微信公众号课程

尚硅谷前端研究院

版本：V 1.0

第 1 章：微信公众号简介

1.1 微信公众号是什么

微信公众平台,是给个人、企业和组织提供业务服务与用户管理能力的全新服务平台。

简单来说，一个提供服务平台。

1.2 微信公众号的分类

1.2.1 订阅号

1.简介

为媒体和个人提供一种新的信息传播方式，主要功能是在微信侧给用户传达资讯；（功能类似报纸杂志，提供新闻信息或娱乐趣事）

2.适用主要人群

个人、媒体。

3.群发次数

订阅号（认证用户、非认证用户）1 天内可群发 1 条消息。

1.2.3 服务号

1.简介

为企业和组织提供更强大的业务服务与用户管理能力，主要偏向服务类交互（功能类似银行，12315，114 等）

2.适用主要人群

企业、政府或其他组织。

3.群发次数

服务号 1 个月（按自然月）内可发送 4 条群发消息。

1.2.4 小程序

小程序是一种新的开放能力，开发者可以快速地开发一个小程序。小程序可以在微信内被便捷地获取和传播，同时具有出色的使用体验。

1.2.5 企业微信（原企业号）

企业微信继承企业号所有能力，同时为企业提供专业的通讯工具、丰富的办公应用与 API，助力企业高效沟通与办公。

1.3 订阅号和服务号的主要区别

1. 推送频率

订阅号：1 天内可群发 1 条消息。

服务号：1 个月内可发送 4 条群发消息。

2. 提供功能

订阅号：包含大部分功能。

服务号：认证的服务号包含全部功能。

3. 适用人群

订阅号：个人、媒体。

服务号：企业、政府或其他组织。

1.4 注册微信公众号

1.3.1 注册网址

官网：https://mp.weixin.qq.com/

1.3.2 注册流程

1) 打开官网，点击右上角的立即注册

2) 选择订阅号注册

3) 依次输入要求的信息，勾上我同意，点击注册

4) 选择中国内地，点击确定

5) 选择订阅号，确定

6) 主体类型选择个人，填写好信息后点击继续

7) 填写好如图所示信息，点击完成。需要注意的是，名称如有敏感词审核可能不过关，或者需要时间审核。

1.5 使用微信号

登录后，页面左边将会有一排功能选项，订阅号中一共有 5 个功能可以直接设置使用。

第 2 章：微信公众号的开发

2.1 验证消息的合法性

2.1.1 填写服务器配置

1) 开发/开发者工具 ==> 开发者文档 ==> 开始开发/接口测试号申请 ==> 进入微信公众帐号测试号申请系统

2) 填写好 URL 和 Token，点击提交

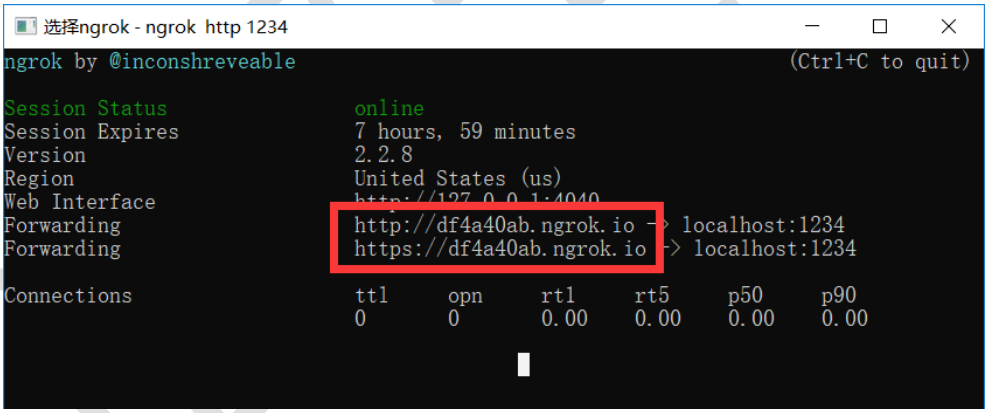
补充：ngrok 内网穿透

1. 下载 ngrok 客户端

2. 本地启动服务器，如 http://localhost:1234，端口号为 1234

3. 打开 ngrok 客户端，输入 ngrok http 1234，启动内网穿透

4. 复制生成后的网址



2.1.2 验证服务器地址的有效性

1）将 token、timestamp、nonce 三个参数进行字典序排序

2）将三个参数字符串拼接成一个字符串进行 sha1 加密

3）开发者获得加密后的字符串可与 signature 对比，标识该请求来源于微信

```
// index.js：
const express = require('express')
```

```
const sha1 = require('sha1')


const app = express()


const config = {
    wechat: {
        appId: 'wxc8e92f7ab70fbca0',
        appsecret: 'b4054e90b75787c78e0af50bf7fc3e87',
        token: 'atguigu0925'
    }
}


app.use(function (req, res, next) {
    console.log(req.query)
    /*
    signature: 微信的加密签名，结合你的 token，timestamp 和 nonce 经过某种算法生成的
    echostr: 随机字符串，微信后台随机生成的
    timestamp: 时间戳，对应当前时间
    nonce: 随机数，微信后台随机生成的


    1. 确认消息发送自微信后台
        - 结合你的 token，timestamp 和 nonce 根据某种算法计算得出的字符串是否与 signature 相等。
            - 相等：下一步
            - 不相等：返回错误信息
        - 具体怎么做的？某种算法是什么呢？
```

```
        - 第一步：将三个字符串按字典序排序
          - 123...abcdefg....    对应方法 arr.sort()
        - 第二步：将三个排序后的字符串拼接在一起，用 sha1 算法加密
        - 第三步：将加密后的字符串与 signature 进行对比，相等下一步，不等报错
    2. 返回 echostr 给微信后台，微信后台才会确认开发者的合法身份
        - 将 echostr 返回给微信后台
    */

    const token = config.wechat.token
    const signature = req.query.signature
    const nonce = req.query.nonce
    const timestamp = req.query.timestamp
    const echostr = req.query.echostr

    // const str = [token, timestamp, nonce]
    // console.log('排序前：' + str)
    // str = str.sort()
    // console.log('排序后：' + str)
    // str = str.join('')
    // console.log('拼接后：' + str)
    // const sha = sha1(str)
    // console.log('加密后：' + sha)

    const str = [token, timestamp, nonce].sort().join('')
    const sha = sha1(str)
```

```
    if (sha === signature) {
        res.send(echostr)
    } else {
        res.send('wrong')
    }
})

app.listen(1234, function () {
    console.log('服务器启动成功')
})
```

2.1.3 依据接口文档实现业务逻辑

验证 URL 有效性成功后即接入生效，成为开发者。你可以在公众平台网站中申请微信认证，认证成功后，将获得更多接口权限，满足更多业务需求。

2.2 获取 access_token

access_token 是公众号的全局唯一接口调用凭据，公众号调用各接口时都需使用 access_token。开发者需要进行妥善保存。

2.2.1 目录结构

```
├── config/              # 配置目录
|    ├── index.js        # 存储配置信息
├─── wechat/            # 核心功能库
|    ├── accessToken.txt  # 存储 access_token
|    ├── auth.js         # 验证服务器功能
|    ├── wechat.js       # 类 wechat
```

```
├── index.js          # 入口启动文件
├── package.json      # 配置文件
```

2.2.2 代码

1) index.js

```
/*
  此模块用来储存关键的配置信息
 */
module.exports = {
  appID: 'wxc8e92f7ab70fbca0',
  appsecret: 'b4054e90b75787c78e0af50bf7fc3e87',
  token: 'atguiguHTML1208'   //token 要严格保密！！！
}
```

2) auth.js

```
/*
  验证服务器的有效性:

    1、填写服务器配置(测试号管理页面)
      - URL 开发者服务器地址（保证能在互联网中能访问）
        通过 ngrok http 端口号 就得到一个网址
      - Token 参与微信签名的加密
    2、验证服务器地址的有效性
      - 将 timestamp、nonce、token 三个参数按照字典序排序
      - 将三个参数拼接在一起，进行 sha1 加密
      - 将加密后生成字符串和微信签名进行对比，
          如果相同说明成功，返回一个 echostr 给微信服务器，
          如果不相同，说明签名算法出了问题，配置不成功
 */
//引入配置对象
const config = require('../config');
//引入 sha1 加密模块
```

```
const sha1 = require('sha1');

module.exports = () => {

  return (req, res, next) => {
    //接受微信服务器发送过来的请求参数
    console.log(req.query);
    /*
      { signature: 'c4409bdd012bf28d8b4aabf7ac5847c5560d6cf0',    微信的加密签名（timestamp、nonce、token）
        echostr: '11283286178012191741',   随机字符串
        timestamp: '1529977721',            时间戳
        nonce: '1462949582' }               随机数字
     */
    //获取参与加密的参数
    const {signature, echostr, timestamp, nonce} = req.query;
    const {token} = config;
    /*// - 将 timestamp、nonce、token 三个参数按照字典序排序
    const arr = [timestamp, nonce, token].sort();
    // - 将三个参数拼接在一起，进行 sha1 加密
    const str = arr.join('');
    const sha1Str = sha1(str);*/
    //简写方式
    const sha1Str = sha1([timestamp, nonce, token].sort().join(''));
    // - 将加密后生成字符串和微信签名进行对比，
    if (sha1Str === signature) {
      //说明成功，返回 echostr 给微信服务器
      res.send(echostr);
    } else {
      //说明失败
      res.send('');
    }
  }
}
```

3) wechat.js

```
/*
  获取 access_token:
    全局唯一的接口调用凭据，今后使用微信的接口基本上都需要携带上这个参数
```

```
    2 小时需要更新一次，提前 5 分钟刷新

    请求地址：

https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID
&secret=APPSECRET
    请求方式：
      GET

    设计思路：
      首先发送请求获取凭据，保存为一个唯一的文件
      然后后面请求先去本地文件读取凭据
        判断凭据是否过期
          如果没有过期，直接使用
          如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件

    总结：
      先去本地查看有没有指定文件（readAccessToken）
        如果有（之前请求过凭据）
          判断凭据是否过期(isValidAccessToken)
            如果没有过期，直接使用
            如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件
(getAccessToken、saveAccessToken)
        如果没有（之前都没有请求过凭据）
          发送请求获取凭据，保存为一个唯一的文件

 */
//引入配置对象
const {appID, appsecret} = require('../config');
//引入发送 http 请求的库
const rp = require('request-promise-native');
//引入 fs 模块
const {readFile, writeFile} = require('fs');

class Wechat {

  getAccessToken () {
    //定义请求地址
    const url =
`https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=${
appID}&secret=${appsecret}`;
```

```
    /*
      问题：需要将回调函数中的数据返回出去？
      解决：用 promise 解决

      所有的异步操作，都应该包装一层 promise，让这个异步操作执行完毕之后，再去执行后面的代码
      简化： 所有的异步操作，都应该包装一层 promise
     */
    return new Promise((resolve, reject) => {
      //发送 http 请求
      //下载 request-promise-native   request
      rp({method: 'GET', json: true, url})
        .then(res => {
          //请求成功的状态
          // console.log(res);
          //重新赋值凭据的过期时间 ：  当前时间 + (7200 - 5 分钟) * 1000
          res.expires_in = Date.now() + (res.expires_in - 300) * 1000;
          // console.log(res);
          resolve(res);
        })
        .catch(err => {
          //请求失败
          reject('getAccessToken 方法出了问题：' + err);
        })
    })
  }
  saveAccessToken (data) {
    /*
      问题：writeFile 方法会将对象转化为字符串
      解决：我将对象转化为 json 字符串
     */
    data = JSON.stringify(data);
    return new Promise((resolve, reject) => {
      //将凭据保存为一个文件
      writeFile('accessToken.txt', data, err => {
        if (!err) {
          //写入成功
          resolve();
        } else {
          //写入失败
          reject('saveAccessToken 方法出了问题：' + err);
```

```
            }
        })
    })
}
readAccessToken () {
    return new Promise((resolve, reject) => {
        //将凭据读取出来
        readFile('accessToken.txt', (err, data) => {
            if (!err) {
                //将读取的 Buffer 数据转化为 json 字符串
                data = data.toString();
                //将 json 字符串转化为对象
                data = JSON.parse(data);
                //读取成功
                resolve(data);
            } else {
                //读取失败
                reject('readAccessToken 方法出了问题：' + err);
            }
        })
    })
}
isValidAccessToken (data) {
    /*
        判断凭据是否过期
            true    凭据没有过期
            false   凭据过期了
     */
    //过滤非法的数据
    if (!data || !data.access_token || !data.expires_in) return false;
    //判断凭据是否过期
    /*if (data.expires_in > Date.now()) {
        //如果凭据的过期时间大于当前时间，说明没有过期
        return true
    } else {
        //如果凭据的过期时间小于当前时间，说明过期了
        return false
    }*/
    //简写方式
    return data.expires_in > Date.now();
}
```

```
    fetchAccessToken () {
        //优化操作,优化不去执行读取文件操作
        if (this.access_token && this.expires_in && this.isValidAccessToken(this)) {
            //说明 this 有凭据和过期时间，并且凭据未过期
            return Promise.resolve({access_token: this.access_token, expires_in:
this.expires_in});
        }

        return this.readAccessToken()
            .then(async res => {
                //判断凭据是否过期(isValidAccessToken)
                if (this.isValidAccessToken(res)) {
                    //没有过期，直接使用
                    return Promise.resolve(res);
                } else {
                    //重新发送请求获取凭据
                    const data = await this.getAccessToken();
                    //保存下来
                    await this.saveAccessToken(data);
                    //将请求回来的凭据返回出去
                    return Promise.resolve(data);
                }
            })
            .catch(async err => {
                console.log(err);
                //重新发送请求获取凭据
                const data = await this.getAccessToken();
                //保存下来
                await this.saveAccessToken(data);
                //将请求回来的凭据返回出去
                return Promise.resolve(data);
            })
            .then(res => {
                //将其请求回来的凭据和过期时间挂载到 this 上
                this.access_token = res.access_token;
                this.expires_in = res.expires_in;
                //指定 fetchAccessToken 方法返回值
                return Promise.resolve(res);
            })
    }
}
```

4) index.js

```
const express = require('express');
const auth = require('./wechat/auth');
const app = express();

//接受微信服务器发送过来的请求 GET
//应用中间级，能够接受处理所有请求
app.use(auth());

app.listen(3000, err => {
  if (!err) console.log('服务器启动成功了~~~');
})
```

2.3 自动回复消息

2.3.1 目录结构

```
├── config/              # 配置目录
|   ├── index.js         # 存储配置信息
├── libs/                # 工具方法库
|   ├── utils.js         # 解析字符串的工具方法
├── wechat/              # 核心功能库
|   ├── auth.js          # 验证服务器和回复用户消息功能
|   ├── reply.js         # 处理返回消息功能
|   ├── template.js      # 回复消息模板文件
|   ├── wechat.js        # 类 wechat
├── index.js             # 入口启动文件
├── package.json         # 配置文件
```

2.3.2 代码

1) index.js

```
/*
   此模块用来储存关键的配置信息
 */
module.exports = {
   appID: 'wxc8e92f7ab70fbca0',
   appsecret: 'b4054e90b75787c78e0af50bf7fc3e87',
   token: 'atguiguHTML1208'   //token 要严格保密！！！
}
```

2) utils.js

```
/*
   工具函数
 */
//引入解析 xml 数据的库
const {parseString} = require('xml2js');

module.exports = {
   getUserDataAsync (req) {
      /*
         用户数据是通过流的方式发送，通过绑定 data 事件接受数据
       */
      return new Promise((resolve, reject) => {
         let data = '';
         req
            .on('data', userData => {
               //将流式数据全部拼接起来
               data += userData;
            })
            .on('end', () => {
               //确保数据全部获取了
               resolve(data);
            })
```

```
        })
    },
    parseXMLAsync (xmlData) {
        return new Promise((resolve, reject) => {
            parseString(xmlData, {trim: true}, (err, data) => {
                if (!err) {
                    //解析成功了
                    resolve(data);
                } else {
                    //解析失败了
                    reject('parseXMLAsync 方法出了问题：' + err);
                }
            })
        })
    },
    formatMessage (jsData) {
        const data = jsData.xml;
        //初始化一个空的对象
        let message = {};
        //判断数据是一个合法的数据
        if (typeof data === 'object') {
            //循环遍历对象中的所有数据
            for (let key in data) {
                //获取属性值
                let value = data[key];
                //过滤掉空的数据和空的数组
                if (Array.isArray(value) && value.length > 0) {
                    //在新对象中添加属性和值
                    message[key] = value[0];
                }
            }
        }
        //将格式化后的数据返回出去
        return message;
    }
}
```

3) auth.js

```
/*
```

```
  验证服务器的有效性:

    1、填写服务器配置(测试号管理页面)
      - URL 开发者服务器地址（保证能在互联网中能访问）
        通过 ngrok http 端口号 就得到一个网址
      - Token 参与微信签名的加密
    2、验证服务器地址的有效性
      - 将 timestamp、nonce、token 三个参数按照字典序排序
      - 将三个参数拼接在一起，进行 sha1 加密
      - 将加密后生成字符串和微信签名进行对比，
          如果相同说明成功，返回一个 echostr 给微信服务器，
          如果不相同，说明签名算法出了问题，配置不成功
 */
//引入配置对象
const config = require('../config');
//引入 sha1 加密模块
const sha1 = require('sha1');
//引入工具函数
const {getUserDataAsync, parseXMLAsync, formatMessage} = require('../libs/utils');
//引入 reply 模块
const reply = require('./reply');

module.exports = () => {

  return async (req, res, next) => {
    //接受微信服务器发送过来的请求参数
    // console.log(req.query);
    /*
      { signature: 'c4409bdd012bf28d8b4aabf7ac5847c5560d6cf0',    微信的加密
签名（timestamp、nonce、token）
        echostr: '11283286178012191741',   随机字符串
        timestamp: '1529977721',            时间戳
        nonce: '1462949582' }               随机数字
     */
    //获取参与加密的参数
    const {signature, echostr, timestamp, nonce} = req.query;
    const {token} = config;
    /*// - 将 timestamp、nonce、token 三个参数按照字典序排序
    const arr = [timestamp, nonce, token].sort();
    // - 将三个参数拼接在一起，进行 sha1 加密
    const str = arr.join('');
```

```
    const sha1Str = sha1(str);*/
    //简写方式
    const sha1Str = sha1([timestamp, nonce, token].sort().join(''));

    /*
      微信服务器会主动发送两种方法的消息
        GET 请求，  验证服务器有效性
        POST 请求，微信服务器会将用户发送过来的消息转发到开发者服务器
上
     */
    if (req.method === 'GET') {
      // - 将加密后生成字符串和微信签名进行对比，
      if (sha1Str === signature) {
        //说明成功，返回 echostr 给微信服务器
        res.send(echostr);
      } else {
        //说明失败
        res.send('');
      }
    } else if (req.method === 'POST') {
      //接受用户发送过来消息
      // console.log(req.query);
      /*
        { signature: 'c67250097842aa50990259fa3df052eeffcb1cee',
          timestamp: '1530000513',
          nonce: '53405765',
          openid: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' }   //用户的 id
       */
      //验证消息是否来自于微信服务器
      if (sha1Str !== signature) {
        //说明消息不是来自于微信服务器
        //过滤掉非法请求
        res.send('error');
        return
      }

      //获取用户的消息，返回的数据格式是 xml
      const xmlData = await getUserDataAsync(req);
      // console.log(xmlData);
      /*
        <xml>
```

```
            <ToUserName><![CDATA[gh_4fe7faab4d6c]]></ToUserName>     //开发者的 id

<FromUserName><![CDATA[oAsoR1iP-_D3LZIwNCnK8BFotmJc]]></FromUserName>
//用户的 openid
            <CreateTime>1530001191</CreateTime>     //消息发送时间
            <MsgType><![CDATA[text]]></MsgType>     //消息的类型
            <Content><![CDATA[666]]></Content>      //消息的具体内容
            <MsgId>6571305078611302153</MsgId>      //消息的 id
          </xml>
          */
        //将 xml 解析成 js 对象
        const jsData = await parseXMLAsync(xmlData);
        // console.log(jsData);
        /*
          { xml:
           { ToUserName: [ 'gh_4fe7faab4d6c' ],
             FromUserName: [ 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' ],
             CreateTime: [ '1530001675' ],
             MsgType: [ 'text' ],
             Content: [ '774' ],
             MsgId: [ '6571307157375473517' ] } }
          */
        //格式化数据
        const message = formatMessage(jsData);
        console.log(message);
        /*
          { ToUserName: 'gh_4fe7faab4d6c',
            FromUserName: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc',
            CreateTime: '1530002262',
            MsgType: 'text',
            Content: '888',
            MsgId: '6571309678521276386' }
          */

        //返回用户消息
        /*
          1. 假如服务器无法保证在五秒内处理并回复
          2. 回复 xml 数据中有多余的空格 *****
          3. 回复文本内容，中 options.content='' ***
          如果有以上现象，就会导致微信客户端中的报错：
```

```
            '该公众号提供服务出现故障，请稍后再试'
          */
        //设置回复用户消息的具体内容
        const replyMessage = await reply(message);
        console.log(replyMessage);   //Promise { <pending> }
        //返回响应给微信服务器
        res.send(replyMessage);

        /*//先返回一个空的响应给微信服务器
        res.send('');*/
      }
    }
}
```

