React全栈项目: 硅谷直聘

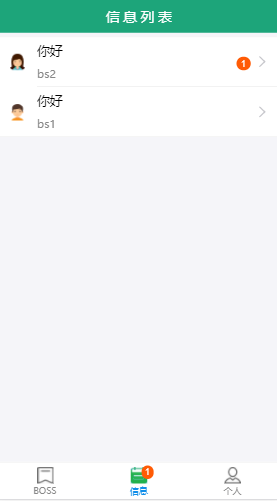
# 第1章: 准备

## 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, Boss/牛人列表, 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

## 项目功能界面

牛人主界面 牛人消息列表 牛人个人中心

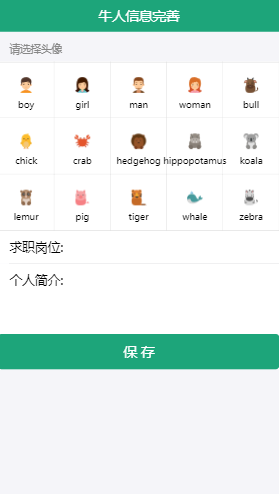
Boss主界面 Boss消息列表 Boss个人中心

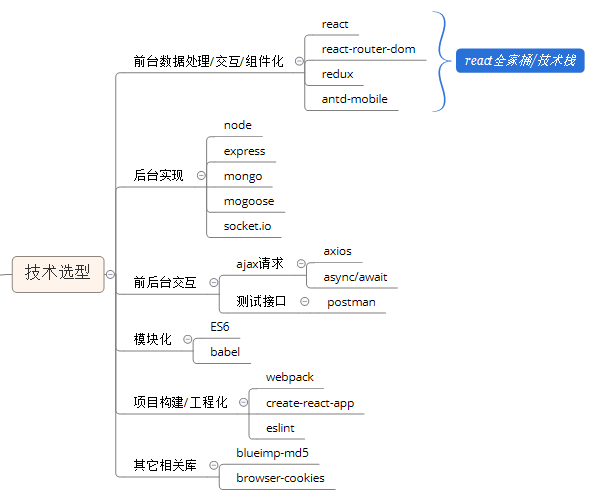
聊天界面 登陆界面 注册界面

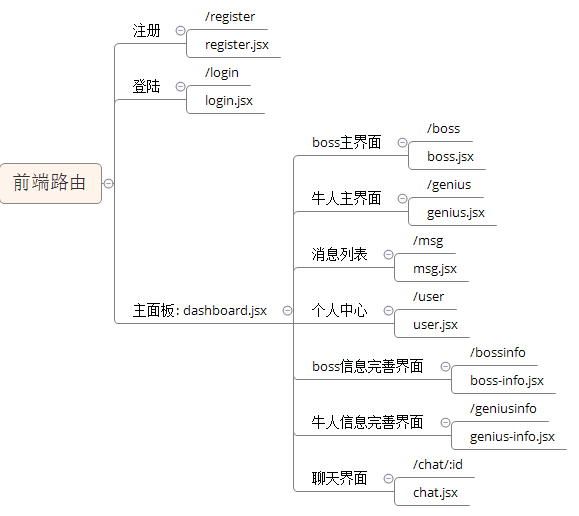
Boss信息完善界面 牛人信息完善界面

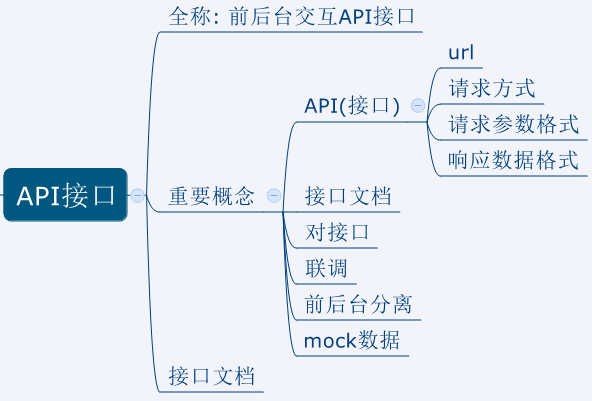
## 技术选型



## 前端路由



## API接口



## 你能从此项目中学到什么?

### 流程及开发方法

1. 熟悉一个项目的**开发流程**
2. 学会**组件化、模块化、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
5. 学会**ES6+eslint**的开发方式

### 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**browser-cookies**操作浏览器端cookie数据

## 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 |

## 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. 在线文档:<https://github.com/facebookincubator/create-react-app>
3. 操作:

npm install -g create-react-app : 全局下载工具

create-react-app gboss :下载模板项目

cd gboss

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](http://localhost:8080)

编码, 自动编译打包(live-reload), 查看效果

1. 打包发布

npm run build

npm install -g serve

serve build

访问: [http://localhost:5000](http://localhost:9000)

## 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

* 下载依赖模块

|  |
| --- |
| **npm install --save-dev babel-plugin-import react-app-rewired** |

* 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; } |

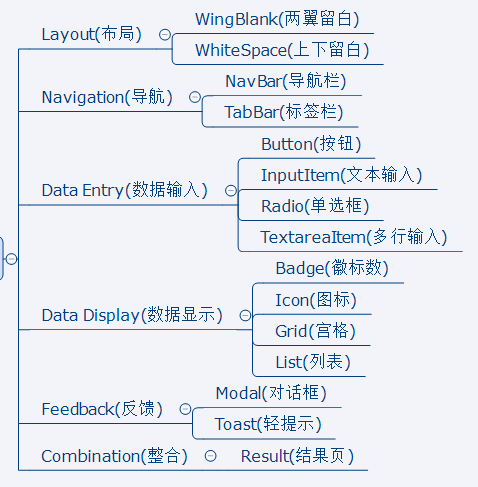
* 修改配置: package.json

|  |
| --- |
| **"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

1. 下载依赖模块

|  |
| --- |
| **npm install --save-dev less@2.7.3 less-loader** |

1. 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/dashboard/dashboard.jsx

|  |
| --- |
| */\* 应用面版的路由组件  \*/* **import** React, {Component} **from 'react'  export default class** Dashboard **extends** Component {  render() {  **return** (  <**div**>Dashboard</**div**>  )  } } |

### 2.5.5. 映射路由: index.js

|  |
| --- |
| */\* 入口JS  \*/*  **import** React **from 'react' import** ReactDOM **from 'react-dom' import** {**BrowserRouter**, **Switch**, **Route**} **from 'react-router-dom'  import** Login **from './containers/login/login' import** Register **from './containers/register/register' import** Dashboard **from './containers/dashboard/dashboard'** ReactDOM.render((  <**BrowserRouter**>  <**Switch**>  <**Route path='/login' component=**{Login}/>  <**Route path='/register' component=**{Register}/>  <**Route component=**{Dashboard}/>  </**Switch**>  </**BrowserRouter**> ), **document**.getElementById(**'root'**)) |

## 2.6. 引入redux

### 2.6.1. 下载相关依赖包

|  |
| --- |
| **npm install --save redux react-redux redux-thunk**  **npm install --save-dev redux-devtools-extension** |

### 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** {**BrowserRouter**, **Switch**, **Route**} **from 'react-router-dom'  import** store **from './redux/store' import** Login **from './containers/login/login' import** Register **from './containers/register/register' import** Dashboard **from './containers/dashboard/dashboard'** ReactDOM.render((  <**Provider store=**{store}>  <**BrowserRouter**>  <**Switch**>  <**Route path='/login' component=**{Login}/>  <**Route path='/register' component=**{Register}/>  <**Route component=**{Dashboard}/>  </**Switch**>  </**BrowserRouter**>  </**Provider**> ), **document**.getElementById(**'root'**)) |

## 2.7. 注册/登陆界面



### 2.7.1. Logo组件

1. components/logo/logo.jsx

|  |
| --- |
| */\* 显示logo图片的UI组件  \*/*  **import** React **from 'react' import** logoImg **from './job.png' import './logo.css'  class** Logo **extends** React.Component {   render() {  **return** (  <**div className="logo-container"**>  <**img src=**{logoImg} **alt="logo"**/>  </**div**>  )  } }  **export default** Logo |

1. components/logo/logo.less

|  |
| --- |
| .**logo-container** {  **margin-top**: 10**px**;  **text-align**: **center**;  **margin-bottom**: 10**px**; } |

### 2.7.2. 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** = {  **name**: **''**,  **pwd**: **''**,  }   *// 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(***JSON***.stringify(**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*(**'name'**, val)}  >  用户名:  </**InputItem**>  <**WhiteSpace**/>  <**InputItem  type='password'  placeholder='输入密码'  onChange=**{val => **this**.*handleChange*(**'pwd'**, 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**>  )  } } |

