微信投票小程序demo说明文档

July 2, 2022

Contents

1 概述

本文档为简易微信投票小程序的说明文档，旨在说明微信小程序的开发流程。本文档分为前端开发说明和后端开发说明两部分。本例程实现用户在线投票以及展示投票结果的功能。后端采用了Django作为框架，数据库采用PostgreSQL。完整demo例程见`https://github.com/zl11250422/wechat_vote`。

2 前端开发

2.1 微信小程序界面开发流程

微信小程序开发流程如下：

1. 安装开发工具。前往开发者工具下载页面，根据自己的操作系统下载对应的安装包进行安装，有关开发者工具更详细的介绍可以查看《开发者工具介绍》。打开小程序开发者工具，用微信扫码登录开发者工具，准备开发小程序，如图1所示。(开发者工具下载页面：`https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html`)

Figure 1: 开发者工具登录页面

2. 新建或导入小程序。扫码登录页面如图2所示，我们可通过点击中间的加号按钮新建一个小程序，或者点击图右上角的导入按钮来导入已有的小程序。

Figure 2: 新建或导入小程序

3. 申请账号。新建小程序后，在创建小程序页面需要填入一系列相关信息，包括项目名称目录等。此外，我们需要填写AppID号，小程序的AppID相当于小程序平台的一个身份证，后续会在很多地方要用到AppID。如图3所示，点击注册按钮，注册AppID。

Figure 3: 创建小程序页面

4. 开发小程序。开发界面如图4所示。

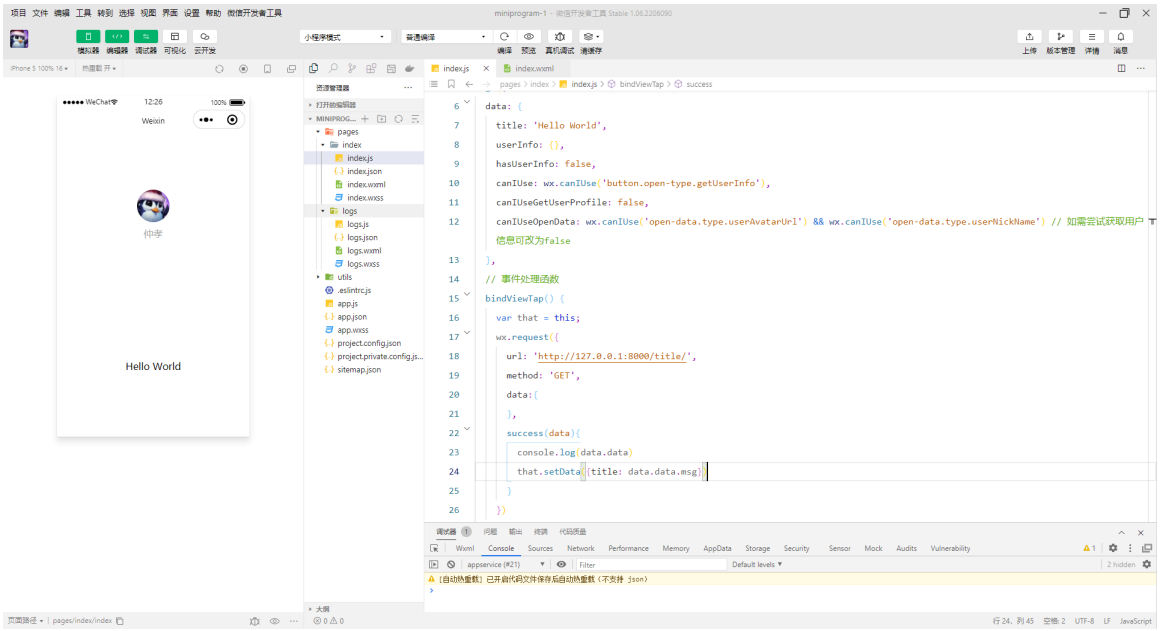

Figure 4: 小程序开发页面

2.2 微信小程序代码构成

下面我们以demo为例说明小程序的代码构成，demo目录如下所示:

```
/wechat_vote
└── /wechat
    ├── /pages
    │   ├── /index
    │   │   ├── /index.js
    │   │   ├── /index.wxml
    │   │   └── /index.wxss
    │   └── /result
    │       ├── /result.js
    │       ├── /result.wxml
    │       └── /result.wxss
    ├── /utils
    ├── /app.js
    ├── /app.json
    └── /app.wxss
```

在以上目录中，前端开发的目录为/wechat_vote/wechat/。
从目录中我们可以看到微信小程序的代码主要由四种类型的文件构成:

- .json文件后缀的JSON配置文件。在小程序种扮演静态配置的角色。
- .wxml文件后缀的WXML模板文件。用来描述程序页面的结构，相当于网页开发的HTML文件。
- .wxss文件后缀的WXSS样式文件。WXSS具有CSS大部分的特性。
- .js文件后缀的脚本逻辑文件。用来控制页面和用户的交互逻辑。