4) reply.js

```
/*
   处理并分析用户发送的消息
   决定返回什么消息给用户
 */
const template = require('./template');

module.exports = async message => {

  //定义 options
  let options = {
    toUserName: message.FromUserName,
    fromUserName: message.ToUserName,
    createTime: Date.now(),
    msgType: 'text'
  }

  //设置回复用户消息的具体内容
  let content = '';

  //判断用户发送消息的类型和内容，决定返回什么消息给用户
  if (message.MsgType === 'text') {
    if (message.Content === '1') {
      content = '大吉大利，今晚吃鸡';
    } else if (message.Content === '2') {
```

```
            content = '落地成盒';
        } else if (message.Content === '3') {
            //回复图文消息
            content = [{
                title: 'Nodejs 开发',
                description: '微信公众号开发',
                picUrl:
'https://ss1.baidu.com/6ONXsjip0QIZ8tyhnq/it/u=1841004364,244945169&fm=58
&bpow=121&bpoh=75',
                url: 'http://nodejs.cn/'
            }, {
                title: 'web 前端',
                description: '这里有最新、最强的技术',
                picUrl:
'https://ss0.baidu.com/6ONWsjip0QIZ8tyhnq/it/u=1981851186,10620031&fm=58
&s=6183FE1ECDA569015C69A554030010F3&bpow=121&bpoh=75',
                url: 'http://www.atguigu.com/'
            }];
            options.msgType = 'news';

        } else if (message.Content.match('爱')) {
            //模糊匹配，只要包含爱
            content = '我爱你~';
        } else {
            content = '您在说啥，我听不懂';
        }
    } else if (message.MsgType === 'image') {
        content = '您的图片地址为：' + message.PicUrl;
    } else if (message.MsgType === 'voice') {
        content = '语音识别结果：' + message.Recognition;
    } else if (message.MsgType === 'video') {
        content = '接受了视频消息';
    } else if (message.MsgType === 'shortvideo') {
        content = '接受了小视频消息';
    } else if (message.MsgType === 'location') {
        content = '纬度：' + message.Location_X + '经度：' + message.Location_Y
            + '缩放大小：' + message.Scale + '详情：' + message.Label;
    } else if (message.MsgType === 'link') {
        content = '标题：' + message.Title + '描述：' + message.Description + '网址：
' + message.Url;
    } else if (message.MsgType === 'event') {
```

```
    if (message.Event === 'subscribe') {
      //用户订阅事件
      content = '欢迎您的订阅~';
      if (message.EventKey) {
        //扫描带参数的二维码的订阅事件
        content = '欢迎您扫二维码的关注';
      }
    } else if (message.Event === 'SCAN') {
      //已经关注了公众号，在扫描带参数二维码进入公众号
      content = '已经关注了公众号，在扫描带参数二维码进入公众号';
    } else if (message.Event === 'unsubscribe') {
      //用户取消关注
      console.log('无情取关~');
    } else if (message.Event === 'LOCATION') {
      //用户进行会话时，上报一次地理位置消息
      content = '纬度：' + message.Latitude + '  经度：' + message.Longitude + '  精
度：' + message.Precision;
    } else if (message.Event === 'CLICK') {
      content = '点击了菜单~~~';
    } else if (message.Event === 'VIEW') {
      //用户点击菜单，跳转到其他链接
      console.log('用户点击菜单，跳转到其他链接');
    }
  }

  //将最终回复消息内容添加到 options 中
  options.content = content;
  //将最终的 xml 数据返回出去
  return template(options);
}
```

5) template.js

```
/*
  设置回复用户的 6 种消息内容
 */

module.exports = options => {

  //回复用户消息
```

```
  let replyMessage = '<xml>' +
    '<ToUserName><![CDATA[' + options.toUserName + ']]></ToUserName>' +
    '<FromUserName><![CDATA[' + options.fromUserName +
']]></FromUserName>' +
    '<CreateTime>' + options.createTime + '</CreateTime>' +
    '<MsgType><![CDATA[' + options.msgType + ']]></MsgType>';

  if (options.msgType === 'text') {
    replyMessage += '<Content><![CDATA[' + options.content + ']]></Content>';
  } else if (options.msgType === 'image') {
    replyMessage += '<Image><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Image>';
  } else if (options.msgType === 'voice') {
    replyMessage += '<Voice><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Voice>';
  } else if (options.msgType === 'video') {
    replyMessage += '<Video>' +
      '<MediaId><![CDATA[' + options.mediaId + ']]></MediaId>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '</Video>';
  } else if (options.msgType === 'music') {
    replyMessage += '<Music>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '<MusicUrl><![CDATA[' + options.musicUrl + ']]></MusicUrl>' +
      '<HQMusicUrl><![CDATA[' + options.hqMusicUrl + ']]></HQMusicUrl>' +
      '<ThumbMediaId><![CDATA[' + options.mediaId + ']]></ThumbMediaId>' +
      '</Music>';
  } else if (options.msgType === 'news') {
    replyMessage += '<ArticleCount>' + options.content.length + '</ArticleCount>'
+
      '<Articles>';

    options.content.forEach(item => {
      replyMessage += '<item>' +
        '<Title><![CDATA[' + item.title + ']]></Title>' +
        '<Description><![CDATA[' + item.description + ']]></Description>' +
        '<PicUrl><![CDATA[' + item.picUrl + ']]></PicUrl>' +
        '<Url><![CDATA[' + item.url + ']]></Url>' +
        '</item>';
```

```
        })

        replyMessage += '</Articles>';
    }

    replyMessage += '</xml>';
    //将拼接好回复用户的数据返回出去
    return replyMessage;

}
```

5) wechat.js

```
/*
    获取 access_token:
        全局唯一的接口调用凭据，今后使用微信的接口基本上都需要携带上这个参数
        2 小时需要更新一次，提前 5 分钟刷新

        请求地址：

https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
        请求方式：
            GET

        设计思路：
            首先发送请求获取凭据，保存为一个唯一的文件
            然后后面请求先去本地文件读取凭据
                判断凭据是否过期
                    如果没有过期，直接使用
                    如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件

        总结：
            先去本地查看有没有指定文件（readAccessToken）
                如果有（之前请求过凭据）
                    判断凭据是否过期(isValidAccessToken)
                        如果没有过期，直接使用
                        如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件(getAccessToken、saveAccessToken)
```

```
        如果没有（之前都没有请求过凭据）
         发送请求获取凭据，保存为一个唯一的文件

 */
//引入配置对象
const {appID, appsecret} = require('../config');
//引入发送 http 请求的库
const rp = require('request-promise-native');
const request = require('request');
//引入 fs 模块
const {readFile, writeFile, createReadStream, createWriteStream} = require('fs');

class Wechat {
  getAccessToken () {
    //定义请求地址
    const url =
`https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=${
appID}&secret=${appsecret}`;
    /*
      问题：需要将回调函数中的数据返回出去？
      解决：用 promise 解决

      所有的异步操作，都应该包装一层 promise，让这个异步操作执行完毕之
后，再去执行后面的代码
      简化： 所有的异步操作，都应该包装一层 promise
     */
    return new Promise((resolve, reject) => {
      //发送 http 请求
      //下载 request-promise-native   request
      rp({method: 'GET', json: true, url})
        .then(res => {
          //请求成功的状态
          // console.log(res);
          //重新赋值凭据的过期时间 ：  当前时间 + (7200 - 5 分钟) * 1000
          res.expires_in = Date.now() + (res.expires_in - 300) * 1000;
          // console.log(res);
          resolve(res);
        })
        .catch(err => {
          //请求失败
          reject('getAccessToken 方法出了问题：' + err);
```

```
            })
        })
    }
    saveAccessToken (data) {
        /*
            问题：writeFile 方法会将对象转化为字符串
            解决：我将对象转化为 json 字符串
         */
        data = JSON.stringify(data);
        return new Promise((resolve, reject) => {
            //将凭据保存为一个文件
            writeFile('accessToken.txt', data, err => {
                if (!err) {
                    //写入成功
                    resolve();
                } else {
                    //写入失败
                    reject('saveAccessToken 方法出了问题：' + err);
                }
            })
        })
    }
    readAccessToken () {
        return new Promise((resolve, reject) => {
            //将凭据读取出来
            readFile('accessToken.txt', (err, data) => {
                if (!err) {
                    //将读取的 Buffer 数据转化为 json 字符串
                    data = data.toString();
                    //将 json 字符串转化为对象
                    data = JSON.parse(data);
                    //读取成功
                    resolve(data);
                } else {
                    //读取失败
                    reject('readAccessToken 方法出了问题：' + err);
                }
            })
        })
    }
    isValidAccessToken (data) {
```

```
    /*
      判断凭据是否过期
        true    凭据没有过期
        false   凭据过期了
     */
    //过滤非法的数据
    if (!data || !data.access_token || !data.expires_in) return false;
    //判断凭据是否过期
    /*if (data.expires_in > Date.now()) {
      //如果凭据的过期时间大于当前时间，说明没有过期
      return true
    } else {
      //如果凭据的过期时间小于当前时间，说明过期了
      return false
    }*/
    //简写方式
    return data.expires_in > Date.now();
  }
  fetchAccessToken () {
    //优化操作,优化不去执行读取文件操作
    if (this.access_token && this.expires_in && this.isValidAccessToken(this)) {
      //说明 this 有凭据和过期时间，并且凭据未过期
      return Promise.resolve({access_token: this.access_token, expires_in:
this.expires_in});
    }

    return this.readAccessToken()
      .then(async res => {
        //判断凭据是否过期(isValidAccessToken)
        if (this.isValidAccessToken(res)) {
          //没有过期，直接使用
          return Promise.resolve(res);
        } else {
          //重新发送请求获取凭据
          const data = await this.getAccessToken();
          //保存下来
          await this.saveAccessToken(data);
          //将请求回来的凭据返回出去
          return Promise.resolve(data);
        }
      })
```

```
        .catch(async err => {
            console.log(err);
            //重新发送请求获取凭据
            const data = await this.getAccessToken();
            //保存下来
            await this.saveAccessToken(data);
            //将请求回来的凭据返回出去
            return Promise.resolve(data);
        })
        .then(res => {
            //将其请求回来的凭据和过期时间挂载到 this 上
            this.access_token = res.access_token;
            this.expires_in = res.expires_in;
            //指定 fetchAccessToken 方法返回值
            return Promise.resolve(res);
        })
    }
}

module.exports = Wechat;
```

6) index.js

```
const express = require('express');
const auth = require('./wechat/auth');
const app = express();

//接受微信服务器发送过来的请求 GET
//应用中间级，能够接受处理所有请求
app.use(auth());

app.listen(3000, err => {
    if (!err) console.log('服务器启动成功了~~~');
})
```

2.4 上传素材

2.4.1 目录结构

```
├── config/              # 配置目录
|     ├── index.js       # 存储配置信息
├── libs/                # 工具方法库
|     ├── util.js        # 解析字符串的工具方法
|     ├── api.js         # 定义接口的文件
├─── wechat/            # 核心功能库
|     ├── 1.jpg          # 需要上传素材图片
|     ├── 2.mp4          # 需要上传素材视频
|     ├── auth.js        # 回复消息功能
|     ├── reply.js       # 处理用户发送的信息
|     ├── template.js    # 返回给用户的信息模板
|     ├── wechat.js      # 类 Wechat
├─── app.js              # 入口启动文件
├─── package.json        # 配置文件
```

2.4.2 代码

1) index.js

```
/*
  此模块用来储存关键的配置信息
 */
module.exports = {
  appID: 'wxc8e92f7ab70fbca0',
  appsecret: 'b4054e90b75787c78e0af50bf7fc3e87',
  token: 'atguiguHTML1208'   //token 要严格保密！！！
```

```
}
```

2) utils.js

```
/*
   工具函数
 */
//引入解析 xml 数据的库
const {parseString} = require('xml2js');

module.exports = {
  getUserDataAsync (req) {
    /*
      用户数据是通过流的方式发送，通过绑定 data 事件接受数据
     */
    return new Promise((resolve, reject) => {
      let data = '';
      req
        .on('data', userData => {
          //将流式数据全部拼接起来
          data += userData;
        })
        .on('end', () => {
          //确保数据全部获取了
          resolve(data);
        })
    })
  },
  parseXMLAsync (xmlData) {
    return new Promise((resolve, reject) => {
      parseString(xmlData, {trim: true}, (err, data) => {
        if (!err) {
          //解析成功了
          resolve(data);
        } else {
          //解析失败了
          reject('parseXMLAsync 方法出了问题：' + err);
        }
      })
    })
  },
```

```
    formatMessage (jsData) {
        const data = jsData.xml;
        //初始化一个空的对象
        let message = {};
        //判断数据是一个合法的数据
        if (typeof data === 'object') {
            //循环遍历对象中的所有数据
            for (let key in data) {
                //获取属性值
                let value = data[key];
                //过滤掉空的数据和空的数组
                if (Array.isArray(value) && value.length > 0) {
                    //在新对象中添加属性和值
                    message[key] = value[0];
                }
            }
        }
        //将格式化后的数据返回出去
        return message;
    }
}
```

3) api.js

```
/*
    所有接口的文件
 */
//提取出来的接口前缀
const prefix = 'https://api.weixin.qq.com/cgi-bin/';

module.exports = {
    accessToken: prefix + 'token?grant_type=client_credential',
    temporary: {
        upload: prefix + 'media/upload?',
        get: prefix + 'media/get?'
    },
    permanent: {
        uploadNews: prefix + 'material/add_news?',
        uploadImg: prefix + 'media/uploadimg?',
        uploadOthers: prefix + 'material/add_material?',
```

```
      get: prefix + 'material/get_material?',
      delete: prefix + 'material/del_material?',
      updateNews: prefix + 'material/update_news?',
      getCount: prefix + 'material/get_materialcount?',
      getMaterialList: prefix + 'material/batchget_material?'
    }
}
```

4) auth.js

```
/*
  验证服务器的有效性:

    1、填写服务器配置(测试号管理页面)
      - URL 开发者服务器地址（保证能在互联网中能访问）
        通过 ngrok http 端口号 就得到一个网址
      - Token 参与微信签名的加密
    2、验证服务器地址的有效性
      - 将 timestamp、nonce、token 三个参数按照字典序排序
      - 将三个参数拼接在一起，进行 sha1 加密
      - 将加密后生成字符串和微信签名进行对比，
          如果相同说明成功，返回一个 echostr 给微信服务器，
          如果不相同，说明签名算法出了问题，配置不成功
 */
//引入配置对象
const config = require('../config');
//引入 sha1 加密模块
const sha1 = require('sha1');
//引入工具函数
const {getUserDataAsync, parseXMLAsync, formatMessage} = require('../libs/utils');
//引入 reply 模块
const reply = require('./reply');

module.exports = () => {

  return async (req, res, next) => {
    //接受微信服务器发送过来的请求参数
    // console.log(req.query);
    /*
      { signature: 'c4409bdd012bf28d8b4aabf7ac5847c5560d6cf0',    微信的加密
```

```
签名（timestamp、nonce、token）
        echostr: '11283286178012191741',   随机字符串
        timestamp: '1529977721',            时间戳
        nonce: '1462949582' }               随机数字
    */
  //获取参与加密的参数
  const {signature, echostr, timestamp, nonce} = req.query;
  const {token} = config;
  /*// - 将 timestamp、nonce、token 三个参数按照字典序排序
  const arr = [timestamp, nonce, token].sort();
  // - 将三个参数拼接在一起，进行 sha1 加密
  const str = arr.join('');
  const sha1Str = sha1(str);*/
  //简写方式
  const sha1Str = sha1([timestamp, nonce, token].sort().join(''));

  /*
    微信服务器会主动发送两种方法的消息
      GET 请求，  验证服务器有效性
      POST 请求，微信服务器会将用户发送过来的消息转发到开发者服务器上
    */
  if (req.method === 'GET') {
    // - 将加密后生成字符串和微信签名进行对比，
    if (sha1Str === signature) {
      //说明成功，返回 echostr 给微信服务器
      res.send(echostr);
    } else {
      //说明失败
      res.send('');
    }
  } else if (req.method === 'POST') {
    //接受用户发送过来消息
    // console.log(req.query);
    /*
      { signature: 'c67250097842aa50990259fa3df052eeffcb1cee',
        timestamp: '1530000513',
        nonce: '53405765',
        openid: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' }   //用户的 id
     */
    //验证消息是否来自于微信服务器
```

```
        if (sha1Str !== signature) {
          //说明消息不是来自于微信服务器
          //过滤掉非法请求
          res.send('error');
          return
        }

        //获取用户的消息，返回的数据格式是 xml
        const xmlData = await getUserDataAsync(req);
        // console.log(xmlData);
        /*
          <xml>
            <ToUserName><![CDATA[gh_4fe7faab4d6c]]></ToUserName>     //开
发者的 id

<FromUserName><![CDATA[oAsoR1iP-_D3LZIwNCnK8BFotmJc]]></FromUserName>
//用户的 openid
            <CreateTime>1530001191</CreateTime>     //消息发送时间
            <MsgType><![CDATA[text]]></MsgType>     //消息的类型
            <Content><![CDATA[666]]></Content>      //消息的具体内容
            <MsgId>6571305078611302153</MsgId>      //消息的 id
          </xml>
         */
        //将 xml 解析成 js 对象
        const jsData = await parseXMLAsync(xmlData);
        // console.log(jsData);
        /*
          { xml:
           { ToUserName: [ 'gh_4fe7faab4d6c' ],
             FromUserName: [ 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' ],
             CreateTime: [ '1530001675' ],
             MsgType: [ 'text' ],
             Content: [ '774' ],
             MsgId: [ '6571307157375473517' ] } }
         */
        //格式化数据
        const message = formatMessage(jsData);
        console.log(message);
        /*
          { ToUserName: 'gh_4fe7faab4d6c',
            FromUserName: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc',
```

```
            CreateTime: '1530002262',
            MsgType: 'text',
            Content: '888',
            MsgId: '6571309678521276386' }
         */

        //返回用户消息
        /*
          1. 假如服务器无法保证在五秒内处理并回复
          2. 回复 xml 数据中有多余的空格 *****
          3. 回复文本内容，中 options.content='' ***
          如果有以上现象，就会导致微信客户端中的报错：
            '该公众号提供服务出现故障，请稍后再试'
         */
        //设置回复用户消息的具体内容
        const replyMessage = await reply(message);
        console.log(replyMessage);    //Promise { <pending> }
        //返回响应给微信服务器
        res.send(replyMessage);

        /*//先返回一个空的响应给微信服务器
        res.send('');*/
    }
  }
}
```

5) reply.js

```
/*
  处理并分析用户发送的消息
  决定返回什么消息给用户
 */
const template = require('./template');
const Wechat = require('./wechat');


const wechatApi = new Wechat();
```

```
module.exports = async message => {

  //定义 options
  let options = {
    toUserName: message.FromUserName,
    fromUserName: message.ToUserName,
    createTime: Date.now(),
    msgType: 'text'
  }

  //设置回复用户消息的具体内容
  let content = '';

  //判断用户发送消息的类型和内容，决定返回什么消息给用户
  if (message.MsgType === 'text') {
    if (message.Content === '1') {
      content = '大吉大利，今晚吃鸡';
    } else if (message.Content === '2') {
      content = '落地成盒';
    } else if (message.Content === '3') {
      //回复图文消息
      content = [{
        title: 'Nodejs 开发',
        description: '微信公众号开发',
        picUrl:
'https://ss1.baidu.com/6ONXsjip0QIZ8tyhnq/it/u=1841004364,244945169&fm=58
```

```
&bpow=121&bpoh=75',
            url: 'http://nodejs.cn/'
        }, {
            title: 'web 前端',
            description: '这里有最新、最强的技术',
            picUrl:
'https://ss0.baidu.com/6ONWsjip0QIZ8tyhnq/it/u=1981851186,10620031&fm=58
&s=6183FE1ECDA569015C69A554030010F3&bpow=121&bpoh=75',
            url: 'http://www.atguigu.com/'
        }];
        options.msgType = 'news';

    } else if (message.Content === '4') {
        //上传一个多媒体素材
        const data = await wechatApi.uploadTemporaryMaterial('image',
'C:\\Users\\web\\Desktop\\图片\\1.jpg');
        console.log(data);
        /*
          { type: 'image',
            media_id:
'nT2v9ObOdrUjMU-klAQrNTy1I3pZI0_ZO8yV6zB3N0KHVg92nlToTEpNXbkTcskV',
            created_at: 1530070685 }
         */
        //返回一个图片消息给用户
        options.msgType = 'image';
        options.mediaId = data.media_id;
```

```
    } else if (message.Content === '5') {
        //获取一个多媒体素材
        await wechatApi.getTemporaryMaterial('nT2v9ObOdrUjMU-kIAQrNTy1I3pZI0_ZO8yV6zB3N0KHVg92nlToTEpNXbkTcskV', __dirname + '/1.jpg');
        content = '获取多媒体素材成功了~~';

    } else if (message.Content === '6') {
        //上传图文素材中的图片
        const {url} = await wechatApi.uploadPermanentMaterial('pic', 'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(url);
//http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2tyQicPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0
        //上传缩略图
        const {media_id} = await wechatApi.uploadPermanentMaterial('image', 'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(media_id);
//1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE
        //上传图文素材
        const newsList = {
          "articles": [{
            "title": '大美女',
            "thumb_media_id": media_id,
            "author": '佚名',
            "digest": '这里有个大美女',
            "show_cover_pic": 1,
```

```
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
            '    <img src="' + url + '">\n' +
            '</body>\n' +
            '</html>',
            "content_source_url": 'http://www.atguigu.com'
        },{
            "title": '大美女',
            "thumb_media_id": media_id,
            "author": '佚名',
            "digest": '这里有个大美女',
            "show_cover_pic": 1,
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
            '    <img src="' + url + '">\n' +
```

```
                '</body>\n' +
                '</html>',
                "content_source_url": 'http://www.atguigu.com'
            }]
        }
        const data = await wechatApi.uploadPermanentMaterial('news', newsList);
        console.log(data); //{ media_id:
'1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas' }


        content = '上传永久素材成功了~~';


    } else if (message.Content === '7') {
        //获取永久素材
        const newsData = await wechatApi.getPermanentMaterial('news',
'1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas');
        console.log(newsData);
        //返回给用户
        content = [];
        newsData.news_item.forEach(item => {
            content.push({
                title: item.title,
                description: item.digest,
                picUrl:
'http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2ty
QicPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0',
                url: item.url
            })
```

```
        })
        options.msgType = 'news';
    } else if (message.Content === '8') {
        //更新永久图文素材
        const body = {
            "media_id": '1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas',
            "index": 0,
            "articles": {
                "title": '大帅哥',
                "thumb_media_id":
'1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE',
                "author": '0315',
                "digest": '这是一个大帅哥',
                "show_cover_pic": 0,
                "content": 'hello 爱他硅谷',
                "content_source_url": 'http://www.baidu.com'
            }
        }
        let data = await wechatApi.updatePermanentNews(body);
        console.log(data); //{ errcode: 0, errmsg: 'ok' }
        //获取永久素材的数量
        data = await wechatApi.getPermanentCount();
        console.log(data);
        /*
            { voice_count: 1,
              video_count: 9,
              image_count: 48,
```

```
            news_count: 20 }
          */
        //获取指定素材的列表
        data = await wechatApi.getPermanentList({
          type: 'news',
          offset: 0,
          count: 20
        });
        console.log(data);
        //删除永久素材
        data = await
wechatApi.deletePermanentMaterial('1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGw
hrTOGas');
        console.log(data);
        //返回给用户
        content = '测试 api';
      } else if (message.Content.match('爱')) {
        //模糊匹配，只要包含爱
        content = '我爱你~';
      } else {
        content = '您在说啥，我听不懂';
      }
    } else if (message.MsgType === 'image') {
      content = '您的图片地址为：' + message.PicUrl;
    } else if (message.MsgType === 'voice') {
      content = '语音识别结果：' + message.Recognition;
    } else if (message.MsgType === 'video') {
```

```
        content = '接受了视频消息';
    } else if (message.MsgType === 'shortvideo') {
        content = '接受了小视频消息';
    } else if (message.MsgType === 'location') {
        content = '纬度：' + message.Location_X + '  经度：' + message.Location_Y
            + '  缩放大小：' + message.Scale + '  详情：' + message.Label;
    } else if (message.MsgType === 'link') {
        content = '标题：' + message.Title + '  描述：' + message.Description + '  网址：
' + message.Url;
    } else if (message.MsgType === 'event') {
        if (message.Event === 'subscribe') {
            //用户订阅事件
            content = '欢迎您的订阅~';
            if (message.EventKey) {
                //扫描带参数的二维码的订阅事件
                content = '欢迎您扫二维码的关注';
            }
        } else if (message.Event === 'SCAN') {
            //已经关注了公众号，在扫描带参数二维码进入公众号
            content = '已经关注了公众号，在扫描带参数二维码进入公众号';
        } else if (message.Event === 'unsubscribe') {
            //用户取消关注
            console.log('无情取关~');
        } else if (message.Event === 'LOCATION') {
            //用户进行会话时，上报一次地理位置消息
            content = '纬度：' + message.Latitude + '  经度：' + message.Longitude + '  精
度：' + message.Precision;
```

```
    } else if (message.Event === 'CLICK') {
      content = '点击了菜单~~~';
    } else if (message.Event === 'VIEW') {
      //用户点击菜单，跳转到其他链接
      console.log('用户点击菜单，跳转到其他链接');
    }
  }

  //将最终回复消息内容添加到 options 中
  options.content = content;
  //将最终的 xml 数据返回出去
  return template(options);
}
```