### 2.7.3. 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'  const** RadioItem = Radio.*RadioItem* **export default class** Register **extends** Component {  **state** = {  **name**: **''**,  **pwd**: **''**,  **pwd2**: **''**,  **type**: **'boss'** }   *// 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*(**'name'**, val)}  >  用户名:  </**InputItem**>  <**WhiteSpace**/>  <**InputItem  type='password'  placeholder='输入密码'  onChange=**{val => **this**.*handleChange*(**'pwd'**, val)}  >  密 码:  </**InputItem**>  <**WhiteSpace**/>  <**InputItem  type='password'  placeholder='输入确认密码'  onChange=**{val => **this**.*handleChange*(**'pwd2'**, val)}  >  确认密码:  </**InputItem**>  <**WhiteSpace**/>  <**RadioItem  checked=**{**type** === **'boss'**}  **onChange=**{() => **this**.*handleChange*(**'type'**, **'boss'**)}  >  BOSS  </**RadioItem**>  <**RadioItem  checked=**{**type** === **'genius'**}  **onChange=**{() => **this**.*handleChange*(**'type'**, **'genius'**)}  >  牛人  </**RadioItem**>  <**WhiteSpace**/>  <**Button type='primary' onClick=**{**this**.*register*}>注 册</**Button**>  <**WhiteSpace**/>  <**Button onClick=**{**this**.*toLogin*}>已经有账号</**Button**>  </**List**>  </**WingBlank**>  </**div**>  )  } } |

## 2.8. 搭建后台(基本)

### 2.8.1. 下载相关依赖包

|  |
| --- |
| **npm install --save express**  **npm install -g nodemon** |

### 2.8.2. 实现最基本的服务器端

1. server/server.js

|  |
| --- |
| */\* 后台启动服务器的入口模块 1. 引入express 2. 生成应用对象(执行express函数) 3. 注册根路由(使用app的use()) 4. 启动服务器(使用app监听指定端口)  \*/  // 1. 引入express* **const** express = require(**'express'**)  *// 2. 生成应用对象(执行express函数)* **const** app = express()  *// 3. 注册根路由(使用app的use())* app.use(**'/'**, **function** (req, res) {  res.send(**'hello server!'**) })  *// 4. 启动服务器(使用app监听指定端口)* app.listen(**'8888'**, () => {  **console**.log(**'start server at port 8888'**) }) |

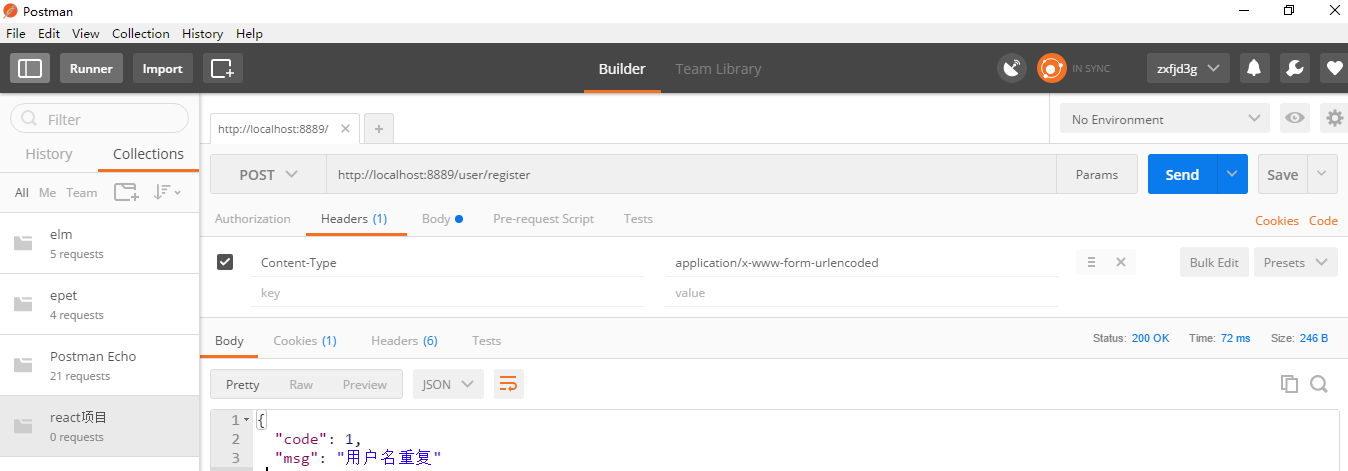
1. 配置启动服务器命令: package.json

|  |
| --- |
| **"server"**: **"nodemon server/server.js"** |

1. 运行访问

|  |
| --- |
| **npm run server**  **http://localhost:8888** |

### 2.8.2. 使用postman测试接口



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

## 2.9. 使用mongoose操作数据库

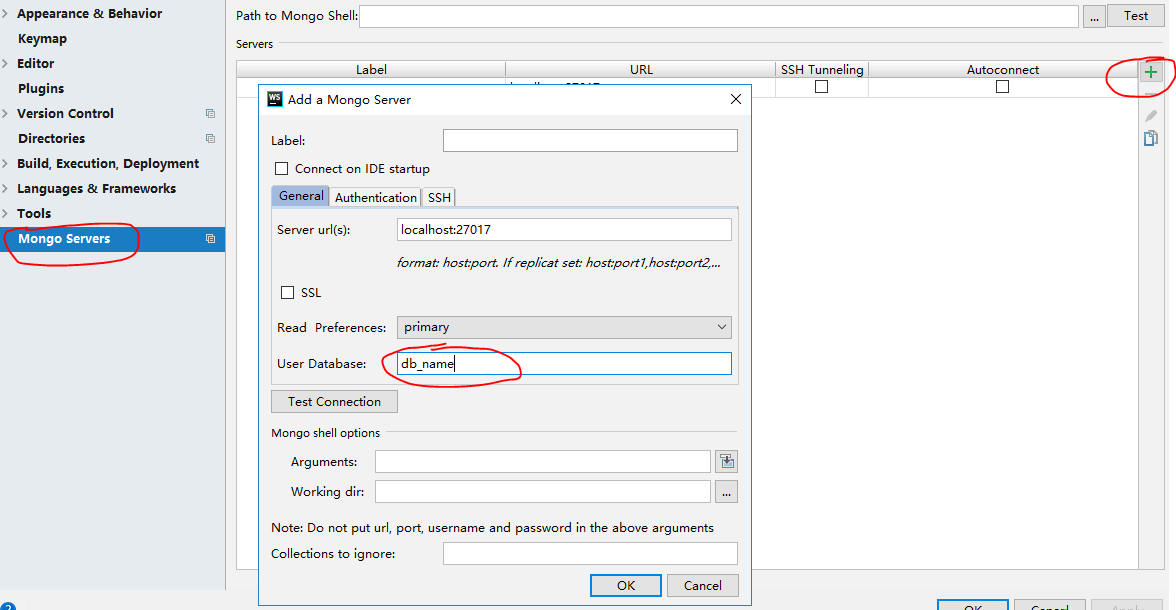
### 2.9.1. 下载依赖包

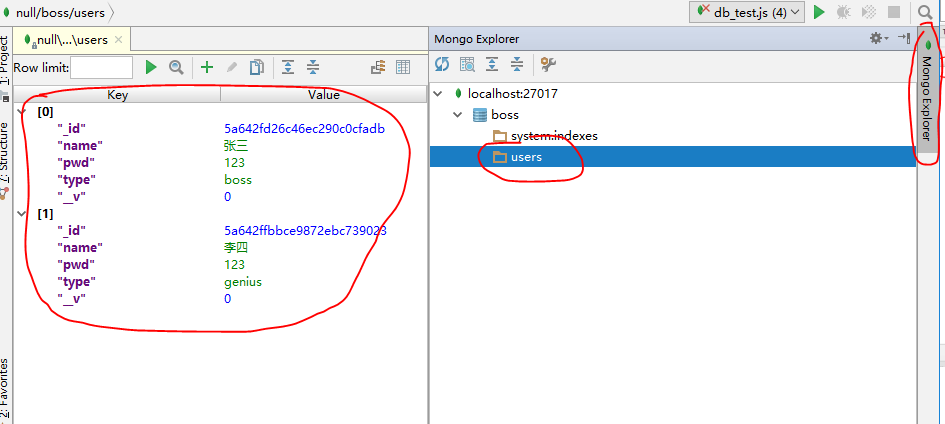
|  |
| --- |
| **npm install --save mongoose** |