在/wechat_vote/wechat/的根目录下包含/pages文件夹，/utils文件夹，以及app.js, app.json, app.wxss文件等。其中以app为文件名的文件为当前小程序的整体配置。/pages文件夹为小程序具体页面设计文件夹，/utils文件夹为工具类文件夹。

2.3 投票小程序页面设计

在投票小程序设计中，我们设计了两个页面，具体见/index和/result目录，其中/index目录为投票页面，/result目录为投票结果展示页面，见图5和6。

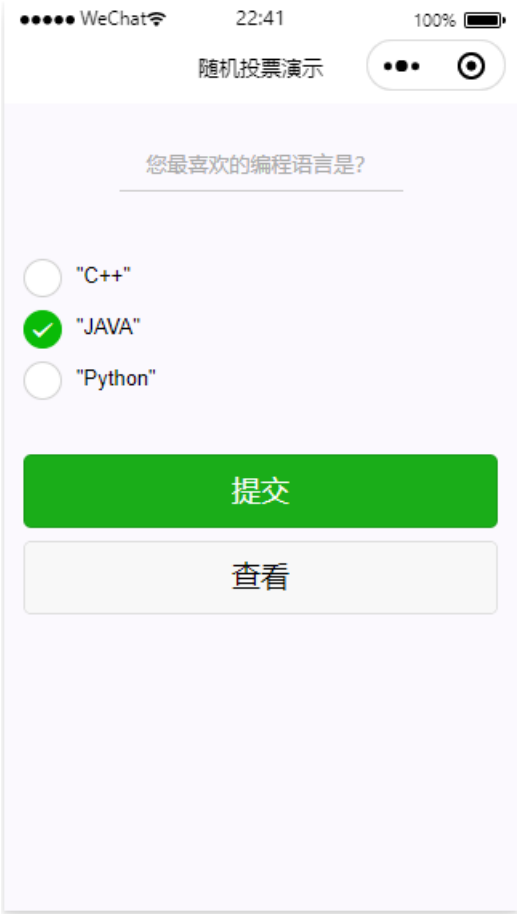

Figure 5: 用户投票页面

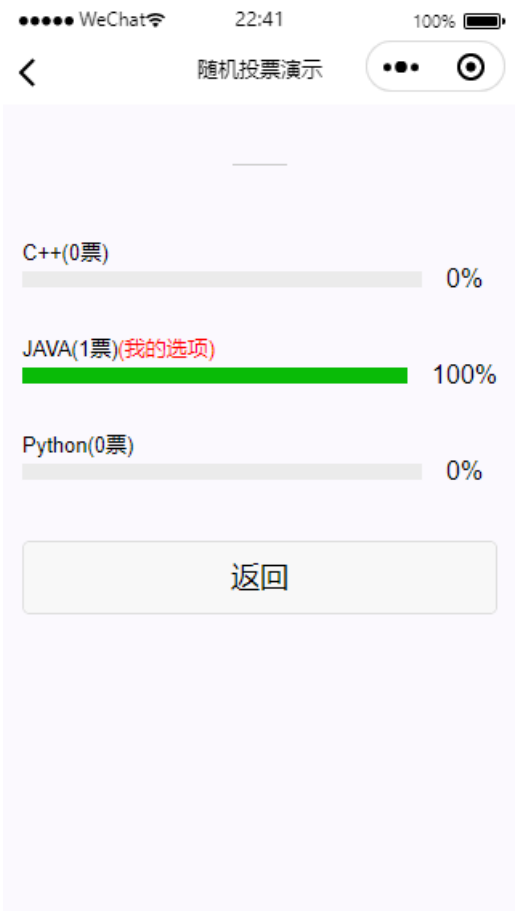

Figure 6: 投票结果展示页面

3 后端开发

3.1 Django代码构成

当今网页开发基本都是前后端分离的模式，因此产生的众多的web框架，在本demo中我们采用Django作为网络框架进行开发。主要开发目录如下：

```
/wechat_vote
├── /vote
│   ├── /settings.py
│   ├── /urls.py
│   └── /wsgi.py
├── /voteapp
│   ├── /models.py
│   └── /views.py
└── manage.py
```

在以上目录中，根目录/wechat_vote下的manage.py为Django项目自动生成的一个用于管理项目的脚本文件，需要通过python命令执行，manage.py接受的是Django提供的内置命令。需要说明的是，鉴于我们开发的原因，我们使用了Django内置的开发服务器，因此在后端开发中我们并没有对具体服务器进行配置。这是为了方便调试和开发，但是由于性能问题，建议只用来测试，不用在生产环境。使用命令如下：

python manage.py runserver 127.0.0.1:8000
下面我们对Django的代码构成做一简单介绍。

- urls.py。网址入口，接受前端发送的网址，采用正则匹配网址，关联到对应的views.py中的一个函数，访问网址就对应一个函数。
- views.py。处理用户发出的请求并返回需要的数据，从urls.py中对应过来。
- models.py。与数据库操作相关，存入或读取数据时用到这个，当然用不到数据库时，你可以不使用。
- settings.py。Django的设置，配置文件，比如debug的开关，数据库接口设置，静态文件的位置等等。