6) template.js

```
/*
  设置回复用户的 6 种消息内容
 */

module.exports = options => {

  //回复用户消息
  let replyMessage = '<xml>' +
    '<ToUserName><![CDATA[' + options.toUserName + ']]></ToUserName>' +
    '<FromUserName><![CDATA[' + options.fromUserName +
']]></FromUserName>' +
    '<CreateTime>' + options.createTime + '</CreateTime>' +
    '<MsgType><![CDATA[' + options.msgType + ']]></MsgType>';

  if (options.msgType === 'text') {
    replyMessage += '<Content><![CDATA[' + options.content + ']]></Content>';
  } else if (options.msgType === 'image') {
    replyMessage += '<Image><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Image>';
```

```
  } else if (options.msgType === 'voice') {
    replyMessage += '<Voice><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Voice>';
  } else if (options.msgType === 'video') {
    replyMessage += '<Video>' +
      '<MediaId><![CDATA[' + options.mediaId + ']]></MediaId>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '</Video>';
  } else if (options.msgType === 'music') {
    replyMessage += '<Music>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '<MusicUrl><![CDATA[' + options.musicUrl + ']]></MusicUrl>' +
      '<HQMusicUrl><![CDATA[' + options.hqMusicUrl + ']]></HQMusicUrl>' +
      '<ThumbMediaId><![CDATA[' + options.mediaId + ']]></ThumbMediaId>' +
      '</Music>';
  } else if (options.msgType === 'news') {
    replyMessage += '<ArticleCount>' + options.content.length + '</ArticleCount>'
+
      '<Articles>';

    options.content.forEach(item => {
      replyMessage += '<item>' +
        '<Title><![CDATA[' + item.title + ']]></Title>' +
        '<Description><![CDATA[' + item.description + ']]></Description>' +
        '<PicUrl><![CDATA[' + item.picUrl + ']]></PicUrl>' +
        '<Url><![CDATA[' + item.url + ']]></Url>' +
        '</item>';
    })

    replyMessage += '</Articles>';
  }

  replyMessage += '</xml>';
  //将拼接好回复用户的数据返回出去
  return replyMessage;

}
```

7) wechat.js

```
/*
  获取 access_token:
    全局唯一的接口调用凭据，今后使用微信的接口基本上都需要携带上这个参数
    2 小时需要更新一次，提前 5 分钟刷新

    请求地址：

https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
    请求方式：
      GET

    设计思路：
      首先发送请求获取凭据，保存为一个唯一的文件
      然后后面请求先去本地文件读取凭据
        判断凭据是否过期
          如果没有过期，直接使用
          如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件

    总结：
      先去本地查看有没有指定文件（readAccessToken）
        如果有（之前请求过凭据）
          判断凭据是否过期(isValidAccessToken)
            如果没有过期，直接使用
            如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件(getAccessToken、saveAccessToken)
        如果没有（之前都没有请求过凭据）
          发送请求获取凭据，保存为一个唯一的文件

 */
//引入配置对象
const {appID, appsecret} = require('../config');
//引入发送 http 请求的库
const rp = require('request-promise-native');
const request = require('request');
//引入 fs 模块
const {readFile, writeFile, createReadStream, createWriteStream} = require('fs');
//引入接口文件
const api = require('../libs/api');
```

```
class Wechat {
  getAccessToken () {
    //定义请求地址
    const url = `${api.accessToken}&appid=${appID}&secret=${appsecret}`;
    /*
      问题：需要将回调函数中的数据返回出去？
      解决：用 promise 解决

      所有的异步操作，都应该包装一层 promise，让这个异步操作执行完毕之后，再去执行后面的代码
      简化： 所有的异步操作，都应该包装一层 promise
     */
    return new Promise((resolve, reject) => {
      //发送 http 请求
      //下载 request-promise-native  request
      rp({method: 'GET', json: true, url})
        .then(res => {
          //请求成功的状态
          // console.log(res);
          //重新赋值凭据的过期时间 ： 当前时间 + (7200 - 5 分钟) * 1000
          res.expires_in = Date.now() + (res.expires_in - 300) * 1000;
          // console.log(res);
          resolve(res);
        })
        .catch(err => {
          //请求失败
          reject('getAccessToken 方法出了问题：' + err);
        })
    })
  }
  saveAccessToken (data) {
    /*
      问题：writeFile 方法会将对象转化为字符串
      解决：我将对象转化为 json 字符串
     */
    data = JSON.stringify(data);
    return new Promise((resolve, reject) => {
      //将凭据保存为一个文件
      writeFile('accessToken.txt', data, err => {
        if (!err) {
```

```
            //写入成功
            resolve();
          } else {
            //写入失败
            reject('saveAccessToken 方法出了问题：' + err);
          }
        })
      })
    }
    readAccessToken () {
      return new Promise((resolve, reject) => {
        //将凭据读取出来
        readFile('accessToken.txt', (err, data) => {
          if (!err) {
            //将读取的 Buffer 数据转化为 json 字符串
            data = data.toString();
            //将 json 字符串转化为对象
            data = JSON.parse(data);
            //读取成功
            resolve(data);
          } else {
            //读取失败
            reject('readAccessToken 方法出了问题：' + err);
          }
        })
      })
    }
    isValidAccessToken (data) {
      /*
        判断凭据是否过期
          true    凭据没有过期
          false   凭据过期了
       */
      //过滤非法的数据
      if (!data || !data.access_token || !data.expires_in) return false;
      //判断凭据是否过期
      /*if (data.expires_in > Date.now()) {
        //如果凭据的过期时间大于当前时间，说明没有过期
        return true
      } else {
        //如果凭据的过期时间小于当前时间，说明过期了
```

```
            return false
        }*/
        //简写方式
        return data.expires_in > Date.now();
    }
    fetchAccessToken () {
        //优化操作,优化不去执行读取文件操作
        if (this.access_token && this.expires_in && this.isValidAccessToken(this)) {
            //说明 this 有凭据和过期时间，并且凭据未过期
            return Promise.resolve({access_token: this.access_token, expires_in:
this.expires_in});
        }

        return this.readAccessToken()
            .then(async res => {
                //判断凭据是否过期(isValidAccessToken)
                if (this.isValidAccessToken(res)) {
                    //没有过期，直接使用
                    return Promise.resolve(res);
                } else {
                    //重新发送请求获取凭据
                    const data = await this.getAccessToken();
                    //保存下来
                    await this.saveAccessToken(data);
                    //将请求回来的凭据返回出去
                    return Promise.resolve(data);
                }
            })
            .catch(async err => {
                console.log(err);
                //重新发送请求获取凭据
                const data = await this.getAccessToken();
                //保存下来
                await this.saveAccessToken(data);
                //将请求回来的凭据返回出去
                return Promise.resolve(data);
            })
            .then(res => {
                //将其请求回来的凭据和过期时间挂载到 this 上
                this.access_token = res.access_token;
                this.expires_in = res.expires_in;
```

```
            //指定 fetchAccessToken 方法返回值
            return Promise.resolve(res);
        })
    }

    //上传临时素材
    uploadTemporaryMaterial (type, filePath) {
        /*
            type: 上传多媒体文件的类型
            filePath: 上传多媒体文件的路径
         */
        return new Promise((resolve, reject) => {
            //获取 access_token
            this.fetchAccessToken()
                .then(res => {
                    //定义请求的地址
                    const url =
`${api.temporary.upload}access_token=${res.access_token}&type=${type}`;
                    //定义要传输过去的媒体数据
                    const formData = {
                        media: createReadStream(filePath)
                    }
                    //发送请求
                    rp({method: 'POST', json: true, url, formData})
                        .then(res => {
                            //将请求回来的数据返回出去
                            resolve(res);
                        })
                        .catch(err => {
                            reject('uploadTemporaryMaterial 方法出了问题：' + err);
                        })
                })
        })


    }
    //获取临时素材
    getTemporaryMaterial (mediaId, filePath, isVideo = false) {
        /*
            mediaId: 要获取的素材 id
            filePath：要保存媒体文件的路径
```

```
        isVideo: 可选值
      */
    return new Promise((resolve, reject) => {
      //获取 access_token
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          const url =
`${api.temporary.get}access_token=${res.access_token}&media_id=${mediaId}`;
          //发送请求
          if (isVideo) {
            //如果是视频消息素材，就返回一个 url 地址
            rp({method: 'GET', json: true, url})
              .then(res => resolve(res))
              .catch(err => reject('getTemporaryMaterial 方法出了问题：' + err))
          } else {
            //如果不是视频消息素材，就返回一个文件接收
            request
              .get(url)
              .pipe(createWriteStream(filePath))
              .once('close', () => {
                //说明文件下载成功了~
                resolve();
              })
          }
        })
    })

  }
  //上传永久素材
  uploadPermanentMaterial (type, material, description) {
    /*
      type: 可以区分我通过什么方式上传素材
      material: 上传素材的路径/请求体中的内容
      description: 针对于视频素材上传
    */
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          let url = '';
```

```
                //定义发送请求的配置对象
                let options = {
                    method: 'POST',
                    json: true
                }
                if (type === 'news') {
                    //上传图文消息
                    url =
`${api.permanent.uploadNews}access_token=${res.access_token}`;
                    options.body = material;
                } else if (type === 'pic') {
                    //上传图文消息的图片
                    url =
`${api.permanent.uploadImg}access_token=${res.access_token}`;
                    options.formData = {
                        media: createReadStream(material)
                    }
                } else {
                    //上传其他素材
                    url =
`${api.permanent.uploadOthers}access_token=${res.access_token}&type=${type}`;
                    options.formData = {
                        media: createReadStream(material)
                    }
                    if (type === 'video') {
                        options.body = description;
                    }
                }
                //将请求地址放到配置对象中
                options.url = url;
                //发送请求
                rp(options)
                    .then(res => resolve(res))
                    .catch(err => reject('uploadPermanentMaterial 方法出了问题:' +
err))
            })
        })
    }
    //获取永久素材
    getPermanentMaterial (type, mediaId, filePath) {
        /*
```

```
        type: 用来如何接受数据
        mediaId: 获取的媒体素材 id
        filePath: 保存的媒体素材的位置
       */
      return new Promise((resolve, reject) => {
        this.fetchAccessToken()
          .then(res => {
            //定义请求地址
            const url = `${api.permanent.get}access_token=${res.access_token}`;
            //定义请求体中的数据
            const body = {
              media_id: mediaId
            }
            if (type === 'news' || 'video') {
              rp({method: 'POST', json: true, url, body})
                .then(res => resolve(res))
                .catch(err => reject('getPermanentMaterial 方法出了问题:' + err))
            } else {
              request({method: 'POST', json: true, url, body})
                .pipe(createWriteStream(filePath))
                .once('close', () => resolve())
            }
          })
      })
    }
    //删除永久素材
    deletePermanentMaterial (mediaId) {
      return new Promise((resolve, reject) => {
        this.fetchAccessToken()
          .then(res => {
            const url =
`${api.permanent.delete}access_token=${res.access_token}`;
            const body = {
              media_id: mediaId
            }
            rp({method: 'POST', json: true, url, body})
              .then(res => resolve(res))
              .catch(err => reject('deletePermanentMaterial 方法出了问题：' +
err))
          })
      })
```

```
  }
  //更新永久图文消息素材
  updatePermanentNews (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url =
`${api.permanent.updateNews}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('updatePermanentNews 方法出了问题：' + err))
        })
    })
  }
  //获取永久素材数量
  getPermanentCount () {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url =
`${api.permanent.getCount}access_token=${res.access_token}`;
          rp({method: 'GET', json: true, url})
            .then(res => resolve(res))
            .catch(err => reject('getPermanentCount 方法出了问题：' + err))
        })
    })
  }
  //获取永久素材列表
  getPermanentList (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url =
`${api.permanent.getMaterialList}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('getPermanentList 方法出了问题：' + err))
        })
    })
  }
}
```

```
//测试
// (async () => {
//
//     const wechatApi = new Wechat();
//
//     console.log(await wechatApi.fetchAccessToken());
//
// })()

module.exports = Wechat;
```

8) index.js

```
const express = require('express');
const auth = require('./wechat/auth');
const app = express();

//接受微信服务器发送过来的请求 GET
//应用中间级，能够接受处理所有请求
app.use(auth());

app.listen(3000, err => {
  if (!err) console.log('服务器启动成功了~~~');
})
```

2.5 菜单

2.5.1 目录结构

```
├── config/              # 配置目录
|    ├── index.js        # 存储配置信息
├── libs/                # 工具方法库
|    ├── util.js         # 解析字符串的工具方法
|    ├── api.js          # 定义接口的文件
├─── wechat/            # 核心功能库
```

```
|   ├── 1.jpg          # 需要上传素材图片
|   ├── 2.mp4          # 需要上传素材视频
|   ├── auth.js        # 回复消息功能
|   ├── menu.js        # 定义菜单
|   ├── reply.js       # 处理用户发送的信息
|   ├── template.js    # 返回给用户的信息模板
|   ├── wechat.js      # 类 Wechat
├── app.js             # 入口启动文件
├── package.json       # 配置文件
```

2.5.2 代码

1) index.js

```
/*
  此模块用来储存关键的配置信息
 */
module.exports = {
  appID: 'wxc8e92f7ab70fbca0',
  appsecret: 'b4054e90b75787c78e0af50bf7fc3e87',
  token: 'atguiguHTML1208'   //token 要严格保密！！！
}
```

2) utils.js

```
/*
  工具函数
 */
//引入解析 xml 数据的库
const {parseString} = require('xml2js');

module.exports = {
  getUserDataAsync (req) {
```

```
    /*
      用户数据是通过流的方式发送，通过绑定 data 事件接受数据
     */
    return new Promise((resolve, reject) => {
      let data = '';
      req
        .on('data', userData => {
          //将流式数据全部拼接起来
          data += userData;
        })
        .on('end', () => {
          //确保数据全部获取了
          resolve(data);
        })
    })
  },
  parseXMLAsync (xmlData) {
    return new Promise((resolve, reject) => {
      parseString(xmlData, {trim: true}, (err, data) => {
        if (!err) {
          //解析成功了
          resolve(data);
        } else {
          //解析失败了
          reject('parseXMLAsync 方法出了问题：' + err);
        }
      })
    })
  },
  formatMessage (jsData) {
    const data = jsData.xml;
    //初始化一个空的对象
    let message = {};
    //判断数据是一个合法的数据
    if (typeof data === 'object') {
      //循环遍历对象中的所有数据
      for (let key in data) {
        //获取属性值
        let value = data[key];
        //过滤掉空的数据和空的数组
        if (Array.isArray(value) && value.length > 0) {
```

```
                //在新对象中添加属性和值
                message[key] = value[0];
            }
        }
    }
    //将格式化后的数据返回出去
    return message;
  }
}
```

3) api.js

```
/*
  所有接口的文件
 */
//提取出来的接口前缀
const prefix = 'https://api.weixin.qq.com/cgi-bin/';


module.exports = {
  accessToken: prefix + 'token?grant_type=client_credential',
  temporary: {
    upload: prefix + 'media/upload?',
    get: prefix + 'media/get?'
  },
  permanent: {
    uploadNews: prefix + 'material/add_news?',
    uploadImg: prefix + 'media/uploadimg?',
    uploadOthers: prefix + 'material/add_material?',
    get: prefix + 'material/get_material?',
    delete: prefix + 'material/del_material?',
```

```
    updateNews: prefix + 'material/update_news?',
    getCount: prefix + 'material/get_materialcount?',
    getMaterialList: prefix + 'material/batchget_material?'
  },
  menu: {
    create: prefix + 'menu/create?',
    delete: prefix + 'menu/delete?',
    get: prefix + 'menu/get?',
    myCreate: prefix + 'menu/addconditional?',
    myDelete: prefix + 'menu/delconditional?',
    myTest: prefix + 'menu/trymatch?'
  }
}
```

4) auth.js

```
/*
  验证服务器的有效性:

    1、填写服务器配置(测试号管理页面)
      - URL 开发者服务器地址（保证能在互联网中能访问）
        通过 ngrok http 端口号 就得到一个网址
      - Token 参与微信签名的加密
    2、验证服务器地址的有效性
      - 将 timestamp、nonce、token 三个参数按照字典序排序
      - 将三个参数拼接在一起，进行 sha1 加密
      - 将加密后生成字符串和微信签名进行对比，
          如果相同说明成功，返回一个 echostr 给微信服务器，
          如果不相同，说明签名算法出了问题，配置不成功
 */
//引入配置对象
const config = require('../config');
//引入 sha1 加密模块
const sha1 = require('sha1');
//引入工具函数
```

```
const {getUserDataAsync, parseXMLAsync, formatMessage} = require('../libs/utils');
//引入 reply 模块
const reply = require('./reply');

module.exports = () => {

  return async (req, res, next) => {
    //接受微信服务器发送过来的请求参数
    // console.log(req.query);
    /*
      { signature: 'c4409bdd012bf28d8b4aabf7ac5847c5560d6cf0',    微信的加密签名（timestamp、nonce、token）
        echostr: '11283286178012191741',   随机字符串
        timestamp: '1529977721',            时间戳
        nonce: '1462949582' }               随机数字
     */
    //获取参与加密的参数
    const {signature, echostr, timestamp, nonce} = req.query;
    const {token} = config;
    /*// - 将 timestamp、nonce、token 三个参数按照字典序排序
    const arr = [timestamp, nonce, token].sort();
    // - 将三个参数拼接在一起，进行 sha1 加密
    const str = arr.join('');
    const sha1Str = sha1(str);*/
    //简写方式
    const sha1Str = sha1([timestamp, nonce, token].sort().join(''));

    /*
      微信服务器会主动发送两种方法的消息
        GET 请求，  验证服务器有效性
        POST 请求，微信服务器会将用户发送过来的消息转发到开发者服务器上
     */
    if (req.method === 'GET') {
      // - 将加密后生成字符串和微信签名进行对比，
      if (sha1Str === signature) {
        //说明成功，返回 echostr 给微信服务器
        res.send(echostr);
      } else {
        //说明失败
        res.send('');
```

```
    }
  } else if (req.method === 'POST') {
    //接受用户发送过来消息
    // console.log(req.query);
    /*
      { signature: 'c67250097842aa50990259fa3df052eeffcb1cee',
        timestamp: '1530000513',
        nonce: '53405765',
        openid: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' }   //用户的 id
     */
    //验证消息是否来自于微信服务器
    if (sha1Str !== signature) {
      //说明消息不是来自于微信服务器
      //过滤掉非法请求
      res.send('error');
      return
    }

    //获取用户的消息，返回的数据格式是 xml
    const xmlData = await getUserDataAsync(req);
    // console.log(xmlData);
    /*
      <xml>
        <ToUserName><![CDATA[gh_4fe7faab4d6c]]></ToUserName>     //开
发者的 id

<FromUserName><![CDATA[oAsoR1iP-_D3LZIwNCnK8BFotmJc]]></FromUserName>
//用户的 openid
        <CreateTime>1530001191</CreateTime>     //消息发送时间
        <MsgType><![CDATA[text]]></MsgType>     //消息的类型
        <Content><![CDATA[666]]></Content>      //消息的具体内容
        <MsgId>6571305078611302153</MsgId>      //消息的 id
      </xml>
     */
    //将 xml 解析成 js 对象
    const jsData = await parseXMLAsync(xmlData);
    // console.log(jsData);
    /*
      { xml:
       { ToUserName: [ 'gh_4fe7faab4d6c' ],
         FromUserName: [ 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' ],
```

```
                CreateTime: [ '1530001675' ],
                MsgType: [ 'text' ],
                Content: [ '774' ],
                MsgId: [ '6571307157375473517' ] } }
            */
            //格式化数据
            const message = formatMessage(jsData);
            console.log(message);
            /*
              { ToUserName: 'gh_4fe7faab4d6c',
                FromUserName: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc',
                CreateTime: '1530002262',
                MsgType: 'text',
                Content: '888',
                MsgId: '6571309678521276386' }
            */

            //返回用户消息
            /*
              1. 假如服务器无法保证在五秒内处理并回复
              2. 回复 xml 数据中有多余的空格 *****
              3. 回复文本内容，中 options.content='' ***
              如果有以上现象，就会导致微信客户端中的报错：
                '该公众号提供服务出现故障，请稍后再试'
            */
            //设置回复用户消息的具体内容
            const replyMessage = await reply(message);
            console.log(replyMessage);    //Promise { <pending> }
            //返回响应给微信服务器
            res.send(replyMessage);

            /*//先返回一个空的响应给微信服务器
            res.send('');*/
        }
    }
}
```