### 2.9.2. server/db\_test.js

|  |
| --- |
| */\* 测试使用mongoose操作mongodb数据库的模块 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. 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  \*/  /\*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/gboss\_test'**) *// 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(**'连接数据库成功！'**) })  */\*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({  *// 用户名* **'name'**: {**type**: String, **'required'**: **true**},  *// 密码* **'pwd'**: {**type**: String, **'required'**: **true**},  *// 类型* **'type'**: {**'type'**: String, **'required'**: **true**},  *// 头像* **'avatar'**: {**'type'**: String},  *// 个人简介或者职位简介* **'desc'**: {**'type'**: String},  *// 职位名* **'title'**: {**'type'**: String},  *// 如果你是boss 还有两个字段  // 公司名称* **'company'**: {**'type'**: String},  *// 工资* **'money'**: {**'type'**: String} }) *// 2.2. 定义Model(与集合对应, 可以操作集合) //const UserModel = mongoose.model('user', userSchema) // 集合的名称为users* mongoose.model(**'user'**, userSchema) **const** UserModel = mongoose.model(**'user'**)  */\*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* **function** *testSave*() {  **const** userModel = **new** UserModel({  **name**: **'Jack'**,  **pwd**: **'123'**,  **type**: **'genius'**,  **avatar**: **'boy'** })  userModel.save(userModel, **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*() {  UserModel.find(**function** (err, users) { *// 如果没有匹配的, 返回[]* **console**.log(**'find()'**, err, users)  })   UserModel.findOne({**\_id**: **'5a69d23dcbe9821448e5f799'**}, **function** (err, user) { *// 如果没有匹配的, 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**: **'5a69d23dcbe9821448e5f799'**}, {**name**: **'BB'**}, **function** (err, user) {  **console**.log(**'findByIdAndUpdate()'**, err, user)  }) }  *// testUpdate()  // 3.4. 通过Model的remove()删除匹配的数据* **function** *testRemove*() {  UserModel.remove({**\_id**: **'5a69d23dcbe9821448e5f799'**}, **function** (err, result) {  **console**.log(**'remove()'**, err, result) *// { ok: 1, n: 1 }* }) } *// testRemove()* |

### 2.9.3. 安装webstorm的mongo插件: Mongo Plugin





## 2.10. 注册/登陆后台处理

### 2.10.1. 下载相关依赖包

|  |
| --- |
| **npm install --save body-parser cookie-parser**  **npm install --save blueimp-md5** |

### 2.10.2. 数据库操作模块: server/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. 向外暴露获取Model的方法  \*/ // 1. 连接数据库* **const** mongoose = require(**'mongoose'**) mongoose.connect(**'mongodb://localhost:27017/bossz'**) **const** conn = mongoose.**connection** conn.**on**(**'connected'**, **function** () {  **console**.log(**'数据库连接成功!'**) })  *// 2. 得到对应特定集合的Model: UserModel* **const** userSchema = mongoose.Schema({  *// 用户名* **'name'**: {**type**: String, **required**: **true**},  *// 密码* **'pwd'**: {**type**: String, **required**: **true**},  *// 类型* **'type'**: {**'type'**: String, **required**: **true**},  *// 头像* **'avatar'**: {**'type'**: String},  *// 个人简介或者职位简介* **'desc'**: {**'type'**: String},  *// 职位名* **'title'**: {**'type'**: String},  *// 如果你是boss 还有两个字段  // 公司名称* **'company'**: {**'type'**: String},  *// 工资* **'money'**: {**'type'**: String} }) mongoose.model(**'user'**, userSchema)  *// 3. 向外暴露* **module**.**exports** = {  getModel(name) {  **return** mongoose.model(name)  } } |

### 2.10.3. 应用路由器模块: server/appRouter.js

|  |
| --- |
| */\* 后台应用的路由器模块 1. 引入express 2. 得到路由器 3. 注册n个路由 4. 向外暴露路由器 5. 通过 app使用上路由器  \*/ // 1. 引入express* **const** express = require(**'express'**) **const** md5 = require(**'blueimp-md5'**) **const models** = require(**'./models'**) **const** UserModel = **models**.getModel(**'user'**) *// 2. 得到路由器* **const** router = express.Router() **const** \_filter = {**'pwd'**: 0, **'\_\_v'**: 0} *// 查询时过滤掉  // 3. 注册n个路由  // 用户注册的路由 /\* 路由回调函数的3步 1. 获取请求参数 2. 处理(操作数据库数据) 3. 返回响应  \*/* router.post(**'/register'**, **function** (req, res) {  *// 1. 获取请求参数* **const** {name, pwd, type} = req.**body** *// 包含所有请求参数的对象   // 2. 处理(操作数据库数据)  // 2.1. 根据name查询是否已经存在,* UserModel.findOne({name}, **function** (err, user) {  *// 3.1. 如果已经存在, 返回一个错误的提示* **if**(user) {  **return** res.send({**code**: 1, **msg**: **'用户名已存在!'**}) *// code: 数据的标识 1: 错误 0: 正确* }  *// 2.2. 如果不存在, 保存到数据库,* **const** userModel = **new** UserModel({name, **pwd**: md5(pwd), type})  userModel.save(**function** (err, user) {  *// 向浏览器端返回cookie(key=value)* res.cookie(**'userid'**, user.**\_id**)  *// 3.2. 返回数据(新的user)* res.send({**code**: 0, **data**: {**\_id**: user.**\_id**, name, pwd, type}})  })  }) })  *// 用户登陆的路由* router.post(**'/login'**, **function** (req, res) {  *// 1. 获取请求参数* **const** {name, pwd} = req.**body** *// 2. 根据name和pwd查询对应的user* UserModel.findOne({name, **pwd**: md5(pwd)}, \_filter, **function** (err, user) {  *// 3.1. 存在, 返回user数据* **if**(user) {  *// 向浏览器端返回cookie(key=value)* res.cookie(**'userid'**, user.**\_id**)  res.json({**code**: 0, **data**: user})  } **else** {  *// 3.2. 不存在, 返回错误信息* res.json({**code**: 1, **msg**: **'用户名或密码错误!'**})  }  }) })  *// 4. 向外暴露路由器* **module**.**exports** = router |

### 2.10.4. 服务器模块: server/server.js

|  |
| --- |
| */\* 使用express启动服务器 1. 引入express 2. 生成应用对象(执行express函数) 3. 注册根路由(使用app的use()) 4. 启动服务器(使用app监听指定端口)  \*/ // 1. 引入express* **const** express = require(**'express'**) **const** bodyParser = require(**'body-parser'**) *// 解析请求体* **const** cookieParser = require(**'cookie-parser'**) *// 引入应用的路由器* **const** appRouter = require(**'./app\_router'**)  *// 2. 生成应用对象(执行express函数)* **const** app = express()  *// 3. 注册根路由(使用app的use()) /\*app.use('/', function (req, res) {  res.send('hello server!2222') })\*/* app.use(cookieParser()) *// 解析cookie数据* app.use(bodyParser.**json**()) *// 解析请求体(ajax请求: json数据格式)* app.use(bodyParser.**urlencoded**({ **extended**: **false** })) *// 解析请求体(表单数据) // 注册路由器* app.use(**'/api'**, appRouter) *// 登陆: /api/login  // 4. 启动服务器(使用app监听指定端口)* app.listen(4000, **function** () {  **console**.log(**'服务器启动成功 port: 4000'**) }) |

## 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  \*/* **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.substr(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* }  } |

1. api/index.js

|  |
| --- |
| **import** *ajax* **from './ajax'** *// 请求注册* **export const** *reqRegister* = (user) => *ajax*(**'/api/register'**, user, **'POST'**) *// 请求登陆* **export const** *reqLogin* = (user) => *ajax*(**'/api/login'**, user, **'POST'**) |

1. 配置ajax请求的代理: package.json

|  |
| --- |
| **"proxy"**: **"http://localhost:8888"** |

### 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'** |

1. 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*({name, pwd, pwd2, type}) {  **if** (!name || !pwd || !type) {  **return** *errorMsg*(**'用户名密码必须输入'**)  }  **if** (pwd !== pwd2) {  **return** *errorMsg*(**'密码和确认密码不同'**)  }  **return async** dispatch => {  **const** result = **await** *reqRegister*({name, pwd, type})  **if** (result.**code** === 0) {  dispatch(*authSuccess*(result.**data**))  } **else** {  dispatch(*errorMsg*(result.**msg**))  }  } }  */\* 异步登陆  \*/* **export const** *login* = ({name, pwd}) => {  **if** (!name || !pwd) {  **return** *errorMsg*(**'用户密码必须输入'**)  }  **return async** dispatch => {  **const** result = **await** *reqLogin*({name, pwd})  **if** (result.**code** === 0) {  dispatch(*authSuccess*(result.**data**))  } **else** {  dispatch(*errorMsg*(result.**msg**))  }  } } |

