移动端适配问题的解决(配合Vant组件库)

1. 安装相关依赖

```
npm install amfe-flexible --save
npm install postcss-pxtorem --save-dev
```

2. 在main.js中引入amfe-flexible

```
import 'amfe-flexible'
```

3. 在vue-cli2中， 在.postcss.js文件中的plugins下新增postcss-pxtorem的配置

```
module.exports = {
  "plugins": {
    "postcss-pxtorem": {
        rootValue: 192, // 根据设计图尺寸写，值是设计图的1/10,设计图是1920，就写192
        propList: ['*'], // 需要被转换的属性
        selectorBlackList: [] // 不进行px转换的选择器
    }
  }
}
```

首页-推荐和新闻头条部分+新冠疫情数据可视化展示

好文-诗词-成语词典这三个点击进去之后都是列表项目，分别展示文章标题，诗词标题，成语名称，使用拼音索引栏目进行列表检索功能，搜索栏目实现搜索文章，诗词，成语词典的功能。趣味答题单独拆分开来，主要实现答题的功能，必须登录，不登录不可以进行答题。开心一刻只需要实现段子的刷新即可，不需要列表的功能，可以收藏，可以评论。根据知乎api的接口实现标准执行即可。

// 文章-顶部添加分类的，按照阅读量，按照点赞量，按照评论量筛选

上拉加载，下拉刷新功能的实现

诗词分类，分为诗词曲古文四个种类进行筛选-按照阅读量，按照点赞量，按照评论量筛选

技术问题记录

- textarea出现不能高度自适应的问题

```
解决方案
使用div的contenteditable属性
<div class="content-editable" placeholder="123"
contenteditable="true"@input="divInput" :value="value" ></div>
.content-editable {
  min-height: 60px;
  outline: none;
  font-size: 15px;
  background-color: #fff;
  padding: 10px 8px 8px 8px;
  line-height: 20px;
}
但是使用contenteditable属性会出现不能使用placeholder的问题，使用下面的方法解决
.content-editable:empty::before {
  content: attr(placeholder);
}
```

- 1px边框的实现

```
.one-px {
   position: absolute;
   height: 1px;
   left: -5px;
   right: -5px;
   background-color: rgba(138, 138, 138, .2);
   top: 0;
   -webkit-transform: scaleY(.5);
   transform:scaleY(.5);
}
```

- 底部的评论输入组件使用contenteditable，切换输入，也就是默认的不能编辑，点击之后进入可以编辑的状态，然后使用v-directives的指令编辑v-focus指令自动获取输入的焦点
- 设备类型的判断，针对输入框的变化

```
function isAndroid () {
  const ua = typeof window === 'object' ? window.navigator.userAgent : ''
  return /Android/i.test(ua)
}

function isIOS () {
  const ua = typeof window === 'object' ? window.navigator.userAgent : ''
  return /iPhone|iPod|iPad/i.test(ua)
}
// 这里记得要将事件监听在页面关闭的时候卸载，避免出现内存泄露的问题
if (isAndroid()) {
    const innerHeight = window.innerHeight
    window.addEventListener('resize', () => {
      const newInnerHeight = window.innerHeight
      if (innerHeight > newInnerHeight) {
        // 键盘弹出事件处理
      } else {
        // 键盘收起事件处理
        this.showInput = false
      }
    })
  } else if (isIOS()) {
    window.addEventListener('focusin', () => {
      // 键盘弹出事件处理
      // alert('iphone 键盘弹出事件处理')
```

```
      // this.$toast('弹起')
    })
     window.addEventListener('focusout', () => {
     // 键盘收起事件处理
     // alert('iphone 键盘收起事件处理')
     // this.$toast('下来')
     this.showInput = false
    })
  }
```

- 建议不要直接使用 cnpm 安装依赖，会有各种诡异的 bug。可以通过如下操作解决 npm 下载速度慢的问题

```
npm install --registry=https://registry.npm.taobao.org
```

- // 根据环境来使用vConsole，避免ESlint错误
  var vc = new VConsole() // eslint-disable-line no-unused-vars

富文本编辑器推荐

https://github.com/nhn/tui.editor

需求记录

- 好文底部增加推荐文字的内容，以点赞量随机推荐
- 诗词下面增加注释模块
- 修改开心一刻为金句模块

淘宝镜像配置

```
npm config set registry https://registry.npm.taobao.org
// 配置后可通过下面方式来验证是否成功
npm config get registry
// 或
npm info express
如果将来你需要发布自己的软件包时，需要将registry字段的值修改回来
```