5) menu.js

```
/*
```

```
  菜单的配置
 */

module.exports = {
  "button":[
    {
      "type":"click",
      "name":"首页",
      "key":"首页"
    },
    {
      "name":"二级菜单",
      "sub_button":[
        {
          "type":"view",
          "name":"跳转到硅谷",
          "url":"http://www.atguigu.com/"
        },
        {
          "type": "scancode_waitmsg",
          "name": "扫码带提示",
          "key": "扫码带提示"
        },
        {
          "type": "scancode_push",
          "name": "扫码推事件\ue348",
          "key": "扫码推事件"
        },
        {
          "type": "pic_sysphoto",
          "name": "系统拍照发图",
          "key": "系统拍照发图"
        },
        {
          "type": "pic_photo_or_album",
          "name": "拍照或者相册发图",
          "key": "拍照或者相册发图"
        }
      ]
    },
    {
```

```
        "name":"二菜单",
        "sub_button":[
          {
            "type": "pic_weixin",
            "name": "微信相册发图",
            "key": "微信相册发图"
          },
          {
            "type": "location_select",
            "name": "发送位置",
            "key": "发送位置"
          },
          {
            "type": "media_id",
            "name": "图片",
            "media_id": "1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE"
          },
          {
            "type": "view_limited",
            "name": "图文消息",
            "media_id": "1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas"
          }
        ]
      }
  ]
}
```

6) reply.js

```
/*
  处理并分析用户发送的消息
  决定返回什么消息给用户
 */
const template = require('./template');
const Wechat = require('./wechat');
```

```
const wechatApi = new Wechat();


module.exports = async message => {


  //定义 options
  let options = {
    toUserName: message.FromUserName,
    fromUserName: message.ToUserName,
    createTime: Date.now(),
    msgType: 'text'
  }


  //设置回复用户消息的具体内容
  let content = '';


  //判断用户发送消息的类型和内容，决定返回什么消息给用户
  if (message.MsgType === 'text') {
    if (message.Content === '1') {
      content = '大吉大利，今晚吃鸡';
    } else if (message.Content === '2') {
      content = '落地成盒';
    } else if (message.Content === '3') {
      //回复图文消息
      content = [{
        title: 'Nodejs 开发',
        description: '微信公众号开发',
        picUrl:
```

```
'https://ss1.baidu.com/6ONXsjip0QIZ8tyhnq/it/u=1841004364,244945169&fm=58
&bpow=121&bpoh=75',
            url: 'http://nodejs.cn/'
          }, {
            title: 'web 前端',
            description: '这里有最新、最强的技术',
            picUrl:
'https://ss0.baidu.com/6ONWsjip0QIZ8tyhnq/it/u=1981851186,10620031&fm=58
&s=6183FE1ECDA569015C69A554030010F3&bpow=121&bpoh=75',
            url: 'http://www.atguigu.com/'
          }];
          options.msgType = 'news';

      } else if (message.Content === '4') {
          //上传一个多媒体素材
          const data = await wechatApi.uploadTemporaryMaterial('image',
'C:\\Users\\web\\Desktop\\图片\\1.jpg');
          console.log(data);
          /*
            { type: 'image',
              media_id:
'nT2v9ObOdrUjMU-kIAQrNTy1I3pZI0_ZO8yV6zB3N0KHVg92nlToTEpNXbkTcskV',
              created_at: 1530070685 }
           */
          //返回一个图片消息给用户
          options.msgType = 'image';
          options.mediaId = data.media_id;
```

```
    } else if (message.Content === '5') {
        //获取一个多媒体素材
        await wechatApi.getTemporaryMaterial('nT2v9ObOdrUjMU-kIAQrNTy1I3pZI0_ZO8yV6zB3N0KHVg92nlToTEpNXbkTcskV', __dirname + '/1.jpg');
        content = '获取多媒体素材成功了~~';

    } else if (message.Content === '6') {
        //上传图文素材中的图片
        const {url} = await wechatApi.uploadPermanentMaterial('pic', 'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(url);
//http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2tyQicPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0
        //上传缩略图
        const {media_id} = await wechatApi.uploadPermanentMaterial('image', 'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(media_id);
//1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE
        //上传图文素材
        const newsList = {
          "articles": [{
            "title": '大美女',
            "thumb_media_id": media_id,
            "author": '佚名',
            "digest": '这里有个大美女',
```

```
            "show_cover_pic": 1,
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
            '    <img src="' + url + '">\n' +
            '</body>\n' +
            '</html>',
            "content_source_url": 'http://www.atguigu.com'
        },{
            "title": '大美女',
            "thumb_media_id": media_id,
            "author": '佚名',
            "digest": '这里有个大美女',
            "show_cover_pic": 1,
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
```

```
                '    <img src="' + url + '">\n' +
                '</body>\n' +
                '</html>',
                "content_source_url": 'http://www.atguigu.com'
            }]
        }
        const data = await wechatApi.uploadPermanentMaterial('news', newsList);
        console.log(data); //{ media_id:
'1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas' }

        content = '上传永久素材成功了~~';

    } else if (message.Content === '7') {
        //获取永久素材
        const newsData = await wechatApi.getPermanentMaterial('news',
'1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas');
        console.log(newsData);
        //返回给用户
        content = [];
        newsData.news_item.forEach(item => {
            content.push({
                title: item.title,
                description: item.digest,
                picUrl:
'http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2ty
QicPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0',
                url: item.url
```

```
            })
        })
        options.msgType = 'news';
    } else if (message.Content === '8') {
        //更新永久图文素材
        const body = {
            "media_id": '1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas',
            "index": 0,
            "articles": {
                "title": '大帅哥',
                "thumb_media_id":
'1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE',
                "author": '0315',
                "digest": '这是一个大帅哥',
                "show_cover_pic": 0,
                "content": 'hello 爱他硅谷',
                "content_source_url": 'http://www.baidu.com'
            }
        }
        let data = await wechatApi.updatePermanentNews(body);
        console.log(data); //{ errcode: 0, errmsg: 'ok' }
        //获取永久素材的数量
        data = await wechatApi.getPermanentCount();
        console.log(data);
        /*
            { voice_count: 1,
              video_count: 9,
```

```
            image_count: 48,
            news_count: 20 }
         */
        //获取指定素材的列表
        data = await wechatApi.getPermanentList({
          type: 'news',
          offset: 0,
          count: 20
        });
        console.log(data);
        //删除永久素材
        data = await
wechatApi.deletePermanentMaterial('1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGw
hrTOGas');
        console.log(data);
        //返回给用户
        content = '测试 api';
      } else if (message.Content.match('爱')) {
        //模糊匹配，只要包含爱
        content = '我爱你~';
      } else {
        content = '您在说啥，我听不懂';
      }
    } else if (message.MsgType === 'image') {
      content = '您的图片地址为：' + message.PicUrl;
    } else if (message.MsgType === 'voice') {
      content = '语音识别结果：' + message.Recognition;
```

```
    } else if (message.MsgType === 'video') {
        content = '接受了视频消息';
    } else if (message.MsgType === 'shortvideo') {
        content = '接受了小视频消息';
    } else if (message.MsgType === 'location') {
        content = '纬度：' + message.Location_X + '  经度：' + message.Location_Y
            + '  缩放大小：' + message.Scale + '  详情：' + message.Label;
    } else if (message.MsgType === 'link') {
        content = '标题：' + message.Title + '  描述：' + message.Description + '  网址：
' + message.Url;
    } else if (message.MsgType === 'event') {
        if (message.Event === 'subscribe') {
            //用户订阅事件
            content = '欢迎您的订阅~';
            if (message.EventKey) {
                //扫描带参数的二维码的订阅事件
                content = '欢迎您扫二维码的关注';
            }
        } else if (message.Event === 'SCAN') {
            //已经关注了公众号，在扫描带参数二维码进入公众号
            content = '已经关注了公众号，在扫描带参数二维码进入公众号';
        } else if (message.Event === 'unsubscribe') {
            //用户取消关注
            console.log('无情取关~');
        } else if (message.Event === 'LOCATION') {
            //用户进行会话时，上报一次地理位置消息
            content = '纬度：' + message.Latitude + '  经度：' + message.Longitude + '  精
```

```
度：' + message.Precision;
        } else if (message.Event === 'CLICK') {
            content = '点击了菜单~~~';
        } else if (message.Event === 'VIEW') {
            //用户点击菜单，跳转到其他链接
            console.log('用户点击菜单，跳转到其他链接');
        }
    }

    //将最终回复消息内容添加到 options 中
    options.content = content;
    //将最终的 xml 数据返回出去
    return template(options);
}
```

7) template.js

```
/*
    设置回复用户的 6 种消息内容
 */

module.exports = options => {

    //回复用户消息
    let replyMessage = '<xml>' +
        '<ToUserName><![CDATA[' + options.toUserName + ']]></ToUserName>' +
        '<FromUserName><![CDATA[' + options.fromUserName +
']]></FromUserName>' +
        '<CreateTime>' + options.createTime + '</CreateTime>' +
        '<MsgType><![CDATA[' + options.msgType + ']]></MsgType>';

    if (options.msgType === 'text') {
        replyMessage += '<Content><![CDATA[' + options.content + ']]></Content>';
    } else if (options.msgType === 'image') {
```

```
        replyMessage += '<Image><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Image>';
    } else if (options.msgType === 'voice') {
        replyMessage += '<Voice><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Voice>';
    } else if (options.msgType === 'video') {
        replyMessage += '<Video>' +
            '<MediaId><![CDATA[' + options.mediaId + ']]></MediaId>' +
            '<Title><![CDATA[' + options.title + ']]></Title>' +
            '<Description><![CDATA[' + options.description + ']]></Description>' +
            '</Video>';
    } else if (options.msgType === 'music') {
        replyMessage += '<Music>' +
            '<Title><![CDATA[' + options.title + ']]></Title>' +
            '<Description><![CDATA[' + options.description + ']]></Description>' +
            '<MusicUrl><![CDATA[' + options.musicUrl + ']]></MusicUrl>' +
            '<HQMusicUrl><![CDATA[' + options.hqMusicUrl + ']]></HQMusicUrl>' +
            '<ThumbMediaId><![CDATA[' + options.mediaId + ']]></ThumbMediaId>' +
            '</Music>';
    } else if (options.msgType === 'news') {
        replyMessage += '<ArticleCount>' + options.content.length + '</ArticleCount>'
+
            '<Articles>';

        options.content.forEach(item => {
            replyMessage += '<item>' +
                '<Title><![CDATA[' + item.title + ']]></Title>' +
                '<Description><![CDATA[' + item.description + ']]></Description>' +
                '<PicUrl><![CDATA[' + item.picUrl + ']]></PicUrl>' +
                '<Url><![CDATA[' + item.url + ']]></Url>' +
                '</item>';
        })

        replyMessage += '</Articles>';
    }

    replyMessage += '</xml>';
    //将拼接好回复用户的数据返回出去
    return replyMessage;

}
```

9) wechat.js

```
/*
  获取 access_token:
    全局唯一的接口调用凭据，今后使用微信的接口基本上都需要携带上这个参数
    2 小时需要更新一次，提前 5 分钟刷新

    请求地址：

https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
    请求方式：
      GET

    设计思路：
      首先发送请求获取凭据，保存为一个唯一的文件
      然后后面请求先去本地文件读取凭据
        判断凭据是否过期
          如果没有过期，直接使用
          如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件

    总结：
      先去本地查看有没有指定文件（readAccessToken）
        如果有（之前请求过凭据）
          判断凭据是否过期(isValidAccessToken)
            如果没有过期，直接使用
            如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件(getAccessToken、saveAccessToken)
        如果没有（之前都没有请求过凭据）
          发送请求获取凭据，保存为一个唯一的文件

 */
//引入配置对象
const {appID, appsecret} = require('../config');
//引入发送 http 请求的库
const rp = require('request-promise-native');
const request = require('request');
//引入 fs 模块
```

```
const {readFile, writeFile, createReadStream, createWriteStream} = require('fs');
//引入接口文件
const api = require('../libs/api');
//引入菜单文件
const menu = require('./menu');

class Wechat {
  getAccessToken () {
    //定义请求地址
    const url = `${api.accessToken}&appid=${appID}&secret=${appsecret}`;
    /*
      问题：需要将回调函数中的数据返回出去？
      解决：用 promise 解决

      所有的异步操作，都应该包装一层 promise，让这个异步操作执行完毕之后，再去执行后面的代码
      简化： 所有的异步操作，都应该包装一层 promise
     */
    return new Promise((resolve, reject) => {
      //发送 http 请求
      //下载 request-promise-native  request
      rp({method: 'GET', json: true, url})
        .then(res => {
          //请求成功的状态
          // console.log(res);
          //重新赋值凭据的过期时间 ： 当前时间 + (7200 - 5 分钟) * 1000
          res.expires_in = Date.now() + (res.expires_in - 300) * 1000;
          // console.log(res);
          resolve(res);
        })
        .catch(err => {
          //请求失败
          reject('getAccessToken 方法出了问题：' + err);
        })
    })
  }
  saveAccessToken (data) {
    /*
      问题：writeFile 方法会将对象转化为字符串
      解决：我将对象转化为 json 字符串
     */
```

```
    data = JSON.stringify(data);
    return new Promise((resolve, reject) => {
      //将凭据保存为一个文件
      writeFile('accessToken.txt', data, err => {
        if (!err) {
          //写入成功
          resolve();
        } else {
          //写入失败
          reject('saveAccessToken 方法出了问题：' + err);
        }
      })
    })
  }
  readAccessToken () {
    return new Promise((resolve, reject) => {
      //将凭据读取出来
      readFile('accessToken.txt', (err, data) => {
        if (!err) {
          //将读取的 Buffer 数据转化为 json 字符串
          data = data.toString();
          //将 json 字符串转化为对象
          data = JSON.parse(data);
          //读取成功
          resolve(data);
        } else {
          //读取失败
          reject('readAccessToken 方法出了问题：' + err);
        }
      })
    })
  }
  isValidAccessToken (data) {
    /*
      判断凭据是否过期
        true    凭据没有过期
        false   凭据过期了
     */
    //过滤非法的数据
    if (!data || !data.access_token || !data.expires_in) return false;
    //判断凭据是否过期
```

```
        /*if (data.expires_in > Date.now()) {
            //如果凭据的过期时间大于当前时间，说明没有过期
            return true
        } else {
            //如果凭据的过期时间小于当前时间，说明过期了
            return false
        }*/
        //简写方式
        return data.expires_in > Date.now();
    }
    fetchAccessToken () {
        //优化操作,优化不去执行读取文件操作
        if (this.access_token && this.expires_in && this.isValidAccessToken(this)) {
            //说明 this 有凭据和过期时间，并且凭据未过期
            return Promise.resolve({access_token: this.access_token, expires_in:
this.expires_in});
        }

        return this.readAccessToken()
            .then(async res => {
                //判断凭据是否过期(isValidAccessToken)
                if (this.isValidAccessToken(res)) {
                    //没有过期，直接使用
                    return Promise.resolve(res);
                } else {
                    //重新发送请求获取凭据
                    const data = await this.getAccessToken();
                    //保存下来
                    await this.saveAccessToken(data);
                    //将请求回来的凭据返回出去
                    return Promise.resolve(data);
                }
            })
            .catch(async err => {
                console.log(err);
                //重新发送请求获取凭据
                const data = await this.getAccessToken();
                //保存下来
                await this.saveAccessToken(data);
                //将请求回来的凭据返回出去
                return Promise.resolve(data);
```

```
        })
        .then(res => {
          //将其请求回来的凭据和过期时间挂载到 this 上
          this.access_token = res.access_token;
          this.expires_in = res.expires_in;
          //指定 fetchAccessToken 方法返回值
          return Promise.resolve(res);
        })
    }

    //上传临时素材
    uploadTemporaryMaterial (type, filePath) {
      /*
        type: 上传多媒体文件的类型
        filePath: 上传多媒体文件的路径
       */
      return new Promise((resolve, reject) => {
        //获取 access_token
        this.fetchAccessToken()
          .then(res => {
            //定义请求的地址
            const url =
`${api.temporary.upload}access_token=${res.access_token}&type=${type}`;
            //定义要传输过去的媒体数据
            const formData = {
              media: createReadStream(filePath)
            }
            //发送请求
            rp({method: 'POST', json: true, url, formData})
              .then(res => {
                //将请求回来的数据返回出去
                resolve(res);
              })
              .catch(err => {
                reject('uploadTemporaryMaterial 方法出了问题：' + err);
              })
          })
      })


    }
```

```
  //获取临时素材
  getTemporaryMaterial (mediaId, filePath, isVideo = false) {
    /*
      mediaId: 要获取的素材 id
      filePath：要保存媒体文件的路径
      isVideo: 可选值
     */
    return new Promise((resolve, reject) => {
      //获取 access_token
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          const url =
`${api.temporary.get}access_token=${res.access_token}&media_id=${mediaId}`;
          //发送请求
          if (isVideo) {
            //如果是视频消息素材，就返回一个 url 地址
            rp({method: 'GET', json: true, url})
              .then(res => resolve(res))
              .catch(err => reject('getTemporaryMaterial 方法出了问题：' + err))
          } else {
            //如果不是视频消息素材，就返回一个文件接收
            request
              .get(url)
              .pipe(createWriteStream(filePath))
              .once('close', () => {
                //说明文件下载成功了~
                resolve();
              })
          }
        })
    })

  }
  //上传永久素材
  uploadPermanentMaterial (type, material, description) {
    /*
      type: 可以区分我通过什么方式上传素材
      material: 上传素材的路径/请求体中的内容
      description: 针对于视频素材上传
     */
```

```
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          let url = '';
          //定义发送请求的配置对象
          let options = {
            method: 'POST',
            json: true
          }
          if (type === 'news') {
            //上传图文消息
            url =
`${api.permanent.uploadNews}access_token=${res.access_token}`;
            options.body = material;
          } else if (type === 'pic') {
            //上传图文消息的图片
            url =
`${api.permanent.uploadImg}access_token=${res.access_token}`;
            options.formData = {
              media: createReadStream(material)
            }
          } else {
            //上传其他素材
            url =
`${api.permanent.uploadOthers}access_token=${res.access_token}&type=${type}`;
            options.formData = {
              media: createReadStream(material)
            }
            if (type === 'video') {
              options.body = description;
            }
          }
          //将请求地址放到配置对象中
          options.url = url;
          //发送请求
          rp(options)
            .then(res => resolve(res))
            .catch(err => reject('uploadPermanentMaterial 方法出了问题:' +
err))
        })
```

```
    })
  }
  //获取永久素材
  getPermanentMaterial (type, mediaId, filePath) {
    /*
      type: 用来如何接受数据
      mediaId: 获取的媒体素材 id
      filePath: 保存的媒体素材的位置
     */
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          const url = `${api.permanent.get}access_token=${res.access_token}`;
          //定义请求体中的数据
          const body = {
            media_id: mediaId
          }
          if (type === 'news' || 'video') {
            rp({method: 'POST', json: true, url, body})
              .then(res => resolve(res))
              .catch(err => reject('getPermanentMaterial 方法出了问题:' + err))
          } else {
            request({method: 'POST', json: true, url, body})
              .pipe(createWriteStream(filePath))
              .once('close', () => resolve())
          }
        })
    })
  }
  //删除永久素材
  deletePermanentMaterial (mediaId) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url =
`${api.permanent.delete}access_token=${res.access_token}`;
          const body = {
            media_id: mediaId
          }
          rp({method: 'POST', json: true, url, body})
```

```
                .then(res => resolve(res))
                .catch(err => reject('deletePermanentMaterial 方法出了问题：' +
err))
            })
        })
    }
    //更新永久图文消息素材
    updatePermanentNews (body) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
            .then(res => {
                const url =
`${api.permanent.updateNews}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                .then(res => resolve(res))
                .catch(err => reject('updatePermanentNews 方法出了问题：' + err))
            })
        })
    }
    //获取永久素材数量
    getPermanentCount () {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
            .then(res => {
                const url =
`${api.permanent.getCount}access_token=${res.access_token}`;
                rp({method: 'GET', json: true, url})
                .then(res => resolve(res))
                .catch(err => reject('getPermanentCount 方法出了问题：' + err))
            })
        })
    }
    //获取永久素材列表
    getPermanentList (body) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
            .then(res => {
                const url =
`${api.permanent.getMaterialList}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                .then(res => resolve(res))
```

```
                .catch(err => reject('getPermanentList 方法出了问题：' + err))
        })
    })
}
//创建菜单
createMenu (body) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.create}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                    .then(res => resolve(res))
                    .catch(err => reject('createMenu 方法出了问题：' + err))
            })
    })
}
//删除菜单
deleteMenu () {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.delete}access_token=${res.access_token}`;
                rp({method: 'GET', json: true, url})
                    .then(res => resolve(res))
                    .catch(err => reject('deleteMenu 方法出了问题：' + err))
            })
    })
}
//获取菜单的配置
getMenu () {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.get}access_token=${res.access_token}`;
                rp({method: 'GET', json: true, url})
                    .then(res => resolve(res))
                    .catch(err => reject('getMenu 方法出了问题：' + err))
            })
    })
}
//创建自定义菜单
```

```
  createMyMenu (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.menu.myCreate}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('createMyMenu 方法出了问题：' + err))
      })
    })
  }
  //删除自定义菜单
  deleteMyMenu (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.menu.myDelete}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('deleteMyMenu 方法出了问题：' + err))
        })
    })
  }
  //测试个性化菜单匹配结果
  testMyMenu (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.menu.myTest}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject(' testMyMenu 方法出了问题：' + err))
        })
    })
  }
}

//测试
(async () => {

  const wechatApi = new Wechat();
```

```
    let data = await wechatApi.deleteMenu();
    console.log(data);
    data = await wechatApi.createMenu(menu);
    console.log(data);

})()

module.exports = Wechat;
```

10) index.js

```
const express = require('express');
const auth = require('./wechat/auth');
const app = express();

//接受微信服务器发送过来的请求 GET
//应用中间级，能够接受处理所有请求
app.use(auth());

app.listen(3000, err => {
    if (!err) console.log('服务器启动成功了~~~');
})
```