1. 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 = {  **name**: **''**, *// 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  } }  **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** {List, WingBlank, InputItem, Radio, WhiteSpace, Button} **from 'antd-mobile' import** {connect} **from 'react-redux' import** {**Redirect**} **from 'react-router-dom'  import** Logo **from '../../components/logo/logo' import** {*regisger*} **from '../../redux/actions'  const** RadioItem = Radio.*RadioItem* **class** Register **extends** Component {   **state** = {  **name**: **''**,  **pwd**: **''**,  **pwd2**: **''**,  **type**: **'genius'** }   *toLogin* = () => **this**.**props**.**history**.replace(**'/login'**)   *handleChange* = (key, val) => **this**.setState({[key]: val})   *handleRegister* = () => **this**.**props**.**regisger**(**this**.**state**)   render() {   **const** {redirectTo, msg} = **this**.**props  if** (redirectTo) {  **return** <**Redirect to=**{redirectTo}/>  }   **return** (  <**div**>  <**Logo**></**Logo**>  <**WingBlank**>  {msg ? <**p className='error-msg'**>{msg}</**p**> : **null**}  <**List**>  <**InputItem  onChange=**{val => **this**.*handleChange*(**'name'**, val)}  >用户名</**InputItem**>  <**WhiteSpace**/>  <**InputItem  type='password'  onChange=**{val => **this**.*handleChange*(**'pwd'**, val)}  >密码</**InputItem**>  <**WhiteSpace**/>  <**InputItem  type='password'  onChange=**{val => **this**.*handleChange*(**'pwd2'**, val)}  >确认密码</**InputItem**>  <**WhiteSpace**/>  <**RadioItem  checked=**{**this**.**state**.**type** === **'genius'**}  **onChange=**{() => **this**.*handleChange*(**'type'**, **'genius'**)}  >  牛人  </**RadioItem**>  <**RadioItem  checked=**{**this**.**state**.**type** === **'boss'**}  **onChange=**{() => **this**.*handleChange*(**'type'**, **'boss'**)}  >  BOSS  </**RadioItem**>  <**WhiteSpace**/>  <**Button type='primary' onClick=**{**this**.*handleRegister*}>注册 </**Button**>  <**Button onClick=**{**this**.*toLogin*}>已经有账号</**Button**>  </**List**>  </**WingBlank**>   </**div**>  )  } }  **export default** connect(  state => state.**user**,  {*regisger*} )(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** {List, InputItem, WingBlank, 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** = {  **name**: **''**,  **pwd**: **''** }   *toRegister* = () => **this**.**props**.**history**.push(**'/register'**)   *handleChange* = (key, val) => **this**.setState({[key]: val})   *handleLogin* = () => **this**.**props**.**login**(**this**.**state**)   render() {   **const** {redirectTo, msg} = **this**.**props  if** (redirectTo) {  **return** <**Redirect to=**{redirectTo}/>  }   **return** (  <**div**>  <**Logo**></**Logo**>  <**WingBlank**>  {msg ? <**p className='error-msg'**>{msg}</**p**> : **null**}  <**List**>  <**InputItem  onChange=**{v => **this**.*handleChange*(**'name'**, v)}  >用户</**InputItem**>  <**WhiteSpace**/>  <**InputItem  onChange=**{v => **this**.*handleChange*(**'pwd'**, v)}  **type='password'** >密码</**InputItem**>  </**List**>  <**WhiteSpace**/>  <**Button type='primary' onClick=**{**this**.*handleLogin*}>登录</**Button**>  <**WhiteSpace**/>  <**Button onClick=**{**this**.*toRegister*}>还没有账号</**Button**>  </**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**: 18**px**; }  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 browser-cookies** |

### 2.12.2. 注册/登陆成功的路由跳转

1. 工具模块: utils/index.js

|  |
| --- |
| */\* 包含一些工具函数的模块  \*/  /\*\*  \* 得到重定向的路径  \*/* **export function** *getRedirectPath*({type, avatar}) {  *// 根据用户信息 返回跳转地址  // user.type /boss /genius  // user.avatar /bossinfo /geniusinfo* **let** url = (type === **'boss'**) ? **'/boss'** : **'/genius'  if** (!avatar) {  url += **'info'** }  **return** url } |

1. reducers中使用工具: redux/reducers.js

|  |
| --- |
| **import** {*getRedirectPath*} **from '../utils'**  **case** AUTH\_SUCCESS: *// 认证成功* **const** user = action.**data  const** {type, avatar} = user  **return** {...user, **redirectTo**: *getRedirectPath*({type, avatar})} |

### 2.12.3. 后台路由: server/appRouter.js

|  |
| --- |
| *// 更新用户信息的路由* router.post(**'/update'**, **function** (req, res) {  *// 检查用户是否登陆, 如果没有, 返回错误提示信息* **const** userid = req.**cookies**.**userid** *//取出请求中cookie包含的userid* **if**(!userid) {  **return** res.send({**code**: 1, **msg**: **'请先登陆'**})  }  *// 更新对应的user* UserModel.findByIdAndUpdate({**\_id**: userid}, req.**body**, **function** (err, user) {*// user不包含新添加的数据* **if**(!user) { *// 更新失败, 需要重新登陆  // 告诉浏览器清除保存的userid cookie* res.clearCookie(**'userid'**)   res.send({**code**: 1, **msg**: **'请先登陆'**})  } **else** { *// 成功了* **const** {\_id, name, type} = user  *// ...在node端不能使用, 需要使用assign来合并对象* user = Object.assign({}, req.**body**, {\_id, name, type})  res.send({**code**: 0, **data**: user})  }  }) }) |

### 2.12.4. 更新的ajax请求: api/index.js

|  |
| --- |
| *// 更新用户信息* **export const** *reqUpdateUser* = (user) => *ajax*(**'/api/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'** |

1. redux/actions.js

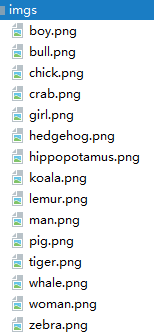
|  |
| --- |
| **import** {  RECEIVE\_USER,  RESET\_USER } **from './action-types'**  **import** {  *reqUpdateUser* } **from '../api'**  *// 同步接收用户* **const** *receiveUser* = (user) => ({**type**: RECEIVE\_USER, **data**: user}) *// 同步重置用户* **export const** *resetUser* = (msg) => ({**type**: RESET\_USER, **data**: msg}) */\* 异步更新用户  \*/* **export const** *updateUser* = (user) => {  **return async** dispatch => {  *// 发送异步ajax请求* **const** result = **await** *reqUpdateUser*(user) **if** (result.**code** === 0) { *// 更新成功* dispatch(*receiveUser*(result.**data**))  } **else** { *// 失败* dispatch(*resetUser*(result.**msg**))  }  } } |

1. 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



1. 组件: components/avatar-selector/avatar-selector.jsx

|  |
| --- |
| */\* 选择头像的UI组件  \*/*  **import** React **from 'react' import** {Grid, List} **from 'antd-mobile' import** PropTypes **from 'prop-types'  class** AvatarSelector **extends** React.Component {   **static** *propTypes* = {  **selectAvatar**: PropTypes.*func*.isRequired  }   **state** = {  **icon**: **null**,  **text**: **''** }   componentWillMount() {  **this**.**avatarList** = **'boy,girl,man,woman,bull,chick,crab,hedgehog,hippopotamus,koala,lemur,pig,tiger,whale,zebra'** .split(**','**)  .map(text => ({  **icon**: require(**`../../assets/imgs/**${text}**.png`**),  text  }))  }   render() {   **const** gridHeader = **this**.**state**.**icon** ? (  <**div**>  <**span**>已选择头像: </**span**>  <**img style=**{{**width**: 20}} **src=**{**this**.**state**.**icon**} **alt="avatar"**/>  </**div**>  )  : **'请选择头像'  return** (  <**div**>  <**List  renderHeader=**{() => gridHeader}  >  <**Grid  data=**{**this**.**avatarList**}  **columnNum=**{5}  **onClick=**{item => {  **this**.setState(item)  **this**.**props**.**selectAvatar**(item.**text**)  }}  />  </**List**>  </**div**>  )  } }  **export default** AvatarSelector |