Vue项目记录

1.路由权限控制

2.登录之后返回原来的页面

CDN推荐

https://www.jsdelivr.com/package/npm/vant

package.json的版本号处理

指定版本号
(1)普通版本号: 表示安装此版本,比如"classnames": "2.2.5"，表示安装2.2.5的版本
(2)表示安装大版本的最小最新子版本: ~版本,比如 "babel-plugin-import": "~1.1.0",表示安装1.1.x的最新版本（不低于1.1.0），但是不安装1.2.x，也就是说安装时不改变大版本号和次要版本号
(3)表示安装大版本的最高中版本: ^版本,比如 "antd": "^3.1.4",，表示安装3.1.4及以上的版本，但是不安装4.0.0，也就是说安装时不改变大版本号。

Node.js+Koa2+Mongoose后端问题记录

Koa书写中间件的时候必须加上async和await，因为按照不加这对属性，就没有办法保证中间件的执行顺序。

在下一个中间件向上一个中间件传递参数，可以通过ctx上下文对象传递，比如ctx.textValue = '传递的参数'

在上一个中间件中通过ctx.textValue获取，同时重点要保证是按照洋葱模型实现的。

REST风格

- 万维网软件架构风格
- Representational State Transfer 数据的表现形式JSON、XML
- State表示当前的状态或者是数据
- Transfer 数据传输

REST的六个限制

- 客户-服务器（CS架构）
- 关注点分离
- 前端专注用户界面，提高了可移植性
- 无状态-Stateless
- 所有的用户会话都保存在客户端
- 每次请求都必须包含所有的信息，不能依赖上下文信息
- 服务端不用保存会话信息，提升了简单性、可靠性、可见性
- 缓存，所有服务端的响应都要被标注为可缓存或者不可缓存，减少前后端的交互，提升了性能
- 统一接口，Uniform Interface，接口设计尽可能统一使用，提升了简单性，可见性，接口与实现解耦，使前后端可以独立开发迭代
- 分层系统（Layered System）每层只知道相邻的一层，后面的隐藏的就不知道了，客户端不知道是和代理还是真是的服务器通信，其他层：安全层，负载均衡、缓存层等。
- 按需代码（Code-On-Demand)，客户端可以下载运行服务端传来的代码，比如JS，通过减少一些功能，简化了服务端。

统一接口的限制

- 资源的标识
  - 资源是任何可以命名的事物，比如用户、评论等
  - 每个资源可以通过URI被唯一标识，如https://api.github.com/users
- 通过表述来操作资源
  - 表述就是REpresentational，比如JSON、XML
  - 客户端不能直接操作比如SQL的服务端资源
  - 客户端应该通过表述比如JSON来操作资源
- 自描述消息
  - 每个消息（请求或者是响应）必须提供足够的信息让接受者理解
  - 媒体类型（application/json、application/xml)
  - HTTP方法：GET（查）、POST（增）、DELETE（删）等
  - 是否缓存：Cache-Control，决定缓存是被可以被缓存

什么是RESTful API

- 具体样子
  - 基本的URI，例如：https://api.github.com/users
  - 标准的HTTP方法：如GET、POST、PATCH、DELETE等
  - 传输的数据媒体类型，如JSON、XML
- 实际举例
  - GET /users #获取user列表
  - GET /users/12 #查看某个具体的user
  - POST /users #新建一个user
  - PUT /users/12 #更新user 12
  - DELETE /users/12 #删除user 12
- 请求设计规范
- URI尽量使用名词而且是复数的形式，如/users
- URI 使用嵌套表示关联关系，如/users/12/repos/15,表示id为12的仓库为15的用户
- 使用正确的HTTP方法，如GET/POST/PUT/DELETE，尽量不要造成歧义,PATCH表示局部替换，PUT方法一般用来整体编辑替换
- 不符合增删改查的情况：POST/action/子资源
- 响应设计规范
  - 查询

    - 分页
    - 字段过滤
    - 状态码
    - 错误处理
- 安全
    - HTTPS
    - 鉴权-用户鉴权，针对后台管理系统
    - 限流，防范刷流量刷接口的狗东西

