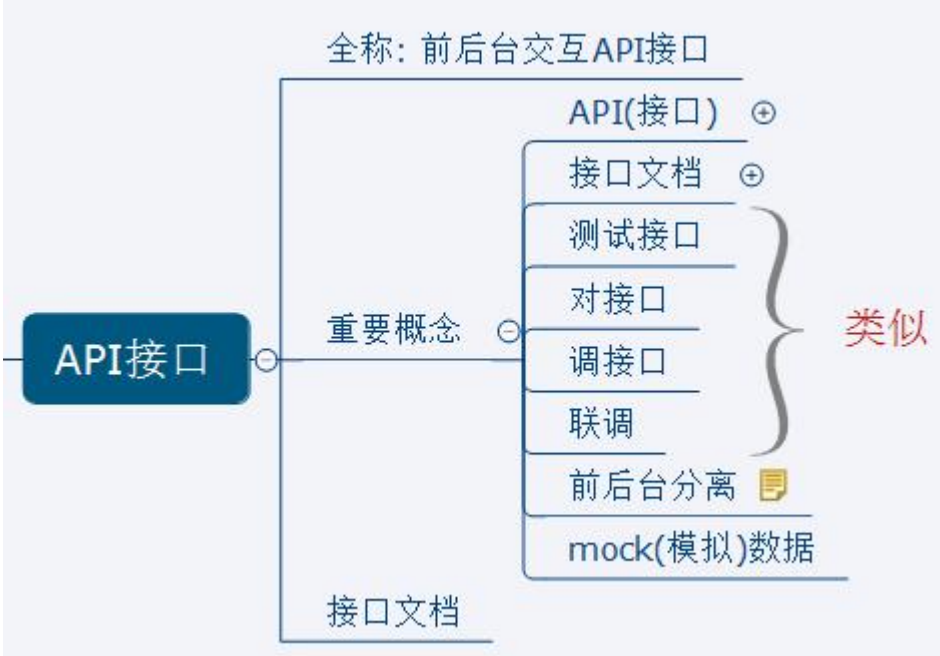
1.3. 技术选型



1.4. 前端路由



1.5. API 接口



1.6. 你能从此项目中学到什么?

1.6.1. 流程及开发方法

1) 熟悉一个项目的**开发流程**
2) 学会**模块化、组件化、工程化**的开发模式
3) 掌握使用 **create-react-app** 脚手架初始化 react 项目开发
4) 学会使用 **node+express+mongoose+mongodb** 搭建后台开发

1.6.2. React 插件或第三方库

1) 学会使用 **react-router-dom** 开发单页应用
2) 学会使用 **axios** 与后端进行数据交互
3) 学会使用 **redux+react-redux+redux-thunk** 管理应用组件状态
4) 学会使用 **antd-mobile** 组件库构建界面
5) 学会使用 **mongoose** 操作 mongodb 数据库

6) 学会使用 **express** 搭建后台路由

7) 学会使用 **socket.io** 实现实时通信

8) 学会使用 **blueimp-md5** 对密码进行 MD5 加密处理

9) 学会使用 **js-cookies** 操作浏览器端 cookie 数据

1.7. npm 常用命令

```
* npm init  //初始化当前应用包，生成 package.json
* npm install  //根据 package.json 下载所有依赖包
* npm install packageName --save   //下载某个运行时依赖包
* npm install packageName --save-dev  //下载某个开发编译期依赖包
* npm install packageName -g  //全局下载某个依赖包
* npm install package@version  //下载指定版本的某个依赖包
* npm info packageName  //查看某个包有远程仓库中的相关信息
* npm rm packageName --save  //移除已下载的运行依赖包
* npm rm packageName –save-dev  //移除已下载的开发依赖包
* npm list  //查看安装的所有的包
* npm help  //查看命令的详细帮助
* npm install -g cnpm --registry=https://registry.npm.taobao.org  //安装淘宝镜像
* npm config set registry="https://registry.npm.taobao.org" //将淘宝镜像设置为 npm 的默认仓库
* npm run xxx //执行 package.json 的 scripts 中配置的命令
* npm root -g //查看全局下载目录
```

1.8. git 常用基本命令

```
* git config --global user.name "username"  //配置用户名
* git config --global user.password "xx@gmail.com" //配置邮箱
* git init  //初始化生成一个本地仓库
* git clone url  //将远程仓库克隆下载到本地
* git add *   //添加到暂存区
* git commit –m “message”  //提交到本地仓库
* git remote add origin url  //关联到远程仓库
* git push origin master  //push 到远程
* git pull origin master  //从远程 pull 更新
```

第 2 章: 应用开发详解

2.1. 开启项目开发

2.1.1. 使用 create-react-app(脚手架)搭建项目

1) create-react-app 是 react 官方提供的用于搭建基于 react+webpack+es6 项目的脚手架

2) 操作:

```
npm install -g create-react-app : 全局下载工具
create-react-app gzhipin-client :下载模板项目
cd gzhipin
npm start
访问: localhost:3000
```

2.1.2. 编码测试与打包发布项目

1) 编码测试

npm start
访问: http://localhost:3000
编码, 自动编译打包刷新(live-reload), 查看效果

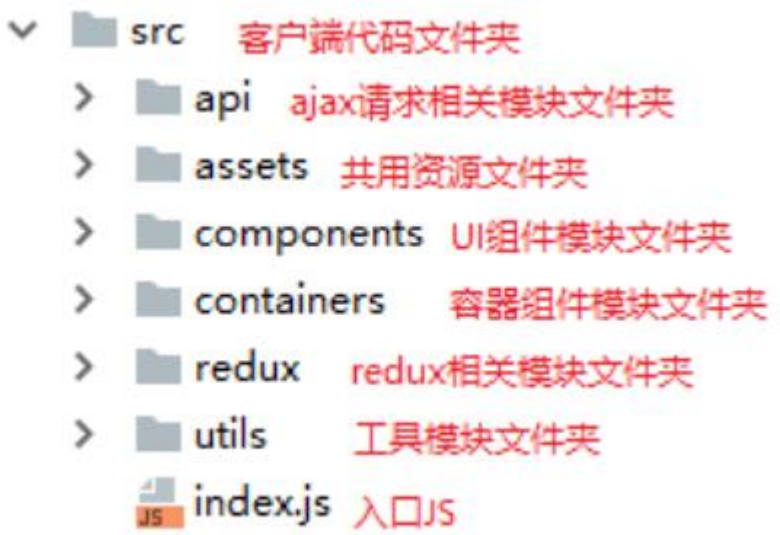
2) 打包发布

npm run build
npm install -g serve
serve build
访问: http://localhost:5000

2.2. 功能需求分析

演示项目功能, 对功能模块进行分析说明

2.3. 项目(前端)源码目录设计



2.4. 引入 antd-mobile

2.4.1. 下载组件库包

```
npm install antd-mobile --save
```

2.4.2. 页面处理: index.html

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no" />
<script src="https://as.alipayobjects.com/g/component/fastclick/1.0.6/fastclick.js"></script>
<script>
  if ('addEventListener' in document) {
    document.addEventListener('DOMContentLoaded', function() {
      FastClick.attach(document.body);
    }, false);
  }
  if(!window.Promise) {
    document.writeln('<script
```

```
src="https://as.alipayobjects.com/g/component/es6-promise/3.2.2/es6-promise.min.js"
'+'>'+'<'+'/'+'script>');
  }
</script>
```

2.4.3. 实现组件的按需打包

1) 下载依赖模块

```
npm install --save-dev babel-plugin-import react-app-rewired
```

2) 定义加载配置的 js 模块: config-overrides.js

```
const {injectBabelPlugin} = require('react-app-rewired');
module.exports = function override(config, env) {
  config = injectBabelPlugin(['import', {libraryName: 'antd-mobile', style: 'css'}],
config);
  return config;
}
```

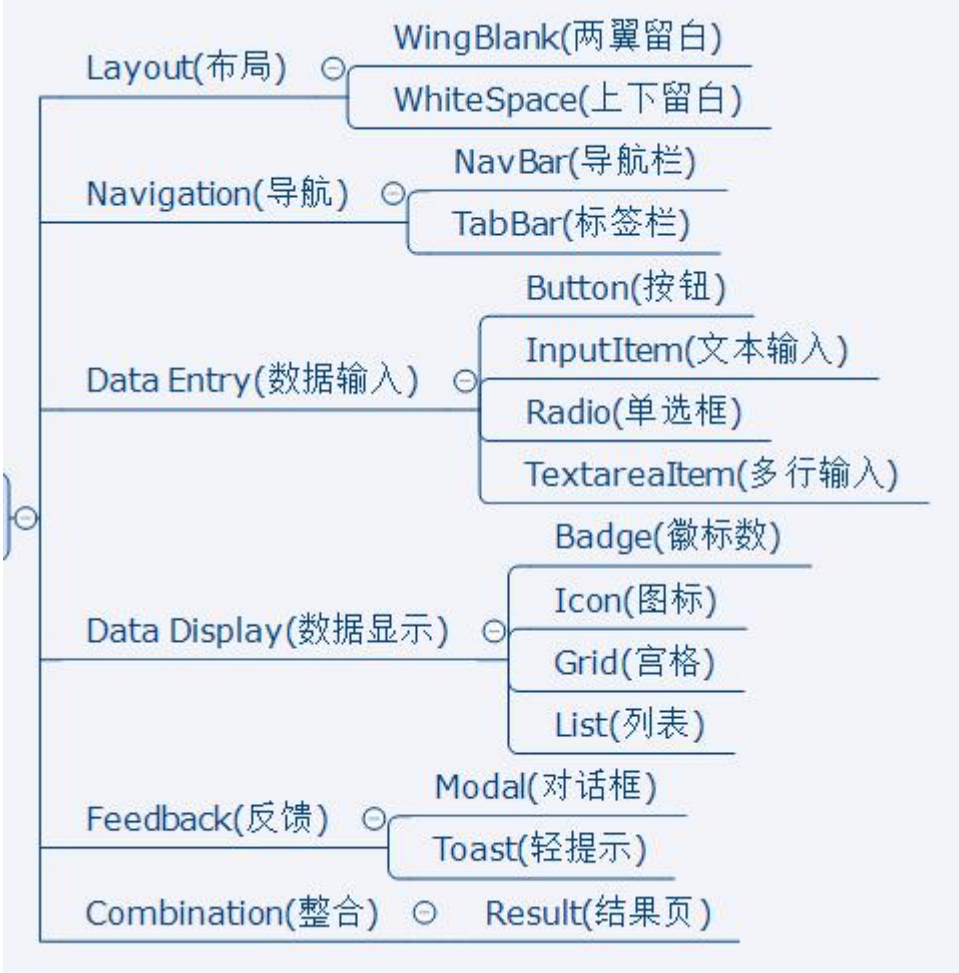
➢ 修改配置: package.json

```
"scripts": {
  "start": "react-app-rewired start",
  "build": "react-app-rewired build",
  "test": "react-app-rewired test --env=jsdom",
  "eject": "react-scripts eject"
}
```

2.4.4. 在应用中使用 antd 组件

```
import React from 'react'
import ReactDOM from 'react-dom'
import {Button} from 'antd-mobile'

ReactDOM.render(
  <Button type='primary'>学习</Button>,
  document.getElementById('root')
)
```

2.4.5. 应用中使用的组件



2.4.6. 自定义主题

1) 目标:

将主体的背景颜色从 blue 变为 green

2) 下载依赖模块

```
npm install --save-dev less@2.7.3 less-loader
```

3) 配置: config-overrides.js

```
const {injectBabelPlugin, getLoader} = require('react-app-rewired');


const fileLoaderMatcher = function (rule) {
  return rule.loader && rule.loader.indexOf(`file-loader`) != -1;
```

```
}


module.exports = function override(config, env) {
  // babel-plugin-import
  config = injectBabelPlugin(['import', {
    libraryName: 'antd-mobile',
    //style: 'css',
    style: true, // use less for customized theme
  }], config);


  // customize theme
  config.module.rules[1].oneOf.unshift(
    {
      test: /\.less$/,
      use: [
        require.resolve('style-loader'),
        require.resolve('css-loader'),
        {
          loader: require.resolve('postcss-loader'),
          options: {
            // Necessary for external CSS imports to work
            // https://github.com/facebookincubator/create-react-app/issues/2677
            ident: 'postcss',
            plugins: () => [
              require('postcss-flexbugs-fixes'),
              autoprefixer({
                browsers: [
                  '>1%',
                  'last 4 versions',
                  'Firefox ESR',
                  'not ie < 9', // React doesn't support IE8 anyway
                ],
                flexbox: 'no-2009',
              }),
            ],
          },
        },
        {
          loader: require.resolve('less-loader'),
          options: {
            // theme vars, also can use theme.js instead of this.
            modifyVars: {
```

```
            "@brand-primary": "#1cae82", // 正常
            "@brand-primary-tap": "#1DA57A", // 按下
          },
        },
      },
    ]
  }
);