下面我们针对投票小程序的具体设计对以上文件做进一步介绍。

3.2 数据库设计

在设计投票小程序时，我们在数据库中主要建立的三个表，分别存储投票项目主题，投票选项，投票结果。建表语句具体在/wechat_vote/voteapp/models.py中，如图7所示。

```
# Create your models here
class DEMO_VOTE(models.Model):
    id = models.AutoField(primary_key=True, null=False)
    title = models.CharField(max_length=32)

class DEMO_VOTE_DETAIL(models.Model):
    id = models.AutoField(primary_key=True, null=False)
    vote_id = models.IntegerField(null=False)
    content = models.CharField(max_length=32)
    sort = models.IntegerField(null=False, default = 0)
    #CREATE_TIME = models.DateTimeField('日期',default=timezone.now)

class DEMO_VOTE_RESULT(models.Model):
    id = models.AutoField(primary_key=True, null=False)
    vote_id = models.IntegerField(null=False)
    detail_id = models.IntegerField(null=False)
    open_id = models.CharField(max_length=100)
    #CREATE_TIME = models.DateTimeField('日期',default=timezone.now)
```

Figure 7: 数据库建表类

3.3 后端请求处理

在后端的处理中，我们主要有三个函数进行处理，具体函数在/wechat_vote/voteapp/views.py中，其中包含三个函数，见图8所示。其中title函数用于返回投票页面加载的数据，submit函数用于处理前端的提交数据并对数据库进行操作，result用于读取数据库并将其返回投票结果页面。

```python
def title(request):
    if request.method == "GET":
        request.params = request.GET
    elif request.method in ['POST', 'PUT', 'DELETE']:
        request.params = json.loads(request.body)
    title = '你最喜欢的编程语言是？'
    query_title = models.DEMO_VOTE.objects.filter(title=title)
    if query_title.exists():
        pass
    else:
        models.DEMO_VOTE.objects.create(title = title)
    #vote_demo = models.DEMO_VOTE.objects.first().__dict__  #model转dict的一种方法，但会产生冗余的对象，不建议使用
    vote_demo = forms.model_to_dict(models.DEMO_VOTE.objects.first())
    vote_demo_detail = models.DEMO_VOTE_DETAIL.objects.all().values()
    vote_demo_detail = list(vote_demo_detail)
    #vote_demo = serializers.serialize('json', vote_demo)
    return JsonResponse({'vote_demo': vote_demo, 'vote_demo_detail': vote_demo_detail},json_dumps_params={'ensure_ascii':False},safe=False)

def submit(request):
    if request.method == "GET":
        request.params = request.GET
    elif request.method in ['POST', 'PUT', 'DELETE']:
        request.params = json.loads(request.body)

    detailId = request.params.dict()["ITEM"]
    openId = request.params.dict()["OPEN_ID"]
    result = models.DEMO_VOTE_RESULT.objects.filter(open_id=openId)
    RES = True
    if result.exists():
        RES = False
    else:
        models.DEMO_VOTE_RESULT.objects.create(vote_id = 0, detail_id = detailId, open_id = openId)
        RES = True
    return JsonResponse({'RES': RES},json_dumps_params={'ensure_ascii':False},safe=False)

def result(request):
    if request.method == "GET":
        request.params = request.GET
    elif request.method in ['POST', 'PUT', 'DELETE']:
        request.params = json.loads(request.body)
    openId = request.params.dict()["OPEN_ID"]
    current_vote = forms.model_to_dict(models.DEMO_VOTE_RESULT.objects.filter(open_id=openId).first())
    total_size = models.DEMO_VOTE_RESULT.objects.count()
    vote_demo_detail = models.DEMO_VOTE_DETAIL.objects.all().values()
    vote_demo_detail = list(vote_demo_detail)
    for i in range(len(vote_demo_detail)):
        vote_demo_detail[i]['size'] = models.DEMO_VOTE_RESULT.objects.filter(detail_id=vote_demo_detail[i]['id']).count()

    return JsonResponse({'total_size': total_size, 'vote_demo_detail': vote_demo_detail, 'current_vote': current_vote},json_dumps_params={'ensure_ascii':False},safe=False)
```

Figure 8: 后端请求处理函数

3.4 服务器配置

待续

Note:

- 在开发阶段可暂不设置服务器，但需注意的是在小程序开发工具中需要关闭“开发环境不校验请求域名以及TLS版本”功能，否则即便在localhost中也访问不到后端。