移动互联网及应用

课程大作业报告书

作业名称： **图片识别小程序**

系 （部）： 信息工程系

专业班级： 电子信息科学与技术 17-2

学生姓名： 张厚今

学　　号： 201723010237

指导教师： 魏光村

完成日期： 2019 年 12 月

山东科技大学

目录

1 概述

1.1 课程大作业目的与要求

课程大作业的目的是：运用在本次课程中学到的知识来指导实践，了解程序设计其实现方法，学会解决实际问题。掌握微信小程序设计的具体步骤与基本方法，针对选定的程序做调研分析。通过课程大作业，提高实践动手技能，培养独立分析分析问题和解决问题的能力。

课程大作业的要求：本次课程大作业的选题比较灵活，可以是自主选题，也可以参考课本中的案例自行修改完善，题目要符合课程大作业的要求，并且具备一定的水平和深度。

1.2 课程大作业简介

图像识别是指利用计算机对图像进行处理、分析和理解，以识别各种不同模式的目标和对像的技术。在众多的图像识别分支中，拍照识别是一个重要的应用。利用图像识别技术，识别拍摄到的图片内容，已经广泛应用于各类图像识别 App 中。

微信小程序是一种不需要下载安装即可使用的应用，它实现了应用的“触手可及”和“即用即走”，用户扫一扫或搜一下即可打开应用。利用微信小程序使用便捷的特点，结合图片识别应用，本次大作业选定了制作微信小程序的图片识别应用——ImageMaster。本应用实现了基于微信小程序的动植物识别和车辆识别，使用便捷，充分发挥了微信小程序“即用即走”的特点。

同时，本次大作业项目制作过程只用了 Git 进行进度跟踪，便于进行版本回退和功能更新。

2 设计思路

根据初步选定的设计题目，查找相关资料，针对系统所涉及内容，查阅相关背景知识，将设计思路在此做一个概括。

本课题需要实现一个基本的图片识别应用，实现动物识别、植物识别和车辆识别三种主要功能。查阅的资料主要包括两方面：图片识别 API 的选择以及微信小程序 UI 框架的选择。

2.1 图片识别 API

首先需要查阅图片识别 API 的相关资料，以便确定课题最终使用哪个 API。下面我将在接口能力、是否有参考例程、个人评价三方面简要分析一下常见的几种图片识别 API。

1. 百度大脑

（1）接口能力

表 2-1 百度图片识别 API 接口能力

接口名称	接口能力简要描述
图像主体检测	识别图像中的主体具体坐标位置。
通用物体和场景识别高级版	识别图片中的场景及物体标签，支持 10w+标签类型。
菜品识别	检测用户上传的菜品图片，返回具体的菜名、卡路里、置信度信息。
自定义菜品识别	入库自定义的单菜品图，实现上传多菜品图的精准识别，返回具体的菜名、位置、置信度
logo 商标识别	识别图片中包含的商品 LOGO 信息，返回 LOGO 品牌名称、在图片中的位置、置信度。
动物识别	检测用户上传的动物图片，返回动物名称、置信度信息。
植物识别	检测用户上传的植物图片，返回植物名称、置信度信息。
果蔬食材识别	检测用户上传的果蔬类图片，返回果蔬名称、置信度信息。
地标识别	检测用户上传的地标图片，返回地标名称。
红酒识别	识别图像中的红酒标签，返回红酒名称、国家、产区、酒庄、类型、糖分、葡萄品种等信息。
货币识别	识别图像中的货币类型，返回货币名称、代码、面值、年份信息，可识别百余种国内外常见货币。

（2）调用方式

API 和 SDK 两种方式，SDK 文档包含 Java、PHP、Python、C#、C++、Node 语言，有丰富的 Demo。

2. 腾讯 AI 开放平台

接口包含 OCR、人体与人脸识别、物体识别、图片特效、图片识别、敏感信息审核、闲聊机器人、基础文本分析、语义解析、语音识别 等等，有 PHP 参考例程。

3. 华为 HiAI

接口包含人脸识别、人体识别、图片识别、图像分辨率、场景识别、文档检测矫正、人像分割、视频语音等等。有详细的开发指南，但是其针对的是 Android 手机平台的开发。

4. 旷世 Face++

接口包含人脸识别、人体识别、证件识别、图像识别，拥有详细的开发指南。

综合比较之后，决定采用百度大脑 API 平台。

2.2 微信小程序 UI 框架

查阅各种微信小程序 UI 框架资料，决定使用哪种 UI 框架作为本项目的 UI 框架。

1. WeUI

（1）[项目地址](https://github.com/Tencent/weui-wxss)

（2）简介

WeUI 是一套同微信原生视觉体验一致的基础样式库，由微信官方设计团队为微信内网页和微信小程序量身设计，令用户的使用感知更加统一。包含 button、cell、dialog、progress、 toast、article、actionsheet、icon 等各式元素。

（3）使用体验

毕竟是微信官方提供的 UI 框架，界面肯定是与微信小程序的适配度最好。但是感觉没有特别的亮点。

2. Vant Weapp

（1）[项目地址](https://github.com/youzan/vant-weapp)

（2）简介

Vant Weapp 是有赞移动端组件库 Vant 的小程序版本，两者基于相同的视觉规范，提供一致的 API 接口，助力开发者快速搭建小程序应用。

（3）使用体验

组件分类做得不错，样式也很简洁美观。

3. iView Weapp

（1）[项目地址](https://github.com/TalkingData/iview-weapp)

（2）简介

一套高质量的微信小程序 UI 组件库

（3）使用体验

界面简单，没亮点。

4. MinUI

（1）[项目地址](https://github.com/meili/minui)

（2）简介

MinUI 是基于微信小程序自定义组件特性开发而成的一套简洁、易用、高效的组件库，适用场景广。覆盖小程序原生框架，各种小程序组件主流框架等，并且提供了专门的命令行工具

（3）使用体验

组件还算齐全，没有亮点。官方 Demo 还有广告链接。

5. Wux Weapp

（1）[项目地址](https://github.com/wux-weapp/wux-weapp)

（2）简介

Wux Weapp 是一套组件化、可复用、易扩展的微信小程序 UI 组件库。UI 样式可配置，拓展灵活，轻松适应不同的设计风格。60 多个丰富的组件，能够满足移动端开发的基本需求。

（3）使用体验

没有特别炫酷的效果，但组件齐全。

6. ColorUI

（1）[项目地址](https://github.com/weilanwl/ColorUI)

（2）简介

鲜亮的高饱和色彩，专注视觉的小程序组件库。

（3）使用体验

功能齐全，各种组件分类明确。

7. Taro UI

（1）[项目地址](https://github.com/NervJS/taro-ui)

（2）简介

一款基于 Taro 框架开发的多端 UI 组件库。基于 Taro 开发 UI 组件一套组件，可以在微信小程序，支付宝小程序，百度小程序，H5 多端适配运行（ReactNative 端暂不支持）。提供友好的 API，可灵活的使用组件。

（3）使用体验

界面简介，功能丰富。

经过亲自体验各种 UI 框架的官方 Demo，最后感觉 Taro UI 使我印象很深刻。界面简洁，组件分类明确。我起初决定使用 Taro UI 框架，但是在项目进行到 UI 设计阶段，发现 Taro 使用有些许难度,为简化开发过程,最终使用了微信小程序的官方 UI 框架——WeUI 框架。