控制器

- 获取HTTP请求参数
    - query string 例如?q=keyword
    - router params，例如/users/:id，路由参数一般都是必传
    - body，例如：{ name: '李磊' }
    - header ，如accept、cookie
- 处理业务逻辑
    - 每个资源的控制器放在不同的文件里
    - 尽量使用类+方法名的形式编写控制器,提高可读性
    - 严谨的错误处理
- 发送HTTP响应
    - 发送status，如200/400等
    - 发送body，如{ name: '李磊'}
    - 发送header，如allow、content-type，返回的格式应该怎么样解析

异常处理

- 运行时错误，都返回500
    - 语法没有错误，代表服务器内部错误
- 逻辑错误，如找不到返回404， 先决条件失败返回412（比如id根本不存在），无法处理得实体如参数格式不对返回422， 409代表冲突已存在的内容
- 主要是为了防止程序挂掉，保证稳定度，使用try catch防止程序挂掉，增加程序的健壮性。
- 告诉用户错误的信息，及时反馈用户，让他知道自己错哪了

- 便于开发者调试代码

中间件集合使用

- koa-body-请求体处理中间件
- koa-router-路由中间件
- koa-json-error--Koa2的错误处理json格式中间件
- koa-parameter-参数校验
- koa-static-静态文件中间件
- koa-jsonp-jsonp中间件
- svg-captcha-svg验证码的中间件
- jsonwebtoken-token生成
- koa-jwt-koa权限校验

JWT-json-web-token

- 新一代的开放标准RFC7519
- 定义了一种紧凑且独立的方式，可以将各方之间的信息作为JSON对象进行安全传输
- 该信息可以验证和信任，因为是经过数字签名的
- 由三部分构成
    - 头部Header
        - typ：token的类型，这里固定为jwt
        - alg：使用的hash算法，如HMAC SHA256或者RSA
    - 有效载荷Payload
        - 存储需要传递的信息，例如ID，用户名等
        - 与Header不同，Payload可以加密
    - 签名Signature

JWT和session的对比

- 可拓展性：session在分布式的时候需要做服务器共享处理，在这里JWT更好
- 安全性，都需要相应的保护措施，加密处理
- RESTful API 无状态，JWT更优秀
- JWT性能相对来说有点不太好，大量的用户信息在JWT中，sessionid会给服务器造成压力，各有优缺点，jwt用空间换时间，jwt体积更大
- jwt时效性不如session。session可以主动清除，jwt不能立即生效，必须等到过期之后才能操作，session可以在本地或者服务器端主动予以清除

1.常见的资源的使用

- koa-router 路由
- @koa/cors跨域处理
- koa-static静态资源处理
- koa-body，koa-json协议处理
- koa-session，koa-jwt安全与鉴权处理
- jsonwebtoken：生成token
- koa-logger日志处理
- koa-helmet 通信头
- koa-combine-routers路由合并
- koa-parameter: 数据校验中间件
- koa-json-error错误处理中间件

```
// koa-json-error的生成阶段与开发阶段的切换设置
// 下载cross-env设置环境变量
// 在package.json里面配置下面的环境变量的设置
"scripts": {
    "start": "cross-env NODE_ENV===production nodemon app"
  },
```

2.jsonwebtoken

- koa-jwt为权限校验
- jsonwebtoken产生token
- 使用bearer+token添加到authorization的header头里面鉴权

3.接口设计原则

- 简单：高内聚，与业务对应
- 高效：属性设计、数据结构、方法抽象
- 兼容：尽量保证兼容，进行版本与状态控制

4.获取get请求的地址栏参数，在koa-body中获取

- users/:id ----> user/1:通过ctx.params.id获取
- query-string
- body，如{name: '李磊'}--post请求，由content-type决定
- Header，如Accept和Cookie

5.Restful增删查改的最佳实践

- 新建：返回新建的信息
- 修改：返回修改后的
- 删除：只返回204表示删除成功DELETE

6.HTTP状态码

- 1xx表示 信息，服务器收到请求，需要请求者继续执行操作
- 2xx表示， 成功，操作被成功接收并处理
    - 200： 请求成功。一般用于GET与POST请求
    - 201： 已创建。成功请求并创建了新的资源
    - 202： 已接受。已经接受请求，但未处理完成
    - 203： 非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本
    - 204： 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
    - 205： 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
    - 206： 部分内容。服务器成功处理了部分GET请求
- 3xx表示， 重定向，需要进一步的操作以完成请求