2.6 用户管理

2.6.1 目录结构

```
├── config/              # 配置目录
|    ├── index.js        # 存储配置信息
├── libs/                # 工具方法库
|    ├── util.js         # 解析字符串的工具方法
|    ├── api.js          # 定义接口的文件
├─── wechat/            # 核心功能库
|    ├── 1.jpg           # 需要上传素材图片
|    ├── 2.mp4           # 需要上传素材视频
```

```
|   ├── auth.js          # 回复消息功能
|   ├── menu.js          # 定义菜单
|   ├── reply.js         # 处理用户发送的信息
|   ├── template.js      # 返回给用户的信息模板
|   └── wechat.js        # 类 Wechat
├── app.js               # 入口启动文件
└── package.json         # 配置文件
```

2.6.2 代码

1) index.js

```
/*
  此模块用来储存关键的配置信息
 */
module.exports = {
  appID: 'wxc8e92f7ab70fbca0',
  appsecret: 'b4054e90b75787c78e0af50bf7fc3e87',
  token: 'atguiguHTML1208'   //token 要严格保密！！！
}
```

2) utils.js

```
/*
  工具函数
 */
//引入解析 xml 数据的库
const {parseString} = require('xml2js');

module.exports = {
  getUserDataAsync (req) {
    /*
      用户数据是通过流的方式发送，通过绑定 data 事件接受数据
     */
```

```
    return new Promise((resolve, reject) => {
      let data = '';
      req
        .on('data', userData => {
          //将流式数据全部拼接起来
          data += userData;
        })
        .on('end', () => {
          //确保数据全部获取了
          resolve(data);
        })
    })
  },
  parseXMLAsync (xmlData) {
    return new Promise((resolve, reject) => {
      parseString(xmlData, {trim: true}, (err, data) => {
        if (!err) {
          //解析成功了
          resolve(data);
        } else {
          //解析失败了
          reject('parseXMLAsync 方法出了问题：' + err);
        }
      })
    })
  },
  formatMessage (jsData) {
    const data = jsData.xml;
    //初始化一个空的对象
    let message = {};
    //判断数据是一个合法的数据
    if (typeof data === 'object') {
      //循环遍历对象中的所有数据
      for (let key in data) {
        //获取属性值
        let value = data[key];
        //过滤掉空的数据和空的数组
        if (Array.isArray(value) && value.length > 0) {
          //在新对象中添加属性和值
          message[key] = value[0];
        }
```

```
        }
      }
      //将格式化后的数据返回出去
      return message;
    }
}
```

3) api.js

```
/*
   所有接口的文件
 */
//提取出来的接口前缀
const prefix = 'https://api.weixin.qq.com/cgi-bin/';


module.exports = {
   accessToken: prefix + 'token?grant_type=client_credential',
   temporary: {
      upload: prefix + 'media/upload?',
      get: prefix + 'media/get?'
   },
   permanent: {
      uploadNews: prefix + 'material/add_news?',
      uploadImg: prefix + 'media/uploadimg?',
      uploadOthers: prefix + 'material/add_material?',
      get: prefix + 'material/get_material?',
      delete: prefix + 'material/del_material?',
      updateNews: prefix + 'material/update_news?',
      getCount: prefix + 'material/get_materialcount?',
```

```
    getMaterialList: prefix + 'material/batchget_material?'
  },
  menu: {
    create: prefix + 'menu/create?',
    delete: prefix + 'menu/delete?',
    get: prefix + 'menu/get?',
    myCreate: prefix + 'menu/addconditional?',
    myDelete: prefix + 'menu/delconditional?',
    myTest: prefix + 'menu/trymatch?'
  },
  tag: {
    create: prefix + 'tags/create?',
    get: prefix + 'tags/get?',
    update: prefix + 'tags/update?',
    delete: prefix + 'tags/delete?',
    getUsers: prefix + 'user/tag/get?'
  },
  user: {
    batchTag: prefix + 'tags/members/batchtagging?',
    unBatchTag: prefix + 'tags/members/batchuntagging?',
    getTags: prefix + 'tags/getidlist?',
    get: prefix + 'user/get?'
  }
}
```

4) auth.js

```
/*
  验证服务器的有效性:
```

```
    1、填写服务器配置(测试号管理页面)
      - URL 开发者服务器地址（保证能在互联网中能访问）
        通过 ngrok http 端口号 就得到一个网址
      - Token 参与微信签名的加密
    2、验证服务器地址的有效性
      - 将 timestamp、nonce、token 三个参数按照字典序排序
      - 将三个参数拼接在一起，进行 sha1 加密
      - 将加密后生成字符串和微信签名进行对比，
          如果相同说明成功，返回一个 echostr 给微信服务器，
          如果不相同，说明签名算法出了问题，配置不成功
 */
//引入配置对象
const config = require('../config');
//引入 sha1 加密模块
const sha1 = require('sha1');
//引入工具函数
const {getUserDataAsync, parseXMLAsync, formatMessage} = require('../libs/utils');
//引入 reply 模块
const reply = require('./reply');

module.exports = () => {

  return async (req, res, next) => {
    //接受微信服务器发送过来的请求参数
    // console.log(req.query);
    /*
      { signature: 'c4409bdd012bf28d8b4aabf7ac5847c5560d6cf0',    微信的加密签名（timestamp、nonce、token）
        echostr: '11283286178012191741',   随机字符串
        timestamp: '1529977721',            时间戳
        nonce: '1462949582' }               随机数字
     */
    //获取参与加密的参数
    const {signature, echostr, timestamp, nonce} = req.query;
    const {token} = config;
    /*// - 将 timestamp、nonce、token 三个参数按照字典序排序
    const arr = [timestamp, nonce, token].sort();
    // - 将三个参数拼接在一起，进行 sha1 加密
    const str = arr.join('');
    const sha1Str = sha1(str);*/
```

```
    //简写方式
    const sha1Str = sha1([timestamp, nonce, token].sort().join(''));

    /*
      微信服务器会主动发送两种方法的消息
        GET 请求，  验证服务器有效性
        POST 请求，微信服务器会将用户发送过来的消息转发到开发者服务器
上
     */
    if (req.method === 'GET') {
      // - 将加密后生成字符串和微信签名进行对比，
      if (sha1Str === signature) {
        //说明成功，返回 echostr 给微信服务器
        res.send(echostr);
      } else {
        //说明失败
        res.send('');
      }
    } else if (req.method === 'POST') {
      //接受用户发送过来消息
      // console.log(req.query);
      /*
        { signature: 'c67250097842aa50990259fa3df052eeffcb1cee',
          timestamp: '1530000513',
          nonce: '53405765',
          openid: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' }   //用户的 id
       */
      //验证消息是否来自于微信服务器
      if (sha1Str !== signature) {
        //说明消息不是来自于微信服务器
        //过滤掉非法请求
        res.send('error');
        return
      }

      //获取用户的消息，返回的数据格式是 xml
      const xmlData = await getUserDataAsync(req);
      // console.log(xmlData);
      /*
        <xml>
          <ToUserName><![CDATA[gh_4fe7faab4d6c]]></ToUserName>     //开
```

```
发者的 id

<FromUserName><![CDATA[oAsoR1iP-_D3LZIwNCnK8BFotmJc]]></FromUserName>
//用户的 openid
            <CreateTime>1530001191</CreateTime>     //消息发送时间
            <MsgType><![CDATA[text]]></MsgType>     //消息的类型
            <Content><![CDATA[666]]></Content>      //消息的具体内容
            <MsgId>6571305078611302153</MsgId>      //消息的 id
          </xml>
        */
        //将 xml 解析成 js 对象
        const jsData = await parseXMLAsync(xmlData);
        // console.log(jsData);
        /*
          { xml:
            { ToUserName: [ 'gh_4fe7faab4d6c' ],
              FromUserName: [ 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' ],
              CreateTime: [ '1530001675' ],
              MsgType: [ 'text' ],
              Content: [ '774' ],
              MsgId: [ '6571307157375473517' ] } }
        */
        //格式化数据
        const message = formatMessage(jsData);
        console.log(message);
        /*
          { ToUserName: 'gh_4fe7faab4d6c',
            FromUserName: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc',
            CreateTime: '1530002262',
            MsgType: 'text',
            Content: '888',
            MsgId: '6571309678521276386' }
        */

        //返回用户消息
        /*
          1. 假如服务器无法保证在五秒内处理并回复
          2. 回复 xml 数据中有多余的空格 *****
          3. 回复文本内容，中 options.content='' ***
          如果有以上现象，就会导致微信客户端中的报错：
            '该公众号提供服务出现故障，请稍后再试'
```

```
            */
            //设置回复用户消息的具体内容
            const replyMessage = await reply(message);
            console.log(replyMessage);    //Promise { <pending> }
            //返回响应给微信服务器
            res.send(replyMessage);

            /*//先返回一个空的响应给微信服务器
            res.send('');*/
        }
    }
}
```

5) menu.js

```
/*
  菜单的配置
 */

module.exports = {
  "button":[
    {
      "type":"click",
      "name":"首页",
      "key":"首页"
    },
    {
      "name":"二级菜单",
      "sub_button":[
        {
          "type":"view",
          "name":"跳转到硅谷",
          "url":"http://www.atguigu.com/"
        },
        {
          "type": "scancode_waitmsg",
          "name": "扫码带提示",
          "key": "扫码带提示"
        },
        {
```

```
            "type": "scancode_push",
            "name": "扫码推事件\ue348",
            "key": "扫码推事件"
        },
        {
            "type": "pic_sysphoto",
            "name": "系统拍照发图",
            "key": "系统拍照发图"
        },
        {
            "type": "pic_photo_or_album",
            "name": "拍照或者相册发图",
            "key": "拍照或者相册发图"
        }
    ]
},
{
    "name":"二菜单",
    "sub_button":[
        {
            "type": "pic_weixin",
            "name": "微信相册发图",
            "key": "微信相册发图"
        },
        {
            "type": "location_select",
            "name": "发送位置",
            "key": "发送位置"
        },
        {
            "type": "media_id",
            "name": "图片",
            "media_id": "1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE"
        },
        {
            "type": "view_limited",
            "name": "图文消息",
            "media_id": "1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas"
        }
    ]
}
```

```
  ]
}
```

6) reply.js

```
/*
  处理并分析用户发送的消息
  决定返回什么消息给用户
 */
const template = require('./template');
const Wechat = require('./wechat');

const wechatApi = new Wechat();

module.exports = async message => {

  //定义 options
  let options = {
    toUserName: message.FromUserName,
    fromUserName: message.ToUserName,
    createTime: Date.now(),
    msgType: 'text'
  }

  //设置回复用户消息的具体内容
  let content = '';

  //判断用户发送消息的类型和内容，决定返回什么消息给用户
```

```
  if (message.MsgType === 'text') {
    if (message.Content === '1') {
      content = '大吉大利，今晚吃鸡';
    } else if (message.Content === '2') {
      content = '落地成盒';
    } else if (message.Content === '3') {
      //回复图文消息
      content = [{
        title: 'Nodejs 开发',
        description: '微信公众号开发',
        picUrl:
'https://ss1.baidu.com/6ONXsjip0QIZ8tyhnq/it/u=1841004364,244945169&fm=58
&bpow=121&bpoh=75',
        url: 'http://nodejs.cn/'
      }, {
        title: 'web 前端',
        description: '这里有最新、最强的技术',
        picUrl:
'https://ss0.baidu.com/6ONWsjip0QIZ8tyhnq/it/u=1981851186,10620031&fm=58
&s=6183FE1ECDA569015C69A554030010F3&bpow=121&bpoh=75',
        url: 'http://www.atguigu.com/'
      }];
      options.msgType = 'news';

    } else if (message.Content === '4') {
      //上传一个多媒体素材
      const data = await wechatApi.uploadTemporaryMaterial('image',
```

```
'C:\\Users\\web\\Desktop\\图片\\1.jpg');
        console.log(data);
        /*
           { type: 'image',
             media_id:
'nT2v9ObOdrUjMU-kIAQrNTy1I3pZI0_ZO8yV6zB3N0KHVg92nlToTEpNXbkTcskV',
             created_at: 1530070685 }
         */
        //返回一个图片消息给用户
        options.msgType = 'image';
        options.mediaId = data.media_id;

    } else if (message.Content === '5') {
        //获取一个多媒体素材
        await
wechatApi.getTemporaryMaterial('nT2v9ObOdrUjMU-kIAQrNTy1I3pZI0_ZO8yV6zB
3N0KHVg92nlToTEpNXbkTcskV', __dirname + '/1.jpg');
        content = '获取多媒体素材成功了~~';

    } else if (message.Content === '6') {
        //上传图文素材中的图片
        const {url} = await wechatApi.uploadPermanentMaterial('pic',
'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(url);
//http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2tyQ
icPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0
        //上传缩略图
```

```
        const {media_id} = await wechatApi.uploadPermanentMaterial('image',
'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(media_id);
//1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE
        //上传图文素材
        const newsList = {
          "articles": [{
            "title": '大美女',
            "thumb_media_id": media_id,
            "author": '佚名',
            "digest": '这里有个大美女',
            "show_cover_pic": 1,
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
            '    <img src="' + url + '">\n' +
            '</body>\n' +
            '</html>',
            "content_source_url": 'http://www.atguigu.com'
          },{
            "title": '大美女',
            "thumb_media_id": media_id,
```

```
            "author": '佚名',
            "digest": '这里有个大美女',
            "show_cover_pic": 1,
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
            '    <img src="' + url + '">\n' +
            '</body>\n' +
            '</html>',
            "content_source_url": 'http://www.atguigu.com'
        }]
    }
    const data = await wechatApi.uploadPermanentMaterial('news', newsList);
    console.log(data); //{ media_id:
'1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas' }

    content = '上传永久素材成功了~~';

} else if (message.Content === '7') {
    //获取永久素材
    const newsData = await wechatApi.getPermanentMaterial('news',
'1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas');
```

```
        console.log(newsData);
        //返回给用户
        content = [];
        newsData.news_item.forEach(item => {
          content.push({
            title: item.title,
            description: item.digest,
            picUrl:
'http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2ty
QicPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0',
            url: item.url
          })
        })
        options.msgType = 'news';
      } else if (message.Content === '8') {
        //更新永久图文素材
        const body = {
          "media_id": '1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas',
          "index": 0,
          "articles": {
            "title": '大帅哥',
            "thumb_media_id":
'1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE',
            "author": '0315',
            "digest": '这是一个大帅哥',
            "show_cover_pic": 0,
            "content": 'hello 爱他硅谷',
```

```
            "content_source_url": 'http://www.baidu.com'
        }
    }
    let data = await wechatApi.updatePermanentNews(body);
    console.log(data); //{ errcode: 0, errmsg: 'ok' }
    //获取永久素材的数量
    data = await wechatApi.getPermanentCount();
    console.log(data);
    /*
      { voice_count: 1,
        video_count: 9,
        image_count: 48,
        news_count: 20 }
     */
    //获取指定素材的列表
    data = await wechatApi.getPermanentList({
      type: 'news',
      offset: 0,
      count: 20
    });
    console.log(data);
    //删除永久素材
    data = await
wechatApi.deletePermanentMaterial('1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas');
    console.log(data);
    //返回给用户
```

```
        content = '测试 api';
    } else if (message.Content === '9') {
        //创建标签
        let data1 = await wechatApi.createTag({
            "tag": {
                "name" : "VIP 客户"
            }
        });
        console.log(data1);    //{ tag: { id: 115, name: 'VIP 客户' } }
        let data = await wechatApi.createTag({
            "tag": {
                "name" : "普通客户"
            }
        });
        console.log(data); // { tag: { id: 116, name: '普通客户' } }
        //获取标签
        data = await wechatApi.getTag();
        console.log(data);    //
        //更新标签
        data = await wechatApi.updateTag({
            "tag": {
                "id": data1.tag.id,       //更新的指定标签
                "name": "钻石用户"     //更新的内容
            }
        });
        console.log(data); //{ errcode: 0, errmsg: 'ok' }
        //删除标签
```

```
    data = await wechatApi.deleteTag({
      tag: {
        "id": data1.tag.id
      }
    });
    console.log(data); //{ errcode: 0, errmsg: 'ok' }
  } else if (message.Content === '10') {
    //获取所有用户
    let data = await wechatApi.getUsers();
    console.log(data);
    /*
    {"total":2,
    "count":2,
    "data":{
      "openid":["OPENID1","OPENID2"]
    },
    "next_openid":"NEXT_OPENID"
    }
     */
    //批量为用户打标签
    data = await wechatApi.batchUserTags({
      openid_list: data.data.openid,
      tagid: 116
    });
    console.log(data);   //{ errcode: 0, errmsg: 'ok' }
    //获取标签下的粉丝列表
    data = await wechatApi.getTagUsers({
```

```
            tagid: 116
        });
        console.log(data);
        //获取粉丝下的标签列表
        data = await wechatApi.getUserTags({
            openid: message.FromUserName
        })
        console.log(data); //{ tagid_list: [ 116 ] }

    } else if (message.Content.match('爱')) {
        //模糊匹配，只要包含爱
        content = '我爱你~';
    } else {
        content = '您在说啥，我听不懂';
    }
} else if (message.MsgType === 'image') {
    content = '您的图片地址为：' + message.PicUrl;
} else if (message.MsgType === 'voice') {
    content = '语音识别结果：' + message.Recognition;
} else if (message.MsgType === 'video') {
    content = '接受了视频消息';
} else if (message.MsgType === 'shortvideo') {
    content = '接受了小视频消息';
} else if (message.MsgType === 'location') {
    content = '纬度：' + message.Location_X + '经度：' + message.Location_Y
        + '缩放大小：' + message.Scale + '详情：' + message.Label;
} else if (message.MsgType === 'link') {
```

```
        content = '标题：' + message.Title + ' 描述：' + message.Description + ' 网址：
' + message.Url;
    } else if (message.MsgType === 'event') {
        if (message.Event === 'subscribe') {
            //用户订阅事件
            content = '欢迎您的订阅~';
            if (message.EventKey) {
                //扫描带参数的二维码的订阅事件
                content = '欢迎您扫二维码的关注';
            }
        } else if (message.Event === 'SCAN') {
            //已经关注了公众号，在扫描带参数二维码进入公众号
            content = '已经关注了公众号，在扫描带参数二维码进入公众号';
        } else if (message.Event === 'unsubscribe') {
            //用户取消关注
            console.log('无情取关~');
        } else if (message.Event === 'LOCATION') {
            //用户进行会话时，上报一次地理位置消息
            content = '纬度：' + message.Latitude + ' 经度：' + message.Longitude + ' 精
度：' + message.Precision;
        } else if (message.Event === 'CLICK') {
            content = '点击了菜单~~~';
        } else if (message.Event === 'VIEW') {
            //用户点击菜单，跳转到其他链接
            console.log('用户点击菜单，跳转到其他链接');
        } else if (message.Event === 'scancode_push') {
            //用户点击菜单，扫二维码推送
```

```
            console.log(message.EventKey + '    ' + message.ScanCodeInfo + '    ' +
message.ScanType + ' ' + message.ScanResult);
        } else if (message.Event === 'scancode_waitmsg') {
            //用户点击菜单，扫二维码推送
            console.log(message.EventKey + '    ' + message.ScanCodeInfo + '    ' +
message.ScanType + ' ' + message.ScanResult);
        } else if (message.Event === 'pic_sysphoto') {
            //用户点击菜单，扫二维码推送
            console.log(message.SendPicsInfo + '    ' + message.PicMd5Sum);
        } else if (message.Event === 'pic_photo_or_album') {
            //用户点击菜单，扫二维码推送
            console.log(message.SendPicsInfo + '    ' + message.PicMd5Sum);
        } else if (message.Event === 'pic_weixin') {
            //用户点击菜单，扫二维码推送
            console.log(message.SendLocationInfo + '    ' + message.Poiname);
        }
    }


    //将最终回复消息内容添加到 options 中
    options.content = content;
    //将最终的 xml 数据返回出去
    return template(options);
}
```

7) template.js

```
/*
  设置回复用户的 6 种消息内容
 */
```

```
module.exports = options => {

  //回复用户消息
  let replyMessage = '<xml>' +
    '<ToUserName><![CDATA[' + options.toUserName + ']]></ToUserName>' +
    '<FromUserName><![CDATA[' + options.fromUserName +
']]></FromUserName>' +
    '<CreateTime>' + options.createTime + '</CreateTime>' +
    '<MsgType><![CDATA[' + options.msgType + ']]></MsgType>';

  if (options.msgType === 'text') {
    replyMessage += '<Content><![CDATA[' + options.content + ']]></Content>';
  } else if (options.msgType === 'image') {
    replyMessage += '<Image><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Image>';
  } else if (options.msgType === 'voice') {
    replyMessage += '<Voice><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Voice>';
  } else if (options.msgType === 'video') {
    replyMessage += '<Video>' +
      '<MediaId><![CDATA[' + options.mediaId + ']]></MediaId>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '</Video>';
  } else if (options.msgType === 'music') {
    replyMessage += '<Music>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '<MusicUrl><![CDATA[' + options.musicUrl + ']]></MusicUrl>' +
      '<HQMusicUrl><![CDATA[' + options.hqMusicUrl + ']]></HQMusicUrl>' +
      '<ThumbMediaId><![CDATA[' + options.mediaId + ']]></ThumbMediaId>' +
      '</Music>';
  } else if (options.msgType === 'news') {
    replyMessage += '<ArticleCount>' + options.content.length + '</ArticleCount>'
+
      '<Articles>';

    options.content.forEach(item => {
      replyMessage += '<item>' +
        '<Title><![CDATA[' + item.title + ']]></Title>' +
        '<Description><![CDATA[' + item.description + ']]></Description>' +
```

```
            '<PicUrl><![CDATA[' + item.picUrl + ']]></PicUrl>' +
            '<Url><![CDATA[' + item.url + ']]></Url>' +
            '</item>';
        })

        replyMessage += '</Articles>';
    }

    replyMessage += '</xml>';
    //将拼接好回复用户的数据返回出去
    return replyMessage;

}
```