// css-modules
config.module.rules[1].oneOf.unshift(
  {
    test: /\.css$/,
    exclude: /node_modules|antd-mobile\.css/,
    use: [
      require.resolve('style-loader'),
      {
        loader: require.resolve('css-loader'),
        options: {
          modules: true,
          importLoaders: 1,
          localIdentName: '[local]___[hash:base64:5]'
        },
      },
      {
        loader: require.resolve('postcss-loader'),
        options: {
          // Necessary for external CSS imports to work
          // https://github.com/facebookincubator/create-react-app/issues/2677
          ident: 'postcss',
          plugins: () => [
            require('postcss-flexbugs-fixes'),
            autoprefixer({
              browsers: [
                '>1%',
                'last 4 versions',
                'Firefox ESR',
                'not ie < 9', // React doesn't support IE8 anyway
              ],
              flexbox: 'no-2009',
            }),
          ],
```

```
        },
      },
    ]
  }
);

// file-loader exclude
let l = getLoader(config.module.rules, fileLoaderMatcher);
l.exclude.push(/\.less$/);

return config;
};
```

2.5. 引入路由

2.5.1. 下载路由包: react-router-dom

```
npm install --save react-router-dom
```

2.5.2. 路由组件: containers/register/register.jsx

```
/*
用户注册的路由组件
 */
import React, {Component} from 'react'

export default class Register extends Component {
  render() {
    return (
      <div>Register</div>
    )
  }
}
```

2.5.3. 路由组件: containers/login/login.jsx

```
/*
用户登陆的路由组件
 */
import React, {Component} from 'react'

export default class Login extends Component {
  render () {
    return (
      <div>login</div>
    )
  }
}
```

2.5.4. 路由组件: containers/main/main.jsx

```
/*
应用主界面路由组件
 */
import React, {Component} from 'react'

export default class Main extends Component {
  render() {
    return (
      <div>Main</div>
    )
  }
}
```

2.5.5. 映射路由: index.js

```
/*
入口 JS
 */
import React from 'react'
import ReactDOM from 'react-dom'
```

```
import {HashRouter, Switch, Route} from 'react-router-dom'

import Login from './containers/login/login'
import Register from './containers/register/register'
import Main from './containers/main/main'

ReactDOM.render((
  <HashRouter>
    <Switch>
      <Route path='/login' component={Login}/>
      <Route path='/register' component={Register}/>
      <Route component={Main}/>
    </Switch>
  </HashRouter>
), document.getElementById('root'))
```

2.6. 引入 redux

2.6.1. 下载相关依赖包

```
npm install --save redux@3.7.2 react-redux redux-thunk

npm install --save-dev redux-devtools-extension
```

注意: redux 不能下载最新版本

2.6.2. reducers: redux/reducers.js

```
/*
包含多个用于生成新的 state 的 reducer 函数的模块
 */
import {combineReducers} from 'redux'

function xxx(state = 0, action) {

  return state
```

```
}

function yyy(state = 0, action) {

  return state
}

// 返回合并后的reducer 函数
export default combineReducers({
  xxx,
  yyy
})
```

2.6.3. store: redux/store.js

```
/*
redux 最核心的store 对象模块
 */

import {createStore, applyMiddleware} from 'redux'
import thunk from 'redux-thunk'
import {composeWithDevTools} from 'redux-devtools-extension'
import reducers from './reducers'

export default createStore(reducers, composeWithDevTools(applyMiddleware(thunk)))
```

2.6.4. 入口 JS: index.js

```
/*
入口JS
 */
import React from 'react'
import ReactDOM from 'react-dom'
import {Provider} from 'react-redux'
import {HashRouter, Switch, Route} from 'react-router-dom'

import store from './redux/store'
import Login from './containers/login/login'
```

```
import Register from './containers/register/register'
import Main from './containers/main/main'

ReactDOM.render((
  <Provider store={store}>
    <HashRouter>
      <Switch>
        <Route path='/login' component={Login}/>
        <Route path='/register' component={Register}/>
        <Route component={Main}/>
      </Switch>
    </HashRouter>
  </Provider>
), document.getElementById('root'))
```

2.7. 注册/登陆界面

2.7.1. Logo 组件

1) 引入 logo 图片

2) components/logo/logo.jsx

```
import React from 'react'
import logo from './logo.png'
import './logo.less'
/*
简单的显示 Logo 的组件
 */
export default class Logo extends Component {
  render () {
    return (
      <div className="logo-container">
        <img src={logo} alt="logo" className='logo-img'/>
      </div>
    )
  }
}
```

3) components/logo/logo.less

```
.logo-container {
  text-align: center;
  margin-top: 10px;
  margin-bottom: 10px;
  .logo-img {
    width: 240px;
    height: 240px;
  }
}
```

2.7.2. 注册组件: containers/register/register.jsx

```
/*
用户注册的路由组件
 */
```

```
import React, {Component} from 'react'
import {
  NavBar,
  WingBlank,
  List,
  InputItem,
  WhiteSpace,
  Radio,
  Button
} from 'antd-mobile'

import Logo from '../../components/logo/logo'

export default class Register extends Component {

  state = {
    username: '',
    password: '',
    password2: '',
    type: 'dashen'
  }

  // 处理输入框/单选框变化，收集数据到state
  handleChange = (name, value) => {
    this.setState({[name]: value})
  }

  // 跳转到login路由
  toLogin = () => {
    this.props.history.replace('/login')
  }

  // 注册
  register = () => {
    console.log(JSON.stringify(this.state))
  }

  render() {
    const {type} = this.state
    return (
      <div>
        <NavBar>硅谷直聘</NavBar>
```

```
        <Logo/>
        <WingBlank>
          <List>
            <InputItem
              placeholder='输入用户名'
              onChange={val => this.handleChange('username', val)}
            >
              用户名:
            </InputItem>
            <WhiteSpace/>
            <InputItem
              type='password'
              placeholder='输入密码'
              onChange={val => this.handleChange('password', val)}
            >
              密   码:
            </InputItem>
            <WhiteSpace/>
            <InputItem
              type='password'
              placeholder='输入确认密码'
              onChange={val => this.handleChange('password2', val)}
            >
              确认密码:
            </InputItem>
            <WhiteSpace/>
            <List.Item>
              <span style={{marginRight: 30}}>用户类型:</span>
              <Radio checked={this.state.type==='dashen'}
                     onClick={() => {this.handleChange('type', 'dashen')}}>大神</Radio>
                    
              <Radio checked={this.state.type==='laoban'}
                     onClick={() => {this.handleChange('type', 'laoban')}}>老板</Radio>
            </List.Item>
            <WhiteSpace/>
            <Button type='primary' onClick={this.register}>注   册
</Button>
            <WhiteSpace/>
            <Button onClick={this.toLogin}>已经有账号</Button>
          </List>
        </WingBlank>
      </div>
```

```
    )
  }
}
```

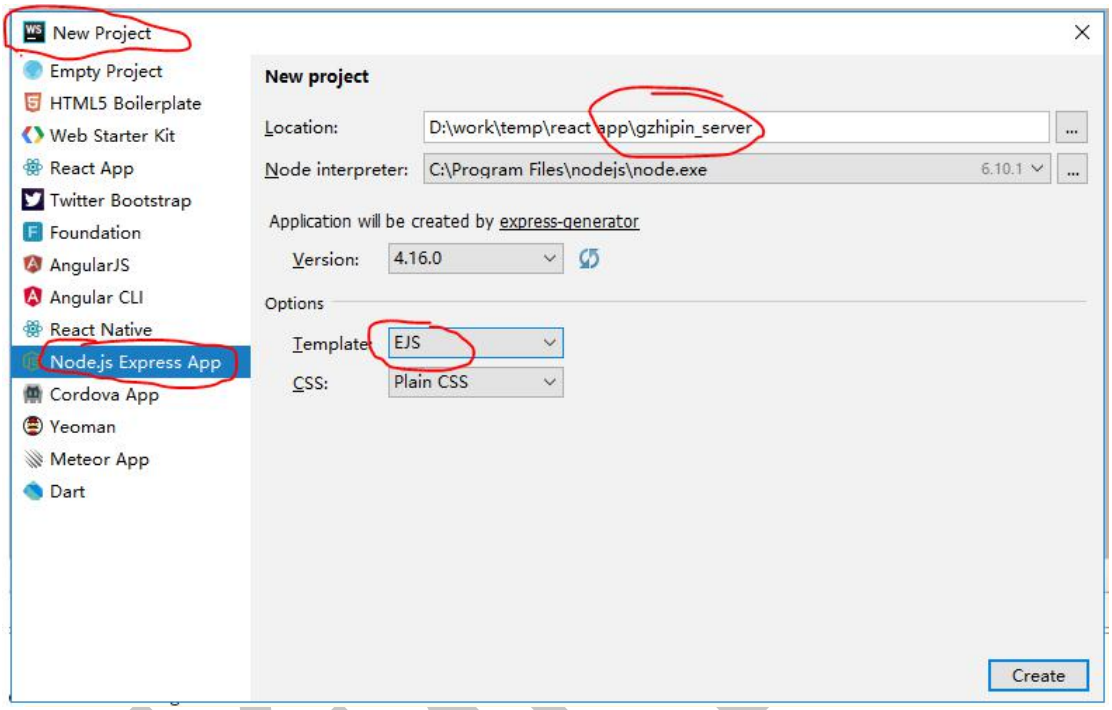
2.7.3. 登陆组件: containers/login/login.jsx

```
/*
用户登陆的路由组件
 */

import React, {Component} from 'react'
import {
  NavBar,
  WingBlank,
  List,
  InputItem,
  WhiteSpace,
  Button
} from 'antd-mobile'

import Logo from '../../components/logo/logo'

export default class Login extends Component {
  state = {
    username: '',
    password: '',
  }

  // 处理输入框/单选框变化, 收集数据到state
  handleChange = (name, value) => {
    this.setState({[name]: value})
  }

  // 跳转到注册路由
  toRegister = () => {
    this.props.history.replace('/register')
  }

  // 注册
```

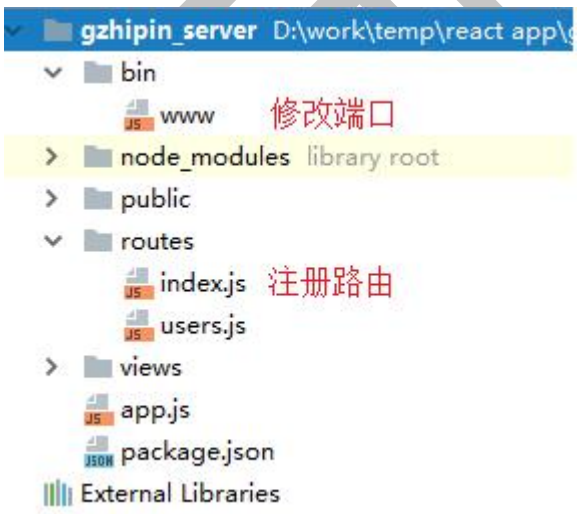
```
  login = () => {
    console.log(this.state)
  }

  render() {
    return (
      <div>
        <NavBar>硅谷直聘</NavBar>
        <Logo/>
        <WingBlank>
          <List>
            <InputItem
              placeholder='输入用户名'
              onChange={val => this.handleChange('username', val)}
            >
              用户名:
            </InputItem>
            <WhiteSpace/>
            <InputItem
              type='password'
              placeholder='输入密码'
              onChange={val => this.handleChange('password', val)}
            >
              密   码:
            </InputItem>
            <WhiteSpace/>
            <Button type='primary' onClick={this.login}>登   陆
</Button>
            <WhiteSpace/>
            <Button onClick={this.toRegister}>还没有账号</Button>
          </List>
        </WingBlank>
      </div>
    )
  }
}
```

2.8. 搭建后台应用

2.8.1. 创建 node+express 应用



2.8.2. 后台应用结构



2.8.3. 启动应用并访问

1) 执行命令: npm start

2) 访问: http://localhost:3000

2.8.4. 后台简单编码并测试

1) 需求:

   a. 后台应用运行端口指定为 4000

   b. 提供一个用户注册的接口

      a) path 为: /register

      b) 请求方式为: POST

      c) 接收 username 和 password 参数

      d) admin 是已注册用户

      e) 注册成功返回: {code: 0, data: {_id: 'abc', username: 'xxx', password:'123'}

      f) 注册失败返回: {code: 1, msg: '此用户已存在'}

2) 修改端口号: bin/www