### 2.12.7. boss信息完善组件: containers/boss-info/boss-info.jsx

|  |
| --- |
| **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** AvatarSelector **from '../../components/avatar-selector/avatar-selector' import** {*updateUser*} **from '../../redux/actions'  class** BossInfo **extends** Component {   **state** = {  *// 头像* **avatar**: **''**,  *// 个人简介或者职位简介* **desc**: **''**,  *// 职位名* **title**: **''**,  *// 公司名称* **company**: **''**,  *// 工资* **money**: **''** }   *handleChange* = (name, val) => {  *//更新状态* **this**.setState({[name]: val})  }   *// 保存  save* = () => {  *// console.log(this.state)* **this**.**props**.**updateUser**(**this**.**state**)  }   *// 设置头像  setAvatar* = (avatar) => {  **this**.setState({avatar})  }   render () {  *// 如果用户信息已经完成, 自动跳转到boss的应用面板: /boss* **const** {avatar} = **this**.**props**.**user  if**(avatar) {  *// this.props.history.replace('/boss')  // return null* **return** <**Redirect to='/boss'**></**Redirect**>  }   **return** (  <**div**>  <**NavBar**>BOSS信息完善</**NavBar**>  <**AvatarSelector setAvatar=**{**this**.*setAvatar*}/>  <**InputItem onChange=**{val => {**this**.*handleChange*(**'title'**, val)}}>招聘职位:</**InputItem**>  <**InputItem onChange=**{val => {**this**.*handleChange*(**'company'**, val)}}>公司名称:</**InputItem**>  <**InputItem onChange=**{val => {**this**.*handleChange*(**'money'**, val)}}>职位薪资:</**InputItem**>  <**TextareaItem  title="职位要求:"  rows=**{3}  **onChange=**{val => {**this**.*handleChange*(**'desc'**, val)}}  />  <**Button type="primary" onClick=**{**this**.*save*}>保存</**Button**>  </**div**>  )  } }  **export default** connect(  state => ({**user**: state.**user**}),  {*updateUser*} )(BossInfo) |

### 2.12.8. genius信息完善组件: containers/genius-info/genius-info.jsx

|  |
| --- |
| */\* 牛人信息完善路由组件  \*/* **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** AvatarSelector **from '../../components/avatar-selector/avatar-selector' import** {*updateUser*} **from '../../redux/actions'  class** GeniusInfo **extends** Component {   **state** = {  *// 头像* **avatar**: **''**,  *// 个人简介或者职位简介* **desc**: **''**,  *// 职位名* **title**: **''**,  }   *handleChange* = (name, val) => {  *//更新状态* **this**.setState({[name]: val})  }   *// 保存  save* = () => {  *// console.log(this.state)* **this**.**props**.**updateUser**(**this**.**state**)  }   *// 设置头像  setAvatar* = (avatar) => {  **this**.setState({avatar})  }   render () {  *// 如果用户信息已经完成, 自动跳转到genius的应用面板: /genius* **const** {avatar} = **this**.**props**.**user  if**(avatar) {  **return** <**Redirect to='/genius'**></**Redirect**>  }   **return** (  <**div**>  <**NavBar**>牛人信息完善</**NavBar**>  <**AvatarSelector setAvatar=**{**this**.*setAvatar*}/>  <**InputItem onChange=**{val => {**this**.*handleChange*(**'title'**, val)}}>求职岗位:</**InputItem**>  <**TextareaItem  title="个人简介:"  rows=**{3}  **onChange=**{val => {**this**.*handleChange*(**'desc'**, val)}}  />  <**Button type="primary" onClick=**{**this**.*save*}>保存</**Button**>  </**div**>  )  } }  **export default** connect(  *// state => state.user, // props* state => ({**user**: state.**user**}),  {*updateUser*} )(GeniusInfo) |

### 2.12.9. dashboard组件: containers/dashboard/dashboard.jsx

|  |
| --- |
| */\* 应用面版的路由组件  \*/* **import** React, {Component} **from 'react' import** {**Switch**, **Route**} **from 'react-router-dom'**  **import** cookies **from 'browser-cookies'**  **import** BossInfo **from '../boss-info/boss-info' import** GeniusInfo **from '../genius-info/genius-info'  export default class** Dashboard **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='/bossinfo' component=**{BossInfo}/>  <**Route path='/geniusinfo' component=**{GeniusInfo}/>  </**Switch**>  </**div**>  )  } } |

## 2.13. 搭建整体界面

### 2.13.2. 后台路由: server/appRoute.js

|  |
| --- |
| */\* 根据cookie查询对应的user  \*/* router.get(**'/user'**, **function** (req, res) {  **const** userid = req.**cookies**.userid  **if**(!userid) {  **return** res.json({**code**: 1, **msg**: **'请先登陆'**})  }  UserModel.findOne({**\_id**: userid}, \_filter, **function** (err, user) {  **if**(!user) {  res.clearCookie(**'userid'**)  res.json({**code**: 1, **msg**: **'请先登陆'**})  } **else** {  res.json({**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*(**'/api/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. 面板的子路由

1. containers/boss/boss.jsx

|  |
| --- |
| */\* boss的主路由组件  \*/*  **import** React, {Component} **from 'react'** **export default class** Boss **extends** Component {  render() {  **return** (  <**div**>牛人列表</**div**>  )  } } |

1. containers/genius/genius.jsx

|  |
| --- |
| */\* 牛人的主路由组件  \*/*  **import** React, {Component} **from 'react'** **export default class** Genius **extends** Component {  render() {  **return** (  <**div**>Boss列表</**div**>  )  } } |

1. containers/msg/msg.jsx

|  |
| --- |
| */\* 对话消息列表组件  \*/*  **import** React, {Component} **from 'react'** **export default class** Msg **extends** Component {  render() {  **return** (  <**div**>msg列表</**div**>  )  } } |

1. containers/user/user.jsx

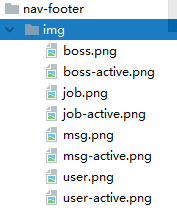
|  |
| --- |
| */\* 用户个人中心路由组件  \*/*  **import** React, {Component} **from 'react'** **export default class** User **extends** Component {  render() {  **return** (  <**div**>个人中心</**div**>  )  } } |

1. 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



1. 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() {   **const** navList = **this**.**props**.**navList**.filter(v => !v.**hide**)  **const** {pathname} = **this**.**props**.**location  return** (  <**TabBar**>  {navList.map(v => (  <**Item  key=**{v.**path**}  **title=**{v.**text**}  **icon=**{{**uri**: require(**`./imgs/**${v.**icon**}**.png`**)}}  **selectedIcon=**{{**uri**: require(**`./imgs/**${v.**icon**}**-active.png`**)}}  **selected=**{pathname === v.**path**}  **onPress=**{() => {  **this**.**props**.**history**.replace(v.**path**)  }}  >  </**Item**>  ))}  </**TabBar**>  )  } }  **export default withRouter**(NavFooter) |

### 2.13.7. 应用面版路由组件

1. containers/dashboard.jsx

|  |
| --- |
| */\* 应用面版的路由组件  \*/* **import** React, {Component} **from 'react' import** {**Switch**, **Route**, **Redirect**} **from 'react-router-dom' import** cookies **from 'browser-cookies' import** {**connect**} **from 'react-redux' import** {NavBar} **from 'antd-mobile'  import** {*getRedirectPath*} **from '../../utils' import** {*getUser*} **from '../../redux/actions' import** BossInfo **from '../boss-info/boss-info' import** GeniusInfo **from '../genius-info/genius-info' import** Genius **from '../genius/genius' import** Boss **from '../boss/boss' import** Msg **from '../msg/msg' import** User **from '../user/user' import** NotFound **from '../not-found/not-found' import** NavFooter **from '../../components/nav-footer/nav-footer'   class** Dashboard **extends** Component {   *// 给组件对象添加navList属性: this.navList获取* **navList** = [  {  **path**: **'/boss'**, *// 路由路径* **component**: Boss,  **title**: **'牛人列表'**,  **icon**: **'boss'**,  **text**: **'牛人'**,  },  {  **path**: **'/genius'**, *// 路由路径* **component**: Genius,  **title**: **'BOSS列表'**,  **icon**: **'job'**,  **text**: **'BOSS'**,  },  {  **path**: **'/msg'**, *// 路由路径* **component**: Msg,  **title**: **'消息列表'**,  **icon**: **'msg'**,  **text**: **'消息'**,  },  {  **path**: **'/me'**, *// 路由路径* **component**: User,  **title**: **'个人中心'**,  **icon**: **'user'**,  **text**: **'我'**,  }  ]   componentDidMount() {  *//如果cookie中有userid, 但state中的user为空, 请求获取用户* **const** userid = cookies.*get*(**'userid'**)  **const** {user} = **this**.**props  if**(userid && !user.**\_id**) {  **this**.**props**.**getUser**()  }  }   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** }   **const** {user, location} = **this**.**props** *// 如果user是空对象, 不做任何渲染* **if**(!user.**\_id**) {  **return null** *// 如果user有数据, 并且请求的是根路径, 自动跳转到/boss或/genius* } **else** {  **if**(location.**pathname**===**'/'**) {  **const** redirectPath = *getRedirectPath*(user.**type**, user.**avatar**)  **return** <**Redirect to=**{redirectPath}/>  }  **if**(user.**type**===**'boss'**) {  **this**.**navList**[1].**hide** = **true** } **else** {  **this**.**navList**[0].**hide** = **true** }  }  *// 得到当前路由对应的nav对象* **const** currentNav = **this**.**navList**.find(nav => nav.**path**===location.**pathname**)   **return** (  <**div**>  {currentNav ? <**NavBar**>{currentNav.**title**}</**NavBar**> : **null**}  <**Switch**>  <**Route path='/bossinfo' component=**{BossInfo}></**Route**>  <**Route path='/geniusinfo' component=**{GeniusInfo}></**Route**>   <**Route path='/genius' component=**{Genius}></**Route**>  <**Route path='/boss' component=**{Boss}></**Route**>  <**Route path='/msg' component=**{Msg}></**Route**>  <**Route path='/me' component=**{User}></**Route**>  <**Route component=**{NotFound}></**Route**>   </**Switch**>  {currentNav ? <**NavFooter navList=**{**this**.**navList**}/> : **null**}  </**div**>  )  } }  **export default connect**(  state => ({**user**: state.**user**}),  {*getUser*} )(Dashboard) |