8) wechat.js

```
/*
    获取 access_token:
        全局唯一的接口调用凭据，今后使用微信的接口基本上都需要携带上这个参数
        2 小时需要更新一次，提前 5 分钟刷新

        请求地址：

https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
        请求方式：
          GET

        设计思路：
          首先发送请求获取凭据，保存为一个唯一的文件
          然后后面请求先去本地文件读取凭据
            判断凭据是否过期
              如果没有过期，直接使用
              如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件

        总结：
          先去本地查看有没有指定文件（readAccessToken）
            如果有（之前请求过凭据）
              判断凭据是否过期(isValidAccessToken)
```

```
                如果没有过期，直接使用
                如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件(getAccessToken、saveAccessToken)
            如果没有（之前都没有请求过凭据）
             发送请求获取凭据，保存为一个唯一的文件

 */
//引入配置对象
const {appID, appsecret} = require('../config');
//引入发送 http 请求的库
const rp = require('request-promise-native');
const request = require('request');
//引入 fs 模块
const {readFile, writeFile, createReadStream, createWriteStream} = require('fs');
//引入接口文件
const api = require('../libs/api');
//引入菜单文件
const menu = require('./menu');

class Wechat {
  getAccessToken () {
    //定义请求地址
    const url = `${api.accessToken}&appid=${appID}&secret=${appsecret}`;
    /*
      问题：需要将回调函数中的数据返回出去？
      解决：用 promise 解决

      所有的异步操作，都应该包装一层 promise，让这个异步操作执行完毕之后，再去执行后面的代码
      简化： 所有的异步操作，都应该包装一层 promise
     */
    return new Promise((resolve, reject) => {
      //发送 http 请求
      //下载 request-promise-native   request
      rp({method: 'GET', json: true, url})
        .then(res => {
          //请求成功的状态
          // console.log(res);
          //重新赋值凭据的过期时间 ：  当前时间 + (7200 - 5 分钟) * 1000
          res.expires_in = Date.now() + (res.expires_in - 300) * 1000;
          // console.log(res);
```

```
            resolve(res);
          })
          .catch(err => {
            //请求失败
            reject('getAccessToken 方法出了问题：' + err);
          })
    })
  }
  saveAccessToken (data) {
    /*
      问题：writeFile 方法会将对象转化为字符串
      解决：我将对象转化为 json 字符串
     */
    data = JSON.stringify(data);
    return new Promise((resolve, reject) => {
      //将凭据保存为一个文件
      writeFile('accessToken.txt', data, err => {
        if (!err) {
          //写入成功
          resolve();
        } else {
          //写入失败
          reject('saveAccessToken 方法出了问题：' + err);
        }
      })
    })
  }
  readAccessToken () {
    return new Promise((resolve, reject) => {
      //将凭据读取出来
      readFile('accessToken.txt', (err, data) => {
        if (!err) {
          //将读取的 Buffer 数据转化为 json 字符串
          data = data.toString();
          //将 json 字符串转化为对象
          data = JSON.parse(data);
          //读取成功
          resolve(data);
        } else {
          //读取失败
          reject('readAccessToken 方法出了问题：' + err);
```

```
            }
        })
    })
}
isValidAccessToken (data) {
    /*
        判断凭据是否过期
            true    凭据没有过期
            false   凭据过期了
     */
    //过滤非法的数据
    if (!data || !data.access_token || !data.expires_in) return false;
    //判断凭据是否过期
    /*if (data.expires_in > Date.now()) {
        //如果凭据的过期时间大于当前时间，说明没有过期
        return true
    } else {
        //如果凭据的过期时间小于当前时间，说明过期了
        return false
    }*/
    //简写方式
    return data.expires_in > Date.now();
}
fetchAccessToken () {
    //优化操作,优化不去执行读取文件操作
    if (this.access_token && this.expires_in && this.isValidAccessToken(this)) {
        //说明 this 有凭据和过期时间，并且凭据未过期
        return Promise.resolve({access_token: this.access_token, expires_in:
this.expires_in});
    }

    return this.readAccessToken()
        .then(async res => {
            //判断凭据是否过期(isValidAccessToken)
            if (this.isValidAccessToken(res)) {
                //没有过期，直接使用
                return Promise.resolve(res);
            } else {
                //重新发送请求获取凭据
                const data = await this.getAccessToken();
                //保存下来
```

```
                await this.saveAccessToken(data);
                //将请求回来的凭据返回出去
                return Promise.resolve(data);
            }
        })
        .catch(async err => {
            console.log(err);
            //重新发送请求获取凭据
            const data = await this.getAccessToken();
            //保存下来
            await this.saveAccessToken(data);
            //将请求回来的凭据返回出去
            return Promise.resolve(data);
        })
        .then(res => {
            //将其请求回来的凭据和过期时间挂载到 this 上
            this.access_token = res.access_token;
            this.expires_in = res.expires_in;
            //指定 fetchAccessToken 方法返回值
            return Promise.resolve(res);
        })
    }

    //上传临时素材
    uploadTemporaryMaterial (type, filePath) {
        /*
            type:  上传多媒体文件的类型
            filePath:  上传多媒体文件的路径
         */
        return new Promise((resolve, reject) => {
            //获取 access_token
            this.fetchAccessToken()
                .then(res => {
                    //定义请求的地址
                    const url =
`${api.temporary.upload}access_token=${res.access_token}&type=${type}`;
                    //定义要传输过去的媒体数据
                    const formData = {
                        media: createReadStream(filePath)
                    }
                    //发送请求
```

```
            rp({method: 'POST', json: true, url, formData})
                .then(res => {
                    //将请求回来的数据返回出去
                    resolve(res);
                })
                .catch(err => {
                    reject('uploadTemporaryMaterial 方法出了问题：' + err);
                })
        })
    })


}
//获取临时素材
getTemporaryMaterial (mediaId, filePath, isVideo = false) {
    /*
      mediaId: 要获取的素材 id
      filePath：要保存媒体文件的路径
      isVideo: 可选值
     */
    return new Promise((resolve, reject) => {
      //获取 access_token
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          const url =
`${api.temporary.get}access_token=${res.access_token}&media_id=${mediaId}`;
          //发送请求
          if (isVideo) {
            //如果是视频消息素材，就返回一个 url 地址
            rp({method: 'GET', json: true, url})
              .then(res => resolve(res))
              .catch(err => reject('getTemporaryMaterial 方法出了问题：' + err))
          } else {
            //如果不是视频消息素材，就返回一个文件接收
            request
              .get(url)
              .pipe(createWriteStream(filePath))
              .once('close', () => {
                //说明文件下载成功了~
                resolve();
```

```
                    })
                }
            })
        })

    }
    //上传永久素材
    uploadPermanentMaterial (type, material, description) {
        /*
          type: 可以区分我通过什么方式上传素材
          material: 上传素材的路径/请求体中的内容
          description: 针对于视频素材上传
         */
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    //定义请求地址
                    let url = '';
                    //定义发送请求的配置对象
                    let options = {
                        method: 'POST',
                        json: true
                    }
                    if (type === 'news') {
                        //上传图文消息
                        url =
`${api.permanent.uploadNews}access_token=${res.access_token}`;
                        options.body = material;
                    } else if (type === 'pic') {
                        //上传图文消息的图片
                        url =
`${api.permanent.uploadImg}access_token=${res.access_token}`;
                        options.formData = {
                            media: createReadStream(material)
                        }
                    } else {
                        //上传其他素材
                        url =
`${api.permanent.uploadOthers}access_token=${res.access_token}&type=${type}`;
                        options.formData = {
                            media: createReadStream(material)
```

```
                }
                if (type === 'video') {
                    options.body = description;
                }
            }
            //将请求地址放到配置对象中
            options.url = url;
            //发送请求
            rp(options)
                .then(res => resolve(res))
                .catch(err => reject('uploadPermanentMaterial 方法出了问题:' +
err))
        })
    })
  }
  //获取永久素材
  getPermanentMaterial (type, mediaId, filePath) {
    /*
      type: 用来如何接受数据
      mediaId: 获取的媒体素材 id
      filePath: 保存的媒体素材的位置
     */
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          const url = `${api.permanent.get}access_token=${res.access_token}`;
          //定义请求体中的数据
          const body = {
            media_id: mediaId
          }
          if (type === 'news' || 'video') {
            rp({method: 'POST', json: true, url, body})
              .then(res => resolve(res))
              .catch(err => reject('getPermanentMaterial 方法出了问题:' + err))
          } else {
            request({method: 'POST', json: true, url, body})
              .pipe(createWriteStream(filePath))
              .once('close', () => resolve())
          }
        })
```

```
        })
    }
    //删除永久素材
    deletePermanentMaterial (mediaId) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    const url =
`${api.permanent.delete}access_token=${res.access_token}`;
                    const body = {
                        media_id: mediaId
                    }
                    rp({method: 'POST', json: true, url, body})
                        .then(res => resolve(res))
                        .catch(err => reject('deletePermanentMaterial 方法出了问题：' +
err))
                })
        })
    }
    //更新永久图文消息素材
    updatePermanentNews (body) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    const url =
`${api.permanent.updateNews}access_token=${res.access_token}`;
                    rp({method: 'POST', json: true, url, body})
                        .then(res => resolve(res))
                        .catch(err => reject('updatePermanentNews 方法出了问题：' + err))
                })
        })
    }
    //获取永久素材数量
    getPermanentCount () {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    const url =
`${api.permanent.getCount}access_token=${res.access_token}`;
                    rp({method: 'GET', json: true, url})
                        .then(res => resolve(res))
```

```
                .catch(err => reject('getPermanentCount 方法出了问题：' + err))
            })
        })
    }
    //获取永久素材列表
    getPermanentList (body) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
            .then(res => {
                const url =
`${api.permanent.getMaterialList}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                .then(res => resolve(res))
                .catch(err => reject('getPermanentList 方法出了问题：' + err))
            })
        })
    }
    //创建菜单
    createMenu (body) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.create}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                .then(res => resolve(res))
                .catch(err => reject('createMenu 方法出了问题：' + err))
            })
        })
    }
    //删除菜单
    deleteMenu () {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.delete}access_token=${res.access_token}`;
                rp({method: 'GET', json: true, url})
                .then(res => resolve(res))
                .catch(err => reject('deleteMenu 方法出了问题：' + err))
            })
        })
    }
```

```
//获取菜单的配置
getMenu () {
  return new Promise((resolve, reject) => {
    this.fetchAccessToken()
      .then(res => {
        const url = `${api.menu.get}access_token=${res.access_token}`;
        rp({method: 'GET', json: true, url})
          .then(res => resolve(res))
          .catch(err => reject('getMenu 方法出了问题：' + err))
      })
  })
}
//创建自定义菜单
createMyMenu (body) {
  return new Promise((resolve, reject) => {
    this.fetchAccessToken()
      .then(res => {
        const url = `${api.menu.myCreate}access_token=${res.access_token}`;
        rp({method: 'POST', json: true, url, body})
          .then(res => resolve(res))
          .catch(err => reject('createMyMenu 方法出了问题：' + err))
      })
  })
}
//删除自定义菜单
deleteMyMenu (body) {
  return new Promise((resolve, reject) => {
    this.fetchAccessToken()
      .then(res => {
        const url = `${api.menu.myDelete}access_token=${res.access_token}`;
        rp({method: 'POST', json: true, url, body})
          .then(res => resolve(res))
          .catch(err => reject('deleteMyMenu 方法出了问题：' + err))
      })
  })
}
//测试个性化菜单匹配结果
testMyMenu (body) {
  return new Promise((resolve, reject) => {
    this.fetchAccessToken()
      .then(res => {
```

```
            const url = `${api.menu.myTest}access_token=${res.access_token}`;
            rp({method: 'POST', json: true, url, body})
              .then(res => resolve(res))
              .catch(err => reject(' testMyMenu 方法出了问题：' + err))
          })
      })
  }
  //创建标签
  createTag (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.tag.create}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('createTag 方法出了问题：' + err))
        })
    })
  }
  //获取标签
  getTag () {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.tag.get}access_token=${res.access_token}`;
          rp({method: 'GET', json: true, url})
            .then(res => resolve(res))
            .catch(err => reject('getTag 方法出了问题：' + err))
        })
    })
  }
  //更新标签
  updateTag (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.tag.update}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('updateTag 方法出了问题：' + err))
        })
```

```
    })
  }
  //更新标签
  deleteTag (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.tag.delete}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('deleteTag 方法出了问题：' + err))
        })
    })
  }
  //获取标签下的粉丝列表
  getTagUsers (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.tag.getUsers}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('getTagUsers 方法出了问题：' + err))
        })
    })
  }
  //批量为用户打标签
  batchUserTags (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.user.batchTag}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('batchUserTags 方法出了问题：' + err))
        })
    })
  }
  //批量为用户取消标签
  unBatchUserTags (body) {
    return new Promise((resolve, reject) => {
```

```
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.user.unBatchTag}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                    .then(res => resolve(res))
                    .catch(err => reject('unBatchUserTags 方法出了问题：' + err))
            })
    })
}
//获取用户下所有的标签
getUserTags (body) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.user.getTags}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                    .then(res => resolve(res))
                    .catch(err => reject('getUserTags 方法出了问题：' + err))
            })
    })
}
//获取所有用户列表
getUsers (next_openid) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                let url = `${api.user.get}access_token=${res.access_token}`;
                if (next_openid) {
                    url += '&next_openid=' + next_openid;
                }
                rp({method: 'GET', json: true, url})
                    .then(res => resolve(res))
                    .catch(err => reject('getUsers 方法出了问题：' + err))
            })
    })
}

}

//测试
/*(async () => {
```

```
    const wechatApi = new Wechat();

    let data = await wechatApi.deleteMenu();
    console.log(data);
    data = await wechatApi.createMenu(menu);
    console.log(data);

})()*/

module.exports = Wechat;
```

11) index.js

```
const express = require('express');
const auth = require('./wechat/auth');
const app = express();

//接受微信服务器发送过来的请求 GET
//应用中间级，能够接受处理所有请求
app.use(auth());

app.listen(3000, err => {
  if (!err) console.log('服务器启动成功了~~~');
})
```

2.7 群发消息

2.7.1 目录结构

```
├── config/              # 配置目录
|    ├── index.js        # 存储配置信息
├── libs/                # 工具方法库
|    ├── util.js         # 解析字符串的工具方法
|    ├── api.js          # 定义接口的文件
├─── wechat/            # 核心功能库
|    ├── 1.jpg           # 需要上传素材图片
```

```
|   ├── 2.mp4          # 需要上传素材视频
|   ├── auth.js        # 回复消息功能
|   ├── menu.js        # 定义菜单
|   ├── reply.js       # 处理用户发送的信息
|   ├── template.js    # 返回给用户的信息模板
|   ├── wechat.js      # 类 Wechat
├── app.js             # 入口启动文件
├── package.json       # 配置文件
```

2.7.2 代码

2) index.js

```
/*
  此模块用来储存关键的配置信息
 */
module.exports = {
  appID: 'wxc8e92f7ab70fbca0',
  appsecret: 'b4054e90b75787c78e0af50bf7fc3e87',
  token: 'atguiguHTML1208'   //token 要严格保密！！！
}
```

3) utils.js

```
/*
  工具函数
 */
//引入解析 xml 数据的库
const {parseString} = require('xml2js');

module.exports = {
  getUserDataAsync (req) {
    /*
      用户数据是通过流的方式发送，通过绑定 data 事件接受数据
```

```
    */
    return new Promise((resolve, reject) => {
      let data = '';
      req
        .on('data', userData => {
          //将流式数据全部拼接起来
          data += userData;
        })
        .on('end', () => {
          //确保数据全部获取了
          resolve(data);
        })
    })
  },
  parseXMLAsync (xmlData) {
    return new Promise((resolve, reject) => {
      parseString(xmlData, {trim: true}, (err, data) => {
        if (!err) {
          //解析成功了
          resolve(data);
        } else {
          //解析失败了
          reject('parseXMLAsync 方法出了问题：' + err);
        }
      })
    })
  },
  formatMessage (jsData) {
    const data = jsData.xml;
    //初始化一个空的对象
    let message = {};
    //判断数据是一个合法的数据
    if (typeof data === 'object') {
      //循环遍历对象中的所有数据
      for (let key in data) {
        //获取属性值
        let value = data[key];
        //过滤掉空的数据和空的数组
        if (Array.isArray(value) && value.length > 0) {
          //在新对象中添加属性和值
          message[key] = value[0];
```

```
            }
          }
        }
        //将格式化后的数据返回出去
        return message;
    }
}
```

4) api.js

```
/*
   所有接口的文件
 */
//提取出来的接口前缀
const prefix = 'https://api.weixin.qq.com/cgi-bin/';


module.exports = {
   accessToken: prefix + 'token?grant_type=client_credential',
   temporary: {
      upload: prefix + 'media/upload?',
      get: prefix + 'media/get?'
   },
   permanent: {
      uploadNews: prefix + 'material/add_news?',
      uploadImg: prefix + 'media/uploadimg?',
      uploadOthers: prefix + 'material/add_material?',
      get: prefix + 'material/get_material?',
      delete: prefix + 'material/del_material?',
      updateNews: prefix + 'material/update_news?',
      getCount: prefix + 'material/get_materialcount?',
      getMaterialList: prefix + 'material/batchget_material?'
   },
   menu: {
      create: prefix + 'menu/create?',
      delete: prefix + 'menu/delete?',
      get: prefix + 'menu/get?',
      myCreate: prefix + 'menu/addconditional?',
```

```
    myDelete: prefix + 'menu/delconditional?',
    myTest: prefix + 'menu/trymatch?'
  },
  tag: {
    create: prefix + 'tags/create?',
    get: prefix + 'tags/get?',
    update: prefix + 'tags/update?',
    delete: prefix + 'tags/delete?',
    getUsers: prefix + 'user/tag/get?'
  },
  user: {
    batchTag: prefix + 'tags/members/batchtagging?',
    unBatchTag: prefix + 'tags/members/batchuntagging?',
    getTags: prefix + 'tags/getidlist?',
    get: prefix + 'user/get?'
  },
  sendAll: {
    tag: prefix + 'message/mass/sendall?',
    users: prefix + 'message/mass/send?'
  }
}
```

5) auth.js

```
/*
  验证服务器的有效性:

    1、填写服务器配置(测试号管理页面)
      - URL 开发者服务器地址（保证能在互联网中能访问）
        通过 ngrok http 端口号 就得到一个网址
      - Token 参与微信签名的加密
    2、验证服务器地址的有效性
      - 将 timestamp、nonce、token 三个参数按照字典序排序
      - 将三个参数拼接在一起，进行 sha1 加密
      - 将加密后生成字符串和微信签名进行对比，
          如果相同说明成功，返回一个 echostr 给微信服务器，
          如果不相同，说明签名算法出了问题，配置不成功
 */
//引入配置对象
const config = require('../config');
//引入 sha1 加密模块
const sha1 = require('sha1');
//引入工具函数
const {getUserDataAsync, parseXMLAsync, formatMessage} = require('../libs/utils');
//引入 reply 模块
const reply = require('./reply');

module.exports = () => {

  return async (req, res, next) => {
    //接受微信服务器发送过来的请求参数
    // console.log(req.query);
    /*
      { signature: 'c4409bdd012bf28d8b4aabf7ac5847c5560d6cf0',    微信的加密签名（timestamp、nonce、token）
        echostr: '11283286178012191741',   随机字符串
        timestamp: '1529977721',           时间戳
        nonce: '1462949582' }              随机数字
     */
    //获取参与加密的参数
    const {signature, echostr, timestamp, nonce} = req.query;
    const {token} = config;
```

```
    /*// - 将 timestamp、nonce、token 三个参数按照字典序排序
    const arr = [timestamp, nonce, token].sort();
    // - 将三个参数拼接在一起，进行 sha1 加密
    const str = arr.join('');
    const sha1Str = sha1(str);*/
    //简写方式
    const sha1Str = sha1([timestamp, nonce, token].sort().join(''));

    /*
      微信服务器会主动发送两种方法的消息
        GET 请求， 验证服务器有效性
        POST 请求，微信服务器会将用户发送过来的消息转发到开发者服务器
上
     */
    if (req.method === 'GET') {
      // - 将加密后生成字符串和微信签名进行对比，
      if (sha1Str === signature) {
        //说明成功，返回 echostr 给微信服务器
        res.send(echostr);
      } else {
        //说明失败
        res.send('');
      }
    } else if (req.method === 'POST') {
      //接受用户发送过来消息
      // console.log(req.query);
      /*
        { signature: 'c67250097842aa50990259fa3df052eeffcb1cee',
          timestamp: '1530000513',
          nonce: '53405765',
          openid: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' }   //用户的 id
       */
      //验证消息是否来自于微信服务器
      if (sha1Str !== signature) {
        //说明消息不是来自于微信服务器
        //过滤掉非法请求
        res.send('error');
        return
      }

      //获取用户的消息，返回的数据格式是 xml
```

```
        const xmlData = await getUserDataAsync(req);
        // console.log(xmlData);
        /*
          <xml>
            <ToUserName><![CDATA[gh_4fe7faab4d6c]]></ToUserName>     //开
发者的 id

<FromUserName><![CDATA[oAsoR1iP-_D3LZIwNCnK8BFotmJc]]></FromUserName>
//用户的 openid
            <CreateTime>1530001191</CreateTime>     //消息发送时间
            <MsgType><![CDATA[text]]></MsgType>     //消息的类型
            <Content><![CDATA[666]]></Content>      //消息的具体内容
            <MsgId>6571305078611302153</MsgId>      //消息的 id
          </xml>
         */
        //将 xml 解析成 js 对象
        const jsData = await parseXMLAsync(xmlData);
        // console.log(jsData);
        /*
          { xml:
           { ToUserName: [ 'gh_4fe7faab4d6c' ],
             FromUserName: [ 'oAsoR1iP-_D3LZIwNCnK8BFotmJc' ],
             CreateTime: [ '1530001675' ],
             MsgType: [ 'text' ],
             Content: [ '774' ],
             MsgId: [ '6571307157375473517' ] } }
         */
        //格式化数据
        const message = formatMessage(jsData);
        console.log(message);
        /*
          { ToUserName: 'gh_4fe7faab4d6c',
            FromUserName: 'oAsoR1iP-_D3LZIwNCnK8BFotmJc',
            CreateTime: '1530002262',
            MsgType: 'text',
            Content: '888',
            MsgId: '6571309678521276386' }
         */

        //返回用户消息
        /*
```

```
            1. 假如服务器无法保证在五秒内处理并回复
            2. 回复 xml 数据中有多余的空格 *****
            3. 回复文本内容，中 options.content='' ***
            如果有以上现象，就会导致微信客户端中的报错：
              '该公众号提供服务出现故障，请稍后再试'
          */
        //设置回复用户消息的具体内容
        const replyMessage = await reply(message);
        console.log(replyMessage);    //Promise { <pending> }
        //返回响应给微信服务器
        res.send(replyMessage);

        /*//先返回一个空的响应给微信服务器
        res.send('');*/
      }
    }
}
```