```
var port = normalizePort(process.env.PORT || '4000');
```

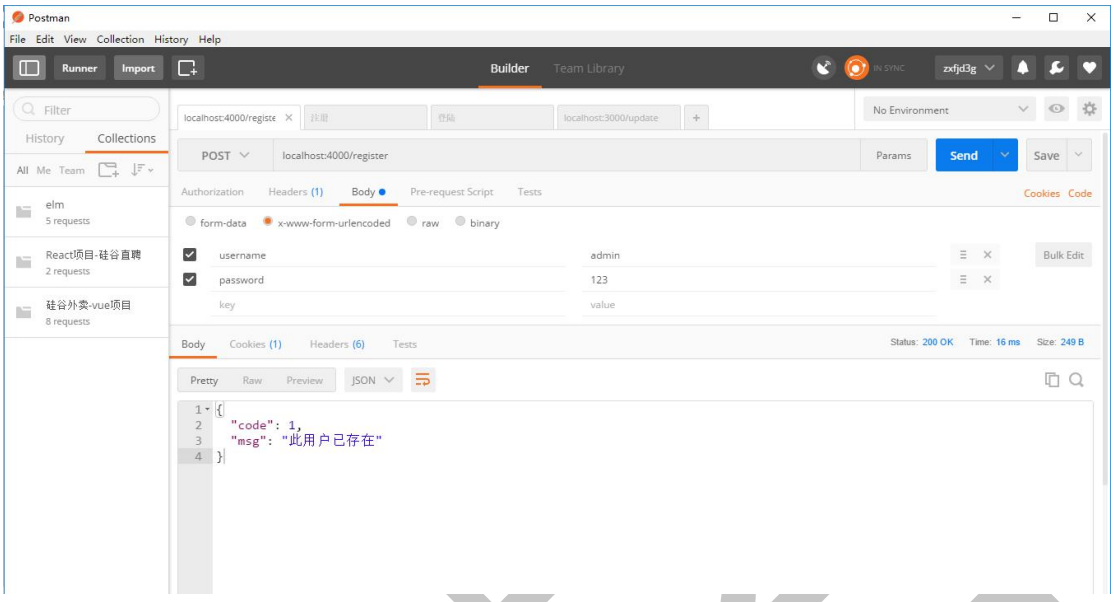
3) 注册新路由: routes/index.js

```
router.post('/register', function (req, res, next) {
  const {username, password} = req.body
  console.log('register', username, password)
  if (username === 'admin') {
    res.send({code: 1, msg: '此用户已存在'})
  } else {
    res.send({code: 0, data: {_id: 'abc', username, password}})
  }
})
```

4) 重启应用

   npm start

5) 使用 postman 测试接口



http://blog.csdn.net/ye1992/article/details/49998511: 不同的 post 提交方式

2.8.5. 后台应用自动重运行

1) 问题: 每次修改后台应用代码, 需要重新运行命令修改才生效
2) 解决: 使用 nodemon 包
3) 下载: npm install --save-dev nodemon
4) 配置: "start": "nodemon ./bin/www"
5) 测试: 修改后台任何代码, 会自动重新运行最新的代码 (按下 ctrl + S)

2.9. 使用 mongoose 操作数据库

2.9.1. 下载依赖包

```
npm install --save mongoose blueimp-md5
```

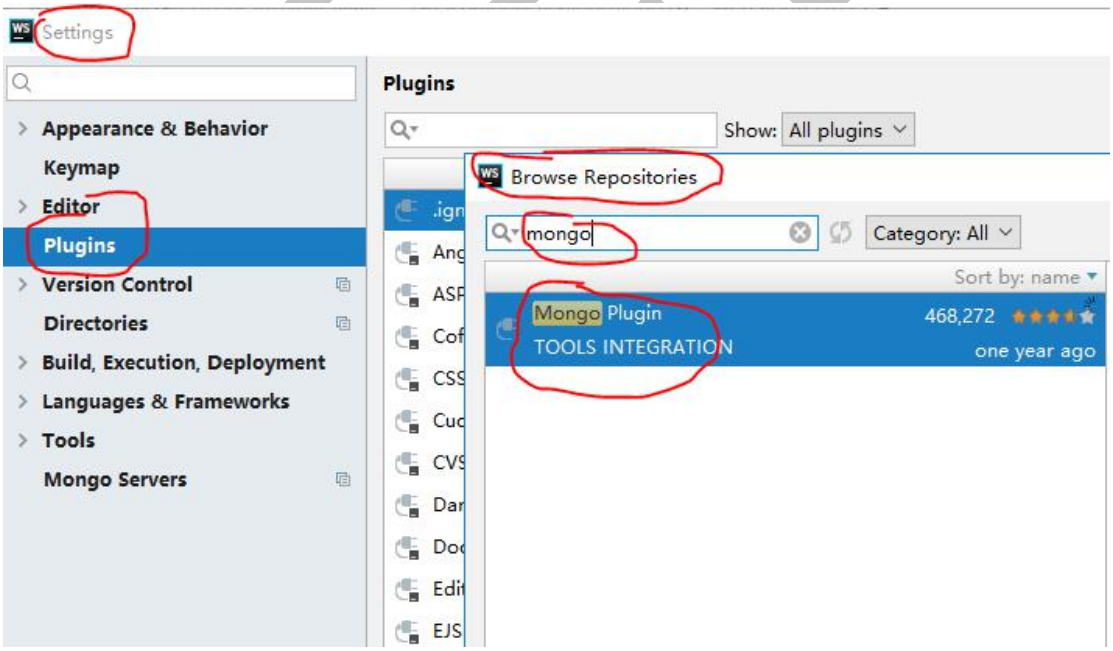
2.9.2. db/db_test.js

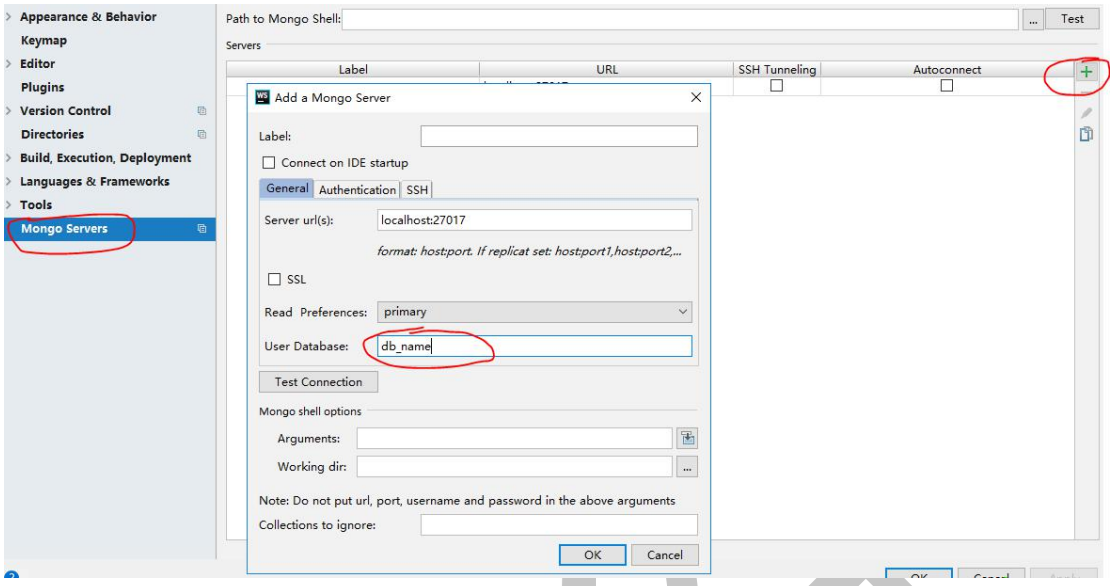
```
/*
使用mongoose 操作 mongodb 的测试文件
1. 连接数据库
  1.1. 引入 mongoose
  1.2. 连接指定数据库(URL 只有数据库是变化的)
  1.3. 获取连接对象
  1.4. 绑定连接完成的监听(用来提示连接成功)
2. 得到对应特定集合的Model
  2.1. 字义 Schema(描述文档结构)
  2.2. 定义 Model(与集合对应, 可以操作集合)
3. 通过Model 或其实例对集合数据进行 CRUD 操作
  3.1. 通过Model 实例的 save()添加数据
  3.2. 通过Model 的 find()/findOne()查询多个或一个数据
  3.3. 通过Model 的 findByIdAndUpdate()更新某个数据
  3.4. 通过Model 的 remove()删除匹配的数据
 */
const md5 = require('blueimp-md5')
// 1. 连接数据库
// 1.1. 引入 mongoose
const mongoose = require('mongoose')
// 1.2. 连接指定数据库(URL 只有数据库是变化的)
mongoose.connect('mongodb://localhost:27017/gzhipin_test2')
// 1.3. 获取连接对象
const conn = mongoose.connection
// 1.4. 绑定连接完成的监听(用来提示连接成功)
conn.on('connected', function () {
  console.log('数据库连接成功')
})

// 2. 得到对应特定集合的Model
// 2.1. 字义 Schema(描述文档结构)
const userSchema = mongoose.Schema({
  username: {type: String, required: true}, // 用户名
  password: {type: String, required: true}, // 密码
  type: {type: String, required: true}, // 用户类型: dashen/laoban
})
// 2.2. 定义 Model(与集合对应, 可以操作集合)
const UserModel = mongoose.model('user', userSchema) // 集合名: users
```

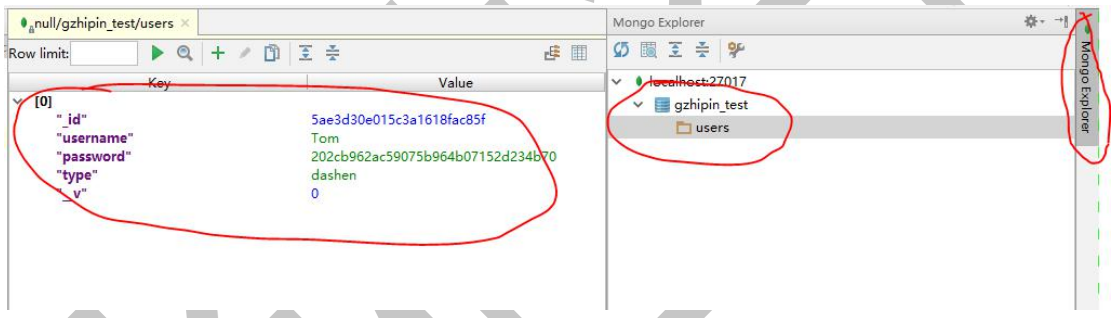
```
// CRUD

// 3.1. 通过Model 实例的save()添加数据
function testSave() {
  // user 数据对象
  const user = {
    username: 'xfzhang',
    password: md5('1234'),
    type: 'dashen',
  }
  const userModel = new UserModel(user)
  // 保存到数据库
  userModel.save(function (err, user) {
    console.log('save', err, user)
  })
}

// testSave()


// 3.2. 通过Model 的find()/findOne()查询多个或一个数据
function testFind() {
  // 查找多个
  UserModel.find(function (err, users) { // 如果有匹配返回的是一个[user, user..], 如果没有一个匹配的返回[]
    console.log('find() ', err, users)
  })
  // 查找一个
  UserModel.findOne({_id: '5ae1d0ab28bd750668b3402c'}, function (err, user) { // 如果有匹配返回的是一个user, 如果没有一个匹配的返回null
    console.log('findOne() ', err, user)
  })
}

// testFind()


// 3.3. 通过Model 的findByIdAndUpdate()更新某个数据
function testUpdate() {
  UserModel.findByIdAndUpdate({_id: '5ae1241cf2dd541a8c59a981'}, {username: 'yyy'},
function (err, user) {
    console.log('findByIdAndUpdate()', err, user)
```

```
  })
}
// testUpdate()

// 3.4. 通过Model 的remove()删除匹配的数据
function testDelete() {
  UserModel.remove({_id: '5ae1241cf2dd541a8c59a981'}, function (err, result) {
    console.log('remove()', err, result)
  })
}
// testDelete()
```

2.9.3. 安装 webstorm 的 mongodb 插件: Mongo Plugin







2.10. 注册/登陆后台处理

2.10.1. 数据库数据操作模块: db/models.js

```
/*
包含 n 个能操作 mongodb 数据库集合的 model 的模块
1. 连接数据库
  1.1. 引入 mongoose
  1.2. 连接指定数据库(URL 只有数据库是变化的)
  1.3. 获取连接对象
  1.4. 绑定连接完成的监听(用来提示连接成功)
2. 定义出对应特定集合的 Model 并向外暴露
  2.1. 字义 Schema(描述文档结构)
  2.2. 定义 Model(与集合对应, 可以操作集合)
  2.3. 向外暴露 Model
 */
```

```
/*1. 连接数据库*/
// 1.1. 引入 mongoose
const mongoose = require('mongoose')
// 1.2. 连接指定数据库(URL 只有数据库是变化的)
mongoose.connect('mongodb://localhost:27017/bossz')
// 1.3. 获取连接对象
const conn = mongoose.connection
// 1.4. 绑定连接完成的监听(用来提示连接成功)
conn.on('connected', function () {
  console.log('数据库连接成功!')
})