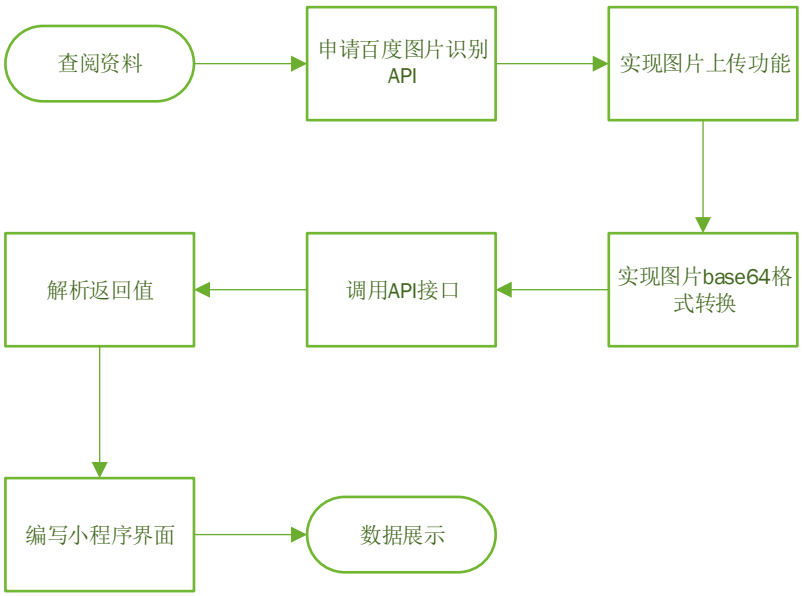
3 设计方案

在需求分析的基础上，查阅资料，对在小程序设计中可能用到的相关技术做一定的调研分析，做一个概要性的描述。

因为微信小程序本身就是联网的应用平台，因此在微信小程序平台进行图片识别，就不必担心网络连接问题。从源头上追溯，实现图片识别需要用户首先上传图片或拍摄图片，需要解决图片上传问题；之后发送 API 请求时，图片需要作为请求的参数，需要解决图片转码的问题；最后 API 请求调用成功后，需要将识别的数据输出，就涉及数据处理以及 UI 界面设计的问题。通过解决关键的“图片上传”、“图片转码”、“API 调用”、“界面设计”这几个问题，就可以实现基本的图片识别小程序了。

4 设计过程

根据初步选定的课程大作业程序设计题目，查找相关资料，结合课本中的案例以及前期的学习，整理设计过程。



4.1 百度开放平台注册

图 4.1 服务条款

首先登陆百度智能云平台，同意它的服务条款。

之后填写相关的信息，在控制台概览中创建应用。

图 4.2 创建新应用

图 4.3 获取密钥

现在已经申请好了百度的 API 接口，下面尝试在微信小程序中进行调用。

4.2 获取 asscee_token

在百度 AI 的官方文档中可以看到，这个 API 接口有两种调用方式，两种不同的调用方式有相同的接口 URL 地址，区别在于请求方式和鉴权方式不同。下面我将尝试使用微信小程序中常用的 POST 请求方式，调用该接口。

图 4.4 调用方式

在官方文档中可以看到，使用 post 请求需要用到 access_token，所以现在去查看如何获取 access_token。

图 4.5 获取 Access_Token

获取 access_token 需要下个授权的服务器地址发送 post 请求，使用固定的参数，即可得到服务器返回的 json 数据。

图 4.6 服务器返回参数

下面就开始进行小程序的编写，实现 access_token 的获取。

首先，在空的小程序中添加一个按钮，在按钮按下后，调用 wx.request 方法向服务器发送 POST 请求。

```
<!--index.wxml-->
<view>
  <button bindtap="get_access_token">
    获取access_token
  </button>
</view>
```

图 4.7 布局文件

按钮绑定事件处理函数，这个事件处理函数就是用来发送 POST 请求的。

```
//index.js
const grant_type = 'client_credentials'
const client_id = 'na7ZiKR[redacted]deOFSrbEh'
const client_secret = 'bMNEYK67[redacted]uL9n0qx8waEl0'

Page({
  get_access_token: function (res) {
    wx.request({
      url: 'https://aip.baidubce.com/oauth/2.0/token?grant_type=' + grant_type + '&client_id=' + client_id + '&client_secret=' + client_secret,
      method: 'POST',
      complete:function(res) {
        console.log('Request complete')
      },
      success: function(res) {
        console.log(res)
      },
      fail: function(res) {
        console.log('Fail to request !')
        console.log(res)
      }
    })
  }
})
```

图 4.8 绑定事件

上面图片就是编写的事件处理函数，利用 wx.request 方法，请求百度 API 的 URL 地址，请求方式按要求是 POST 方式。然后分别编写了请求完成、请求成功、请求失败的回调函数，目的就是让我能在控制台中看到请求的状态。

下面就需要在微信小程序的后台添加合法域名，以便 wx.request 方法能够正确使用。

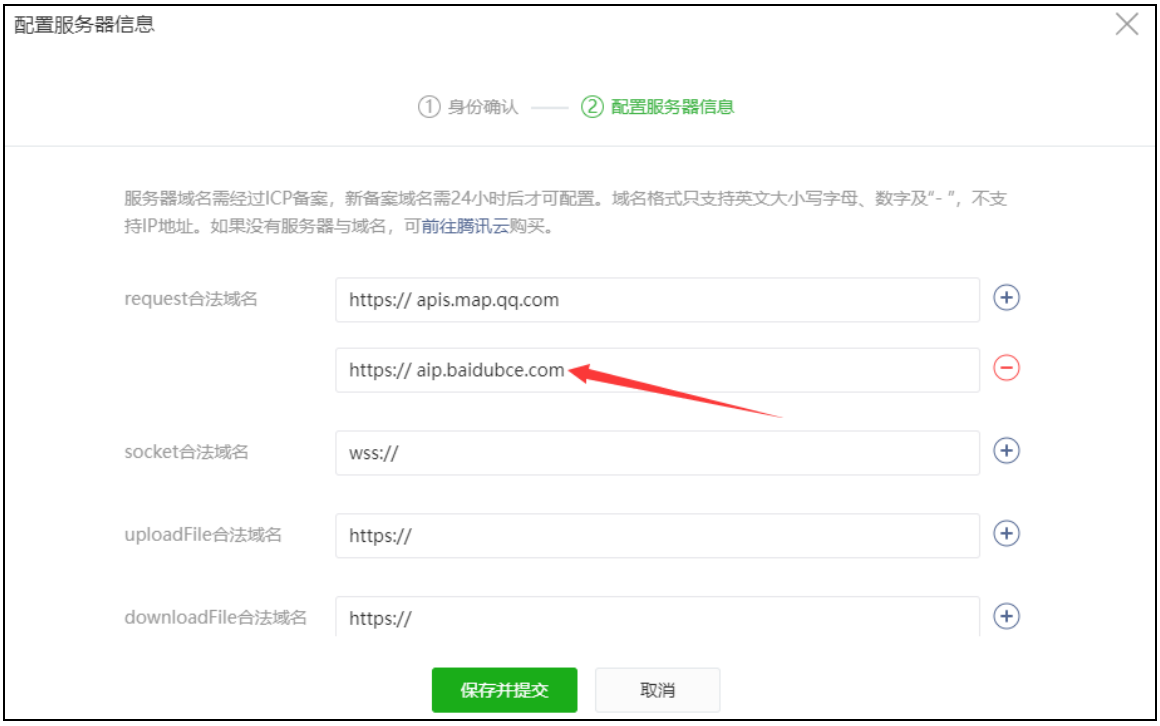


图 4.9 添加域名

箭头所指的就是百度大脑的请求网址，上面那个 URL 是以前做地图 API 添加的，这里没有用到。运行模拟器之后成功获取到了 access_token 的值。

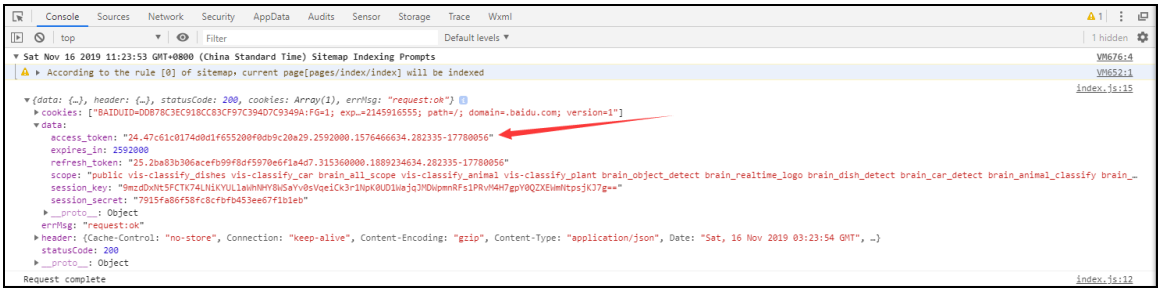


图 4.10 获取 token

可以看到，控制台执行了 success 回调函数，成功返回了 access_token。现在 Access_token 的问题已经解决了。接下来就要尝试去请求百度图片识别的 API 接口了。