6) menu.js

```
/*
  菜单的配置
 */

module.exports = {
  "button":[
    {
      "type":"click",
      "name":"首页",
      "key":"首页"
    },
    {
      "name":"二级菜单",
      "sub_button":[
        {
          "type":"view",
          "name":"跳转到硅谷",
          "url":"http://www.atguigu.com/"
        },
        {
```

```
            "type": "scancode_waitmsg",
            "name": "扫码带提示",
            "key": "扫码带提示"
        },
        {
            "type": "scancode_push",
            "name": "扫码推事件\ue348",
            "key": "扫码推事件"
        },
        {
            "type": "pic_sysphoto",
            "name": "系统拍照发图",
            "key": "系统拍照发图"
        },
        {
            "type": "pic_photo_or_album",
            "name": "拍照或者相册发图",
            "key": "拍照或者相册发图"
        }
    ]
},
{
    "name":"二菜单",
    "sub_button":[
        {
            "type": "pic_weixin",
            "name": "微信相册发图",
            "key": "微信相册发图"
        },
        {
            "type": "location_select",
            "name": "发送位置",
            "key": "发送位置"
        },
        {
            "type": "media_id",
            "name": "图片",
            "media_id": "1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE"
        },
        {
            "type": "view_limited",
```

```
            "name": "图文消息",
            "media_id": "1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas"
          }
        ]
      }
    ]
}
```

7) reply.js

```
/*
  处理并分析用户发送的消息
  决定返回什么消息给用户
 */
const template = require('./template');
const Wechat = require('./wechat');

const wechatApi = new Wechat();

module.exports = async message => {

  //定义 options
  let options = {
    toUserName: message.FromUserName,
    fromUserName: message.ToUserName,
    createTime: Date.now(),
    msgType: 'text'
  }

  //设置回复用户消息的具体内容
```

```
let content = '';


//判断用户发送消息的类型和内容，决定返回什么消息给用户
if (message.MsgType === 'text') {
    if (message.Content === '1') {
        content = '大吉大利，今晚吃鸡';
    } else if (message.Content === '2') {
        content = '落地成盒';
    } else if (message.Content === '3') {
        //回复图文消息
        content = [{
            title: 'Nodejs 开发',
            description: '微信公众号开发',
            picUrl:
'https://ss1.baidu.com/6ONXsjip0QIZ8tyhnq/it/u=1841004364,244945169&fm=58
&bpow=121&bpoh=75',
            url: 'http://nodejs.cn/'
        }, {
            title: 'web 前端',
            description: '这里有最新、最强的技术',
            picUrl:
'https://ss0.baidu.com/6ONWsjip0QIZ8tyhnq/it/u=1981851186,10620031&fm=58
&s=6183FE1ECDA569015C69A554030010F3&bpow=121&bpoh=75',
            url: 'http://www.atguigu.com/'
        }];
        options.msgType = 'news';
```

```
    } else if (message.Content === '4') {
        //上传一个多媒体素材
        const data = await wechatApi.uploadTemporaryMaterial('image',
'C:\\Users\\web\\Desktop\\图片\\1.jpg');
        console.log(data);
        /*
          { type: 'image',
            media_id:
'nT2v9ObOdrUjMU-kIAQrNTy1I3pZI0_ZO8yV6zB3N0KHVg92nlToTEpNXbkTcskV',
            created_at: 1530070685 }
         */
        //返回一个图片消息给用户
        options.msgType = 'image';
        options.mediaId = data.media_id;

    } else if (message.Content === '5') {
        //获取一个多媒体素材
        await
wechatApi.getTemporaryMaterial('nT2v9ObOdrUjMU-kIAQrNTy1I3pZI0_ZO8yV6zB
3N0KHVg92nlToTEpNXbkTcskV', __dirname + '/1.jpg');
        content = '获取多媒体素材成功了~~';

    } else if (message.Content === '6') {
        //上传图文素材中的图片
        const {url} = await wechatApi.uploadPermanentMaterial('pic',
'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(url);
```

```
//http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2tyQicPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0
        //上传缩略图
        const {media_id} = await wechatApi.uploadPermanentMaterial('image', 'C:\\Users\\web\\Desktop\\图片\\9.jpg');
        console.log(media_id);
//1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE
        //上传图文素材
        const newsList = {
          "articles": [{
            "title": '大美女',
            "thumb_media_id": media_id,
            "author": '佚名',
            "digest": '这里有个大美女',
            "show_cover_pic": 1,
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
            '    <img src="' + url + '">\n' +
            '</body>\n' +
            '</html>',
            "content_source_url": 'http://www.atguigu.com'
```

```
        },{
            "title": '大美女',
            "thumb_media_id": media_id,
            "author": '佚名',
            "digest": '这里有个大美女',
            "show_cover_pic": 1,
            "content": '<!DOCTYPE html>\n' +
            '<html lang="en">\n' +
            '<head>\n' +
            '    <meta charset="UTF-8">\n' +
            '    <title>test</title>\n' +
            '</head>\n' +
            '<body>\n' +
            '    <h1>这是一个测试</h1>\n' +
            '    <img src="' + url + '">\n' +
            '</body>\n' +
            '</html>',
            "content_source_url": 'http://www.atguigu.com'
        }]
    }
    const data = await wechatApi.uploadPermanentMaterial('news', newsList);
    console.log(data); //{ media_id:
'1_821D3VHxMTbMuZ5-DSoDEugAPDwV6w58xZCxPy7LE' }

    content = '上传永久素材成功了~~';

} else if (message.Content === '7') {
```

```
        //获取永久素材
        const newsData = await wechatApi.getPermanentMaterial('news',
'1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas');
        console.log(newsData);
        //返回给用户
        content = [];
        newsData.news_item.forEach(item => {
          content.push({
            title: item.title,
            description: item.digest,
            picUrl:
'http://mmbiz.qpic.cn/mmbiz_jpg/l6hEPf9t1fFUdjGSkqfWVexQ9USA1g0Gec4G2ty
QicPUrDNaEL0pxkiaJf1mQfh8PGEia77DW0HhvJJm7U1YMw2lw/0',
            url: item.url
          })
        })
        options.msgType = 'news';
      } else if (message.Content === '8') {
        //更新永久图文素材
        const body = {
          "media_id": '1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwhrTOGas',
          "index": 0,
          "articles": {
            "title": '大帅哥',
            "thumb_media_id":
'1_821D3VHxMTbMuZ5-DSoFvgSbQCngMIAwITtEBCZJE',
            "author": '0315',
```

```
            "digest": '这是一个大帅哥',
            "show_cover_pic": 0,
            "content": 'hello  爱他硅谷',
            "content_source_url": 'http://www.baidu.com'
        }
    }
    let data = await wechatApi.updatePermanentNews(body);
    console.log(data); //{ errcode: 0, errmsg: 'ok' }
    //获取永久素材的数量
    data = await wechatApi.getPermanentCount();
    console.log(data);
    /*
        { voice_count: 1,
            video_count: 9,
            image_count: 48,
            news_count: 20 }
     */
    //获取指定素材的列表
    data = await wechatApi.getPermanentList({
        type: 'news',
        offset: 0,
        count: 20
    });
    console.log(data);
    //删除永久素材
    // data = await
wechatApi.deletePermanentMaterial('1_821D3VHxMTbMuZ5-DSoIBjlGzb2e9R3jGwh
```

```
rTOGas');
        // console.log(data);
        //返回给用户
        content = '测试 api';
    } else if (message.Content === '9') {
        //创建标签
        let data1 = await wechatApi.createTag({
          "tag": {
            "name" : "VIP 客户"
          }
        });
        console.log(data1);   //{ tag: { id: 115, name: 'VIP 客户' } }
        let data = await wechatApi.createTag({
          "tag": {
            "name" : "普通客户"
          }
        });
        console.log(data); // { tag: { id: 116, name: '普通客户' } }
        //获取标签
        data = await wechatApi.getTag();
        console.log(data);   //
        //更新标签
        data = await wechatApi.updateTag({
          "tag": {
            "id": data1.tag.id,      //更新的指定标签
            "name": "钻石用户"     //更新的内容
          }
```

```
    });
    console.log(data); //{ errcode: 0, errmsg: 'ok' }
    //删除标签
    data = await wechatApi.deleteTag({
      tag: {
        "id": data1.tag.id
      }
    });
    console.log(data); //{ errcode: 0, errmsg: 'ok' }
  } else if (message.Content === '10') {
    //获取所有用户
    let data = await wechatApi.getUsers();
    console.log(data);
    /*
    {"total":2,
    "count":2,
    "data":{
      "openid":["OPENID1","OPENID2"]
    },
    "next_openid":"NEXT_OPENID"
    }
     */
    //批量为用户打标签
    data = await wechatApi.batchUserTags({
      openid_list: data.data.openid,
      tagid: 116
    });
```

```
        console.log(data);   //{ errcode: 0, errmsg: 'ok' }
        //获取标签下的粉丝列表
        data = await wechatApi.getTagUsers({
          tagid: 116
        });
        console.log(data);
        //获取粉丝下的标签列表
        data = await wechatApi.getUserTags({
          openid: message.FromUserName
        })
        console.log(data); //{ tagid_list: [ 116 ] }

    } else if (message.Content === '11') {
        //根据标签来群发消息
        const data = await wechatApi.sendAllByTag('text', 116, '大吉大利，落地成盒
');
        console.log(data);
        /*
        { errcode: 0,
          errmsg: 'send job submission success',
          msg_id: 1000000005 }
         */
    } else if (message.Content === '12') {
        //获取用户列表
        const {data} = await wechatApi.getUsers();
        //根据标签来群发消息
        const result = await wechatApi.sendAllByUsers('text', data.openid, '爱你一
```

```
万年');
        console.log(result);
        /*
        { errcode: 0,
          errmsg: 'send job submission success',
          msg_id: 3147483654 }
         */
    } else if (message.Content.match('爱')) {
        //模糊匹配，只要包含爱
        content = '我爱你~';
    } else {
        content = '您在说啥，我听不懂';
    }
} else if (message.MsgType === 'image') {
    content = '您的图片地址为：' + message.PicUrl;
} else if (message.MsgType === 'voice') {
    content = '语音识别结果：' + message.Recognition;
} else if (message.MsgType === 'video') {
    content = '接受了视频消息';
} else if (message.MsgType === 'shortvideo') {
    content = '接受了小视频消息';
} else if (message.MsgType === 'location') {
    content = '纬度：' + message.Location_X + '  经度：' + message.Location_Y
        + '  缩放大小：' + message.Scale + '  详情：' + message.Label;
} else if (message.MsgType === 'link') {
    content = '标题：' + message.Title + '  描述：' + message.Description + '  网址：
' + message.Url;
```

```
    } else if (message.MsgType === 'event') {
        if (message.Event === 'subscribe') {
            //用户订阅事件
            content = '欢迎您的订阅~';
            if (message.EventKey) {
                //扫描带参数的二维码的订阅事件
                content = '欢迎您扫二维码的关注';
            }
        } else if (message.Event === 'SCAN') {
            //已经关注了公众号，在扫描带参数二维码进入公众号
            content = '已经关注了公众号，在扫描带参数二维码进入公众号';
        } else if (message.Event === 'unsubscribe') {
            //用户取消关注
            console.log('无情取关~');
        } else if (message.Event === 'LOCATION') {
            //用户进行会话时，上报一次地理位置消息
            content = '纬度：' + message.Latitude + '  经度：' + message.Longitude + '  精度：' + message.Precision;
        } else if (message.Event === 'CLICK') {
            content = '点击了菜单~~~';
        } else if (message.Event === 'VIEW') {
            //用户点击菜单，跳转到其他链接
            console.log('用户点击菜单，跳转到其他链接');
        } else if (message.Event === 'scancode_push') {
            //用户点击菜单，扫二维码推送
            console.log(message.EventKey + '    ' + message.ScanCodeInfo + '    ' + message.ScanType + ' ' + message.ScanResult);
```

```
    } else if (message.Event === 'scancode_waitmsg') {
      //用户点击菜单，扫二维码推送
      console.log(message.EventKey + '  ' + message.ScanCodeInfo + '  ' +
message.ScanType + ' ' + message.ScanResult);
    } else if (message.Event === 'pic_sysphoto') {
      //用户点击菜单，扫二维码推送
      console.log(message.SendPicsInfo + '  ' + message.PicMd5Sum);
    } else if (message.Event === 'pic_photo_or_album') {
      //用户点击菜单，扫二维码推送
      console.log(message.SendPicsInfo + '  ' + message.PicMd5Sum);
    } else if (message.Event === 'pic_weixin') {
      //用户点击菜单，扫二维码推送
      console.log(message.SendLocationInfo + '  ' + message.Poiname);
    }
  }

  //将最终回复消息内容添加到 options 中
  options.content = content;
  //将最终的 xml 数据返回出去
  return template(options);
}
```

8) template.js

```
/*
  设置回复用户的 6 种消息内容
 */

module.exports = options => {

  //回复用户消息
```

```
  let replyMessage = '<xml>' +
    '<ToUserName><![CDATA[' + options.toUserName + ']]></ToUserName>' +
    '<FromUserName><![CDATA[' + options.fromUserName +
']]></FromUserName>' +
    '<CreateTime>' + options.createTime + '</CreateTime>' +
    '<MsgType><![CDATA[' + options.msgType + ']]></MsgType>';

  if (options.msgType === 'text') {
    replyMessage += '<Content><![CDATA[' + options.content + ']]></Content>';
  } else if (options.msgType === 'image') {
    replyMessage += '<Image><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Image>';
  } else if (options.msgType === 'voice') {
    replyMessage += '<Voice><MediaId><![CDATA[' + options.mediaId +
']]></MediaId></Voice>';
  } else if (options.msgType === 'video') {
    replyMessage += '<Video>' +
      '<MediaId><![CDATA[' + options.mediaId + ']]></MediaId>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '</Video>';
  } else if (options.msgType === 'music') {
    replyMessage += '<Music>' +
      '<Title><![CDATA[' + options.title + ']]></Title>' +
      '<Description><![CDATA[' + options.description + ']]></Description>' +
      '<MusicUrl><![CDATA[' + options.musicUrl + ']]></MusicUrl>' +
      '<HQMusicUrl><![CDATA[' + options.hqMusicUrl + ']]></HQMusicUrl>' +
      '<ThumbMediaId><![CDATA[' + options.mediaId + ']]></ThumbMediaId>' +
      '</Music>';
  } else if (options.msgType === 'news') {
    replyMessage += '<ArticleCount>' + options.content.length + '</ArticleCount>'
+
      '<Articles>';

    options.content.forEach(item => {
      replyMessage += '<item>' +
        '<Title><![CDATA[' + item.title + ']]></Title>' +
        '<Description><![CDATA[' + item.description + ']]></Description>' +
        '<PicUrl><![CDATA[' + item.picUrl + ']]></PicUrl>' +
        '<Url><![CDATA[' + item.url + ']]></Url>' +
        '</item>';
```

```
        })

        replyMessage += '</Articles>';
    }

    replyMessage += '</xml>';
    //将拼接好回复用户的数据返回出去
    return replyMessage;

}
```

