**//启动文件devmain.py[[1]](#footnote-0)**

from flask import Flask

from flask.ext import restful

from dapis.v1 import routedict as api\_v1

from orml import createtable

from conf import httpserver

from conf import httpport

from conf import debugstatus

app = Flask(\_\_name\_\_)

api = restful.Api(app)

app.config['MAX\_CONTENT\_LENGTH'] = 16 \* 1024 \* 1024

for k, v in api\_v1.items():

api.add\_resource(k, v)

if \_\_name\_\_ == '\_\_main\_\_':

print("dbserver started on http://" +

str(httpserver) + ":" + str(httpport))

app.run(host=httpserver, port=int(httpport), debug=debugstatus)

**//全局配置文件conf.py**

t# 用于处理mysql中文的存储

table\_args = {

'mysql\_engine': 'InnoDB',

'mysql\_charset': 'utf8'

}

# 用于测试的服务器和端口，注意这里不能用http://

httpserver = "192.168.1.21"

httpport = "5000" # 5100

# 静态文件的服务器# 用于处理mysql中文的存储

table\_args = {

'mysql\_engine': 'InnoDB',

'mysql\_charset': 'utf8'

}

# 用于测试的服务器和端口，注意这里不能用http://

httpserver = "192.168.1.103"

httpport = "5000" # 5000

# 静态文件的服务器和端口

staticserver = "192.168.1.103"

staticport = "5500" # 5500

# 用于日志配置

formatstr = "%(asctime)s\t%(name)s\t%(pathname)s\t%(lineno)d\t%(message)s\t%(funcName)s\t%(levelname)s"

logdir = "log"

loggername = "reachmedown"

# 清除过时的文件

plansched = "04:00"

# 用于sqlalchemy配置

sqlecho = True

# 测试配置

debugstatus = True

**//交互数据封装格式info.py**

class Info:

def \_\_init\_\_(self, infostatus, infomsg, inforesult=None,infodebug=None):

self.infostatus = infostatus

self.infomsg = infomsg

self.inforesult = inforesult

def tojson(self):

return {'infostatus': self.infostatus,

'infomsg': self.infomsg,

'inforesult': self.inforesult}

**//与数据库连接文件dbbase.py**

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from sqlalchemy.ext.declarative import declarative\_base

from conf import sqlecho

Base = declarative\_base()

engine1 = create\_engine('mysql+mysqlconnector://root:root@0.0.0.0/mysql?charset=utf8', echo=sqlecho)

DBSession1 = sessionmaker(bind=engine1)

session = DBSession1()

try:

session.execute('use mysql')

session.execute('create database CampusPost')

except Exception as a:

print(a)

print('数据库重复建立')

finally:

session.close()

engine = create\_engine('mysql+mysqlconnector://root:root@0.0.0.0/CampusPost?charset=utf8', echo=sqlecho)

DBSession = sessionmaker(bind=engine)

**//创建用户表usermodel.py**

from sqlalchemy import Column, String, Integer

from orml.dbbase import Base

from conf import table\_args

class User(Base):

\_\_table\_args\_\_ = table\_args

\_\_tablename\_\_ = 'user'

user\_id = Column(Integer, primary\_key=True)

user\_name = Column(String(20))

user\_password = Column(String(20))

user\_privilege = Column(Integer)

user\_userlogo = Column(String(50))

**//创建用户认证表usertokenmodel.py**

from sqlalchemy import Column, String, Integer

from orml.dbbase import Base

from conf import table\_args

from sqlalchemy import ForeignKey

class Usertoken(Base):

\_\_table\_args\_\_ = table\_args

\_\_tablename\_\_ = 'usertoken'

usertoken\_str = Column(String(100),primary\_key=True)

usertoken\_userid = Column(Integer, ForeignKey('user.user\_id'))

**//创建指定帖子表toppostmodel.py**

from sqlalchemy import Column, String, Integer

from orml.dbbase import Base

from conf import table\_args

from sqlalchemy import ForeignKey

class Toppost(Base):