通过阅读百度大脑 API 接口文档，我大概理解了百度 API 的接口使用方式。看一下下面的 API 文档的截图就可以了解其流程。

请求说明

请求示例

HTTP 方法： POST

请求URL： https://aip.baidubce.com/rest/2.0/image-classify/v2/advanced_general

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	是否必选	类型	可选值范围	说明
image	true	string	-	Base64编码字符串，以图片文件形式请求时必填。(支持图片格式：jpg，bmp，png，jpeg)，图片大小不超过4M。最短边至少15px，最长边最大4096px。**注意：图片需要base64编码、去掉编码头后再进行urlencode。**
baike_num	否	integer	0-5	返回百科信息的结果数，默认为0，不返回；2为返回前2个结果的百科信息，以此类推。

图 4.11 请求说明

我们首先需要使用微信封装的 request 方法请求这个 API 的 URL 地址，注意要使用该 POST 方式。然后这个 URL 的具体内容就是文档中给出的地址，不同的 API 接口有不同的请求地址。URL 地址需要添加一些参数，如 access_token、Content-Type 等等。同时，还需要一个 image 参数作为图片的标识，这个 image 是将图片转换成了 base64 编码的格式，也就是将图片传换成了一串字符。

再查阅微信小程序 wx.request 方法就可以知道，image 这个参数可以通过小程序的 data 属性表示。这样，API 接口的请求过程就可以在小程序中表示出来了。

data 参数说明

最终发送给服务器的数据是 String 类型，如果传入的 data 不是 String 类型，会被转换成 String 。转换规则如下：

- 对于 `GET` 方法的数据，会将数据转换成 query string （`encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...`）
- 对于 `POST` 方法且 `header['content-type']` 为 `application/json` 的数据，会对数据进行 JSON 序列化
- 对于 `POST` 方法且 `header['content-type']` 为 `application/x-www-form-urlencoded` 的数据，会将数据转换成 query string （`encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...`）

图 4.12 参数说明

下面尝试在小程序中进行程序编写。

首先添加一个变量“token”用来存储我获取到的 access_token 值，然后将 success 回调函数修改成下图中的代码。

```
success: function(res) {
  console.log('Request successful !')
  console.log(res.data)
  token = res.data.access_token;
  console.log('My token is : ' + token);
},
```

图 4.13 调用方式

也就是将 token 从服务器返回的 json 数据中提取出来。现在，token 已经存储到了变量中。

4.3 图片上传

现在有一个问题，微信小程序怎样获取图片数据呢？微信常用的方式是将用户的图片文件上传到微信开发者的服务器上，服务器接收到图片数据后再进行相关的处理。不管怎样，都首先需要使用微信提供的接口上传图片文件。那么首先研究一下怎样使用微信的图片上传接口。

微信提供了“从本地选择图片或使用相机拍照”的接口“wx.chooseImage”。

wx.chooseImage(Object object)

从本地相册选择图片或使用相机拍照。

参数

Object object

属性	类型	默认值	必填	说明
count	number	9	否	最多可以选择的图片张数
sizeType	Array.<string>	['original', 'compressed']	否	所选的图片的尺寸
sourceType	Array.<string>	['album', 'camera']	否	选择图片的来源
success	function		否	接口调用成功的回调函数
fail	function		否	接口调用失败的回调函数
complete	function		否	接口调用结束的回调函数（调用成功、失败都会执行）

图 4.14 参数说明

现在尝试使用该接口上传一下图片文件。在逻辑文件中编写图片上传按钮的事件处理函数。

```
get_image: function(res) {
  wx.chooseImage({
    count: 1,
    sizeType: ['original', 'compressed'],
    sourceType: ['album', 'camera'],
    success(res) {
      const tempFilePaths = res.tempFilePaths
      console.log('Image Path is : ' + tempFilePaths)
    }
  })
}
```

图 4.15 获取图片

下面运行小程序，点击上传图片按钮，会弹出系统资源管理器，在里面选择图片文件点击确定，即可成功上传图片文件。在 success 回调函数中，把上传图片的文件路径放到变量中，再通过控制台显示出来即可。在控制台中显示了图片路径。

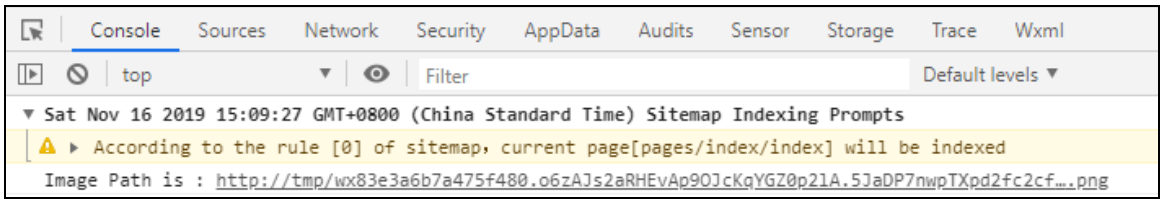


图 4.16 控制台信息

为了确定图片真的已经上传到了微信小程序中，我现在把上传的图片显示在小程序界面中。小程序文档中指出，tempFilePaths 可以作为<image>组件的 src 地址，所以显示图片就很方便了。

经过一番调试，该功能终于实现了。首先，在页面布局中添加<image>组件，该组件的 src 设置为动态变量 imageUrl，以便在图片上传后动态改变。之后编写相应的事件处理函数，首先添加一个默认的图片索引地址，将该图片放到向程序工程目录下，使用 data 属性保存其路径信息。

```
data: {
  imageUrl: "/images/upload_test.png"
},
```

图 4.17 图片地址

然后在原有的图片上传事件处理函数的基础上，编写增加代码如下：

```
get_image: function(res) {
  var that = this
  wx.chooseImage({
    count: 1,
    sizeType: ['original', 'compressed'],
    sourceType: ['album', 'camera'],
    success(res) {
      var tempFilePaths = res.tempFilePaths
      that.setData({
        imageUrl: tempFilePaths
      })
      console.log('Image Path is : ' + tempFilePaths)
      console.log(res)
    }
  })
```

图 4.18 获取图片地址

画横线的代码是图片动态显示的关键代码。这样，就是实现了图片的上传和动态更新。PS：这部分内容参考课本 P132“上传头像”部分。

4.4 图片格式转化

好的，现在就可以看一下怎样把图片转换成 base64 格式的数据了。

经过查找资料，发现将图片转换成 base64 格式的方式也有不少。我是用一种微信提供的文件管理接口实现的。在上传图片的成功回调函数中（此时已经得到了本地图片地址），使用文件系统管理方法，读取本地文件的内容。这里指定读取文件的编码格式为 base64，然后通过控制台显示出来就可以了。

```
wx.getFileSystemManager().readFile ({
  filePath: res.tempFilePaths[0],
  encoding: 'base64',
  complete: res=> {
    console.log('complete')
    //console.log(res)
  },
  success: res => {
    console.log('' + res.data)
  }
})
```