/*2. 定义出对应特定集合的 Model 并向外暴露*/
// 2.1. 字义 Schema(描述文档结构)
const userSchema = mongoose.Schema({
  username: {type: String, required: true}, // 用户名
  password: {type: String, required: true}, // 密码
  type: {type: String, required: true}, // 用户类型: dashen/laoban
  header: {type: String}, // 头像名称
  post: {type: String}, // 职位
  info: {type: String}, // 个人或职位简介
  company: {type: String}, // 公司名称
  salary: {type: String} // 工资
})
// 2.2. 定义 Model(与集合对应, 可以操作集合)
const UserModel = mongoose.model('user', userSchema)
// 2.3. 向外暴露 Model
exports.UserModel = UserModel
```

2.10.2. 路由器模块: routes/index.js

```
// 引入 md5 加密函数库
const md5 = require('blueimp-md5')
// 引入 UserModel
const UserModel = require('../db/models').UserModel
const filter = {password: 0} // 查询时过滤出指定的属性

// 注册路由
router.post('/register', function (req, res) {
  // 1. 获取请求参数数据(username, password, type)
```

```
  const {username, password, type} = req.body
  // 2. 处理数据
  // 3. 返回响应数据
  // 2.1. 根据username查询数据库，看是否已存在user
  UserModel.findOne({username}, function (err, user) {
    // 3.1. 如果存在，返回一个提示响应数据：此用户已存在
    if(user) {
      res.send({code: 1, msg: '此用户已存在'}) // code是数据是否是正常数据的标识
    } else {
      // 2.2. 如果不存在，将提交的user保存到数据库
      new UserModel({username, password: md5(password), type}).save(function (err,
user) {
        // 生成一个cookie(userid: user._id)，并交给浏览器保存
        res.cookie('userid', user._id, {maxAge: 1000*60*60*24*7}) // 持久化cookie，浏
览器会保存在本地文件
        // 3.2. 保存成功，返回成功的响应数据：user
        res.send({code: 0, data: {_id: user._id, username, type}})  // 返回的数据中不要
携带pwd
      })
    }
  })
})

// 登陆路由
router.post('/login', function (req, res) {
  // 1. 获取请求参数数据(username，password)
  const {username, password} = req.body
  // 2. 处理数据：根据username和password去数据库查询得到user
  UserModel.findOne({username, password: md5(password)}, filter, function (err, user)
{
    // 3. 返回响应数据
    // 3.1. 如果user没有值，返回一个错误的提示：用户名或密码错误
    if(!user) {
      res.send({code: 1, msg: '用户名或密码错误'})
    } else {
      // 生成一个cookie(userid: user._id)，并交给浏览器保存
      res.cookie('userid', user._id, {maxAge: 1000*60*60*24*7})
      // 3.2. 如果user有值，返回user
      res.send({code: 0, data: user}) // user中没有pwd
    }
  })
})
```

2.10.3. 使用 postman 测试接口

2.11. 注册/登陆前台处理

2.11.1. 下载相关依赖包

```
npm install --save axios
```

2.11.2. 封装 ajax 请求代码

1) api/ajax.js

```
/*
使用 axios 封装的 ajax 请求函数
函数返回的是 promise 对象
 */

import axios from 'axios'

export default function ajax(url = '', data = {}, type = 'GET') {
  if (type === 'GET') {
    // 准备 url query 参数数据
    let dataStr = '' //数据拼接字符串
    Object.keys(data).forEach(key => {
      dataStr += key + '=' + data[key] + '&'
    })
    if (dataStr !== '') {
      dataStr = dataStr.substring(0, dataStr.lastIndexOf('&'))
      url = url + '?' + dataStr
    }
    // 发送 get 请求
    return axios.get(url)
  } else {
    // 发送 post 请求
    return axios.post(url, data)  // data: 包含请求体数据的对象
  }
```

```
}
```

2) api/index.js

```
/*
包含 n 个接口请求函数的模块
每个函数返回的都是 promise 对象
 */
import ajax from './ajax'

// 请求注册
export const reqRegister = (user) => ajax('/register', user, 'POST')
// 请求登陆
export const reqLogin = (user) => ajax('/login', user, 'POST')
```

3) 配置 ajax 请求的代理: package.json

```
"proxy": "http://localhost:4000"
```

2.11.3. 使用 redux 管理用户信息

1) redux/action-types.js

```
/*
包含所有 action 的 type 常量名称的模块
*/

// 验证成功
export const AUTH_SUCCESS = 'AUTH_SUCCESS'
// 请求出错
export const ERROR_MSG = 'ERROR_MSG'
```

2) redux/actions.js

```
/*
包含所有 action creator 函数的模块
 */

import {
```

```
  AUTH_SUCCESS,
  ERROR_MSG
} from './action-types'
import {
  reqRegister,
  reqLogin
} from '../api'


// 同步错误消息
const errorMsg = (msg) => ({type:ERROR_MSG, data: msg})
// 同步成功响应
const authSuccess = (user) => ({type: AUTH_SUCCESS, data: user})

/*
异步注册
 */
export function register({username, password, password2, type}) {
  // 进行前台表单验证, 如果不合法返回一个同步 action 对象, 显示提示信息
  if (!username || !password || !type) {
    return errorMsg('用户名密码必须输入')
  }
  if (password !== password2) {
    return errorMsg('密码和确认密码不同')
  }
  return async dispatch => {
    // 异步 ajax 请求, 得到响应
    const response = await reqRegister({username, password, type})
    // 得到响应体数据
    const result = response.data
    // 如果是正确的
    if (result.code === 0) {
      // 分发成功的 action
      dispatch(authSuccess(result.data))
    } else {
      // 分发提示错误的 action
      dispatch(errorMsg(result.msg))
    }
  }
}

/*
```

```
异步登陆
 */
export const login = ({username, password}) => {
  if (!username || !password) {
    return errorMsg('用户密码必须输入')
  }
  return async dispatch => {
    const response = await reqLogin({username, password})
    const result = response.data
    if (result.code === 0) {
      dispatch(authSuccess(result.data))
    } else {
      dispatch(errorMsg(result.msg))
    }
  }
}
```

3) redux/reducers.js

```
/*
包含n个根据老的state和action返回新的state的函数的模块
 */
import {combineReducers} from 'redux'

import {
  AUTH_SUCCESS,
  ERROR_MSG
} from './action-types'

const initUser = {
  username: '', // 用户名
  type: '', // 类型
  msg: '', // 错误提示信息
  redirectTo: '' // 需要自动跳转的路由path
}
function user(state = initUser, action) {
  switch (action.type) {
    case AUTH_SUCCESS: // 认证成功
      return {...action.data, redirectTo: '/'}
    case ERROR_MSG: // 错误信息提示
      return {...state, msg: action.data}
```

```
    default:
      return state
  }
}

// 返回合并的reducer
export default combineReducers({
  user
})
```

2.11.4. 注册组件: register.jsx

```
/*
用户注册的路由组件
 */
import React, {Component} from 'react'
import {
  NavBar,
  WingBlank,
  List,
  InputItem,
  WhiteSpace,
  Radio,
  Button
} from 'antd-mobile'
import {connect} from 'react-redux'
import {Redirect} from 'react-router-dom'

import Logo from '../../components/logo/logo'
import {register} from '../../redux/actions'

class Register extends Component {

  state = {
    username: '',
    password: '',
    password2: '',
    type: 'dashen'
  }
```

```
// 处理输入框/单选框变化，收集数据到state
handleChange = (name, value) => {
  this.setState({[name]: value})
}

// 跳转到login路由
toLogin = () => {
  this.props.history.replace('/login')
}

// 注册
register = () => {
  this.props.register(this.state)
}

render() {
  const {type} = this.state
  const {redirectTo, msg} = this.props
  if (redirectTo) {
    return <Redirect to={redirectTo}/>
  }

  return (
    <div>
      <NavBar>硅谷直聘</NavBar>
      <Logo/>
      <WingBlank>
        {msg ? <p className='error-msg'>{msg}</p> : null}
        <List>
          <InputItem
            placeholder='输入用户名'
            onChange={val => this.handleChange('username', val)}
          >
            用户名:
          </InputItem>
          <WhiteSpace/>
          <InputItem
            type='password'
            placeholder='输入密码'
            onChange={val => this.handleChange('password', val)}
          >
```

```
            密   码:
          </InputItem>
          <WhiteSpace/>
          <InputItem
            type='password'
            placeholder='输入确认密码'
            onChange={val => this.handleChange('password2', val)}
          >
            确认密码:
          </InputItem>
          <WhiteSpace/>
          <List.Item>
            <span style={{marginRight: 30}}>用户类型:</span>
            <Radio checked={this.state.type==='dashen'}
                   onClick={() => {this.handleChange('type', 'dashen')}}>大神</Radio>
                  
            <Radio checked={this.state.type==='laoban'}
                   onClick={() => {this.handleChange('type', 'laoban')}}>老板</Radio>
          </List.Item>
          <WhiteSpace/>
          <Button type='primary' onClick={this.register}>注   册
</Button>
          <WhiteSpace/>
          <Button onClick={this.toLogin}>已经有账号</Button>
        </List>
      </WingBlank>
    </div>
  )
 }
}

export default connect(
 state => state.user,
 {register}
)(Register)
```

2.11.5. 登陆组件: login.jsx

```
/*
用户登陆的路由组件
```

```
 */

import React, {Component} from 'react'
import {
  NavBar,
  WingBlank,
  List,
  InputItem,
  WhiteSpace,
  Button
} from 'antd-mobile'
import {connect} from 'react-redux'
import {Redirect} from 'react-router-dom'

import Logo from '../../components/logo/logo'
import {login} from '../../redux/actions'

class Login extends Component {
  state = {
    username: '',
    password: '',
  }

  // 处理输入框/单选框变化，收集数据到state
  handleChange = (name, value) => {
    this.setState({[name]: value})
  }

  // 跳转到注册路由
  toRegister = () => {
    this.props.history.replace('/register')
  }

  // 注册
  login = () => {
    this.props.login(this.state)
  }

  render() {
    const {redirectTo, msg} = this.props
    if (redirectTo) {
      return <Redirect to={redirectTo}/>
```

```
    }

    return (
      <div>
        <NavBar>硅谷直聘</NavBar>
        <Logo/>
        <WingBlank>
          {msg ? <p className='error-msg'>{msg}</p> : null}
          <List>
            <InputItem
              placeholder='输入用户名'
              onChange={val => this.handleChange('username', val)}
            >
              用户名:
            </InputItem>
            <WhiteSpace/>
            <InputItem
              type='password'
              placeholder='输入密码'
              onChange={val => this.handleChange('password', val)}
            >
              密   码:
            </InputItem>
            <WhiteSpace/>
            <Button type='primary' onClick={this.login}>登   陆
</Button>
            <WhiteSpace/>
            <Button onClick={this.toRegister}>还没有账号</Button>
          </List>
        </WingBlank>
      </div>
    )
  }
}

export default connect(
  state => state.user,
  {login}
)(Login)
```

2.11.6. 样式: assets/css/index.less

```
.error-msg {
  color: red;
  text-align: center;
  font-size: 18px;
}

需要在 index.js 中引入
```

2.12. 实现 user 信息完善功能

2.12.1. 下载相关依赖包

```
npm install --save js-cookie
```

2.12.2. 注册/登陆成功的路由跳转

1) 工具模块: utils/index.js

```
/*
包含 n 个工具函数的模块
 */
/*
注册 laoban--> /laobaninfo
注册大神--> /dasheninfo
登陆 laoban --> /laobaninfo 或者 /laoban
登陆大神 --> /dasheninfo 或者 /dashen
 */
export function getRedirectPath(type, header) {
  let path = ''

  // 根据 type 得到 path
  path += type==='laoban' ? '/laoban' : '/dashen'
  // 如果没有头像添加 info
  if(!header) {
    path += 'info'
```

```
  }

  return path
}
```

2) reducers 中使用工具: redux/reducers.js

```
import {getRedirectPath} from '../utils'

case AUTH_SUCCESS: // 认证成功
  const redirectTo = getRedirectPath(action.data.type, action.data.header)
  return {...action.data, redirectTo}
```

2.12.3. 后台路由: routes/index.js

```
// 更新用户路由
router.post('/update', function (req, res) {
  // 得到请求 cookie 的 userid
  const userid = req.cookies.userid
  if(!userid) {// 如果没有, 说明没有登陆, 直接返回提示
    return res.send({code: 1, msg: '请先登陆')};
  }

  // 更新数据库中对应的数据
  UserModel.findByIdAndUpdate({_id: userid}, req.body, function (err, user) {// user
是数据库中原来的数据
    const {_id, username, type} = user
    // node 端 ...不可用
    // const data = {...req.body, _id, username, type}
    // 合并用户信息
    const data = Object.assign(req.body, {_id, username, type})
    // assign(obj1, obj2, obj3,...) // 将多个指定的对象进行合并, 返回一个合并后的对象
    res.send({code: 0, data})
  })
})
```

2.12.4. 更新的 ajax 请求: api/index.js