\_\_table\_args\_\_ = table\_args

\_\_tablename\_\_ = 'toppost'

top\_id = Column(Integer, primary\_key=True)

top\_postid = Column(Integer, ForeignKey('post.post\_id'))

top\_time = Column(String(100))

top\_categoryid = Column(Integer,ForeignKey('category.category\_id'))

**//创建帖子表postmodel.py**

from sqlalchemy import Column, String, Integer

from orml.dbbase import Base

from conf import table\_args

from sqlalchemy import ForeignKey

class Post(Base):

\_\_table\_args\_\_ = table\_args

\_\_tablename\_\_ = 'post'

post\_id = Column(Integer, primary\_key=True)

post\_categoryid = Column(Integer,ForeignKey('category.category\_id'))

post\_userid = Column(Integer,ForeignKey('user.user\_id'))

post\_title = Column(String(500))

post\_time = Column(String(100))

post\_content = Column(String(2000))

visitcount\_count = Column(Integer)

**//创建分类表cateforymodel.py**

from sqlalchemy import Column, String, Integer

from orml.dbbase import Base

from conf import table\_args

from sqlalchemy import ForeignKey

class Category(Base):

\_\_table\_args\_\_ = table\_args

\_\_tablename\_\_ = 'category'

category\_id = Column(Integer,primary\_key=True)

category\_name = Column(String(20))

**//创建图片表picturemodel.py**

from sqlalchemy import Column, String, Integer

from orml.dbbase import Base

from conf import table\_args

from sqlalchemy import ForeignKey

class Picture(Base):

\_\_table\_args\_\_ = table\_args

\_\_tablename\_\_ = 'picture'

picture\_id = Column(Integer, primary\_key=True)

picture\_name = Column(String(200))

picture\_url = Column(String(200))

picture\_postid = Column(Integer,ForeignKey('post.post\_id'))

**//创建评论回复表commentmodel.py**

from sqlalchemy import Column, String, Integer

from orml.dbbase import Base

from conf import table\_args

from sqlalchemy import ForeignKey

class Comment(Base):

\_\_table\_args\_\_ = table\_args

\_\_tablename\_\_ = 'comment'

comment\_id = Column(Integer, primary\_key=True)

comment\_postid = Column(Integer,ForeignKey('post.post\_id'))

comment\_userid = Column(Integer,ForeignKey('user.user\_id'))

comment\_text = Column(String(2000))

comment\_datetime = Column(String(100))

**//创建所有表的文件createtable.py**#注意这里必须先引入model

from dbmodels.categorymodel import Category

from dbmodels.commentmodel import Comment

from dbmodels.picturemodel import Picture

from dbmodels.postmodel import Post

from dbmodels.toppostmodel import Toppost

from dbmodels.usermodel import User

from dbmodels.usertokenmodel import Usertoken

from orml.dbbase import engine

from orml.dbbase import Base

Base.metadata.create\_all(engine)

import orml.createvalue

**//有关用户的orml文件userorml.py**

from orml.dbbase import DBSession

from dbmodels.usermodel import User

from dbmodels.usertokenmodel import Usertoken

from tools.info import Info

import uuid

class Userorml:

def checkuserexist(self, username):

session = DBSession()

try:

if session.query(User.user\_id).filter(User.user\_name == username).count() == 0:

session.close()

return Info(True, '可以使用的用户名', None).tojson()

else:

session.close()

return Info(False, '该用户名已被占用', None).tojson()

except Exception as a:

print(a)

session.close()

return Info(False, '数据库错误', None).tojson()

def adduser(self,username, userpassword):

session = DBSession()

try:

if len(username) > 16:

return Info(False, '用户名不合法', None).tojson()

if session.query(User).filter\_by(user\_name=username).count() > 0:

session.close()

return Info(False, '用户名已被占用', None).tojson()

else:

session.add(User(user\_name = username,

user\_password = userpassword,

user\_privilege = 0,

user\_userlogo = "default.jpg",

))

session.commit()

session.close()

return Info(True, '注册成功',None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close()

return Info(False, '数据库错误', None).tojson()

def checklogin(self, username, password):

session = DBSession()

try:

if session.query(User).filter\_by(user\_name=username, user\_password=password).count() > 0:

user = session.query(User.user\_id,User.user\_privilege).filter\_by(

user\_name=username).all()

userid = user[0][0]

userprivilege = user[0][1]

usertoken = str(uuid.uuid4())

userdict = {}

userdict['usertoken\_str'] = usertoken

userdict['user\_privilege'] = userprivilege

if session.query(Usertoken).filter\_by(usertoken\_userid=userid).count() > 0:

session.query(Usertoken).filter\_by(usertoken\_userid=userid).delete()

session.add(Usertoken(usertoken\_str=usertoken, usertoken\_userid=userid))

session.commit()

session.close()

else:

session.add(Usertoken(usertoken\_str=usertoken,usertoken\_userid=userid))

session.commit()

session.close()

return Info(True, '登录成功', userdict).tojson()

else:

session.close()

return Info(False, '用户名或密码错误', None).tojson()

except Exception as a:

print(a)

session.close()

return Info(False, '数据库错误', None).tojson()

def tokendelete(self,usertoken):

session = DBSession()

try:

session.query(Usertoken).filter\_by(usertoken\_str=usertoken).delete()

session.commit()

session.close()

return Info(True,'成功删除usertoken',None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close()

return Info(False, '数据库错误', None).tojson()

**//有关用户的service文件userservice.py**

from flask.ext import restful

from flask import request

from flask import jsonify

from flask import Response

from conf import httpserver, httpport

from tools.crossdomain import allow\_cross\_domain

from orml.userorml import Userorml

class CheckUserExist(restful.Resource):

@allow\_cross\_domain

def get(self): #请求方式为GET

username = request.args.get("user\_name") #"username"为测试传过来的参数

if not all([username]):

return jsonify(Info(False,'参数不足', None ).tojson())

uo = Userorml() #类实例化的对象

Infoa = uo.checkuserexist(username)

return jsonify(Infoa)

class AddUser(restful.Resource):

@allow\_cross\_domain

def post(self):#请求方式为POST

username = request.form["user\_name"]

userpassword = request.form["user\_password"]

if not all([username,userpassword]):

return jsonify(Info(False,'参数不足', None ).tojson())

uo = Userorml()

Infoa = uo.adduser(username,userpassword)

return jsonify(Infoa)

class CheckLogin(restful.Resource):

@allow\_cross\_domain

def post(self):

username = request.form["user\_name"]

password = request.form["user\_password"]

if not all([username,password]):

return jsonify(Info(False,'参数不足', None ).tojson())

uo = Userorml()

Infoa = uo.checklogin(username, password)

return jsonify(Infoa)

class TokenDelete(restful.Resource):

@allow\_cross\_domain

def post(self):

usertoken = request.form["usertoken\_str"]

uo = Userorml()

Infoa = uo.tokendelete(usertoken)

return jsonify(Infoa)

**//有关帖子的orml文件postorml.py**

from orml.dbbase import DBSession

from dbmodels.usermodel import User

from dbmodels.postmodel import Post

from dbmodels.usertokenmodel import Usertoken

from dbmodels.categorymodel import Category

from dbmodels.toppostmodel import Toppost

from dbmodels.picturemodel import Picture

from dbmodels.commentmodel import Comment

from tools.info import Info

from sqlalchemy import desc

from conf import staticserver, staticport

import time

from tools.timetools import Timetools

import os

import demjson

class Postorml:

def postlist(self,startposi,pagesize,categoryname):

session = DBSession()

try:

category\_id = session.query(Category.category\_id).filter\_by(category\_name=categoryname).all()

categoryid = category\_id[0][0]

listdict = []

if int(startposi) > 0 :