图 4.19 图片转码

上面截了一下关键位置的代码，这部分代码是放在 success 回调函数中，因为回调函数中已经获取到了上传图片的本地地址。

看一下控制台的信息：

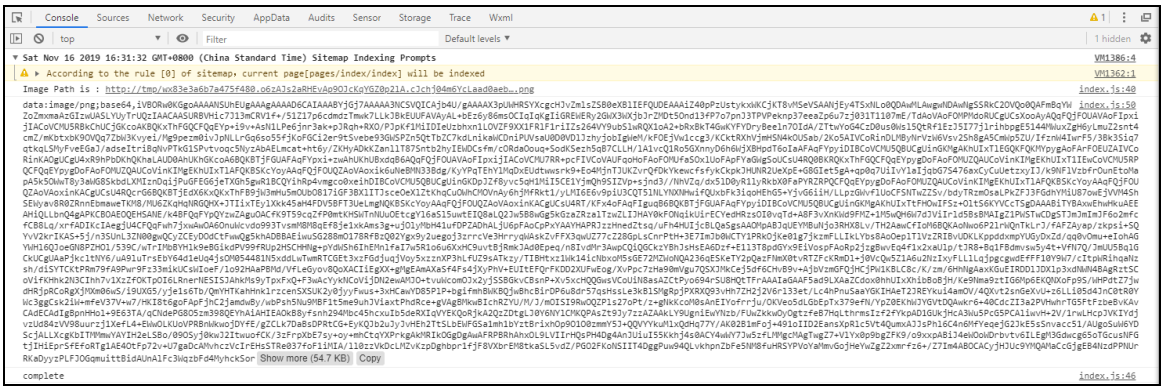


图 4.20 控制台信息

可以看到，控制台输出了图片的 base64 编码值。稍微了解一下 base64 编码，该编码可以用于 HTML 环境下的较长信息的标识，只要用于 canvas 画布的图片显示。它的好处在于，可以在没有上传图片文件的情况下，使用该编码在 HTML 中插入该图片。这种编码有固定的表示形式：“ .....”，也有固定的格式转换格式。具体的内容就不再详细研究了。在我的测试中，上传的原图片大小是 20.5KB，转换成 base64 编码之后的数据大小为 54.7KB。

4.5 API 请求

图片已经准备好了，现在已经可以去调用百度图片识别的 API 了。下面需要对包括“access_token”、“base64 编码”等数据进行整合，并以动物识别为例尝试该 API 的调用。

首先添加识别按钮，然后编写相应的事件处理函数。

```
<view>
  <button bindtap="recognition_image">
    识别该图片
  </button>
</view>
```

图 4.21 识别按钮

事件处理函数就是要进行 API 接口调用了。

```
recognition_image: function(res) {
  wx.request({
    url: 'https://aip.baidubce.com/rest/2.0/image-classify/v1/animal?access_token=' + token,
    method: 'POST',
    header: {
      'content-type': 'application/x-www-form-urlencoded'
    },
    data: {
      image: base64
    },
    complete: res => {
      console.log('Complete recognition_image')
      console.log(res)
    },
    success: res => {
      console.log('Success')
      console.log(res)
    }
  })
}
```

图 4.22 识别图片

上面就是按照百度动物识别 API 文档的要求，结合小程序 wx.request 方法的属性写出来的事件处理函数。

我在网上搜了一个狗的图片，添加到了小程序文件夹中作为测试图片。运行模拟器进行测试。依次点击“获取 access_token”、“上传图片”以及“识别该图片”按钮，在控制台看到 API 返回的数据。

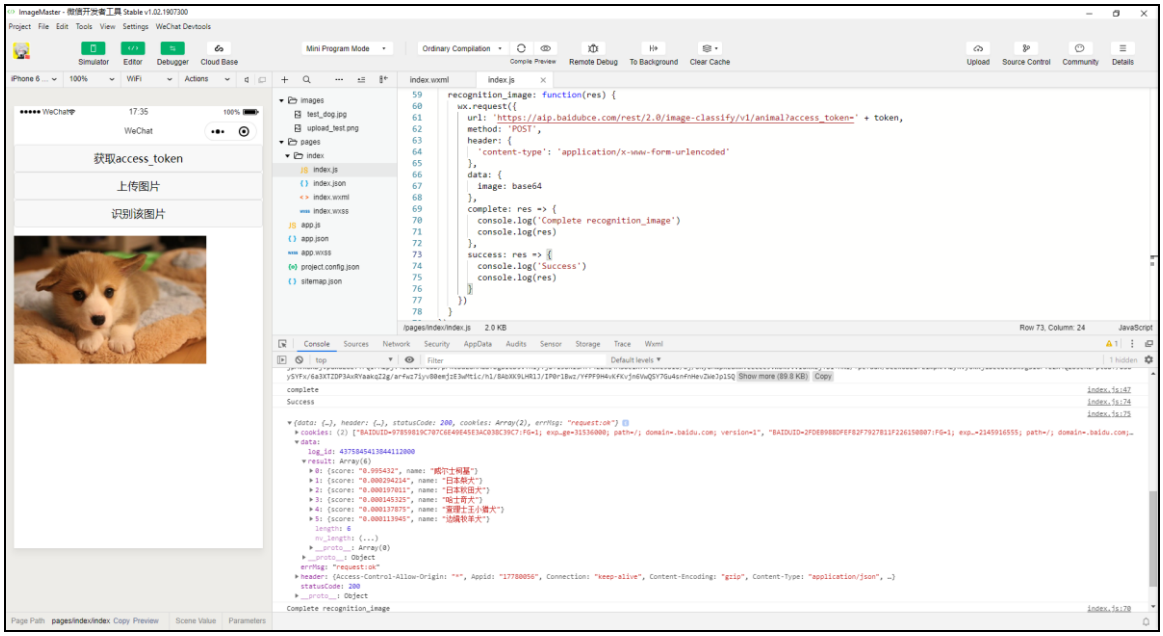


图 4.23 接口返回值

可以看到，在 API 返回的 JSON 数据中，标注了识别结果，其中置信度最高的结果是“威尔士柯基”。我特意搜了一下这个“威尔士柯基”，识别结果还是挺准的。

还要测试一下非动物图片的识别情况。现在我继续上传一个头像图片进行测试，看看

测试结果。可以看到，非动物也是可以被识别出来的，识别结果只有一个高置信度的“非动物”结果。

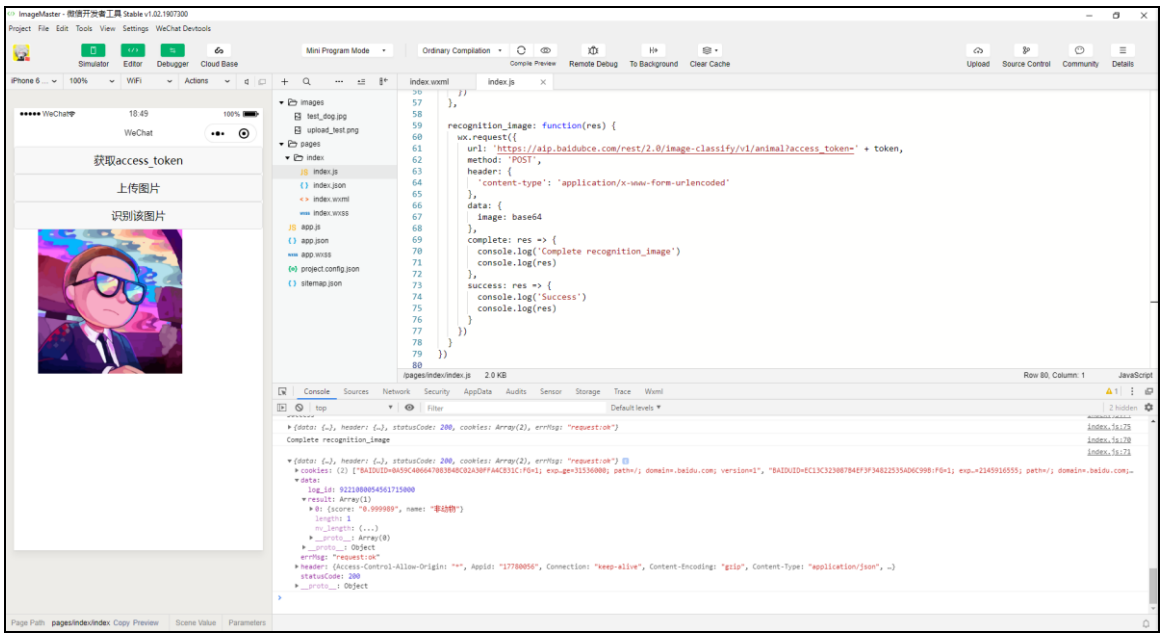


图 4.24 非动物识别

程序做到这里，可以说最大的障碍已经被克服了。目前已经实现了“动物识别”，下面就是逐步实现“植物识别”、“车型识别”等功能就可以了，他们的基本流程是一样的。

4.6 实现植物识别

现在来整理一下代码，尝试实现“植物识别”功能。

通过阅读 API 文档可以看出，其实这几种不同的图像识别的接口，只是 URL 地址不同，其他参数都是一样的。所以我设置了一个变量 apiUrl 用来存放不同接口的 URL 地址，为“植物识别”编写一个新的事件处理函数。这个事件处理函数与“动物识别”唯一的不同就是 apiUrl 不一样罢了。

```
get_Plant_image: function (res) {
  var that = this
  wx.chooseImage({
    count: 1,
    sizeType: ['original', 'compressed'],
    sourceType: ['album', 'camera'],
    success(res) {
      var tempFilePaths = res.tempFilePaths
      apiUrl = 'https://aip.baidubce.com/rest/2.0/image-classify/v1/plant'
      that.setData({
        imageUrl: tempFilePaths,
      })
      console.log('My API URL is : ' + apiUrl)
      console.log('Image Path is : ' + tempFilePaths)
      // console.log(res)
      wx.getFileSystemManager().readFile({
        filePath: res.tempFilePaths[0],
        encoding: 'base64',
        // complete: res=> {
        //   console.log('complete')
        //   console.log(res)
        // },
        success: res => {
          base64 = res.data
          // console.log('' + base64)
        }
      })
    }
  })
},
```