1. 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/user/user.jsx

1. 静态组件

|  |
| --- |
| */\* 用户个人中心路由组件  \*/* **import** React **from 'react' import** {Result, List, WhiteSpace, Modal} **from 'antd-mobile' import** browserCookies **from 'browser-cookies'**  **const** Item = List.*Item* **const** Brief = Item.*Brief* **class** User **extends** React.Component {  render() { **return** (  <**div**>  <**Result  img=**{<**img src=**{require(**`../../assets/imgs/boy.png`**)} **style=**{{**width**: 50}} **alt="avatar"**/>}  **title='bs1'  message='百度'** />   <**List renderHeader=**{() => **'相关信息'**}>  <**Item multipleLine**>  <**Brief**>职位: Web前端工程师</**Brief**>  <**Brief**>简介: 熟悉react, 熟悉vue</**Brief**>  <**Brief**>薪资: 15K</**Brief**>  </**Item**>  </**List**>  <**WhiteSpace**/>  <**List**>  <**Item**>退出登录</**Item**>  </**List**>  </**div**>  )} }  **export default** User |

1. 动态组件

|  |
| --- |
| */\* 用户个人中心路由组件  \*/* **import** React **from 'react' import** {connect} **from 'react-redux' import** {Result, List, WhiteSpace, Modal} **from 'antd-mobile' import** browserCookies **from 'browser-cookies' import** {*resetUser*} **from '../../redux/actions'  const** Item = List.*Item* **const** Brief = Item.*Brief* **class** User **extends** React.Component {   *logout* = () => {  Modal.*alert*(**'注销'**, **'确认退出登录吗?'**, [  {  **text**: **'取消'**,  onPress: () => **console**.log(**'cancel'**)},  {  **text**: **'确认'**,  onPress: () => {  browserCookies.*erase*(**'userid'**)  **this**.**props**.**resetUser**()  }  }  ])  }   render() {  **const** {name, avatar, type, title, desc, money, company} = **this**.**props   return** (  <**div**>  <**Result  img=**{<**img src=**{require(**`../../assets/imgs/**${avatar}**.png`**)} **style=**{{**width**: 50}} **alt="avatar"**/>}  **title=**{name}  **message=**{type === **'boss'** ? company : **null**}  />   <**List renderHeader=**{() => **'相关信息'**}>  <**Item multipleLine**>  <**Brief**>职位: {title}</**Brief**>  <**Brief**>简介: {desc}</**Brief**>  {money ? <**Brief**>薪资: {money}</**Brief**> : **null**}  </**Item**>  </**List**>  <**WhiteSpace**/>  <**List**>  <**Item onClick=**{**this**.*logout*}>退出登录</**Item**>  </**List**>  </**div**>  )  } }  **export default** connect(  state => state.**user**,  {*resetUser*} )(User) |

### 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. Boss/牛人列表功能

### 2.15.1. 后台路由: server/appRouter.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*(**'/api/list'**, {type}) |

### 2.15.3. redux管理状态

1. redux/action-types.js

|  |
| --- |
| *// 获取用户列表* **export const** USER\_LIST = **'USER\_LIST'** |

1. redux/actions.js

|  |
| --- |
| **import** {  USER\_LIST } **from './action-types' import** {  *reqUserList* } **from '../api'**  *// 用户列表* **const** *userList* = (users) => ({**type**: USER\_LIST, **data**: users}) *// 异步获取用户列表* **export const** *getUserList* = (type) => {  **return async** dispatch => {  **const** result = **await** *reqUserList*(type)  **if** (result.**code** === 0) {  dispatch(*userList*(result.**data**))  }  } } |

1. redux/reducers.js

|  |
| --- |
| **const** initUserList = []  **function** *userList*(state = initUserList, action) {  **switch** (action.**type**) {  **case** 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** PropTypes **from 'prop-types' 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  title='bs1'  thumb=**{require(**`../../assets/imgs/boy.png`**)}  **extra=**{<**span**>web前端工程师</**span**>}  />  <**Body**>  <**div**>公司: 百度</**div**>  <**div**>描述: 熟悉react, redux</**div**>  <**div**>薪资: 15k</**div**>  </**Body**>  </**Card**>  </**div**>  <**div**>  <**WhiteSpace**/>  <**Card**>  <**Header  title='br1'  thumb=**{require(**`../../assets/imgs/boy.png`**)}  **extra=**{<**span**>web前端工程师</**span**>}  />  <**Body**>  <**div**>描述: 熟悉react, redux</**div**>  </**Body**>  </**Card**>  </**div**>  </**WingBlank**>  )  } }  **export default** UserList |

1. 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' import** {**withRouter**} **from 'react-router-dom'  const** Header = Card.*Header* **const** Body = Card.Body  **class** UserList **extends** React.Component {   **static** *propTypes* = {  **userList**: PropTypes.*array*.isRequired  }   *handleClick* = user => {  **console**.log(**'click'**, user)  **this**.**props**.**history**.**push**(**`/chat/**${user.**\_id**}**`**, {**name**: user.**name**})  }   render() {  **return** (  <**div**>  <**WingBlank size="lg"**>  {**this**.**props**.**userList**.map(v => (   <**div**>  <**WhiteSpace**/>  <**Card key=**{v.**\_id**} **onClick=**{() => **this**.*handleClick*(v)}>  <**Header  title=**{v.**name**}  **thumb=**{v.**avatar** ? require(**`../../assets/imgs/**${v.**avatar**}**.png`**) : **null**}  **extra=**{<**span**>{v.**title**}</**span**>}  />  <**Body**>  {v.**type** === **'boss'** ? (<**div**>公司: {v.**company**}</**div**>) : **null**}  <**div**>描述: {v.**desc**}</**div**>  {v.**type** === **'boss'** ? (<**div**>薪资: {v.**money**}</**div**>) : **null**}  </**Body**>  </**Card**>  </**div**>  ))}  </**WingBlank**>  </**div**>  )  } }  **export default withRouter**(UserList) |

### 2.15.5. BOSS路由组件: containers/boss/boss.jsx

|  |
| --- |
| */\* boss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** Boss **extends** React.Component {   componentDidMount() {  **this**.**props**.**getUserList**(**'genius'**)  }   render() {  **return** <**UserList userList=**{**this**.**props**.**userList**}></**UserList**>  }  }  **export default** connect(  state => ({**userList**: state.**userList**}),  {*getUserList*} )(Boss) |

### 2.15.6. 牛人路由组件: containers/genius/genius.jsx

|  |
| --- |
| */\* 牛人的主路由组件  \*/* **import** React **from 'react' import** {connect} **from 'react-redux'  import** {*getUserList*} **from '../../redux/actions' import** UserList **from '../../components/user-list/user-list'  class** Genius **extends** React.Component {   componentDidMount() {  **this**.**props**.**getUserList**(**'boss'**)  }   render() {  **return** <**UserList userList=**{**this**.**props**.**userList**}></**UserList**>  } }  **export default** connect(  state => ({**userList**: state.**userList**}),  {*getUserList*} )(Genius) |