9) wechat.js

```
/*
    获取 access_token:
        全局唯一的接口调用凭据，今后使用微信的接口基本上都需要携带上这个参数
        2 小时需要更新一次，提前 5 分钟刷新

        请求地址：

https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
        请求方式：
          GET

        设计思路：
          首先发送请求获取凭据，保存为一个唯一的文件
          然后后面请求先去本地文件读取凭据
            判断凭据是否过期
              如果没有过期，直接使用
              如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件

        总结：
          先去本地查看有没有指定文件（readAccessToken）
            如果有（之前请求过凭据）
              判断凭据是否过期(isValidAccessToken)
                如果没有过期，直接使用
                如果过期了，重新发送请求获取凭据，保存下来覆盖之前的文件(getAccessToken、saveAccessToken)
```

```
        如果没有（之前都没有请求过凭据）
         发送请求获取凭据，保存为一个唯一的文件

 */
//引入配置对象
const {appID, appsecret} = require('../config');
//引入发送 http 请求的库
const rp = require('request-promise-native');
const request = require('request');
//引入 fs 模块
const {readFile, writeFile, createReadStream, createWriteStream} = require('fs');
//引入接口文件
const api = require('../libs/api');
//引入菜单文件
const menu = require('./menu');

class Wechat {
  getAccessToken () {
    //定义请求地址
    const url = `${api.accessToken}&appid=${appID}&secret=${appsecret}`;
    /*
      问题：需要将回调函数中的数据返回出去？
      解决：用 promise 解决

      所有的异步操作，都应该包装一层 promise，让这个异步操作执行完毕之后，再去执行后面的代码
      简化： 所有的异步操作，都应该包装一层 promise
     */
    return new Promise((resolve, reject) => {
      //发送 http 请求
      //下载 request-promise-native   request
      rp({method: 'GET', json: true, url})
        .then(res => {
          //请求成功的状态
          // console.log(res);
          //重新赋值凭据的过期时间 ：  当前时间 + (7200 - 5 分钟) * 1000
          res.expires_in = Date.now() + (res.expires_in - 300) * 1000;
          // console.log(res);
          resolve(res);
        })
        .catch(err => {
```

```
            //请求失败
            reject('getAccessToken 方法出了问题：' + err);
        })
    })
}
saveAccessToken (data) {
    /*
      问题：writeFile 方法会将对象转化为字符串
      解决：我将对象转化为 json 字符串
     */
    data = JSON.stringify(data);
    return new Promise((resolve, reject) => {
      //将凭据保存为一个文件
      writeFile('accessToken.txt', data, err => {
        if (!err) {
          //写入成功
          resolve();
        } else {
          //写入失败
          reject('saveAccessToken 方法出了问题：' + err);
        }
      })
    })
}
readAccessToken () {
    return new Promise((resolve, reject) => {
      //将凭据读取出来
      readFile('accessToken.txt', (err, data) => {
        if (!err) {
          //将读取的 Buffer 数据转化为 json 字符串
          data = data.toString();
          //将 json 字符串转化为对象
          data = JSON.parse(data);
          //读取成功
          resolve(data);
        } else {
          //读取失败
          reject('readAccessToken 方法出了问题：' + err);
        }
      })
    })
```

```
  }
  isValidAccessToken (data) {
    /*
      判断凭据是否过期
        true    凭据没有过期
        false   凭据过期了
     */
    //过滤非法的数据
    if (!data || !data.access_token || !data.expires_in) return false;
    //判断凭据是否过期
    /*if (data.expires_in > Date.now()) {
      //如果凭据的过期时间大于当前时间，说明没有过期
      return true
    } else {
      //如果凭据的过期时间小于当前时间，说明过期了
      return false
    }*/
    //简写方式
    return data.expires_in > Date.now();
  }
  fetchAccessToken () {
    //优化操作,优化不去执行读取文件操作
    if (this.access_token && this.expires_in && this.isValidAccessToken(this)) {
      //说明 this 有凭据和过期时间，并且凭据未过期
      return Promise.resolve({access_token: this.access_token, expires_in:
this.expires_in});
    }

    return this.readAccessToken()
      .then(async res => {
        //判断凭据是否过期(isValidAccessToken)
        if (this.isValidAccessToken(res)) {
          //没有过期，直接使用
          return Promise.resolve(res);
        } else {
          //重新发送请求获取凭据
          const data = await this.getAccessToken();
          //保存下来
          await this.saveAccessToken(data);
          //将请求回来的凭据返回出去
          return Promise.resolve(data);
```

```
            }
        })
        .catch(async err => {
            console.log(err);
            //重新发送请求获取凭据
            const data = await this.getAccessToken();
            //保存下来
            await this.saveAccessToken(data);
            //将请求回来的凭据返回出去
            return Promise.resolve(data);
        })
        .then(res => {
            //将其请求回来的凭据和过期时间挂载到 this 上
            this.access_token = res.access_token;
            this.expires_in = res.expires_in;
            //指定 fetchAccessToken 方法返回值
            return Promise.resolve(res);
        })
}

//上传临时素材
uploadTemporaryMaterial (type, filePath) {
    /*
        type: 上传多媒体文件的类型
        filePath: 上传多媒体文件的路径
     */
    return new Promise((resolve, reject) => {
        //获取 access_token
        this.fetchAccessToken()
            .then(res => {
                //定义请求的地址
                const url =
`${api.temporary.upload}access_token=${res.access_token}&type=${type}`;
                //定义要传输过去的媒体数据
                const formData = {
                    media: createReadStream(filePath)
                }
                //发送请求
                rp({method: 'POST', json: true, url, formData})
                    .then(res => {
                        //将请求回来的数据返回出去
```

```
                resolve(res);
            })
            .catch(err => {
                reject('uploadTemporaryMaterial 方法出了问题：' + err);
            })
        })
    })


}
//获取临时素材
getTemporaryMaterial (mediaId, filePath, isVideo = false) {
    /*
      mediaId: 要获取的素材 id
      filePath：要保存媒体文件的路径
      isVideo: 可选值
     */
    return new Promise((resolve, reject) => {
      //获取 access_token
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          const url =
`${api.temporary.get}access_token=${res.access_token}&media_id=${mediaId}`;
          //发送请求
          if (isVideo) {
            //如果是视频消息素材，就返回一个 url 地址
            rp({method: 'GET', json: true, url})
              .then(res => resolve(res))
              .catch(err => reject('getTemporaryMaterial 方法出了问题：' + err))
          } else {
            //如果不是视频消息素材，就返回一个文件接收
            request
              .get(url)
              .pipe(createWriteStream(filePath))
              .once('close', () => {
                //说明文件下载成功了~
                resolve();
              })
          }
        })
```

```
    })

  }
  //上传永久素材
  uploadPermanentMaterial (type, material, description) {
    /*
      type: 可以区分我通过什么方式上传素材
      material: 上传素材的路径/请求体中的内容
      description: 针对于视频素材上传
     */
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          let url = '';
          //定义发送请求的配置对象
          let options = {
            method: 'POST',
            json: true
          }
          if (type === 'news') {
            //上传图文消息
            url =
`${api.permanent.uploadNews}access_token=${res.access_token}`;
            options.body = material;
          } else if (type === 'pic') {
            //上传图文消息的图片
            url =
`${api.permanent.uploadImg}access_token=${res.access_token}`;
            options.formData = {
              media: createReadStream(material)
            }
          } else {
            //上传其他素材
            url =
`${api.permanent.uploadOthers}access_token=${res.access_token}&type=${type}`;
            options.formData = {
              media: createReadStream(material)
            }
            if (type === 'video') {
              options.body = description;
```

```
                }
            }
            //将请求地址放到配置对象中
            options.url = url;
            //发送请求
            rp(options)
                .then(res => resolve(res))
                .catch(err => reject('uploadPermanentMaterial 方法出了问题:' +
err))
        })
    })
}
//获取永久素材
getPermanentMaterial (type, mediaId, filePath) {
    /*
      type: 用来如何接受数据
      mediaId: 获取的媒体素材 id
      filePath: 保存的媒体素材的位置
     */
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          //定义请求地址
          const url = `${api.permanent.get}access_token=${res.access_token}`;
          //定义请求体中的数据
          const body = {
            media_id: mediaId
          }
          if (type === 'news' || 'video') {
            rp({method: 'POST', json: true, url, body})
              .then(res => resolve(res))
              .catch(err => reject('getPermanentMaterial 方法出了问题:' + err))
          } else {
            request({method: 'POST', json: true, url, body})
              .pipe(createWriteStream(filePath))
              .once('close', () => resolve())
          }
        })
    })
}
//删除永久素材
```

```
  deletePermanentMaterial (mediaId) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.permanent.delete}access_token=${res.access_token}`;
          const body = {
            media_id: mediaId
          }
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('deletePermanentMaterial 方法出了问题：' +
err))
        })
    })
  }
  //更新永久图文消息素材
  updatePermanentNews (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url =
`${api.permanent.updateNews}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('updatePermanentNews 方法出了问题：' + err))
        })
    })
  }
  //获取永久素材数量
  getPermanentCount () {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url =
`${api.permanent.getCount}access_token=${res.access_token}`;
          rp({method: 'GET', json: true, url})
            .then(res => resolve(res))
            .catch(err => reject('getPermanentCount 方法出了问题：' + err))
        })
    })
  }
```

```
  //获取永久素材列表
  getPermanentList (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url =
`${api.permanent.getMaterialList}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('getPermanentList 方法出了问题：' + err))
        })
    })
  }
  //创建菜单
  createMenu (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.menu.create}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('createMenu 方法出了问题：' + err))
        })
    })
  }
  //删除菜单
  deleteMenu () {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.menu.delete}access_token=${res.access_token}`;
          rp({method: 'GET', json: true, url})
            .then(res => resolve(res))
            .catch(err => reject('deleteMenu 方法出了问题：' + err))
        })
    })
  }
  //获取菜单的配置
  getMenu () {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
```

```
            .then(res => {
                const url = `${api.menu.get}access_token=${res.access_token}`;
                rp({method: 'GET', json: true, url})
                    .then(res => resolve(res))
                    .catch(err => reject('getMenu 方法出了问题：' + err))
            })
    })
}
//创建自定义菜单
createMyMenu (body) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.myCreate}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                    .then(res => resolve(res))
                    .catch(err => reject('createMyMenu 方法出了问题：' + err))
        })
    })
}
//删除自定义菜单
deleteMyMenu (body) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.myDelete}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                    .then(res => resolve(res))
                    .catch(err => reject('deleteMyMenu 方法出了问题：' + err))
            })
    })
}
//测试个性化菜单匹配结果
testMyMenu (body) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.menu.myTest}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                    .then(res => resolve(res))
                    .catch(err => reject(' testMyMenu 方法出了问题：' + err))
```

```
            })
        })
    }
    //创建标签
    createTag (body) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    const url = `${api.tag.create}access_token=${res.access_token}`;
                    rp({method: 'POST', json: true, url, body})
                        .then(res => resolve(res))
                        .catch(err => reject('createTag 方法出了问题：' + err))
                })
        })
    }
    //获取标签
    getTag () {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    const url = `${api.tag.get}access_token=${res.access_token}`;
                    rp({method: 'GET', json: true, url})
                        .then(res => resolve(res))
                        .catch(err => reject('getTag 方法出了问题：' + err))
                })
        })
    }
    //更新标签
    updateTag (body) {
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    const url = `${api.tag.update}access_token=${res.access_token}`;
                    rp({method: 'POST', json: true, url, body})
                        .then(res => resolve(res))
                        .catch(err => reject('updateTag 方法出了问题：' + err))
                })
        })
    }
    //更新标签
    deleteTag (body) {
```

```
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.tag.delete}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('deleteTag 方法出了问题：' + err))
        })
    })
  }
  //获取标签下的粉丝列表
  getTagUsers (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.tag.getUsers}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('getTagUsers 方法出了问题：' + err))
        })
    })
  }
  //批量为用户打标签
  batchUserTags (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.user.batchTag}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
            .then(res => resolve(res))
            .catch(err => reject('batchUserTags 方法出了问题：' + err))
        })
    })
  }
  //批量为用户取消标签
  unBatchUserTags (body) {
    return new Promise((resolve, reject) => {
      this.fetchAccessToken()
        .then(res => {
          const url = `${api.user.unBatchTag}access_token=${res.access_token}`;
          rp({method: 'POST', json: true, url, body})
```

```
                .then(res => resolve(res))
                .catch(err => reject('unBatchUserTags 方法出了问题：' + err))
        })
    })
}
//获取用户下所有的标签
getUserTags (body) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                const url = `${api.user.getTags}access_token=${res.access_token}`;
                rp({method: 'POST', json: true, url, body})
                    .then(res => resolve(res))
                    .catch(err => reject('getUserTags 方法出了问题：' + err))
            })
    })
}
//获取所有用户列表
getUsers (next_openid) {
    return new Promise((resolve, reject) => {
        this.fetchAccessToken()
            .then(res => {
                let url = `${api.user.get}access_token=${res.access_token}`;
                if (next_openid) {
                    url += '&next_openid=' + next_openid;
                }
                rp({method: 'GET', json: true, url})
                    .then(res => resolve(res))
                    .catch(err => reject('getUsers 方法出了问题：' + err))
            })
    })
}
//定义根据标签群发消息
sendAllByTag (type, tag_id, content, is_to_all = false, send_ignore_reprint = 0) {
    /*
        type: 媒体数据类型
        tag_id: 指定标签
        content: 媒体消息内容
        is_to_all: 是否保存历史消息记录中
        send_ignore_reprint:    图文消息被判定为转载时，是否继续群发。 1 为
继续群发（转载），0 为停止群发。 该参数默认为0。
```

```
        */
        return new Promise((resolve, reject) => {
            this.fetchAccessToken()
                .then(res => {
                    const url = `${api.sendAll.tag}access_token=${res.access_token}`;
                    let body = {
                        filter: {
                            is_to_all,
                            tag_id
                        }
                    }
                    //判断群发的消息类型
                    if (type === 'text') {
                        body.text = {
                            content
                        }
                    } else if (type === 'mpnews') {
                        body.send_ignore_reprint = send_ignore_reprint;
                        body[type] = {
                            media_id: content
                        }
                    } else {
                        body[type] = {
                            media_id: content
                        }
                    }
                    body.msgtype = type;
                    console.log(body);
                    //发送请求
                    rp({method: 'POST', json: true, url, body})
                        .then(res => resolve(res))
                        .catch(err => reject('sendAllByTag 方法出了问题：' + err))
                })
        })
    }
    //定义根据用户列表 openid 群发消息
    sendAllByUsers (type, openid_list, content, send_ignore_reprint = 0, title,
description) {
        /*
            type: 媒体数据类型
            openid_list: 指定用户列表
```

```
        content: 媒体消息内容
        send_ignore_reprint:    图文消息被判定为转载时，是否继续群发。 1 为继续群发（转载），0 为停止群发。 该参数默认为0。
        title: 视频文件的标题
        description: 视频文件的描述
       */
      return new Promise((resolve, reject) => {
        this.fetchAccessToken()
          .then(res => {
            const url = `${api.sendAll.users}access_token=${res.access_token}`;
            let body = {
              touser: openid_list
            }
            //判断群发的消息类型
            if (type === 'text') {
              body.text = {
                content
              }
            } else if (type === 'mpnews') {
              body.send_ignore_reprint = send_ignore_reprint;
              body[type] = {
                media_id: content
              }
            } else if (type === 'mpvideo') {
              body[type] = {
                media_id: content,
                title: title,
                description: description
              }
            } else {
              body[type] = {
                media_id: content
              }
            }
            body.msgtype = type;
            // console.log(body);
            //发送请求
            rp({method: 'POST', json: true, url, body})
              .then(res => resolve(res))
              .catch(err => reject('sendAllByUsers 方法出了问题：' + err))
          })
```

```
        })
    }
}

//测试
/*(async () => {

    const wechatApi = new Wechat();

    let data = await wechatApi.deleteMenu();
    console.log(data);
    data = await wechatApi.createMenu(menu);
    console.log(data);

})()*/

module.exports = Wechat;
```

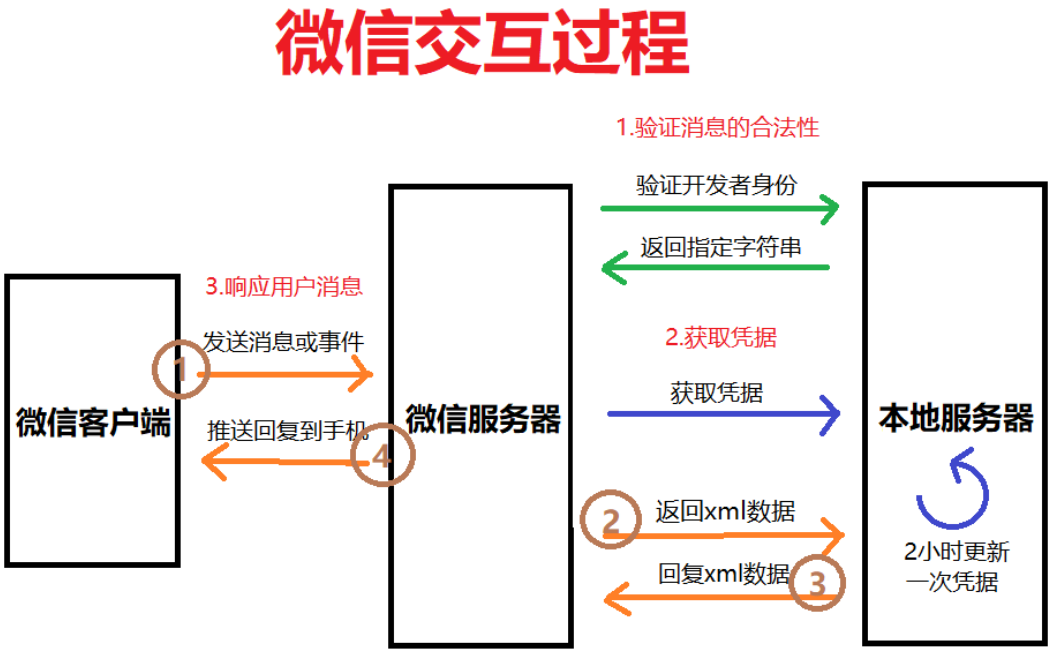
12) index.js

```
const express = require('express');
const auth = require('./wechat/auth');
const app = express();

//接受微信服务器发送过来的请求 GET
//应用中间级，能够接受处理所有请求
app.use(auth());

app.listen(3000, err => {
    if (!err) console.log('服务器启动成功了~~~');
})
```

第 3 章：微信公众号交互流程



第 4 章：微信公众号 JS-SDK

4.1 简介

微信 JS-SDK (JavaScript Software Development Kit)是微信公众平台面向网页开发者提供的基于微信内的网页开发工具包。

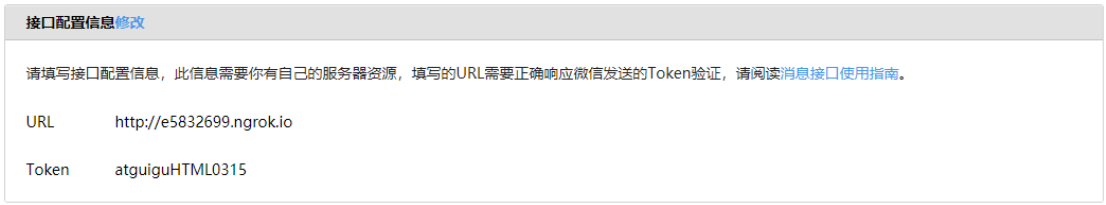
4.2 功能

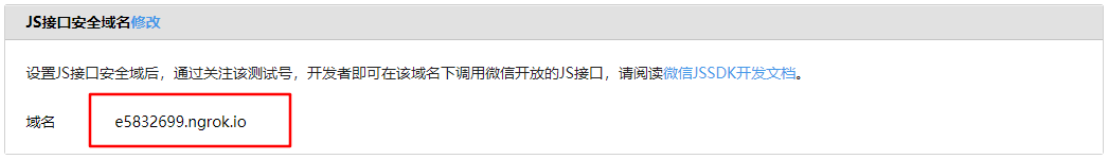
通过使用微信 JS-SDK，网页开发者可借助微信高效地使用拍照、选图、语音、位置等手机系统的能力，同时可以直接使用微信分享、扫一扫、卡券、支付等微信特有的能力，为微信用户提供更优质的网页体验。

简单来说，使用 JS-SDK 我们可以引入外部网页。

4.3 使用步骤

4.3.1 绑定域名





4.3.2 注入权限验证配置

1) app.js，此处有一个签名算法，最终生成一个签名

```
const express = require('express');
const auth = require('./wechat/auth');
const Wechat = require('./wechat/wechat');
const {url} = require('./config');
const sha1 = require('sha1');
const app = express();

app.set('views', 'views');
app.set('view engine', 'ejs');

const wechatApi = new Wechat();

app.get('/search', async (req, res) => {
```

```
//用户访问微信中的网页触发的路由
/*
    验证的签名算法
        1. 参与签名的字段包括 noncestr（随机字符串），有效的 jsapi_ticket, timestamp（时间戳）, url（当前网页的 URL，不包含#及其后面部分）
        2. 对所有待签名参数按照字段名的 ASCII 码从小到大排序（字典序）后，
        3. 使用 URL 键值对的格式（即 key1=value1&key2=value2…）拼接成字符串 string1。这里需要注意的是所有参数名均为小写字符。
        4. 对 string1 作 sha1 加密，字段名和字段值都采用原始值，不进行 URL 转义。
 */
//获取 ticket
const {ticket} = await wechatApi.fetchTicket();
//获取随机字符串
const noncestr = Math.random().toString().split('.')[1];
//获取当前时间戳
const timestamp = Date.now();
//2. 按照 key=value 将其拼接在一起
const params = [
    'jsapi_ticket=' + ticket,
    'noncestr=' + noncestr,
    'timestamp=' + timestamp,
    'url=' + url + '/search'
]
// 3. 按照字典序排序，再使用'&'将四个参数拼接在一起
const string = params.sort().join('&');
// 4. sha1 加密
```

```
    const signature = sha1(string);
    //将数据渲染到页面上
    res.render('search', {
        signature,
        noncestr,
        timestamp
    })

})

//应用中间级，能够接受处理所有请求
app.use(auth());

app.listen(3000, err => {
    if (!err) console.log('服务器启动成功了~~~');
})
```

2) 新建 views 目录，下面新建一个模板文件 movie.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width,initial-scale=1.0,maximum-scale=1.0,minimum-scale=1.0,user-scalable=no">
    <title>猜电影</title>
```

```
</head>
<body>
<h1>点击标题，开始录音翻译</h1>
<div id="director"></div>
<p id="title"></p>
<p id="year"></p>
<p id="poster"></p>

<script src="http://zeptojs.com/zepto-docs.min.js"></script>
<script src="http://res.wx.qq.com/open/js/jweixin-1.2.0.js"></script>
<script>
  $(function () {
    wx.config({
      debug: false, // 开启调试模式,调用的所有 api 的返回值会在客户端 alert 出来，若要查看传入的参数，可以在 pc 端打开，参数信息会通过 log 打出，仅在 pc 端时才会打印。
      appId: 'wxc8e92f7ab70fbca0', // 必填，公众号的唯一标识
      timestamp: '<%= timestamp %>', // 必填，生成签名的时间戳
      nonceStr: '<%= noncestr %>', // 必填，生成签名的随机串
      signature: '<%= signature %>',// 必填，签名
      jsApiList: [
        'startRecord',
        'stopRecord',
        'onVoiceRecordEnd',
        'translateVoice'
      ] // 必填，需要使用的 JS 接口列表
    })
```

```
wx.ready(function(){


    wx.checkJsApi({
      jsApiList: ['onVoiceRecordEnd'], // 需要检测的 JS 接口列表，所有 JS 接口列表见附录 2,
      success: function(res) {
        // 以键值对的形式返回，可用的 api 值 true，不可用为 false
        // 如：{"checkResult":{"chooseImage":true},"errMsg":"checkJsApi:ok"}
        console.log(res)
      }
    })

    const isRecording = false

    $('h1').on('tap', function () {
      if (!isRecording) {
        isRecording = true
        wx.startRecord({
          cancel: function () {
            window.alert('您取消了音频录制，需要打开此功能')
          }
        })
        return
      }

      isRecording = false
```

```
            wx.stopRecord({
                success: function (res) {
                    const localId = res.localId
                    console.log(localId)

                    wx.translateVoice({
                        localId: localId,
                        isShowProgressTips: 1, // 默认为1，显示进度提示
                        success: function (res) {
                            const result = res.translateResult
                            alert(res.translateResult); // 语音识别的结果
                        }
                    })
                }
            })
        })
    })
</script>
</body>
</html>
```

4.3.3 实现豆瓣电影查询和分享功能

1) moive.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width,initial-scale=1.0,maximum-scale=1.0,minimum-scale=1.0,user-scalable=no">
    <title>猜电影</title>
</head>
<body>
<h1>点击标题，开始录音翻译</h1>
<div id="director"></div>
<p id="title"></p>
<p id="year"></p>
<p id="poster"></p>
<!--用来实现功能的移动端库-->
<script src="http://zeptojs.com/zepto-docs.min.js"></script>
<!--需要调用JS接口，就必须引入此文件 -->
<script src="http://res.wx.qq.com/open/js/jweixin-1.2.0.js"></script>
<script>
    $(function () {
        //微信全局配置，注入配置信息
        wx.config({
            debug: false, // 开启调试模式,调用的所有api的返回值会在客户端alert出来，若要查看传入的参数，可以在pc端打开，参数信息会通过log打出，仅在pc端时才会打印。
            appId: 'wxc8e92f7ab70fbca0', // 必填，公众号的唯一标识
            timestamp: '<%= timestamp %>', // 必填，生成签名的时间戳
            nonceStr: '<%= noncestr %>', // 必填，生成签名的随机串
```

```
        signature: '<%= signature %>',// 必填，签名
        jsApiList: [
            'startRecord',
            'stopRecord',
            'onVoiceRecordEnd',
            'translateVoice',
            'onMenuShareTimeline',
            'onMenuShareAppMessage',
            'onMenuShareQQ',
            'onMenuShareWeibo',
            'onMenuShareQZone'
        ] // 必填，需要使用的JS接口列表，没填的JS接口不能使用
    })
    //config 信息验证后会执行 ready 方法，所有微信接口都在 ready 函数中调用
    wx.ready(function(){
        //判断当前客户端版本是否支持指定JS接口，此处只举一个例子
        wx.checkJsApi({
            jsApiList: ['onVoiceRecordEnd'], // 需要检测的JS接口列表，所有JS接口列表见附录2,
            success: function(res) {
                // 以键值对的形式返回，可用的api值true，不可用为false
                // 如：{"checkResult":{"chooseImage":true},"errMsg":"checkJsApi:ok"}
                console.log(res)
            }
        })
```

```
const isRecording = false


$('h1').on('tap', function () {
  if (!isRecording) {
    isRecording = true
    //开始录音
    wx.startRecord({
      cancel: function () {
        window.alert('您取消了音频录制，需要打开此功能')
      }
    })
    return
  }
  isRecording = false
  //停止录音
  wx.stopRecord({
    success: function (res) {
      const localId = res.localId
      //识别音频并返回识别结果
      wx.translateVoice({
        localId: localId,
        isShowProgressTips: 1, // 默认为1，显示进度提示
        success: function (res) {
          //识别后的结果，文本字符串
          const result = res.translateResult
          //根据结果请求豆瓣电影查询API
          $.getJSON('https://api.douban.com/v2/movie/search?q=' +
```

```
result + '&callback=?', function (data) {
                    console.log(data)
                    const subjects = data.subjects[0]
                    console.log(subjects)
                    //将请求后返回的结果内容显示到页面上去
                    $('#director').html(subjects.directors[0].name)
                    $('#title').html(subjects.title)
                    $('#year').html(subjects.year)
                    $('#poster').html('<img src="' + subjects.images.large + '" />')
                    //添加一个分享给朋友接口
                    wx.onMenuShareAppMessage({
                      title: subjects.title, // 分享标题
                      desc: '我搜出来了' + subjects.title, // 分享描述
                      link: 'http://079e92c1.ngrok.io/share', // 分享链接，该链接域名或路径必须与当前页面对应的公众号 JS 安全域名一致
                      imgUrl: subjects.images.large, // 分享图标
                      type: 'link', // 分享类型,music、video 或 link，不填默认为 link
                      dataUrl: '', // 如果 type 是 music 或 video，则要提供数据链接，默认为空
                      success: function () {
                        // 用户确认分享后执行的回调函数
                        alert('分享成功')
                      },
                      cancel: function () {
                        // 用户取消分享后执行的回调函数
                        alert('分享失败')
```

```
                    }
                })
            })
            alert(res.translateResult); // 语音识别的结果
        }
    })
}
})
})
})
</script>
</body>
</html>
```

2) share.ejs，这个就是分享显示的页面，此处为模拟页面，没有数据

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width,initial-scale=1.0,maximum-scale=1.0,minimum-scale
=1.0,user-scalable=no">
    <title>分享</title>
</head>
```

```
<body>
<h1>这是一个电影分享页面</h1>
</body>
</html>
```