```
// 更新用户信息
export const reqUpdateUser = (user) => ajax('/update', user, 'POST')
```

2.12.5. redux 更新状态:

1) redux/action-types.js

```
// 接收用户
export const RECEIVE_USER = 'RECEIVE_USER'
// 重置用户
export const RESET_USER = 'RESET_USER'
```

2) redux/actions.js

```
import {
  RECEIVE_USER,
  RESET_USER
} from './action-types'
import {
  reqUpdateUser
} from '../api'

// 同步接收用户
const receiveUser = (user) => ({type: RECEIVE_USER, data: user})
// 同步重置用户
export const resetUser = (msg) => ({type: RESET_USER, data: msg})

/*
异步更新用户
 */
export const updateUser = (user) => {
  return async dispatch => {
    // 发送异步 ajax 请求
    const response = await reqUpdateUser(user)
    const result = response.data
    if (result.code === 0) { // 更新成功
      dispatch(receiveUser(result.data))
    } else { // 失败
```

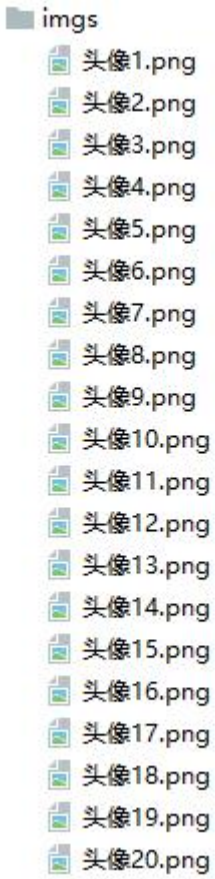
```
      dispatch(resetUser(result.msg))
    }
  }
}
```

3) redux/reducers.js

```
import {
  RECEIVE_USER,
  RESET_USER
} from './action-types'

function user(state = initUser, action) {
  switch (action.type) {
    case RECEIVE_USER: // 接收用户
      return action.data
    case RESET_USER: // 重置用户
      return {...initUser, msg: action.data}
  }
}
```

2.12.6. 用户头像选择组件

1) 添加多个头像图片文件: assets/imgs



2) 组件: components/header-selector/header-selector.jsx

```
/*
选择头像的组件
 */
import React, {Component} from 'react'
import {List, Grid} from 'antd-mobile'
import PropTypes from 'prop-types'

export default class HeaderSelector extends Component {

  static propTypes = {
    setHeader: PropTypes.func.isRequired
  }

  state = {
    icon: null
  }

  constructor(props) {
```

```
    super(props)
    this.headerList = []
    for (var i = 0; i < 20; i++) {
      const text = `头像${i+1}`
      this.headerList.push({text, icon: require(`../../assets/imgs/${text}.png`)})
    }
  }

  selectHeader = ({icon, text}) => {
    // 更新当前组件的状态
    this.setState({icon})
    // 更新父组件的状态
    this.props.setHeader(text)
  }


  render () {

    // 计算头部显示
    const {icon} = this.state
    const gridHeader = icon ? <p>已选择头像: <img src={icon} alt="header"/></p> : '请选择头像'

    return (
      <List renderHeader={() => gridHeader}>
        <Grid data={this.headerList}
              columnNum={5}
              onClick={this.selectHeader}/>
      </List>
    )
  }
}
```

2.12.7. 老板信息完善组件: containers/laoban-info/laoban-info.jsx

```
/*
laoban 信息完善路由组件
 */
import React, {Component} from 'react'
import {NavBar, InputItem, TextareaItem, Button} from 'antd-mobile'
import {connect} from 'react-redux'
```

```
import {Redirect} from 'react-router-dom'

import HeaderSelector from '../../components/header-selector/header-selector'
import {updateUser} from '../../redux/actions'

class LaobanInfo extends Component {

  state = {
    header: '', // 头像名称
    info: '', // 职位简介
    post: '', // 职位名称
    company: '', // 公司名称
    salary: '' // 工资
  }

  handleChange = (name, val) => {
    this.setState({[name]: val})
  }

  // 设置更新header
  setHeader = (header) => {
    this.setState({header})
  }



  render() {
    const {user} = this.props
    // 如果用户信息已完善, 自动跳转到laoban主界面
    if(user.header) {
      return <Redirect to='/laoban'/>
    }

    return (
      <div>
        <NavBar>老板信息完善</NavBar>
        <HeaderSelector setHeader={this.setHeader}/>
        <InputItem onChange={val => this.handleChange('post', val)}>招聘职
位:</InputItem>
        <InputItem onChange={val => this.handleChange('company', val)}>公司名
称:</InputItem>
        <InputItem onChange={val => this.handleChange('salary', val)}>职位薪
资:</InputItem>
```

```
        <TextareaItem title="职位要求:"
                      rows={3}
                      onChange={val => this.handleChange('info', val)}/>

        <Button type='primary' onClick={() => this.props.updateUser(this.state)}>保存
</Button>
      </div>
    )
  }
}

export default connect(
  state => ({user: state.user}),
  {updateUser}
)(LaobanInfo)
```

2.12.8. 大神信息完善组件: containers/dashen-info/dashen-info.jsx

```
/*
大神信息完善路由组件
 */
import React, {Component} from 'react'
import {NavBar, InputItem, TextareaItem, Button} from 'antd-mobile'
import {connect} from 'react-redux'
import {Redirect} from 'react-router-dom'

import HeaderSelector from '../../components/header-selector/header-selector'
import {updateUser} from '../../redux/actions'

class DashenInfo extends Component {

  state = {
    header: '', // 头像名称
    info: '', // 个人简介
    post: '', // 求职岗位
  }

  handleChange = (name, val) => {
    this.setState({[name]: val})
  }
```

```
  // 设置更新header
  setHeader = (header) => {
    this.setState({header})
  }


  render() {

    const {user} = this.props
    // 如果用户信息已完善，自动跳转到大神主界面
    if (user.header) {
      return <Redirect to='/dashen'/>
    }

    return (
      <div>
        <NavBar>大神信息完善</NavBar>
        <HeaderSelector setHeader={this.setHeader}/>
        <InputItem onChange={val => this.handleChange('post', val)}>求职岗
位:</InputItem>
        <TextareaItem title="个人介绍:"
                      rows={3}
                      onChange={val => this.handleChange('info', val)}/>

        <Button type='primary' onClick={() => this.props.updateUser(this.state)}>保存
</Button>
      </div>
    )
  }
}

export default connect(
  state => ({user: state.user}),
  {updateUser}
)(DashenInfo)
```

2.12.9. main 组件: containers/main/main.jsx

```
/*
应用主界面路由组件
 */
import React, {Component} from 'react'
import {Switch, Route} from 'react-router-dom'
import Cookies from 'js-cookies'

import LaobanInfo from '../laoban-info/laoban-info'
import DashenInfo from '../dashen-info/dashen-info'

export default class Main extends Component {
  render() {

// 如果浏览器中没有保存userid的cookie, 直接跳转到Login
    const userid = Cookies.get('userid')
    if (!userid) {
      this.props.history.replace('/login')
      return null
    }

    return (
      <div>
        <Switch>
          <Route path='/laobaninfo' component={LaobanInfo}/>
          <Route path='/dasheninfo' component={DashenInfo}/>
        </Switch>
      </div>
    )
  }
}
```

2.13. 搭建整体界面

2.13.2. 后台路由: routes/index.js

```
// 根据cookie获取对应的user
router.get('/user', function (req, res) {
```

```
  // 取出cookie 中的userid
  const userid = req.cookies.userid
  if(!userid) {
    return res.send({code: 1, msg: '请先登陆'})
  }

  // 查询对应的user
  UserModel.findOne({_id: userid}, filter, function (err, user) {
    return res.send({code: 0, data: user})
  })
})
```

2.13.3. ajax 请求模块: api/index.js

```
// 查看用户信息(根据cookie)
export const reqUser = () => ajax('/user')
```

2.13.4. redux 管理状态

1) redux/actions.js

```
/*
异步获取用户
 */
export const getUser = () => {
  return async dispatch => {
    const response = await reqUser()
    const result = response.data
    if (result.code === 0) {
      dispatch(receiveUser(result.data))
    } else {
      dispatch(resetUser(result.msg))
    }
  }
}
```

2.13.5. main 路由的子路由

1) containers/laoban/laoban.jsx

```
/*
老板的主路由组件
 */

import React, {Component} from 'react'

export default class Laoban extends Component {
  render() {
    return (
      <div>老板列表</div>
    )
  }
}
```

2) containers/dashen/dashen.jsx

```
/*
大神的主路由组件
 */

import React, {Component} from 'react'

export default class Dashen extends Component {
  render() {
    return (
      <div>大神列表</div>
    )
  }
}
```

3) containers/message/message.jsx

```
/*
对话消息列表组件
 */
```

```
import React, {Component} from 'react'

export default class Message extends Component {
  render() {
    return (
      <div>Message 列表</div>
    )
  }
}
```

4) containers/personal/personal.jsx

```
/*
用户个人中心路由组件
 */

import React, {Component} from 'react'

export default class Personal extends Component {
  render() {
    return (
      <div>个人中心</div>
    )
  }
}
```

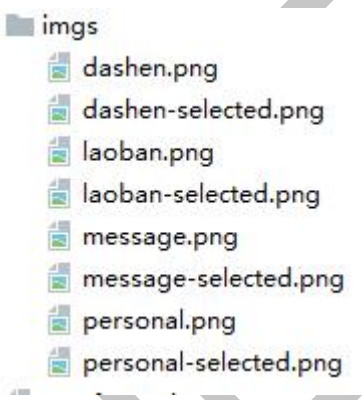
5) components/not-found/not-found.jsx

```
/*
提示找不到页面的 UI 路由组件
 */

import React from "react"
import {Button} from "antd-mobile"

class NotFound extends React.Component {
  render() {
    return (
      <div>
        <div>
          <h2>抱歉，找不到该页面!</h2>
```

```
          <Button
            type="primary"
            onClick={() => this.props.history.replace("/")}
          >
            回到首页
          </Button>
        </div>
      </div>
    )
  }
}

export default NotFound
```

2.13.6. 底部导航组件

1) 相关图标图片



2) 组件: components/nav-footer/nav-footer.jsx

```
/*
底部导航的UI 组件
 */

import React from 'react'
import PropTypes from 'prop-types'
import {TabBar} from 'antd-mobile'
import {withRouter} from 'react-router-dom'

const Item = TabBar.Item
```

```
class NavFooter extends React.Component {

  static propTypes = {
    navList: PropTypes.array.isRequired
  }

  render() {
    // nav.hide = true/false hide 代表当前项应该被隐藏
    const navList = this.props.navList.filter(nav => !nav.hide) // 回调函数返回值为true, 当前元素就会留下, 否则不留
    // 当前请求的路径
    const {pathname} = this.props.location
    return (
      <TabBar>
        {
          navList.map((nav, index) => (
            <Item key={nav.path}
                  title={nav.text}
                  icon={{uri: require(`./imgs/${nav.icon}.png`)}}
                  selectedIcon={{uri: require(`./imgs/${nav.icon}-selected.png`)}}
                  selected={pathname===nav.path}
                  onPress={() => {
                    this.props.history.replace(nav.path)
                  }}
            />
          ))
        }
      </TabBar>
    )
  }
}

export default withRouter(NavFooter) // 让非路由组件可以访问到路由组件的API
```

2.13.7. 应用主界面路由组件

1) containers/main/main.jsx

```
/*
应用主界面路由组件
 */
```

```
import React from 'react'
import {Route, Switch, Redirect} from 'react-router-dom'
import Cookies from 'js-cookie' // get(), set()
import {connect} from 'react-redux'
import {NavBar} from 'antd-mobile'

import LaobanInfo from '../laoban-info/laoban-info'
import DashenInfo from '../dashen-info/dashen-info'
import Dashen from '../dashen/dashen'
import Laoban from '../laoban/laoban'
import Message from '../message/message'
import Personal from '../personal/personal'
import NotFound from '../../components/not-found/not-found'
import NavFooter from '../../components/nav-footer/nav-footer'

import {getUser} from '../../redux/actions'
import {getRedirectPath} from '../../utils'

class Main extends React.Component {
// 组件类和组件对象
  // 给组件对象添加属性
  navList = [
    {
      path: '/laoban', // 路由路径
      component: Laoban,
      title: '大神列表',
      icon: 'dashen',
      text: '大神',
    },
    {
      path: '/dashen', // 路由路径
      component: Dashen,
      title: '老板列表',
      icon: 'laoban',
      text: '老板',
    },
    {
      path: '/message', // 路由路径
      component: Message,
      title: '消息列表',
      icon: 'message',
      text: '消息',
```