- 4xx表示， 客户端错误，请求包含语法错误或无法完成请求
    - 400： 客户端请求的语法错误，服务器无法理解
    - 401： 请求要求用户的身份认证
    - 402： 保留，将来使用
    - 403： 服务器理解请求客户端的请求，但是拒绝执行此请求
    - 404： 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
    - 405： 客户端请求中的方法被禁止
    - 406： 服务器无法根据客户端请求的内容特性完成请求
    - 407： 请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权
    - 408： 服务器等待客户端发送的请求时间过长，超时
    - 409： 服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突
    - 410： 客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置
    - 411： 服务器无法处理客户端发送的不带Content-Length的请求信息
    - 412： 客户端请求信息的先决条件错误
- 5xx表示， 服务器错误，服务器在处理请求的过程中发生了错误
    - 500： 服务器内部错误，无法完成请求

7.koa错误处理

- ctx.throw抛出错误：ctx.throw(状态码，错误提示信息)

8.mongoose的增删改查

8.1mongoose支持的数据类型

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId

- Array

增加个人信息的所在地

用户的个人信息可以包括所在城市，城市由三级解耦实现，用户的毕业学校，用户的个人签名，头像修改，生日

8.2mongoose的操作

- findOne：查找一条数据
- find({name: new RegExp(ctx.query.q)}) // 实现模糊搜索
- findById：根据id查询
- findByIdAndUpdate：根据id查询并且更新
- findByIdAndRemove：根据id查询到并且删除
- ctx.findByIdAndUpdata(ctx.params.id, {$inc: { countNumber: 1 }})-->指定的地方递增1

####8.3ctx.origin表示获取网站的原地址；http://localhost:3000

9.Mongoose自带的时间戳

new Schema({}, {timestamps: true})----> 这样就会有自带的创建时间和更新时间

###10.ali-oss上传图片无法读取的问题-将读取权限设置为公共读即可，后面的问题可以通过设置域名来进行配置

11.前端方法实现分页

```
// 返回我的关注
  async listFollowing(ctx) {
    const { perpage = 2 } = ctx.query
    const perPage = Math.max(perpage * 1, 1)
    // 默认从第一页开始
    const page = Math.max(ctx.query.current_page * 1, 1)
    const user = await
userModel.findById(ctx.params.id).select('+following').populate('following'
)
```

```
    if (!user) {
      ctx.throw(404, '用户不存在')
    }

    // 前端方法的分页实现 slice的效率比splice的高
    const newUser = user.following.slice((page - 1) * perPage, page *
perPage)
    const allUsers = await
userModel.findById(ctx.params.id).select('+following').populate('following'
)
    ctx.body = {
      errno: 0,
      data: newUser,
      total: allUsers.following.length
    }
  }
```

腾讯云部署记录

1.关机停止重启

2.重新安装系统盘

3.常用的操作Linux命令

- cd / 返回根路径
- cd ~ 返回默认的home目录下面
- 连接远程服务器 ssh root@ 42.192.120.52 ----> ssh 用户名@远程ip

4.配置ssh key避免反复登录的问题（这个不重要，看不懂可以跳过）

- cd ~
- ls -al 获取当前目录下的所有文件
- cd .ssh/ 进入.shh文件夹，找到known_hosts这个文件

- 在.ssh目录下输入ssh-keygen -t rsa -f 'chengang-key'也就是生成的公钥名称
- 将生成的chengang-key.pub拷贝到远程服务器 scp chengang-key.pub root@42.192.120.52:.，其中:.表示拷贝到远程服务器的默认路径，也就是~这个路径下
- 登录ssh远程服务器，将~目录下面拷贝过来的chengang-key.pub文件拷贝到.ssh文件夹下面的authorized_keys下面：cat ~/chengang-key.pub >> ~/.ssh/authorized_keys
- 重启服务器：systemctl restart sshd
- 在本地的.ssh中创建config文件

5.安装Node.js环境

- yum update -y更新软件环境
- 安装Node.js：yum install nodejs -y

6.安装git

```
yum install –y git
```

7.在VSCode中插件模块找到sftp并安装

ENOENT: no such file or directory, open 'C:\Users\cghbh.ssh\chengang-key'