postlist = session.query(Post.post\_title,Post.post\_time,Post.post\_content,Post.post\_userid,Post.post\_id).filter\_by(post\_categoryid=categoryid).order\_by(desc(Post.post\_time)).offset(startposi).limit(pagesize).all()

if len(postlist) == 0 :

session.close()

return Info(True,0,"已经没有帖子了").tojson()

for lista in postlist:

dicta = {}

dicta["post\_id"] = lista[4]

dicta["post\_title"] = lista[0]

dicta["post\_time"] = lista[1]

dicta["post\_content"] = lista[2]

user\_name = session.query(User.user\_name,User.user\_userlogo).filter\_by(user\_id=lista[3]).all()

username = user\_name[0][0]

dicta["user\_name"] = username

userlogo = user\_name[0][1]

userlogo = 'http://' + str(staticserver) + ":" + str(staticport) + '/logo/' + userlogo

dicta["post\_userhead"] = userlogo

if session.query(Toppost).filter\_by(top\_postid=lista[4]).count() == 0:

dicta["post\_topstatu"] = "0"

else:

dicta["post\_topstatu"] = "1"

listdict.append(dicta)

session.close()

return Info(True,0,listdict).tojson()

toplistid = session.query(Toppost.top\_postid).filter\_by(top\_categoryid=categoryid).order\_by(Toppost.top\_time).offset(startposi).limit(pagesize).all()

toplist\_id = []

for postid in toplistid:

toplist\_id.append(postid)

dict1 = {}

postdetail = session.query(Post.post\_title,Post.post\_time,Post.post\_content,Post.post\_userid,Post.post\_id).filter\_by(post\_id=postid[0]).all()

dict1["post\_id"] = postdetail[0][4]

dict1["post\_title"] = postdetail[0][0]

dict1["post\_time"] = postdetail[0][1]

dict1["post\_content"] = postdetail[0][2]

user\_name = session.query(User.user\_name,User.user\_userlogo).filter\_by(user\_id=postdetail[0][3]).all()

username = user\_name[0][0]

dict1["user\_name"] = username

userlogo = user\_name[0][1]

if not userlogo:

suserlogo = "default.jpg"

userlogo = 'http://' + str(staticserver) + ":" + str(staticport) + '/logo/' + userlogo

dict1["post\_userhead"] = userlogo

if session.query(Toppost).filter\_by(top\_postid=postdetail[0][4]).count() == 0:

dict1["post\_topstatu"] = "0"

else:

dict1["post\_topstatu"] = "1"

listdict.append(dict1)

listdictlen = len(listdict)

if len(listdict) != 0:

pagesizeafter = int(pagesize) - int(listdictlen)

else:

pagesizeafter = int(pagesize)

postlist = session.query(Post.post\_title,Post.post\_time,Post.post\_content,Post.post\_userid,Post.post\_id).filter\_by(post\_categoryid=categoryid).order\_by(desc(Post.post\_time)).offset(str(startposi)).limit(str(pagesizeafter)).all()

if len(postlist) == 0 :

session.close()

print(111111111111111111111111)

return Info(True,listdictlen,listdict).tojson()

for lista in postlist:

dicta = {}

dicta["post\_id"] = lista[4]

dicta["post\_title"] = lista[0]

dicta["post\_time"] = lista[1]

dicta["post\_content"] = lista[2]

user\_name = session.query(User.user\_name,User.user\_userlogo).filter\_by(user\_id=lista[3]).all()

username = user\_name[0][0]

dicta["user\_name"] = username

userlogo = user\_name[0][1]

userlogo = 'http://' + str(staticserver) + ":" + str(staticport) + '/logo/' + userlogo

dicta["post\_userhead"] = userlogo

if session.query(Toppost).filter\_by(top\_postid=lista[4]).count() == 0:

dicta["post\_topstatu"] = "0"

else:

dicta["post\_topstatu"] = "1"

listdict.append(dicta)

listdictafter = []