图 4.25 植物识别

可以看到，我把请求接口的 URL 放到了 apiUrl 变量中。同时精简了控制台显示的数据，只保留关键的数据。

至于图像识别的事件处理函数，只需要把之前的 POST 请求地址改成用 apiUrl 表示就可以了。

```
recognition_image: function(res) {
  wx.request({
    url: apiUrl + '?access_token=' + token,
    method: 'POST',
    header: {
      'content-type': 'application/x-www-form-urlencoded'
    },
    data: {
      image: base64
    },
    success: res => {
      console.log('recognition_image Success')
      console.log(res.data.result[0].name)
    }
  })
}
```

图 4.26 识别图片

可以看到画横线的那句话，就是把 POST 地址改了一下，使程序的使用范围更广了。同时还要注意到，我精简了控制台的输出信息，只保留置信度最高的结果。

现在就可以识别植物了。

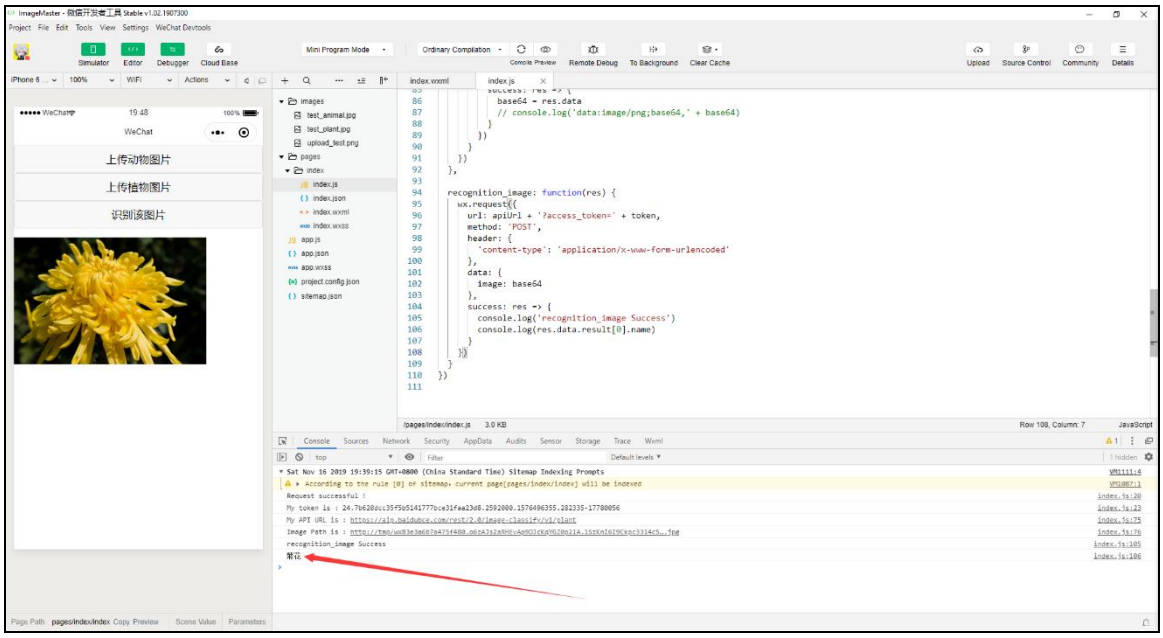


图 4.27 识别植物

现在控制台简洁多了，而且也能正确显示置信度最高的识别结果了。

4.7 实现车辆识别

现在继续做车型识别。现在添加功能就比较简单了，只需修改 apiUrl。

这时遇到了一个问题，上传图片的大小错误。

图 4.28 控制台信息

为了方便调试，我将控制台信息输出添加了条件判断语句，如果发生错误就输出错误信息，没有错误就输出识别结果。首先测试了图片大小错误的情况，然后测试了正常的图片，控制台输出的信息如下：

图 4.29 控制台信息

这样调试就方便多了。现在来解决图片 size 的问题。以“车型识别”为例，其图片要求如下：

参数	是否必选	类型	可选值范围	说明
image	是	string	-	图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式。**注意：图片需要base64编码、去掉编码头后再进行urlencode。**

图 4.30 图片要求

我刚才上传的“image size error”的图片，其像素为 1920px*1080px，符合文档中图片长宽的要求。原图大小为 460KB，其编码为 base64 之后的大小为 1.2MB，同样符合要求。这就奇怪了，因为这些参数都是符合文档要求的。

之后我又进行了其他测试，换了一张 1920px*1080px 的图片，不会报错。所以看起来图片不合适的原因应该是图片大小的问题。然而，当我继续测试使用 2048px*1335px，原图大小为 1.5MB 的图片时，竟然可以正常识别。所以这张图片报错的原因暂时没有搞清楚。

先不管这里了，因为我测试了其他图片，都没有出现问题，只有这一张出现图片大小的问题。就先放一下吧。下面整理一下小程序，优化一下界面。

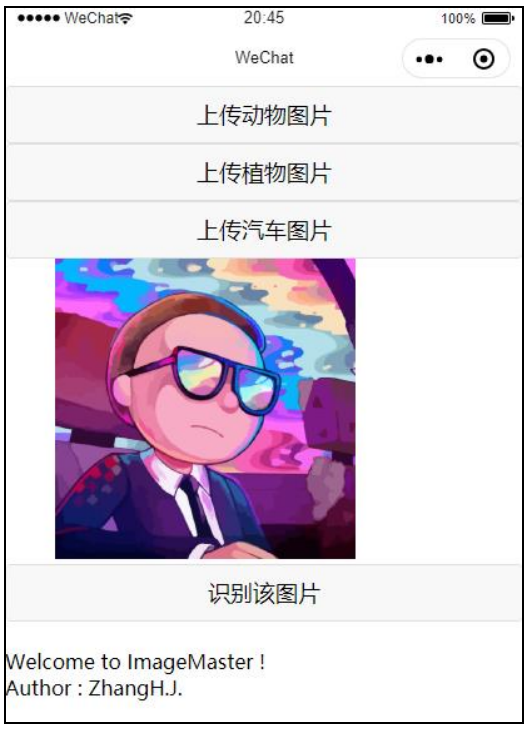


图 4.31 界面展示

上图是初始化界面，调整了“识别图片”按钮和图片之间的相对位置，同时添加了文本框显示提示信息。下面以“动物识别”为例，展示识别效果。可以看到，通过添加文本框，可以直观地显示识别结果，非常方便。

图 4.32 识别结果

下面就来编写一下样式文件吧。

4.8 Taro 样式测试

（1）Taro 样式介绍

样式文件我想尝试使用 UI 框架编写，经过对比多种 UI 框架，最终决定使用 Taro UI 作为我的 UI 框架使用。以下是 Taro 官网介绍：

Taro 是由京东凹凸实验室打造的多端开发解决方案。现如今市面上端的形态多种多样，Web、ReactNative、微信小程序等各种端大行其道，当业务要求同时在不同的端都要求有所表现的时候，针对不同的端去编写多套代码的成本显然非常高，这时候只编写一套代码就能够适配到多端的能力就显得极为需要。使用 Taro，我们可以只书写一套代码，再通过 Taro 的编译工具，将源代码分别编译出可以在不同端（微信小程序、H5、RN 等）运行的代码。