```
  },
  {
    path: '/personal', // 路由路径
    component: Personal,
    title: '用户中心',
    icon: 'personal',
    text: '个人',
  }
]

componentDidMount() {
  // cookie 中有 userid
  // redux 中的 user 是空对象
  const userid = Cookies.get('userid')
  const {user} = this.props
  if (userid && !user._id) {
    this.props.getUser()  // 获取 user 并保存到 redux 中
  }
}


render() {
  // 得到当前请求的 path
  const pathname = this.props.location.pathname
  // 判断用户是否已登陆(过)(cookie 中 userid 是否有值)
  const userid = Cookies.get('userid')
  if (!userid) { // 如果没值, 自动跳转到登陆界面
    return <Redirect to='/login'/>
  }
  // cookie 中有 userid
  // redux 中的 user 是否有数据
  const {user} = this.props
  if (!user._id) {
    return null  // 不做任何显示
  } else {
    // 请求根路径时, 自动 跳转到对应的用户主界面
    if (pathname === '/') {
      const path = getRedirectPath(user.type, user.header)
      return <Redirect to={path}/>
    }

    // 指定哪个 nav 应该被隐藏
```

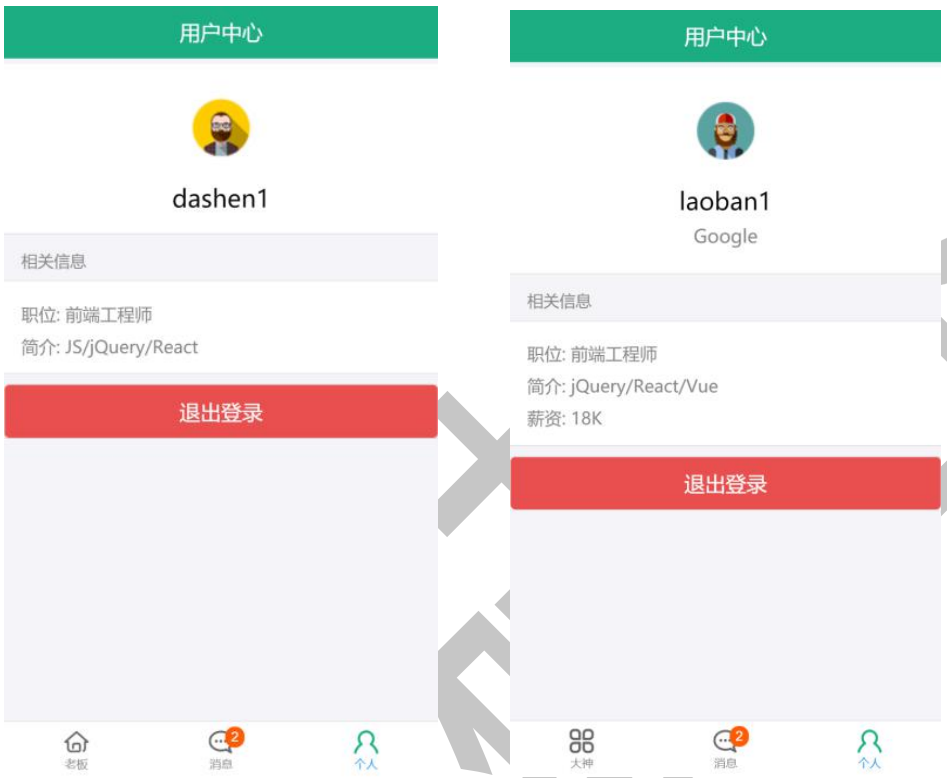
```
      if (user.type === 'laoban') {
        this.navList[1].hide = true
      } else {
        this.navList[0].hide = true
      }
    }

    // 得到当前的nav
    const currentNav = this.navList.find(nav => nav.path === pathname)

    return (
      <div>
        {currentNav ? <NavBar className='stick-top'>{currentNav.title}</NavBar> : null}
        <Switch>
          <Route path='/laobaninfo' component={LaobanInfo}></Route>
          <Route path='/dasheninfo' component={DashenInfo}></Route>

          <Route path='/dashen' component={Dashen}></Route>
          <Route path='/laoban' component={Laoban}></Route>
          <Route path='/message' component={Message}></Route>
          <Route path='/personal' component={Personal}></Route>
          <Route component={NotFound}></Route>
        </Switch>

        {currentNav ? <NavFooter unReadCount={this.props.unReadCount}
navList={this.navList}/> : null}
      </div>
    )
  }
}

export default connect(
  state => ({user: state.user}),
  {getUser}
)(Main)
```

2) 样式: assets/css/index.less

```
.am-tab-bar{  /*使导航始终在底部*/
  position: fixed;
  bottom: 0;
  width: 100%;
```

```
}
```

2.14. 个人中心功能



2.14.1. 个人中心组件: containers/personal/personal.jsx

1) 静态组件

```
/*
用户个人中心路由组件
 */

import React from 'react'
import {Result, List, WhiteSpace, Button} from 'antd-mobile'

const Item = List.Item
const Brief = Item.Brief

export default class Personal extends React.Component {
```

```
  render() {

    return (
      <div>
        <Result
          img={<img src={require(`../../assets/imgs/头像1.png`)} style={{width: 50}}
alt="header"/>}
          title='张三'
          message='IBM'
        />

        <List renderHeader={() => '相关信息'}>
          <Item multipleLine>
            <Brief>职位: 前端工程师</Brief>
            <Brief>简介: React/Vue/jQuery</Brief>
            <Brief>薪资: 20k</Brief>
          </Item>
        </List>
        <WhiteSpace/>
        <List>
          <Button type='warning'>退出登录</Button>
        </List>
      </div>
    )
  }
}
```

2) 动态组件

```
/*
用户个人中心路由组件
 */

import React from 'react'
import {Result, List, WhiteSpace, Modal, Button} from 'antd-mobile'
import {connect} from 'react-redux'
import Cookies from 'js-cookie'

import {resetUser} from '../../redux/actions'
const Item = List.Item
const Brief = Item.Brief
```

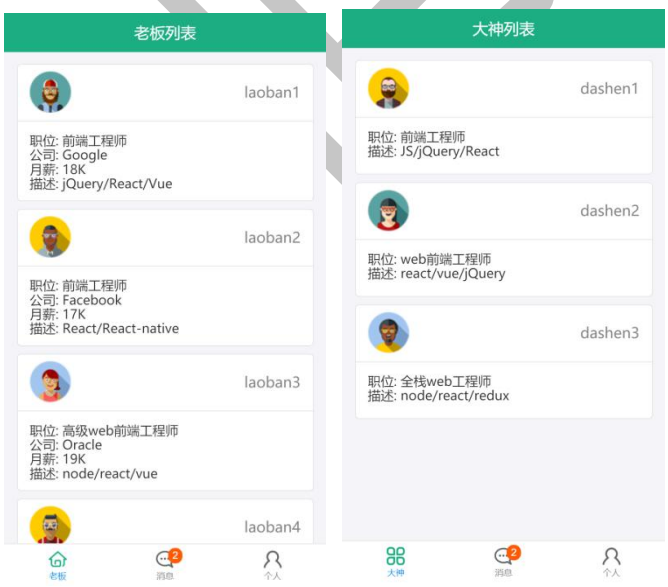
```
class Personal extends React.Component {

  handleLogout = () => {
    Modal.alert('退出', '确认退出登录吗?', [
      {
        text: '取消',
        onPress: () => console.log('cancel')
      },
      {
        text: '确认',
        onPress: () => {
          // 清除 cookie 中的 userid
          Cookies.remove('userid')
          // 重置 redux 中的 user 状态
          this.props.resetUser()
        }
      }
    ])
  }

  render() {

    const {username, header, post, info, salary, company} = this.props.user

    return (
      <div style={{marginTop: 50}}>
        <Result
          img={<img src={require(`../../assets/imgs/${header}.png`)} style={{width:
50}} alt="header"/>}
          title={username}
          message={company}
        />

        <List renderHeader={() => '相关信息'}>
          <Item multipleLine>
            <Brief>职位: {post}</Brief>
            <Brief>简介: {info}</Brief>
            {salary ? <Brief>薪资: {salary}</Brief> : null}
          </Item>
        </List>
        <WhiteSpace/>
        <List>
```

```
          <Button onClick={this.handleLogout} type='warning'>退出登录</Button>
        </List>
      </div>
    )
  }
}

export default connect(
  state => ({user: state.user}),
  {resetUser}
)(Personal)
```

2.14.2. 样式: assets/css/index.less

```
.am-tab-bar{  /*使导航始终在底部*/
  position: fixed;
  bottom: 0;
  width: 100%;
  height: inherit;  /*高度包裹内容*/
}
```

2.15. 老板/大神列表功能



2.15.1. 后台路由: routes/index.js

```
/*
查看用户列表
 */
router.get('/list',function(req, res){
  const { type } = req.query
  UserModel.find({type},function(err,users){
    return res.json({code:0, data: users})
  })
})
```

2.15.2. ajax 请求模块: api/index.js

```
// 请求获取用户列表
export const reqUserList = (type) => ajax('/list', {type})
```

2.15.3. redux 管理状态

1) redux/action-types.js

```
export const RECEIVE_USER_LIST = 'receive_user_list' // 接收用户列表
```

2) redux/actions.js

```
import {
  RECEIVE_USER_LIST
} from './action-types'
import {
  reqUserList
} from '../api'

// 用户列表
const receiveUserList = (users) => ({type: RECEIVE_USER_LIST, data: users})
// 异步获取用户列表
export const getUserList = (type) => {
  return async dispatch => {
    const response = await reqUserList(type)
```

```
    const result = response.data
    if (result.code === 0) {
      dispatch(receiveUserList(result.data))
    }
  }
}
```

3) redux/reducers.js

```
const initUserList = []

function userList(state = initUserList, action) {
  switch (action.type) {
    case RECEIVE_USER_LIST:
      return action.data
    default:
      return state
  }
}

export default combineReducers({
  user,
  userList
})
```

2.15.4. 用户列表组件: components/user-list/user-list.jsx

1) 静态组件

```
/*
用户列表的UI 组件
 */
import React from 'react'
import {Card, WingBlank, WhiteSpace} from 'antd-mobile'

const Header = Card.Header
const Body = Card.Body

class UserList extends React.Component {
```

```
  render() {
    return (
      <WingBlank>
        <div>
          <WhiteSpace/>
          <Card>
            <Header
              thumb={require(`../../assets/imgs/头像 1.png`)}
              extra='aa'
            />
            <Body>
              <div>职位: 前端工程师</div>
              <div>公司: Google</div>
              <div>月薪: 18K</div>
              <div>描述: React/Vue</div>
            </Body>
          </Card>
        </div>
      </WingBlank>
    )
  }
}

export default UserList
```

2) 动态组件

```
/*
用户列表的UI 组件
 */
import React from 'react'
import PropTypes from 'prop-types'
import {Card, WingBlank, WhiteSpace} from 'antd-mobile'

const Header = Card.Header
const Body = Card.Body

class UserList extends React.Component {

  static propsTypes = {
    userList: PropTypes.array.isRequired
  }
```

```
  render() {
    return (
      <WingBlank>
        {
          this.props.userList.map(user => (
            <div key={user._id}>
              <WhiteSpace/>
              <Card>
                <Header
                  thumb={user.header ?
require(`../../assets/imgs/${user.header}.png`) : null}
                  extra={user.username}
                />
                <Body>
                <div>职位: {user.post}</div>
                {user.company ? <div>公司: {user.company}</div> : null}
                {user.salary ? <div>月薪: {user.salary}</div> : null}
                <div>描述: {user.info}</div>
                </Body>
              </Card>
            </div>
          ))
        }
      </WingBlank>
    )
  }
}

export default UserList
```

2.15.5. 老板路由组件: containers/laoban/laoban.jsx

```
/*
老板的主路由组件
 */
import React from 'react'
import {connect} from 'react-redux'

import {getUserList} from '../../redux/actions'
```

```
import UserList from '../../components/user-list/user-list'

class Laoban extends React.Component {

  componentDidMount() {
    this.props.getUserList('dashen')
  }

  render() {
    return <UserList userList={this.props.userList}></UserList>
  }
}

export default connect(
  state => ({userList: state.userList}),
  {getUserList}
)(Laoban)
```

2.15.6. 大神路由组件: containers/dashen/dashen.jsx

```
/*
大神的主路由组件
 */
import React from 'react'
import {connect} from 'react-redux'

import {getUserList} from '../../redux/actions'
import UserList from '../../components/user-list/user-list'

class Dashen extends React.Component {

  componentDidMount() {
    this.props.getUserList('laoban')
  }

  render() {
    return <UserList userList={this.props.userList}></UserList>
  }
}
```

```
export default connect(
  state => ({userList: state.userList}),
  {getUserList}
)(Dashen)
```

2.15.7. 优化头部和底部的布局效果

1) 问题:

列表滑动, 顶部会跟着滑动而不可见

底部导航会遮挡列表的部分显示

2) Index.less

```
.stick-top { /*固定在顶部*/
  z-index: 10;
  position: fixed;
  top: 0;
  width: 100%;
}
```

3) main.jsx