### 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

1. Index.less

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

1. dashboard.jsx

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

1. user-list.jsx

|  |
| --- |
| <**WingBlank style=**{{**marginTop**: 50, **marginBottom**: 50}}> |

## 2.16. 实现实时对话通信功能

### 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

#### 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>

#### **编码例子**

* 服务器端

|  |
| --- |
| *// 得到服务器对象* **const** server = require(**'http'**).Server(app) *// 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**(daa) {  **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)  }) })  *// 启动服务器监听*  server.listen(4000, () => {  **console**.log(**'服务器启动成功 port: 4000'**) }) |

* 客户端

|  |
| --- |
| *// 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()}) |

### 2.16.3. 后台实现

#### 添加新的数据库集合模型: server/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\_to组成字符串* **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* mongoose.model(**'chat'**, chatSchema) |

#### **新增路由: server/appRouter.js**

|  |
| --- |
| **const** ChatModel = models.getModel(**'chat'**)  */\* 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avatar组件的user对象* **const** users = {}  userDocs.forEach(doc => {  users[doc.**\_id**] = {**name**: doc.**name**, **avatar**: doc.**avatar**}  })  */\*  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**})  }) }) |

#### **使用上socket.io实现实时通信: server/server.js(初步)**

|  |
| --- |
| **const** ChatModel = require(**'./models'**).getModel(**'chat'**) *// 得到操作chats的Model* **const** server = require(**'http'**).Server(app) *// 得到服务器对象* **const** io = require(**'socket.io'**)(server) *// 得到操作服务器端sokectIO的io对象* *// 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**(**'sendMsg'**, **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)  })  }) })  server.listen(4000, **function**(){  **console**.log(**'app listening at port 4000'**); }) |

#### **优化socketIO的消息分发**

* **问题: 后台将消息保存到数据库后, 会将消息全局分发, 但应该只用发送给2个客户端**
* **客户端: actions.js**

|  |
| --- |
| */\* 初始化浏览器端的socketio的函数(在注册/登陆/获取用户信息成功后调用)  连接服务器端(指定userid参数)  绑定用来接收服务器发送消息的监听  \*/* **const** *initIO* = (dispatch, userid) => {  *// 连接服务器* io.**socket** = io(**`ws://localhost:4000?userid=**${userid}**`**)  io.**socket**.on(**'receiveMsg'**, (chatMsg) => {  **console**.log(**'浏览器接收到服务器消息'**, chatMsg)  dispatch(*receiveMsg*(chatMsg, chatMsg.**to** === userid))  }) } |

* **服务器端**

|  |
| --- |
| **const** server = require(**'http'**).Server(app) **const** io = require(**'socket.io'**)(server) **const** sockets = {} *//用来缓存所有与客户端连接socket* **const** ChatModel = require(**'./models'**).getModel(**'chat'**)  *// 绑定连接监听, 一旦有客户端与服务连接上了, 就自动调用回调函数* io.on(**'connection'**, **function** (socket) {  **console**.log(**'有客户端连接上了'**)  *// 得到连接url中包含的参数userid* **const** userid = socket.**handshake**.**query**.**userid** *// 如果userid不存在, 不做任何处理, 直接结束* **if**(!userid) {  **return** }  *// 如果缓存中已存在, 移除, 断开连接* **const** savedSocket = sockets[userid]  **if**(savedSocket) {  **delete** sockets[userid]  savedSocket.disconnect()  }  *// 将新的连接缓存起来* sockets[userid] = socket   **console**.log(**'sockets'**, Object.keys(sockets))   *// 绑定sendMsg监听, 用来接收客户端发送的聊天消息* socket.on(**'sendMsg'**, **function** ({from, to, content}) {  **console**.log(**'服务器接收到客户端的消息'**, from, to, content)  *// 将消息保存到数据库* **const** create\_time = Date.now()  **const** chat\_id = [from, to].sort().join(**'\_'**)  **const** chatModel = **new** ChatModel({from, to, content, create\_time, chat\_id})  chatModel.save(**function** (err, chatMsg) {  **console**.log(err, chatMsg, {from, to, content, create\_time, chat\_id})  *//保存完成后, 向from和to对应的客户端分发receiveMsg消息* sockets[from] && sockets[from].emit(**'receiveMsg'**, chatMsg)  sockets[to] && sockets[to].emit(**'receiveMsg'**, chatMsg)  })  }) }) |

### 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*(**'/api/msglist'**)  *// 标识查看了指定用户发送的聊天信息* **export const** *reqReadChatMsg* = (from) => *ajax*(**'/api/readmsg'**, {from}, **'POST'**) |

### 2.16.5. redux管理状态

#### 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* |

#### **redux/actions.js**

|  |
| --- |
| **import** io **from 'socket.io-client' import** {  *reqChatMsgList*,  *reqReadChatMsg* } **from '../api' import** {  RECEIVE\_MSG,  MSG\_READ, } **from "./action-types"**  */\* 初始socketIO, 绑定接收监听异步接收服务器发送的消息*  *(在注册/登陆/获取用户信息成功后调用)  \*/* **const** *initIO* = (dispatch, userid) => {  **if** (!io.**socket**) {  *// 连接服务器* io.**socket** = io(**`ws://localhost:4000?userid=**${userid}**`**)  io.**socket**.on(**'receiveMsg'**, (chatMsg) => {  **console**.log(**'浏览器接收到服务器消息'**, chatMsg)  dispatch(*receiveMsg*(chatMsg, chatMsg.**to** === userid))  })  } }  */\* 异步获取当前用户所有相关消息的列表 (在注册/登陆/获取用户信息成功后调用)  \*/* **const** *getMsgList* = **async** (dispatch, userid) => {  **const** response = **await** *reqChatMsgList*()  **const** result = response.**data  if** (result.**code** === 0) {  **const** {chatMsgs, users} = result.**data** dispatch(*receiveMsgList*({chatMsgs, users, userid}))  } }  *// 接收一个消息的同步action* **const** *receiveMsg* = (chatMsg, isToMe) => ({**type**: RECEIVE\_MSG, **data**: {chatMsg, isToMe}}) *// 接收消息列表的同步action* **const** *receiveMsgList* = ({chatMsgs, users, userid}) => ({**type**: RECEIVE\_MSG\_LIST, **data**: {chatMsgs, users, userid}}) *// 接收读取了消息的同步action* **const** *msgRead* = ({from, to, count}) => ({**type**: MSG\_READ, **data**: {from, to, count}})  */\* 异步发送消息  \*/* **export const** *sendMsg* = ({from, to, content}) => {  **return** dispatch => {  io.**socket**.emit(**'sendMsg'**, {from, to, content})  **console**.log(**'浏览器向服务器发送消息'**, {from, to, content})  } }  */\* 异步更新读取消息的action  \*/* **export const** *readMsg* = (userid) => {   **return async** (dispatch, state) => {  **const** response = **await** *reqReadChatMsg*(userid)  **const** result = response.**data  if**(result.**code**===0) {  **const** count = result.**data  const** from = userid  **const** to = state.**user**.**\_id** dispatch(*msgRead*({from, to, count}))  }  } } |

#### **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\_LIST:  **const** {chatMsgs, users, userid} = action.**data  return** {  chatMsgs,  users,  **unReadCount**: chatMsgs.reduce((preTotal, msg) => !msg.**read**&&msg.**to**===userid, 0) *// 统计未读数量* }  **case** RECEIVE\_MSG:  **const** {chatMsg, isToMe} = action.**data  return** {  **chatMsgs**: [...state.**chatMsgs**, chatMsg],  **users**: state.**users**,  **unReadCount**: state.**unReadCount** + (isToMe ? 1 : 0)  }  **case** MSG\_READ:  **const** {from, to, count} = action.**data  return** {  **chatMsgs**: state.**chatMsgs**.map(msg => {  **if**(msg.**from**===from && msg.**to**===to && !msg.**read**) {  **return** {...msg, **read**: **true**}  }  **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

#### 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/boy.png'**)}  >  你好  </**Item**>  <**Item  thumb=**{require(**'../../assets/imgs/boy.png'**)}  >  你好2  </**Item**>  <**Item  className='chat-me'  extra=**{<**img src=**{require(**'../../assets/imgs/girl.png'**)}/>}  >  很好  </**Item**>  <**Item  className='chat-me'  extra=**{<**img src=**{require(**'../../assets/imgs/girl.png'**)}/>}  >  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**: 15**px**;  **text-align**: **right**; } |