下面就按照官网的教程，尝试在本地小程序中使用该框架进行练习。为了防止原有的小程序结构被破坏，首先新建一个小程序作为练习使用。

Taro 需要使用 Node.js，要求版本在 v8.x 或以上。但我之前安装的版本是 v4.4.3，所以现在先重新安装新版本的 Node.js。

```
PS C:\Users\ZHJ> node -v
v4.4.3
```

图 4.33 版本查看

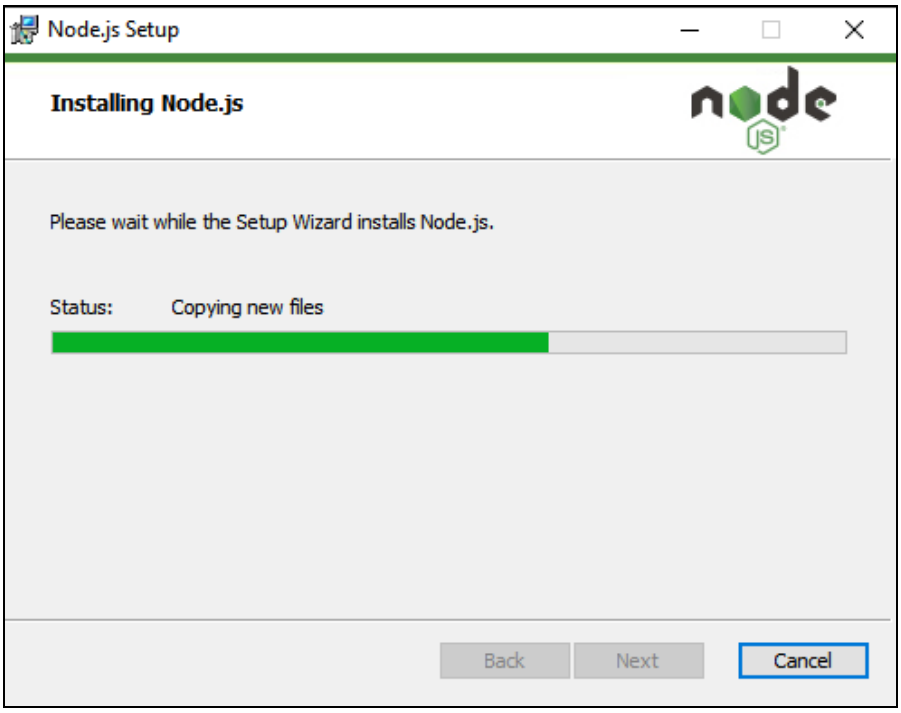


图 4.34 安装新版本 NodeJs

经过重新安装，现在已经升级到了最新的长期支持版本 V12.13.1。

```
PS C:\Users\ZHJ> node -v
v12.13.1
```

图 4.35 检查新版本

下面开始项目初始化安装。

（2） 使用前的准备

首先安装 Taro 脚手架工具，这里需要使用 npm 包管理工具进行安装。

```
PS C:\Users\ZHJ> npm install -g @tarojs/cli
npm WARN deprecated babel-plugin-remove-dead-code@1.3.2: use babel-plugin-minify-dead-code-elimination and babel-plugin-minify-guarded-expressions
npm WARN deprecated crypto@1.0.1: This package is no longer supported. It's now a built-in Node module. If you've depended on crypto, you should switch to the one that's built-in.
npm WARN deprecated eslint-plugin-typescript@0.12.0: Deprecated: Use @typescript-eslint/eslint-plugin instead
npm WARN deprecated joi@14.3.1: This module has moved and is now available at @hapi/joi. Please update your dependencies as this version is no longer maintained an may contain bugs and security issues.
npm WARN deprecated core-js@2.6.10: core-js@<3.0 is no longer maintained and not recommended for usage due to the number of issues. Please, upgrade your dependencies to the actual version of core-js@3.
npm WARN deprecated fsevents@1.2.9: One of your dependencies needs to upgrade to fsevents v2: 1) Proper nodejs v10+ support 2) No more fetching binaries from AWS, smaller package size
npm WARN deprecated topo@3.0.3: This module has moved and is now available at @hapi/topo. Please update your dependencies as this version is no longer maintained an may contain bugs and security issues.
npm WARN deprecated hoek@6.1.3: This module has moved and is now available at @hapi/hoek. Please update your dependencies as this version is no longer maintained an may contain bugs and security issues.
npm WARN deprecated cross-spawn-async@2.2.5: cross-spawn no longer requires a build toolchain, use it instead
npm WARN deprecated circular-json@0.3.3: CircularJSON is in maintenance only, flatted is its successor.
C:\Users\ZHJ\AppData\Roaming\npm\taro -> C:\Users\ZHJ\AppData\Roaming\npm\node_modules\@tarojs\cli\bin\taro

> core-js@2.6.10 postinstall C:\Users\ZHJ\AppData\Roaming\npm\node_modules\@tarojs\cli\node_modules\core-js
> node postinstall || echo "ignore"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)


> ejs@2.7.4 postinstall C:\Users\ZHJ\AppData\Roaming\npm\node_modules\@tarojs\cli\node_modules\ejs
> node ./postinstall.js

Thank you for installing EJS: built with the Jake JavaScript build tool (https://jakejs.com/)

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\@tarojs\cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ @tarojs/cli@1.3.25
added 1098 packages from 797 contributors in 117.159s
PS C:\Users\ZHJ>
```

图 4.36 安装脚手架

之后需要使用 Yarn 依赖管理工具安装相应的依赖，先来安装 Yarn。对于 Windows 系统，可以下载官方提供的安装器进行安装。安装 Taro 完成后，可以使用命令安装相应的依赖。

```
PS C:\Users\ZHJ> yarn global add @tarojs/cli
yarn global v1.19.1
[1/4] Resolving packages...
warning @tarojs/cli > babel-plugin-remove-dead-code@1.3.2: use babel-plugin-minify-dead-code-elimination and babel-plugin-minify-guarded-expressions
warning @tarojs/cli > crypto@1.0.1: This package is no longer supported. It's now a built-in Node module. If you've depended on crypto, you should switch to the one that's built-in.
warning @tarojs/cli > eslint-plugin-typescript@0.12.0: Deprecated: Use @typescript-eslint/eslint-plugin instead
warning @tarojs/cli > joi@14.3.1: This module has moved and is now available at @hapi/joi. Please update your dependencies as this version is no longer maintained an may contain bugs and security issues.
warning @tarojs/cli > chokidar > fsevents@1.2.9: One of your dependencies needs to upgrade to fsevents v2: 1) Proper nodejs v10+ support 2) No more fetching binaries from AWS, smaller package size
warning @tarojs/cli > fbjs > core-js@2.6.10: core-js@<3.0 is no longer maintained and not recommended for usage due to the number of issues. Please, upgrade your dependencies to the actual version of core-js@3
.

warning @tarojs/cli > babel-plugin-preval > babel-register > core-js@2.6.10: core-js@<3.0 is no longer maintained and not recommended for usage due to the number of issues. Please, upgrade your dependencies to
 the actual version of core-js@3.
warning @tarojs/cli > babel-generator > babel-runtime > core-js@2.6.10: core-js@<3.0 is no longer maintained and not recommended for usage due to the number of issues. Please, upgrade your dependencies to the
actual version of core-js@3.
warning @tarojs/cli > joi > topo@3.0.3: This module has moved and is now available at @hapi/topo. Please update your dependencies as this version is no longer maintained an may contain bugs and security issues
.
warning @tarojs/cli > joi > hoek@6.1.3: This module has moved and is now available at @hapi/hoek. Please update your dependencies as this version is no longer maintained an may contain bugs and security issues
.
warning @tarojs/cli > joi > topo > hoek@6.1.3: This module has moved and is now available at @hapi/hoek. Please update your dependencies as this version is no longer maintained an may contain bugs and security
 issues.
warning @tarojs/cli > npm-check > execa > cross-spawn-async@2.2.5: cross-spawn no longer requires a build toolchain, use it instead
warning @tarojs/cli > stylelint > file-entry-cache > flat-cache > circular-json@0.3.3: CircularJSON is in maintenance only, flatted is its successor.
[2/4] Fetching packages...
info There appears to be trouble with your network connection. Retrying...
info fsevents@1.2.9: The platform "win32" is incompatible with this module.
info "fsevents@1.2.9" is an optional dependency and failed compatibility check. Excluding it from installation.
[3/4] Linking dependencies...
warning "@tarojs/cli > @typescript-eslint/parser > @typescript-eslint/typescript-estree > tsutils@3.17.1" has unmet peer dependency "typescript@>=2.8.0 || >= 3.2.0-dev || >= 3.3.0-dev || >= 3.4.0-dev || >= 3.5
.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev || >= 3.7.0-beta".
[4/4] Building fresh packages...
success Installed "@tarojs/cli@1.3.25" with binaries:
      - taro
Done in 161.22s.
PS C:\Users\ZHJ>
```