```
<NavBar className='stick-top'>{currentNav.title}</NavBar>
```

4) user-list.jsx

```
<WingBlank style={{marginTop: 50, marginBottom: 50}}>
```

2.16. 实时聊天功能

2.16.1. 下载相关依赖包

```
npm install --save socket.io
```

2.16.2 socket.io 介绍和使用

1) 介绍

- socket.io 是一个能实现多人远程实时通信(聊天)的库
- 它包装的是 H5 WebSocket 和轮询，如果是较新的浏览器内部使用 WebSocket，如果浏览器不支持，那内部就会使用轮询实现实时通信
- 学习资料
  https://socket.io/get-started/chat/
  http://blog.csdn.net/neuq_zxy/article/details/77531126

2) 编码例子

- 服务器端

socketIO/test.js

```
module.exports = function (server) {
  // 得到IO 对象
  const io = require('socket.io')(server)
  // 监视连接(当有一个客户连接上时回调)
  io.on('connection', function (socket) {
    console.log('soketio connected')
    // 绑定 sendMsg 监听，接收客户端发送的消息
    socket.on('sendMsg', function (data) {
      console.log('服务器接收到浏览器的消息', data)
      // 向客户端发送消息(名称，数据)
      io.emit('receiveMsg', data.name + '_' + data.date)
      console.log('服务器向浏览器发送消息', data)
    })
  })
}
```

bin/ww

```
// start up socketio server
require('../socketIO/test')(server)
```

- 客户端

src/test/socketio_test.js

```
// 引入客户端 io
import io from 'socket.io-client'
```

```
// 连接服务器, 得到代表连接的socket 对象
const socket = io('ws://localhost:4000')

// 绑定'receiveMessage'的监听, 来接收服务器发送的消息
socket.on('receiveMsg', function (data) {
  console.log('浏览器端接收到消息:', data)
})

// 向服务器发送消息
socket.emit('sendMsg', {name: 'Tom', date: Date.now()})
console.log('浏览器端向服务器发送消息:', {name: 'Tom', date: Date.now()})
```

src/index.js

```
import './test/socketio_test'
```

2.16.3. 后台实现

1) 添加新的数据库集合模型: db/models.js

```
// 定义chats 集合的文档结构
const chatSchema = mongoose.Schema({
  from: {type: String, required: true}, // 发送用户的id
  to: {type: String, required: true}, // 接收用户的id
  chat_id: {type: String, required: true}, // from 和to 组成的字符串
  content: {type: String, required: true}, // 内容
  read: {type:Boolean, default: false}, // 标识是否已读
  create_time: {type: Number} // 创建时间
})
// 定义能操作chats 集合数据的Model
const ChatModel = mongoose.model('chat', chatSchema)
// 向外暴露Model
exports.ChatModel = ChatModel
```

2) 新增路由: routes/index.js

```
// 引入 UserModel,ChatModel
const models = require('../db/models')
const UserModel = models.UserModel
const ChatModel = models.ChatModel

/*
获取当前用户所有相关聊天信息列表
 */
router.get('/msglist', function (req, res) {
  // 获取 cookie 中的 userid
  const userid = req.cookies.userid
  // 查询得到所有 user 文档数组
  UserModel.find(function (err, userDocs) {
    // 用对象存储所有 user 信息: key 为 user 的 _id, val 为 name 和 header 组成的 user 对象
    const users = {} // 对象容器
    userDocs.forEach(doc => {
      users[doc._id] = {username: doc.username, header: doc.header}
    })
    /*
    查询 userid 相关的所有聊天信息
     参数 1: 查询条件
     参数 2: 过滤条件
     参数 3: 回调函数
    */
    ChatModel.find({'$or': [{from: userid}, {to: userid}]}, filter, function (err,
chatMsgs) {
      // 返回包含所有用户和当前用户相关的所有聊天消息的数据
      res.send({code: 0, data: {users, chatMsgs}})
    })
  })
})

/*
修改指定消息为已读
 */
router.post('/readmsg', function (req, res) {
  // 得到请求中的 from 和 to
  const from = req.body.from
  const to = req.cookies.userid
```

```
  /*
  更新数据库中的 chat 数据
  参数 1: 查询条件
  参数 2: 更新为指定的数据对象
  参数 3: 是否 1 次更新多条, 默认只更新一条
  参数 4: 更新完成的回调函数
   */
  ChatModel.update({from, to, read: false}, {read: true}, {multi: true}, function (err,
doc) {
    console.log('/readmsg', doc)
    res.send({code: 0, data: doc.nModified}) // 更新的数量
  })
})
```

3) 使用上 socket.io 实现实时通信

server_socket.js

```
/*
启动 socket.io 服务的函数
 */
module.exports = function (server) {
  // 引入操作 chats 集合数据的 Model
  const ChatModel = require('./db/models').ChatModel
  // 得到操作服务器端 sokectIO 的 io 对象
  const io = require('socket.io')(server)

  // 绑定监听回调: 客户端连接上服务器
  io.on('connection', function(socket) { // socket 代表连接
    console.log('有客户端连接上了服务器')
    // 绑定 sendMsg 监听, 接收客户端发送的消息
    socket.on('sendMessage', function({from, to, content}) {
      console.log('服务器接收到数据', {from, to, content})
      // 将接收到的消息保存到数据库
      const chat_id = [from, to].sort().join('_')
      const create_time = Date.now()
      const chatModel = new ChatModel({chat_id, from, to, create_time, content})
      chatModel.save(function (err, chatMsg) {
        // 保存完成后, 向所有连接的客户端发送消息
        io.emit('receiveMessage', chatMsg) // 全局发送, 所有连接的客户端都可以收到
```

```
        console.log('向所有连接的客户端发送消息', chatMsg)
      })
    })
  })
}
```

bin/www

```
// use server socket
require('../server_socket')(server)
```

2.16.4. ajax 请求模块: api/index.js

```
// 请求获取当前用户的所有聊天记录
export const reqChatMsgList = () => ajax('/msglist')
// 标识查看了指定用户发送的聊天信息
export const reqReadChatMsg = (from) => ajax('/readmsg', {from}, 'POST')
```

2.16.5. redux 管理状态

1) redux/action-types.js

```
export const RECEIVE_MSG_LIST = 'receive_msg_list' // 接收消息列表
export const RECEIVE_MSG = 'receive_msg' // 接收一条消息
export const MSG_READ = 'msg_read' // 标识消息已读
```

2) redux/actions.js

```
import io from 'socket.io-client'
import {
  RECEIVE_MSG_LIST,
  RECEIVE_MSG,
  MSG_READ
} from './action-types'
import {
  reqChatMsgList,
  reqReadChatMsg
```

```
} from '../api'


// 接收消息列表的同步action
const receiveMsgList = ({users, chatMsgs, userid}) => ({type: RECEIVE_MSG_LIST, data: {users, chatMsgs, userid}})
// 接收消息的同步action
const receiveMsg = (chatMsg, isToMe) => ({type: RECEIVE_MSG, data: {chatMsg, isToMe}})
// 读取了消息的同步action
const msgRead = ({from, to, count}) => ({type: MSG_READ, data: {from, to, count}})

/*
初始化客户端socketio
1. 连接服务器
2. 绑定用于接收服务器返回chatMsg的监听
 */
function initIO(dispatch, userid) {
  if(!io.socket) {
    io.socket = io('ws://localhost:4000')
    io.socket.on('receiveMsg', (chatMsg) => {
      if(chatMsg.from===userid || chatMsg.to===userid) {
        dispatch(receiveMsg(chatMsg, chatMsg.to === userid))
      }
    })
  }
}

/*
获取当前用户相关的所有聊天消息列表
(在注册/登陆/获取用户信息成功后调用)
 */
async function getMsgList(dispatch, userid) {
  initIO(dispatch, userid)
  const response = await reqChatMsgList()
  const result = response.data
  if(result.code===0) {
    const {chatMsgs, users} = result.data
    dispatch(receiveMsgList({chatMsgs, users, userid}))
  }
}

/*
```

```
发送消息的异步action
 */
export const sendMsg = ({from, to, content}) => {
  return async dispatch => {
    io.socket.emit('sendMsg', {from, to, content})
  }
}

/*
更新读取消息的异步action
 */
export const readMsg = (userid) => {

  return async (dispatch, getState) => {
    const response = await reqReadChatMsg(userid)
    const result = response.data
    if(result.code===0) {
      const count = result.data
      const from = userid
      const to = getState().user._id
      dispatch(msgRead({from, to, count}))
    }
  }
}
```

3) redux/reducers.js

```
import {
  RECEIVE_MSG_LIST,
  RECEIVE_MSG,
  MSG_READ
} from "./action-types"

// 初始chat对象
const initChat = {
  chatMsgs: [], // 消息数组 [{from: id1, to: id2}{}]
  users: {}, // 所有用户的集合对象{id1: user1, id2: user2}
  unReadCount: 0 // 未读消息的数量
}
```

```
// 管理聊天相关信息数据的reducer
function chat(state=initChat, action) {
  switch (action.type) {
    case RECEIVE_MSG:
      var {chatMsg, userid} = action.data
      return {
        chatMsgs: [...state.chatMsgs, chatMsg],
        users: state.users,
        unReadCount: state.unReadCount + (!chatMsg.read && chatMsg.to===userid ? 1 : 0)
      }
    case RECEIVE_MSG_LIST:
      var {chatMsgs, users, userid} = action.data
      return {
        chatMsgs,
        users,
        unReadCount: chatMsgs.reduce((preTotal, msg) => { // 别人发给我的未读消息
          return preTotal + (!msg.read&&msg.to===userid ? 1 : 0)
        }, 0)
      }
    case MSG_READ:
      const {count, from, to} = action.data

      return {
        chatMsgs: state.chatMsgs.map(msg => {
          if(msg.from===from && msg.to===to && !msg.read) {
            // msg.read = true  // 不能直接修改状态
            return {...msg, read: true}
          } else {
            return msg
          }
        }),
        users: state.users,
        unReadCount: state.unReadCount-count
      }
    default:
      return state
  }
}
// 向外暴露整合所有reducer 函数的结果
export default combineReducers({ // 返回的依然是一个reducer 函数
  user,
  userList,
```

```
  chat
})
```

2.16.6. 聊天组件: containers/chat/chat.jsx

1) 静态组件

```
/*
对话聊天的路由组件
 */

import React, {Component} from 'react'
import {NavBar, List, InputItem} from 'antd-mobile'

const Item = List.Item

export default class Chat extends Component {

  render() {
    return (
      <div id='chat-page'>
        <NavBar>aa</NavBar>
        <List>
          <Item
            thumb={require('../../assets/imgs/头像 1.png')}
          >
            你好
          </Item>
          <Item
            thumb={require('../../assets/imgs/头像 1.png')}
          >
            你好 2
          </Item>
          <Item
            className='chat-me'
            extra='我'
          >
            很好
          </Item>
```

```
          <Item
            className='chat-me'
            extra='我'
          >
            很好 2
          </Item>
        </List>

        <div className='am-tab-bar'>
          <InputItem
            placeholder="请输入"
            extra={
              <span>发送</span>
            }
          />
        </div>
      </div>
    )
  }
}
```

css/index.less

```
/*消息文本居右*/
#chat-page .chat-me .am-list-extra {
  flex-basis: auto;  /*宽度包裹内容*/
}
#chat-page .chat-me .am-list-content {
  padding-right: 15px;
  text-align: right;
}
```

2) 动态组件

```
/*
对话聊天的路由组件
 */
import React, {Component} from 'react'
import {NavBar, List, InputItem, Icon} from 'antd-mobile'
import {connect} from 'react-redux'
```

```
import {sendMsg} from '../../redux/actions'

const Item = List.Item

class Chat extends Component {
  state = {
    content: ''
  }

  submit = () => {
    const content = this.state.content.trim()
    const to = this.props.match.params.userid
    const from = this.props.user._id
    this.props.sendMsg({from, to, content})
    this.setState({content: ''})
  }

  render() {
    const {user} = this.props
    const {chatMsgs, users} = this.props.chat
    const targetId = this.props.match.params.userid
    if(!users[targetId]) {
      return null
    }
    const meId = user._id
    const chatId = [targetId, meId].sort().join('_')
    const msgs = chatMsgs.filter(msg => msg.chat_id===chatId)
    const targetIcon = users[targetId] ?
require(`../../assets/imgs/${users[targetId].header}.png`) : null

    return (
      <div id='chat-page'>
        <NavBar
          className='stick-top'
          icon={<Icon type='left'/>}
          onLeftClick={() => this.props.history.goBack()}
        >
          {users[targetId].username}
        </NavBar>
        <List style={{marginBottom:50, marginTop:50}}>
          {
            msgs.map(msg => {
```

```
            if(msg.from===targetId) {
              return (
                <Item
                  key={msg._id}
                  thumb={targetIcon}
                >
                  {msg.content}
                </Item>
              )
            } else {
              return (
                <Item
                  key={msg._id}
                  className='chat-me'
                  extra='我'
                >
                  {msg.content}
                </Item>
              )
            }
          })
        }
      </List>