8.安装NGINX

```
yum install nginx –y

// 打开NGINX的配置文件
vim /etc/nginx/nginx.conf

// 退出vim编辑器
输入：，然后输入wq按住enter键退出

// 文件拷贝，同步到VSCode方便Nginx的配置与处理
 cp /etc/nginx/nginx.conf /home/xianyu-api/
```

9.Nginx配置文件修改

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request"
'
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 2048;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    server {
        listen       80 default_server;
        server_name  localhost;

        # 反向代理转发
        location / {
          proxy_pass http://127.0.0.1:3001; # 不要忘记分号
        }

        error_page 404 /404.html;
            location = /40x.html {
```

```
        }

        error_page 500 502 503 504 /50x.html;
            location = /50x.html {
        }
    }
}
mongodb://sa:sa@localhost:27027/xianyu
```

10.Nginx配置文件反向覆盖并且重启，并不重要，会vim可以直接操作

```
cp /home/xianyu-api/nginx.conf /etc/nginx/nginx.conf
并且点击确定覆盖操作
重启Nginx
systemctl restart nginx
// 如果Nginx报错，有可能是配置缺少分号，请注意
```

11.在Centos上面安装MongoDB

```
// 输入这个命令，也就是告诉系统MongoDB安装的源在哪里
vi /etc/yum.repos.d/mongodb-org-4.2.repo

// 在上面创建的文件中输入下面的内容，注意最前面的文字可能会被粘贴缺失掉，检查一遍
[mongodb-org-4.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-
org/4.2/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.2.asc
// 按住ESC按键退出编辑，按住冒号之后输入wq退出vim
// 最后输入下面的命令执行安装MongoDB
yum install mongodb-org -y
// 安装完成之后，通过下面的命令找到MongoDB的配置文件
whereis mongod
打开配置文件
将配置文件最后的bindIp：127.0.0.1 ----> 修改为0.0.0.0，这样才能作为服务器数据库被外网访问
// 重启MongoDB的服务
```

```
service mongod start
// 进入MongoDB数据库
mongo
// 展示数据库的表与文档
show dbs
启动：systemctl start mongod
重启：systemctl restart mongod
关闭：systemctl stop mongod
查看运行状态:systemctl status mongod
// 查看MongoDB是否开启
systemctl status mongod
// 查看MongoDB的日志
more /var/log/mongodb/mongod.log

// 删除因为Linux系统非正常退出产生的临时文件
rm *.swp



// mongodb的卸载
sudo yum erase $(rpm -qa | grep mongodb-org)     #卸载MongoDB
sudo rm -r /var/log/mongodb      #删除日志文件
sudo rm -r /var/lib/mongo           #删除数据文件
```

- 验证MongoDB的安装效果 rpm -qa |grep mongodb ， rpm -ql mongodb-org-server
- 启动/usr/bin/mongod --config /etc/mongod.conf 停止/usr/bin/mongod --config /etc/mongod.conf --shutdown
- 查看mongodb的状态 netstat -nltp|grep mongo

12.域名解析

添加域名至解析列表后，还需要去设置解析记录才可以正常使用解析服务

```
主机记录：www
记录类型：CNAME
线路类型：默认
记录值：cghbh.com
```

13.安装Redis

```
yum install redis -y
// 修改/etc/redis.conf的配置文件
找到redis.conf 并修改 daemonize no（第128行） 为 daemonize yes
开启客户端要确保服务端启动，重启Redis服务
redis-server /etc/redis.conf
```

14.MongoDB新的安装方式，方便后面的权限控制操作

- 在根目录下操作

```
// 下载MongoDB文件
wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel80-4.4.1.tgz
// 解压文件
tar -zxvf mongodb-linux-x86_64-rhel80-4.4.1.tgz
// 迁移文件
mv ./mongodb-linux-x86_64-rhel80-4.4.1  /usr/local/mongodb
```

- 在/usr/local/mongodb文件夹下面操作

```
// 创建db目录和日志文件
mkdir -p ./data/db
mkdir -p ./logs
touch ./logs/mongodb.log
// 创建mongodb.conf配置文件
vim mongodb.conf
// 将下面的内容粘贴


#端口号（如果是企业使用尽量不要用默认端口）
port=27027
#db目录
dbpath=/usr/local/mongodb/data/db
#日志目录
logpath=/usr/local/mongodb/logs/mongodb.log
#后台
fork=true
#日志输出
```