图 4.37 安装依赖

（3） 安装项目模板

现在到小程序工程目录下，进行 Taro 模板的创建。这里注意，需要提前安装 python2 版本。

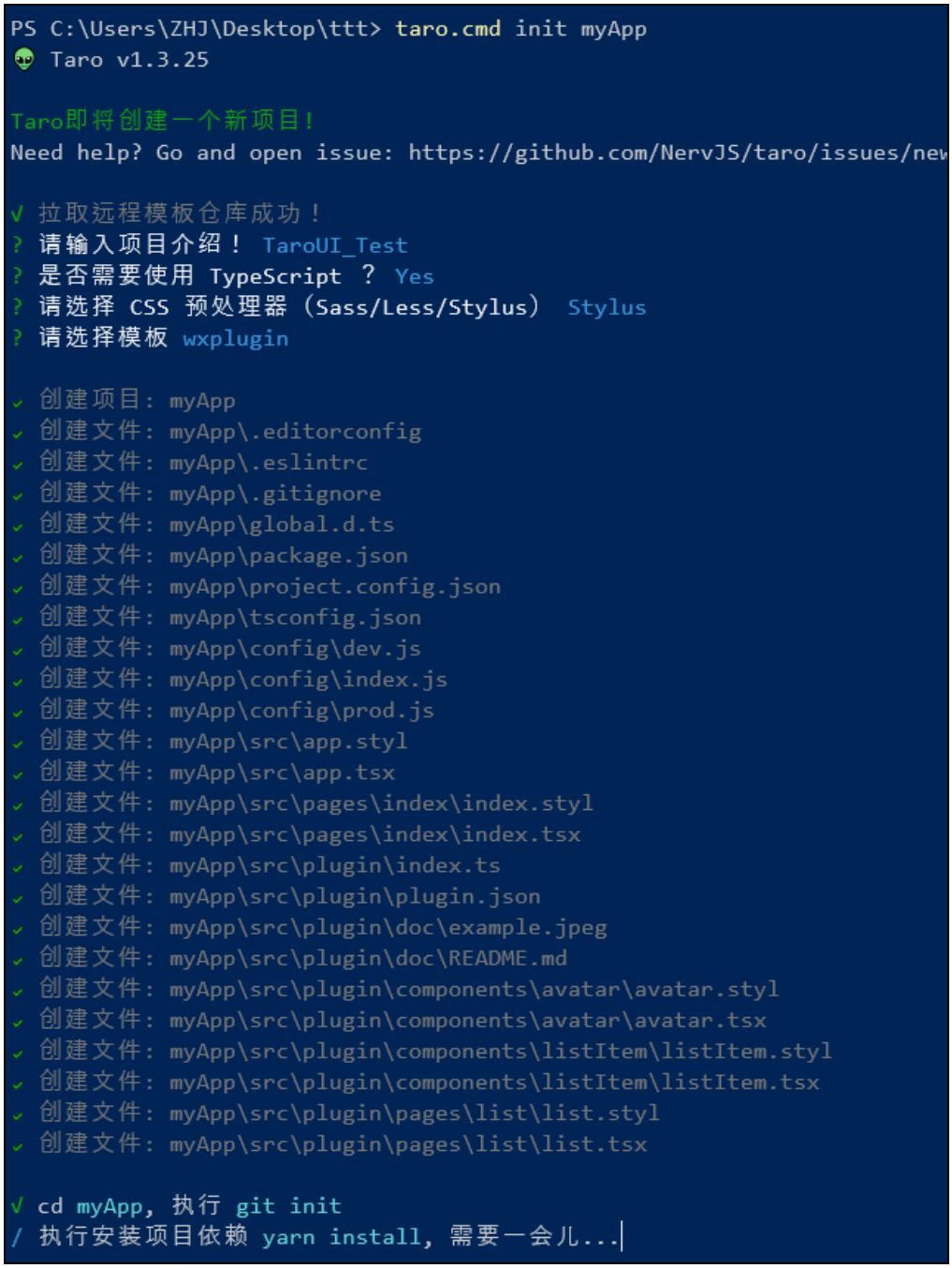


图 4.38 安装模板

现在模板安装完成了，可以进行模块引入了。

（4） 引入 Taro 模块

下面进行 taro 模块的导入，需要使用 npm 本地编译 taro 模块。

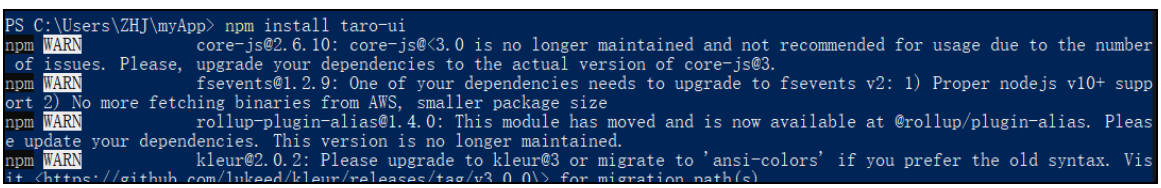


图 4.39 编译模块

编译完成后，我发现 taro 工程的目录结构与微信小程序的工程结构不一样。而且 taro 的样式文件编写的格式也不是 wxss 格式，这就导致我还要学习一些其他的样式格式规则才能上手编写 taro。为了尽快完成小程序，我转向了使用 WeUI 框架。

4.9 WeUI 样式测试

（1） WeUI 介绍

WeUI 是一套与微信原生 UI 一致的 UI 库，核心文件是 weui.css，只需要获取到该文件，然后在页面中引入，即可使用 WeUI 的组件。首先到其 GitHub 网站下载源代码，新建一个工程文件用来进行测试和修改。在微信开发者工具中打开该工程。

图 4.40 工程目录

此时模拟器中就可以查看到 WeUI 框架的各种组件样式。

图 4.41 样式展示

（2） 样例编写

下面来研究一下 WeUI 是怎样组织样式编写的。

以 Grid 样式为例，先看一下他的效果。

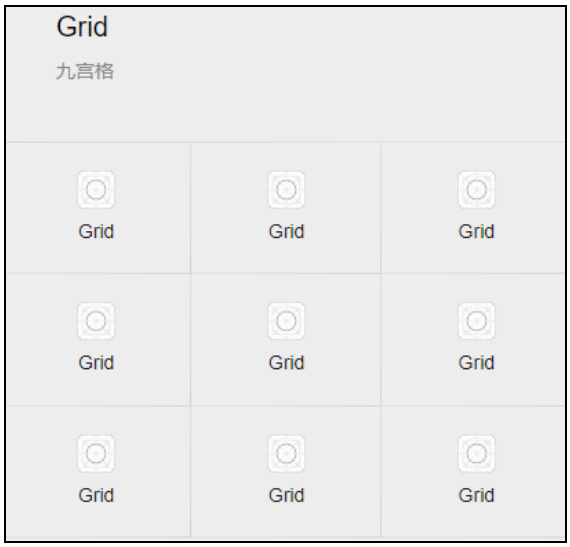


图 4.42 gird 样式效果

再看一下它的布局文件。

```
<view class="page">
    <view class="page__hd">
        <view class="page__title">Grid</view>
        <view class="page__desc">九宫格</view>
    </view>
    <view class="page__bd">
        <view class="weui-grids">
            <block wx:for="{{grids}}" wx:key="*this">
                <navigator url="" class="weui-grid" hover-class="weui-grid_active">
                    <image class="weui-grid__icon" src="../images/icon_tabbar.png" />
                    <view class="weui-grid__label">Grid</view>
                </navigator>
            </block>
        </view>
    </view>
</view>
```