#### **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**: **''** }   *handleSubmit* = () => {  **const** content = **this**.**state**.**content**.trim()  **if**(content) {  **const** from = **this**.**props**.**user**.**\_id  const** to = **this**.**props**.**match**.params.**userid  this**.**props**.**sendMsg**({from, to, content})  **this**.setState({**content**: **''**})  }  }   render() {   **const** {user} = **this**.**props  const** {users, chatMsgs} = **this**.**props**.*chat* **if**(!user.**\_id** || users.**length**===0) {  **return null** }  **const** targetId = **this**.**props**.**match**.params.**userid  const** meId = user.**\_id  const** chat\_id = [targetId, meId].sort().join(**'\_'**)  **const** msgs = chatMsgs.filter(msg => msg.**chat\_id**===chat\_id)  **const** targetIcon = users[targetId] ? require(**`../../assets/imgs/**${users[targetId].**avatar**}**.png`**) : **null  const** meIcon = require(**`../../assets/imgs/**${user.**avatar**}**.png`**)   **return** (  <**div id='chat-page'**>  <**NavBar className='stick-top'  icon=**{<**Icon type='left'**/>}  **onLeftClick=**{() => {  **this**.**props**.**history**.**goBack**()  }}>  {users[targetId].**name**}  </**NavBar**>  <**List style=**{{**marginTop**: 50}}>  {  msgs.map((msg) => {  **if**(msg.**to**===meId) {  **return** (  <**Item  key=**{msg.**\_id**}  **thumb=**{targetIcon}  >  {msg.**content**}  </**Item**>  )  } **else** {  **return** (  <**Item  key=**{msg.**\_id**}  **className='chat-me'  extra=**{<**img src=**{meIcon}/>}  >  {msg.**content**}  </**Item**>  )  }  })  }  </**List**>   <**div className='am-tab-bar'**>  <**InputItem  placeholder="请输入"  extra=**{  <**span onClick=**{**this**.*handleSubmit*}>发送</**span**>  }  **value=**{**this**.**state**.**content**}  **onChange=**{val => **this**.setState({**content**: val})}  />  </**div**>  </**div**>  )  } }  **export default connect**(  state => ({  **user**: state.**user**,  **chat**: state.*chat* }),  {*sendMsg*} )(Chat) |

#### **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**) } |

#### **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)  } }  <**span  onClick=**{**this**.*toggleShow*}  **style=**{{**marginRight**: 10}}>笑脸</**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** } |

### 2.16.7. 消息列表组件: containers/msg/msg.jsx

#### 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** Msg **extends** Component {   render() {  **return** (  <**List**>  <**Item  extra=**{<**Badge text=**{3}/>}  **thumb=**{require(**`../../assets/imgs/boy.png`**)}  **arrow='horizontal'** >  你好  <**Brief**>nr1</**Brief**>  </**Item**>  <**Item  extra=**{<**Badge text=**{0}/>}  **thumb=**{require(**`../../assets/imgs/girl.png`**)}  **arrow='horizontal'** >  你好2  <**Brief**>nr2</**Brief**>  </**Item**>  </**List**>  )} }  **export default Msg** |

#### **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** Msg **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**>  {  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** }) )(Msg) |

### 2.16.8. 应用面板组件: containers/dashboard/dashboard.jsx

|  |
| --- |
| */\* 应用面板的路由组件  \*/* **import** React, {Component} **from 'react' import** {**Route**, **Switch**} **from 'react-router-dom' import** cookies **from 'browser-cookies' import** {connect} **from 'react-redux' import** {NavBar} **from 'antd-mobile'  import** BossInfo **from '../boss-info/boss-info' import** GeniusInfo **from '../genius-info/genius-info' import** Boss **from '../boss/boss' import** Genius **from '../genius/genius' import** Msg **from '../msg/msg' import** User **from '../user/user' import** NotFound **from '../../components/not-found/not-found' import** NavFooter **from '../../components/nav-footer/nav-footer' import** Chat **from '../chat/chat' import** {*getRedirectPath*} **from '../../utils' import** *{getUser}* **from '../../redux/actions'  class** Dashboard **extends** Component {   *// 给组件对象添加navList属性: this.navList获取* **navList** = [  {  **path**: **'/boss'**, *// 路由路径* **component**: Boss,  **title**: **'牛人列表'**,  **icon**: **'boss'**,  **text**: **'牛人'**,  **hide**: **false** },  {  **path**: **'/genius'**, *// 路由路径* **component**: Genius,  **title**: **'BOSS列表'**,  **icon**: **'job'**,  **text**: **'BOSS'**,  **hide**: **false** },  {  **path**: **'/msg'**, *// 路由路径* **component**: Msg,  **title**: **'消息列表'**,  **icon**: **'msg'**,  **text**: **'消息'**,  },  {  **path**: **'/me'**, *// 路由路径* **component**: User,  **title**: **'个人中心'**,  **icon**: **'user'**,  **text**: **'我'**,  }  ]   componentDidMount () {  *// 取出cookie中的userid* **const** userid = cookies.*get*(**'userid'**)  *// 从props中取出user* **const** {user} = **this**.**props** *// userid有值, 而user中没有数据, 才去请求获取userinfo* **if**(userid && !user.**\_id**) {  **this**.**props**.**getUser**()  }  }   render () {  *// 检查用户是否登陆, 如果没有, 跳转到login* **const** userid = cookies.*get*(**'userid'**)  **const** {user, location} = **this**.**props  if**(!userid && !user.**type**) {  **this**.**props**.**history**.**replace**(**'/login'**)  **return null** }   **if**(user.**type**) { *// 已经登陆* **if**(location.**pathname**===**'/'**) { *// 访问的/  // 计算需要跳转的路由路径* **const** path = *getRedirectPath*(user.**type**, user.**avatar**)  *// 跳转* **this**.**props**.**history**.**replace**(path)   **return null** }  **if**(user.**type**===**'boss'**) {  *// 隐藏第2个* **this**.**navList**[1].**hide** = **true** } **else** {  *// 隐藏第2个* **this**.**navList**[0].**hide** = **true** }  } **else** { *// redux中user为空, 但cookie有userid* **return null** *//不做任何显示* }   *// 计算出当前的nav对象  /\*  {  path: '/me', // 路由路径  component: User,  title: '个人中心',  icon: 'user',  text: '我',  }  \*/* **const** currentNav = **this**.**navList**.find(nav => nav.**path**===location.**pathname**)  *// 得到props中的unReadCount* **const** unReadCount = **this**.**props**.**unReadCount   return** (  <**div**>  {currentNav ? <**NavBar**>{currentNav.**title**}</**NavBar**> : **null**}   <**Switch**>  <**Route path='/bossinfo' component=**{BossInfo}/>  <**Route path='/geniusinfo' component=**{GeniusInfo}/>  <**Route path='/chat/:userid' component=**{Chat}/>  {  **this**.**navList**.map((nav, index) => (  <**Route key=**{index} **path=**{nav.**path**} **component=**{nav.**component**}/>  ))  }  <**Route component=**{NotFound}/>  </**Switch**>   {currentNav ? <**NavFooter navList=**{**this**.**navList**} **unReadCount=**{unReadCount}/> : **null**}  </**div**>  )  } }  **export default** connect(  state => ({  **user**: state.**user**, *// 登陆的user* **unReadCount**: state.*chat*.**unReadCount**} *// 未读消息数量* ),  {*getUserInfo*} )(Dashboard) |

### 2.16.9. 列表显示未读消息数量: 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() {  *// 过滤掉hide为true的* **const** navList = **this**.**props**.**navList**.filter(v => !v.*hide*)  *// 当前请求的path* **const** {pathname} = **this**.**props**.**location** */\*  {  path: '/me', // 路由路径  component: User,  title: '个人中心',  icon: 'user',  text: '我',  }  \*/* **return** (  <**TabBar**>  {navList.map(v => (  <**Item  key=**{v.**path**}  **badge=**{v.**path** === **'/msg'** ? **this**.**props**.**unReadCount** : 0}  **title=**{v.**text**}  **icon=**{{**uri**: require(**`./imgs/**${v.**icon**}**.png`**)}}  **selectedIcon=**{{**uri**: require(**`./imgs/**${v.**icon**}**-active.png`**)}}  **selected=**{pathname === v.**path**}  **onPress=**{() => {  **this**.**props**.**history**.**replace**(v.**path**)  }}  >  </**Item**>  ))}  </**TabBar**>  )  } }  **export default withRouter**(NavFooter) |

### 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** |

1. 在聊天组件使用: containers/chat/chat.jsx

|  |
| --- |
| **import** QueueAnim **from 'rc-queue-anim'**  <**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**> |