lengh = int(pagesize) - int(startposi)

for a in listdict:

if a not in listdictafter:

listdictafter.append(a)

listdict = listdictafter

if len(listdict) < lengh:

session.close()

return Info(True,listdictlen,listdict).tojson()

for a in listdict:

if a not in listdictafter:

listdictafter.append(a)

allpagesize = session.query(Post).filter\_by(post\_categoryid=categoryid).count()

while len(listdictafter) < int(pagesize) and len(listdictafter) < allpagesize:

b = int(startposi) + len(listdictafter)

c = int(pagesize) - len(listdictafter)

postlistb = session.query(Post.post\_title,Post.post\_time,Post.post\_content,Post.post\_userid,Post.post\_id).filter\_by(post\_categoryid=categoryid).order\_by(desc(Post.post\_time)).offset(str(b)).limit(str(c)).all()

listdict = listdictafter

for lista in postlistb:

dicta = {}

dicta["post\_id"] = lista[4]

dicta["post\_title"] = lista[0]

dicta["post\_time"] = lista[1]

dicta["post\_content"] = lista[2]

user\_name = session.query(User.user\_name,User.user\_userlogo).filter\_by(user\_id=lista[3]).all()

username = user\_name[0][0]

dicta["user\_name"] = username

userlogo = user\_name[0][1]

userlogo = 'http://' + str(staticserver) + ":" + str(staticport) + '/logo/' + userlogo

dicta["post\_userhead"] = userlogo

if session.query(Toppost).filter\_by(top\_postid=lista[4]).count() == 0:

dicta["post\_topstatu"] = "0"

else:

dicta["post\_topstatu"] = "1"

listdict.append(dicta)

listdictafter = []

for a in listdict:

if a not in listdictafter:

listdictafter.append(a)

session.close()

return Info(True,listdictlen,listdictafter).tojson()

except Exception as a:

print(a)

session.close()

return Info(False, '数据库错误', None).tojson()

def postappend(self,usertoken,posttitle,postcontent,categoryname,postpics):

session = DBSession()

try:

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken).count() == 0:

return Info(False, '无效的用户，登陆断开连接', None).tojson()

user\_id = session.query(Usertoken.usertoken\_userid).filter\_by(usertoken\_str=usertoken).all()

userid = user\_id[0][0]

category\_id = session.query(Category.category\_id).filter\_by(category\_name=categoryname).all()

categoryid = category\_id[0][0]

posttime = time.strftime("%Y-%m-%d %H:%M:%S",time.localtime())

session.add(Post(post\_categoryid=categoryid,

post\_userid=userid,

post\_title=posttitle,

post\_time=posttime,

post\_content=postcontent))

post\_id = session.query(Post.post\_id).filter\_by(post\_userid=userid,post\_title=posttitle,post\_time=posttime).all()

postid = post\_id[0][0]

postpics = demjson.decode(postpics)

for pics in postpics:

picslist = pics.split('/')

picsname = picslist[-1]

session.add(Picture(picture\_name=picsname,

picture\_postid=postid,

picture\_url=picsname))

session.commit()

session.close()

return Info(True,'发布成功',None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close

return Info(False, '数据库错误', None).tojson()

def postdelete(self,usertoken,postid):

session = DBSession()

try:

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken).count() == 0:

return Info(False, '无效的用户，登陆断开连接', None).tojson()

user\_id = session.query(Usertoken.usertoken\_userid).filter\_by(usertoken\_str=usertoken).all()

userid1 = user\_id[0][0]

if session.query(Post).filter\_by(post\_id=postid).count() == 0:

return Info(False, '没有该帖子', None).tojson()

user\_id2 = session.query(Post.post\_userid).filter\_by(post\_id=postid).all()

userid2 = user\_id2[0][0]

user\_privilege = session.query(User.user\_privilege).filter\_by(user\_id=userid1).all()

userprivilege = user\_privilege[0][0]

if userid1 == userid2 or userprivilege == 1:

if session.query(Picture).filter\_by(picture\_postid=postid).count() > 0:

session.query(Picture).filter\_by(picture\_postid=postid).delete()

if session.query(Comment).filter\_by(comment\_postid=postid).count() > 0 :

session.query(Comment).filter\_by(comment\_postid=postid).delete()

if session.query(Toppost).filter\_by(top\_postid=postid).count() > 0:

session.query(Toppost).filter\_by(top\_postid=postid).delete()

session.query(Post).filter\_by(post\_id=postid).delete()

session.commit()

session.close()

return Info(True,'删除帖子成功',None).tojson()

else:

return Info(False, '没有操作的权限', None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close

return Info(False, '数据库错误', None).tojson()

def mypostlist(self,startposi,pagesize,usertoken,categoryname):

session = DBSession()

try:

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken).count() == 0:

return Info(False, '无效的用户，登陆断开连接', None).tojson()

user\_id = session.query(Usertoken.usertoken\_userid).filter\_by(usertoken\_str=usertoken).all()

userid = user\_id[0][0]

user\_logo = session.query(User.user\_userlogo).filter\_by(user\_id = userid).all()

userlogo = user\_logo[0][0]

if not userlogo:

userlogo = "default.jpg"

userlogo = 'http://' + str(staticserver) + ":" + str(staticport) + '/logo/' + userlogo

category\_id = session.query(Category.category\_id).filter\_by(category\_name=categoryname).all()

categoryid = category\_id[0][0]

postlist = session.query(Post.post\_title,Post.post\_time,Post.post\_content,Post.post\_categoryid,Post.post\_id).filter\_by(post\_userid=userid,post\_categoryid=categoryid).order\_by(desc(Post.post\_time)).offset(startposi).limit(pagesize).all()

listpost = []

for postdetail in postlist:

postdict = {}

postdict["post\_id"] = postdetail[4]

postdict["post\_title"] = postdetail[0]

postdict["post\_content"] = postdetail[2]

user\_name = session.query(User.user\_name).filter\_by(user\_id=userid).all()

username = user\_name[0][0]

postdict["user\_name"] = username

tt = Timetools()

postdict["post\_time"] = tt.timetonow(postdetail[1])

category\_name = session.query(Category.category\_name).filter\_by(category\_id=postdetail[3]).all()

categoryname = category\_name[0][0]

postdict["category\_name"] = categoryname

postdict["post\_userhead"] = userlogo

picturelist = []

picture = session.query(Picture.picture\_url).filter\_by(picture\_postid=postdetail[4]).all()

for pic in picture:

pictureurl = 'http://' + str(staticserver) + ":" + str(staticport) + '/pics/' + pic[0]

picturelist.append(pictureurl)

postdict["picture\_url"] = picturelist

listpost.append(postdict)

session.close()

return Info(True,'返回我发布的帖子列表成功',listpost).tojson()

except Exception as a:

print(a)

session.close()

return Info(False, '数据库错误', None).tojson()

def posttop(self,usertoken,postid,categoryname):

session = DBSession()

try:

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken).count() == 0:

return Info(False, '无效的用户，登陆断开连接', None).tojson()

user\_id = session.query(Usertoken.usertoken\_userid).filter\_by(usertoken\_str=usertoken).all()

userid =user\_id[0][0]

user\_privilege = session.query(User.user\_privilege).filter\_by(user\_id=userid).all()

userprivilege = user\_privilege[0][0]

if userprivilege == 0:

return Info(False, '用户没有操作权限', None).tojson()

category\_id = session.query(Category.category\_id).filter\_by(category\_name=categoryname).all()

categoryid = category\_id[0][0]

topposttime = time.strftime("%Y-%m-%d %H:%M:%S",time.localtime())

if session.query(Toppost).filter\_by(top\_postid=postid).count() > 0:

return Info(False, '该帖子已经置顶，不能重复置顶帖子', None).tojson()

session.add(Toppost(top\_postid=postid,

top\_time=topposttime,

top\_categoryid=categoryid))

session.commit()

session.close()

return Info(True,'置顶成功',None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close()

return Info(False, '数据库错误', None).tojson()

def posttopcancel(self,usertoken,postid):

session = DBSession()

try:

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken).count() == 0:

return Info(False, '无效的用户，登陆断开连接', None).tojson()

user\_id = session.query(Usertoken.usertoken\_userid).filter\_by(usertoken\_str=usertoken).all()

userid =user\_id[0][0]

user\_privilege = session.query(User.user\_privilege).filter\_by(user\_id=userid).all()

userprivilege = user\_privilege[0][0]

if userprivilege == 0:

return Info(False, '用户没有操作权限', None).tojson()

session.query(Toppost).filter\_by(top\_postid=postid).delete()

session.commit()

session.close()

return Info(True,'删除成功',None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close()

return Info(False, '数据库错误', None).tojson()

def postiddetail(self,postid):

session = DBSession()

try:

postlist = session.query(Post.post\_userid,Post.post\_title,Post.post\_content,Post.post\_time).filter\_by(post\_id=postid).all();

postdetail = {}

postdetail["post\_id"] = postid

postdetail["post\_userid"] = postlist[0][0]

userdetail = session.query(User.user\_name,User.user\_userlogo).filter\_by(user\_id=postlist[0][0]).all()

postdetail["post\_username"] = userdetail[0][0]

userlogo = userdetail[0][1]

if not userlogo:

userlogo = "default.jpg"

userlogo = 'http://' + str(staticserver) + ":" + str(staticport) + '/logo/' + userlogo

postdetail["post\_userhead"] = userlogo

postdetail["post\_title"] = postlist[0][1]

postdetail["post\_content"] = postlist[0][2]

tt = Timetools()

postdetail["post\_time"] = tt.timetonow(postlist[0][3])

picturelist = []

picture = session.query(Picture.picture\_url).filter\_by(picture\_postid=postid).all()

for pic in picture:

pictureurl = 'http://' + str(staticserver) + ":" + str(staticport) + '/pics/' + pic[0]

picturelist.append(pictureurl)

postdetail["post\_pics"] = picturelist

print(postdetail["post\_pics"])

comment = session.query(Comment.comment\_id,Comment.comment\_userid,Comment.comment\_text,Comment.comment\_datetime).filter\_by(comment\_postid=postid).order\_by(desc(Comment.comment\_datetime)).all()

commentlist = []

for y in comment:

commentdict = {}

commentdict['comment\_id'] = y[0]

commentdict['comment\_userid'] = y[1]

username1 = session.query(User.user\_name).filter\_by(user\_id=y[1]).all()

username1 = username1[0][0]

commentdict['comment\_username'] = username1

userlogo = session.query(User.user\_userlogo).filter\_by(user\_id=y[1]).all()

userlogo = userlogo[0][0]

if not userlogo:

userlogo = "default.jpg"

userlogo = 'http://' + staticserver + ':' + staticport +'/logo/'+userlogo

commentdict['comment\_userhead'] = userlogo

commentdict['comment\_text'] = y[2]

commentdict['comment\_datetime'] = y[3]

commentlist.append(commentdict)

commentlist.sort(key = lambda x : x['comment\_datetime'],reverse=False)

for x in commentlist:

x['comment\_datetime'] = tt.timetonow(x['comment\_datetime'])

postdetail['post\_comments'] = commentlist

return Info(True,'返回成功',postdetail).tojson()

session.commit()

session.close()

except Exception as a:

print(a)

session.close()

return Info(False, '数据库错误', None).tojson()

def postcomment(self,usertoken\_str,post\_id,comment\_str):

session = DBSession()

try:

if not comment\_str:

session.close()

return Info(False, '评论不能为空', None).tojson()