      <div className='am-tab-bar'>
        <InputItem
          placeholder="请输入"
          value={this.state.content}
          onChange={val => this.setState({content: val})}
          extra={
            <span onClick={this.submit}>发送</span>
          }
        />
      </div>
    </div>
  )
 }
}

export default connect(
  state => ({user: state.user, chat: state.chat}),
  {sendMsg}
```

```
)(Chat)
```

3) 列表自动滑到底部显示

```
componentDidMount() {
  // 初始显示列表
  window.scrollTo(0, document.body.scrollHeight)
}

componentDidUpdate () {
  // 更新显示列表
  window.scrollTo(0, document.body.scrollHeight)
}
```

4) 表情功能

本质就是一个字符文本, 可以作用为字符串直接使用, 各个操作系统能显示
在线可用表情: https://emojipedia.org/

```
state = {
  content: '', //输入的聊天内容
  isShow: false // 是否显示表情列表
}

componentWillMount () {
  this.emojis = ['', '', '', '', '', '', '', '', '', '', '', '', '', '',
    '', '', '', '', '', '', '', '', '', '', '', '', '', '',
    '', '', '', '', '', '', '', '', '', '', '', '', '', '']
  this.emojis = this.emojis.map(value => ({text: value}))
  // console.log(this.emojis)
}

// 切换表情列表的显示
toggleShow = () => {
  const isShow = !this.state.isShow
  this.setState({isShow})
  if(isShow) {
    // 异步手动派发 resize 事件,解决表情列表显示的 bug
```

```
    setTimeout(() => {
      window.dispatchEvent(new Event('resize'))
    }, 0)
  }
}

<div className='am-tab-bar'>
  <InputItem
    placeholder="请输入"
    value={this.state.content}
    onChange={val => this.setState({content: val})}
    onFocus={() => this.setState({isShow: false})}
    extra={
      <span>
        <span onClick={this.toggleShow}
              style={{marginRight: 10}}></span>
        <span onClick={this.submit}>发送</span>
      </span>
    }
  />
  {
    this.state.isShow ? (
      <Grid
        data={this.emojis}
        columnNum={8}
        carouselMaxRow={4}
        isCarousel={true}
        onClick={(item) => {
          this.setState({content: this.state.content + item.text})
        }}
      />
    ) : null
  }
</div>
```

2.16.7. 消息列表组件: containers/message/message.jsx

1) 静态组件

```
/*
对话消息列表组件
 */
import React, {Component} from 'react'
import {connect} from 'react-redux'
import {List, Badge} from 'antd-mobile'

const Item = List.Item
const Brief = Item.Brief

class Message extends Component {

  render() {
    return (
      <List>
        <Item
          extra={<Badge text={3}/>}
          thumb={require(`../../assets/images/头像 1.png`)}
          arrow='horizontal'
        >
          你好
          <Brief>nr1</Brief>
        </Item>
        <Item
          extra={<Badge text={0}/>}
          thumb={require(`../../assets/images/头像 2.png`)}
          arrow='horizontal'
        >
          你好 2
          <Brief>nr2</Brief>
        </Item>
      </List>
    )
  }
}
```

```
export default Message
```

2) 动态组件

```
/*
对话消息列表组件
 */
import React, {Component} from 'react'
import {connect} from 'react-redux'
import {List, Badge} from 'antd-mobile'

const Item = List.Item
const Brief = Item.Brief

/*
得到所有聊天的最后msg组成的数组
[msg1, msg2, msg3..]

// 1. 使用{}进行分组(chat_id), 只保存每个组最后一条msg:  {chat_id1: lastMsg1, chat_id2: lastMsg2}

// 2. 得到所有分组的lastMsg组成数组: Object.values(lastMsgsObj)  [lastMsg1, lastMsg2]

// 3. 对数组排序(create_time, 降序)
 */
function getLastMsgs(chatMsgs, userid) {
  // 1. 使用{}进行分组(chat_id), 只保存每个组最后一条msg:  {chat_id1: lastMsg1, chat_id2: lastMsg2}
  const lastMsgsObj = {}
  chatMsgs.forEach(msg => {

    msg.unReadCount = 0

    // 判断当前msg对应的lastMsg是否存在
    const chatId = msg.chat_id
    const lastMsg = lastMsgsObj[chatId]
    // 不存在
    if(!lastMsg) {
      // 将msg保存为lastMsg
      lastMsgsObj[chatId] = msg
      // 别人发给我的未读消息
```

```
      if(!msg.read && userid===msg.to) {
        // 指定msg 上的未读数量为1
        msg.unReadCount = 1
      }

    } else {// 存在
      // 如果msg 的创建时间晚于lastMsg 的创建时间, 替换
      if (msg.create_time>lastMsg.create_time) {
        lastMsgsObj[chatId] = msg
        // 将原有保存的未读数量保存到新的lastMsg
        msg.unReadCount = lastMsg.unReadCount
      }
      // 别人发给我的未读消息
      if(!msg.read && userid===msg.to) {
        // 指定msg 上的未读数量为1
        msg.unReadCount++
      }
    }
  })

  // 2. 得到所有分组的lastMsg 组成数组: Object.values(lastMsgsObj)  [lastMsg1, lastMsg2]
  const lastMsgs = Object.values(lastMsgsObj)

  // 3. 对数组排序(create_time, 降序)
  lastMsgs.sort(function (msg1, msg2) {
    return msg2.create_time-msg1.create_time
  })

  return lastMsgs
}

class Message extends Component {

  render() {
    // 得到props 中的user 和chat
    const {user, chat} = this.props
    // 得到当前用户的id
    const meId = user._id
    // 得到所用用户的集合对象users 和所有聊天的数组
    const {users, chatMsgs} = chat
    // 得到所有聊天的最后消息的数组
    const lastMsgs = getLastMsgs(chatMsgs, meId)
```

```
    return (
      <List style={{marginTop:50, marginBottom: 50}}>
        {
          lastMsgs.map(msg => {
            const targetId = msg.from === meId ? msg.to : msg.from
            const targetUser = users[targetId]
            const avatarImg = targetUser.avatar ?
require(`../../assets/imgs/${targetUser.avatar}.png`) : null

            return (
              <Item
                key={msg._id}
                extra={<Badge text={msg.unReadCount}/>}
                thumb={avatarImg}
                arrow='horizontal'
                onClick={() => this.props.history.push(`/chat/${targetId}`)}
              >
                {msg.content}
                <Brief>{targetUser.name}</Brief>
              </Item>
            )
          })
        }
      </List>
    )
  }
}

export default connect(
  state => ({
    user: state.user,
    chat: state.chat
  })
)(Message)
```

2.16.8. 应用主界面路由组件: containers/main/main.jsx

```
/*
应用主界面路由组件
```

```
 */
import React from 'react'
import {Route, Switch, Redirect} from 'react-router-dom'
import Cookies from 'js-cookie' // get(), set()
import {connect} from 'react-redux'
import {NavBar} from 'antd-mobile'

import LaobanInfo from '../laoban-info/laoban-info'
import DashenInfo from '../dashen-info/dashen-info'
import Dashen from '../dashen/dashen'
import Laoban from '../laoban/laoban'
import Message from '../message/message'
import Personal from '../personal/personal'
import NotFound from '../../components/not-found/not-found'
import NavFooter from '../../components/nav-footer/nav-footer'
import Chat from '../chat/chat'

import {getUser} from '../../redux/actions'
import {getRedirectPath} from '../../utils'

class Main extends React.Component {
// 组件类和组件对象
  // 给组件对象添加属性
  navList = [
    {
      path: '/laoban', // 路由路径
      component: Laoban,
      title: '大神列表',
      icon: 'dashen',
      text: '大神',
    },
    {
      path: '/dashen', // 路由路径
      component: Dashen,
      title: '老板列表',
      icon: 'laoban',
      text: '老板',
    },
    {
      path: '/message', // 路由路径
      component: Message,
      title: '消息列表',
```

```
    icon: 'message',
    text: '消息',
  },
  {
    path: '/personal', // 路由路径
    component: Personal,
    title: '用户中心',
    icon: 'personal',
    text: '个人',
  }
]

componentDidMount() {
  // cookie 中有 userid
  // redux 中的 user 是空对象
  const userid = Cookies.get('userid')
  const {user} = this.props
  if (userid && !user._id) {
    this.props.getUser()  // 获取 user 并保存到 redux 中
  }
}


render() {
  // 得到当前请求的 path
  const pathname = this.props.location.pathname
  // 判断用户是否已登陆(过)(cookie 中 userid 是否有值)
  const userid = Cookies.get('userid')
  if (!userid) { // 如果没值, 自动跳转到登陆界面
    return <Redirect to='/login'/>
  }
  // cookie 中有 userid
  // redux 中的 user 是否有数据
  const {user} = this.props
  if (!user._id) {
    return null  // 不做任何显示
  } else {
    // 请求根路径时, 自动 跳转到对应的用户主界面
    if (pathname === '/') {
      const path = getRedirectPath(user.type, user.header)
      return <Redirect to={path}/>
    }
```

```
      // 指定哪个nav 应该被隐藏
      if (user.type === 'laoban') {
        this.navList[1].hide = true
      } else {
        this.navList[0].hide = true
      }
    }

    // 得到当前的nav
    const currentNav = this.navList.find(nav => nav.path === pathname)

    // 得到props 中的unReadCount
    const unReadCount = this.props.unReadCount

    return (
      <div>
        {currentNav ? <NavBar className='stick-top'>{currentNav.title}</NavBar> : null}
        <Switch>
          <Route path='/laobaninfo' component={LaobanInfo}></Route>
          <Route path='/dasheninfo' component={DashenInfo}></Route>

          <Route path='/dashen' component={Dashen}></Route>
          <Route path='/laoban' component={Laoban}></Route>
          <Route path='/message' component={Message}></Route>
          <Route path='/personal' component={Personal}></Route>
          <Route path='/chat/:userid' component={Chat}></Route>
          <Route component={NotFound}></Route>
        </Switch>

        {currentNav ? <NavFooter className='stick-top' navList={this.navList}
unReadCount={unReadCount}/> : null}
      </div>
    )
  }
}

export default connect(
  state => ({
    user: state.user,
    unReadCount: state.chat.unReadCount // 未读消息数量
  }),
```

```
  {getUser}
)(Main)
```

2.16.9. 显示总未读消息数量: components/nav-footer/nav-footer.jsx

```
/*
底部导航的UI 组件
 */

import React from 'react'
import PropTypes from 'prop-types'
import {TabBar} from 'antd-mobile'
import {withRouter} from 'react-router-dom'

const Item = TabBar.Item

class NavFooter extends React.Component {

  static propTypes = {
    navList: PropTypes.array.isRequired,
    unReadCount: PropTypes.number.isRequired
  }

  render() {
    // nav.hide = true/false hide 代表当前项应该被隐藏
    const navList = this.props.navList.filter(nav => !nav.hide) // 回调函数返回值为true,
当前元素就会留下, 否则不留
    // 当前请求的路径
    const {pathname} = this.props.location
    return (
      <TabBar>
        {
          navList.map((nav, index) => (
            <Item key={nav.path}
                  badge={nav.path === '/message' ? this.props.unReadCount : 0}
                  title={nav.text}
                  icon={{uri: require(`./imgs/${nav.icon}.png`)}}
                  selectedIcon={{uri: require(`./imgs/${nav.icon}-selected.png`)}}
                  selected={pathname===nav.path}
```

```
                onPress={() => {
                  this.props.history.replace(nav.path)
                }}
              />
            ))
          }
        </TabBar>
      )
    }
}

export default withRouter(NavFooter) // 让非路由组件可以访问到路由组件的API
```

2.16.10. 查看聊天后，更新未读数量

chat.jsx

```
componentDidMount() {
  // 初始显示列表
  window.scrollTo(0, document.body.scrollHeight)
  this.props.readMsg(this.props.match.params.userid)
}

componentWillUnmount() {
  this.props.readMsg(this.props.match.params.userid)
}
```

2.16.11. 添加聊天的页面的动画效果

1) 下载依赖包

```
npm install --save rc-queue-anim
```

2) 在聊天组件使用: containers/chat/chat.jsx

```
import QueueAnim from 'rc-queue-anim'

{/*alpha left right top bottom scale scaleBig scaleX scaleY*/}
<QueueAnim type='scale' delay={100}>
  {chatMsgs.map(msg => {
    const avatar = require(`../../assets/imgs/${users[msg.from].avatar}.png`)
    if (msg.from === userid) {
      return (
        <Item
          key={msg._id}
          thumb={avatar}
        >{msg.content}</Item>
      )
    } else {
      return (
        <Item
          key={msg._id}
          extra={<img src={avatar}/>}
          className='chat-me'
        >{msg.content}</Item>
      )
    }
  })}
</QueueAnim>
```