```
logappend=true
auth=true
#允许远程IP连接
bind_ip=127.0.0.1
```

- 启动数据库
  ./bin/mongod --config mongodb.conf
- 连接数据库

  ./bin/mongo --port 27027端口号打开方式

14，端口安全扫描修改配置

- 尽量不要使用0-1024的端口，因为系统一般使用的都在这些范围内，避免和系统权限发生冲突

15.mongodb数据库的删除操作

```
//显示数据库
show dbs
//切换数据库
use xianyu
// 执行删除库的命令
db.dropDatebase()
// 删除数据库中的集合
db.集合名.drop()
```

开发部署问题记录

1.在Vue-Router的history部署的时候，刷新非首页的页面出现 Cannot GET/ 的问题

因为 单页应用通常只有一个浏览器可以访问的index.html。但是如果使用history模式，用户直接访问或者刷新非index.html时，例如访问'/list'，web服务器会绕过index.html，去'/list'位置找相应的页面，这样就会导致404。而connect-history-api-fallback中间件，会把访问地址改成'/'，然后vue-router渲染对应的list组件 。

解决办法：在服务端部署的时候，使用express部署，安装connect-history-api-fallback中间件

```
const express = require('express')
const app = express()
const history = require('connect-history-api-fallback')
app.use(history())

app.listen(3000)
```

2.关于短信验证码的使用问题

- 配置Redis

```
// ../config/redis.js'
const redis = require('redis')
const { promisifyAll } = require('bluebird')

const options = {
  host: '127.0.0.1',
  port: 6379,
  detect_buffers: true,
  retry_strategy: function (options) {
    if (options.error && options.error.code === 'ECONNREFUSED') {
      return new Error('The server refused the connection');
    }
    if (options.total_retry_time > 1000 * 60 * 60) {
      return new Error('Retry time exhausted');
    }
    if (options.attempt > 10) {
      return undefined;
    }
    return Math.min(options.attempt * 100, 3000);
  }
}

const client = promisifyAll(redis.createClient(options))

// 连接Redis
client.on('error', (err) => {
  console.log('Redis Client Error:' + err)
})
```

```
// 设置值
const setValue = (key, value, time) => {
  if (typeof value === 'undefined' || value == null || value === '') {
    return
  }
  if (typeof value === 'string') {
    if (typeof time !== 'undefined') {
      // EX过期时间为秒级，PX为毫秒级
      client.set(key, value, 'EX', time)
    } else {
      client.set(key, value)
    }
  } else if (typeof value === 'object') {
    Object.keys(value).forEach((item) => {
      client.hset(key, item, value[item], redis.print)
    })
  }
}

// 获取值
const getValue = (key) => {
  return client.getAsync(key)
}

const getHValue = (key) => {
  return client.hgetallAsync(key)
}

const delValue = (key) => {
  client.del(key, (err, res) => {
    if (res === 1) {
      console.log('delete successfully');
    } else {
      console.log('delete redis key error:' + err)
    }
  })
}

module.exports = {
  client,
  setValue,
```

```
  getValue,
  getHValue,
  delValue
}
```

- 产生验证码

```
const { setValue, getValue } = require('../config/redis.js')
const randomCode = Math.floor(Math.random() * 899999) + 100000
const { telephone } = ctx.request.body
const client = new Core({
  accessKeyId: 'accessKeyId',
  accessKeySecret: 'accessKeySecret',
  endpoint: 'https://dysmsapi.aliyuncs.com',
  apiVersion: '2017-05-25'
})

const params = {
  "RegionId": "cn-hangzhou",
  "PhoneNumbers": telephone,
  "SignName": "闲语",
  "TemplateCode": "SMS_205611827",
  "TemplateParam": `{ \"code\": ${randomCode} }`
}

const requestOption = { method: 'POST' }
 try {
  const result = await client.request('SendSms', params, requestOption)
  const {Code }= result
  if (Code === 'OK') {
    // 验证码发送成功之后，将当前telephone存储到redis中
    setValue(`${telephone}`, `${randomCode}`, 10 * 60)
    ctx.body = {
      errno: 0,
      message: '验证码发送成功'
    }
  } else {
      ctx.body = {
        errno: 1,
        message: '验证码发送失败'
      }
    }
```

```
  } catch (err) {
    ctx.body = {
      errno: 1,
      message: '验证码发送失败'
    }
  }
```