图 4.43 布局文件

可以看到，他是直接使用了 WeUI 提供的 class，利用 wx:for 循环实现 Grid 组件的控制。而逻辑文件中给出了 grids 变量数组的数据，从 0 到 8 表示循环 9 次，显示 9 个 Grid 宫格。

```
Page({
    data: {
        grids: [0, 1, 2, 3, 4, 5, 6, 7, 8]
    }
});
```

图 4.44 Grid 逻辑文件

现在尝试将该样式移植到我的小程序中。移植样式最关键的就是它的 weui.wxss，按照教程，可以使用外联样式引入的方式将改样式文件引入到项目中。

首先将 style 文件夹整体复制到我的工程目录中，然后在全局样式中导入 weui 的样式文件。

```
/**app.wxss**/
@import 'style/weui.wxss';
```

图 4.45 导入样式

之后创建一个新的页面用来测试样式内容，并将其页面作为默认显示的页面。

```
"pages": [
  "pages/weui/weui",
  "pages/index/index"
],
```

图 4.46 创建新页面

之后就可以分别在新创建的页面中编写 grid 布局和逻辑文件，内容与示例代码一样。这样效果就与示例代码相似了。

官方例程中 Grid 的九个组件，是由<navigator>组件组成的。<navigator>组件用于实现页面之间的跳转，而我想要通过 button 组件实现用户选择不同类型图片的功能，所以需要使用 button 组件。下面再测试一下 button 组件的使用。

首先是从官方历程中抽取出我们需要的组件。

```
<!--pages/weui/weui.wxml-->
<view class="page">
    <view class="page__bd page__bd_spacing">
        <button class="weui-btn" type="primary">页面主操作</button>
        <button class="weui-btn" type="default">页面次要操作</button>
        <button class="weui-btn" type="warn">警告类操作</button>
    </view>
</view>
```

图 4.47 组件布局

现在就需要移植一下样式文件了。在例程样式文件中找到”weui-btn”、”class”、”page__bd”、”page__bd__spacing”的文件内容，复制到本页面的样式文件中即可。最后实现的效果如下图所示。

图 4.48 button 效果

下面尝试将原先的逻辑功能与样式结合起来。首先将样式文件复制到 index.wxss 文件中，然后将原先的 button 组件的样式改成测试时的样式。

于是模拟器中的样式就改为如下图所示：

图 4.49 按钮样式

下面修改标题文本样式，将 WeUI 的标题文本样式移植到我的程序中。

图 4.50 标题样式

下面我想将这三个上传图片的按钮进行水平均匀排布，找到按钮样式相关的样式文件，修改后的样式如下图所示。

图 4.51 按钮样式修改

之后，我又修改了上传按钮和文字输出按钮的样式，添加了提示图片和提示信息。最终的样式如下图所示。

图 4.52 最终样式

5 关键问题

针对完成大作业过程中遇到的问题，分析描述在此过程中的关键问题，如重点难点等。

本次大作业“微信小程序的图片识别”程序实现过程中，重点问题是“图片上传”、“图片转码”、“API 调用”以及“界面设计”。其中图片上传主要使用了微信小程序中的 wx.chooseImage 方法;图片转码使用 wx.getFileSystemMangaer 方法解决；API 调用使用 wx.request 方法实现；界面设计使用了微信小程序的官方 UI 框架实现。

6 使用说明

提供微信小程序体验版的访问方式(小程序码)。

图 6.1 小程序码

下面给出完成的微信小程序的使用说明。使用体验版微信小程序，在自己的手机上进行测试。下面将测试过程及截图展示如下。

（1）上传图片

用户进入小程序后，首先需要通过相册或拍照，上传需要识别的图片。

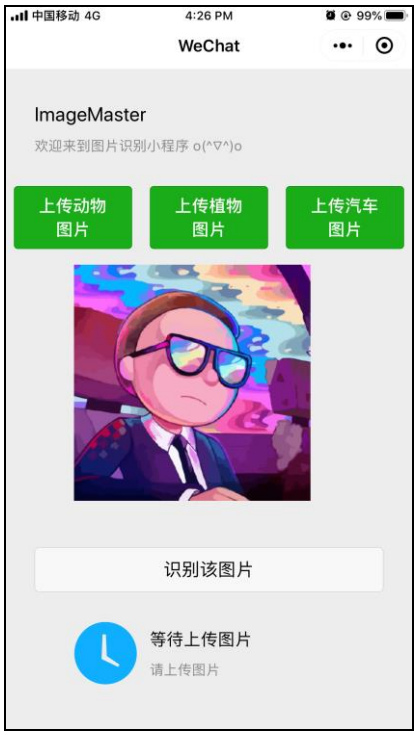


图 6.2 上传图片

以“动物识别”为例，点击“上传动物图片”按钮，进行图片上传。

（2）选择图片

以从相册上传图片为例，点击从相册上传图片按钮，在相册中选择想要识别的图片。

图 6.3 选择图片

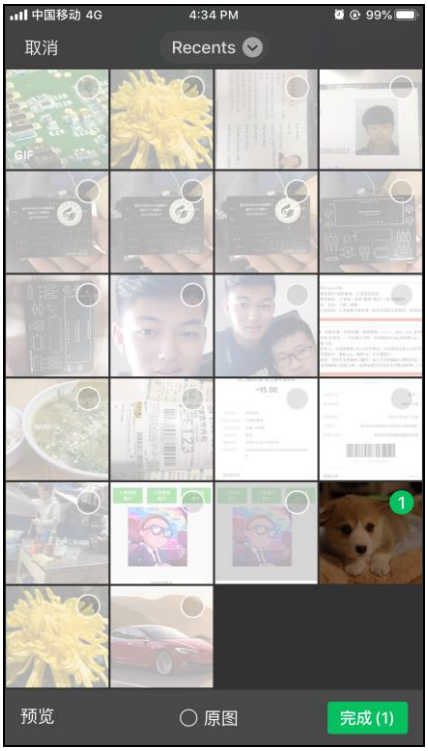


图 6.4 相册选择

（3）识别图片

上传图片完成后，点击小程序界面的“识别该图片”按钮，进行图片识别。经过短时间的等待，即可显示识别结果。

（4）识别结果

界面下方可以展示图片识别的结果。

图 6.5 上传成功

图 6.6 识别结果

7 课程大作业总结

本次大作业进行地还算顺利，通过阅读 API 文档，逐步实现了项目要求。就大作业本身来说，它的逻辑并不复杂。通过阅读相关的技术文档，将整个 API 调用过程分成几个主要的步骤，分步解决每步出现的问题，最终即可完成开发。

虽然没有经历过真正的项目开发，但本次大作业从需求分析到整个功能的实现，我都尽可能去贴近真实的开发过程。本项目需要实现的功能其实并不困难，逻辑也是非常简单，但我认为更重要的是从本次开发过程中积攒的微薄的项目经验。为跟踪项目进度，本次大作业尝试使用了 Git 来进行版本控制管理，使得整个项目的完整进度都有据可循。项目文件也上传至了我的 GitHub 仓库。

本次大作业本应实现的项目需求已经实现了，包括“动植物识别和车辆识别”。如果以后有机会改进，会加入更多可供用户选择的识别类型。小程序的界面也需要进行相关的优化，这也是以后需要改进的重点。

另外，因为我需要实现的功能过于简单，很多组件和逻辑功能都没能在本次大作业中使用。如果以后有机会或其他想法时，再进行其他功能的拓展。本次大作业实现的图片识别小程序 ImageMaster 已经上线了，欢迎测试使用。