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken\_str).count() == 0:

session.close()

return Info(False, '用户登录已过期，请重新登录', None).tojson()

if session.query(Post).filter\_by(post\_id=post\_id).count() == 0:

session.close()

return Info(False, '该货单不存在', None).tojson()

userid = session.query(Usertoken.usertoken\_userid).filter\_by(usertoken\_str=usertoken\_str).all()

userid = userid[0][0]

nowtime = time.strftime('%Y-%m-%d %H:%M:%S',time.localtime())

session.add(Comment(comment\_postid=post\_id,

comment\_userid=userid,

comment\_text=comment\_str,

comment\_datetime=nowtime))

session.commit()

session.close()

return Info(True, '评论成功', None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close()

return Info(False, '数据库错误', None).tojson()

def postreply(self,usertoken\_str,comment\_id,reply\_str):

session = DBSession()

try:

if not reply\_str:

session.close()

return Info(False, '回复不能为空', None).tojson()

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken\_str).count() == 0:

session.close()

return Info(False, '用户登录已过期，请重新登录', None).tojson()

user\_id = session.query(Usertoken.usertoken\_userid).filter\_by(usertoken\_str=usertoken\_str).all()

userid = user\_id[0][0]

if session.query(Comment).filter\_by(comment\_id=comment\_id).count() == 0:

session.close()

return Info(False, '该评论不存在', None).tojson()

post\_id = session.query(Comment.comment\_postid).filter\_by(comment\_id=comment\_id).all()

postid = post\_id[0][0]

nowtime = time.strftime('%Y-%m-%d %H:%M:%S',time.localtime())

session.add(Comment(comment\_postid=postid,

comment\_userid=userid,

comment\_text=reply\_str,

comment\_datetime=nowtime))

session.commit()

session.close()

return Info(True, '回复成功', None).tojson()

except Exception as a:

print(a)

session.rollback()

session.close()

return Info(False, '数据库错误', None).tojson()

def pictureremove(self,usertoken,pic\_address):

session = DBSession()

try:

if session.query(Usertoken).filter\_by(usertoken\_str=usertoken).count() == 0:

session.close()

return Info(False, '用户登录已过期，请重新登录', None).tojson()

pic\_address = pic\_address.split('/')

pic\_address = pic\_address[-1]

if not pic\_address:

return Info(False, '输入的地址格式错误，"/"分割后最后一个元素应该为图片名称', None).tojson()

else:

if os.path.exists('/home/fris/公共的/CampusPost/picture/pics/'+pic\_address):

os.remove('/home/fris/公共的/CampusPost/picture/pics/'+pic\_address)

else:

session.close()

return Info(True, '你要删除的图片不存在', None).tojson()

session.close()

return Info(True, '图片删除成功', None).tojson()

except Exception as a:

print(a)

session.close()

return Info(False, '数据库错误', None).tojson()

**//有关帖子的service文件postservice.py**

from flask.ext import restful

from flask import request

from flask import jsonify

from flask import Response

from conf import httpserver, httpport

from tools.crossdomain import allow\_cross\_domain

from orml.postorml import Postorml

from tools.info import Info

class Postlist(restful.Resource):

@allow\_cross\_domain

def get(self): #请求方式为GET

categoryname = request.args.get("category\_name")

startposi = request.args.get("startposi")

pagesize = request.args.get("pagesize")

if not all([startposi,pagesize,categoryname]):

return jsonify(Info(False,'参数不足', None ).tojson())

po = Postorml() #类实例化的对象

Infoa = po.postlist(startposi,pagesize,categoryname)

return jsonify(Infoa)

class Postappend(restful.Resource):

@allow\_cross\_domain

def post(self):#请求方式为POST

usertoken = request.form["usertoken\_str"]

post\_title = request.form["post\_title"]

post\_content = request.form["post\_content"]

category\_name = request.form["category\_name"]

postpics = request.form["post\_pics"]

po = Postorml()

Infoa = po.postappend(usertoken,post\_title,post\_content,category\_name,postpics)

return jsonify(Infoa)

class Postdelete(restful.Resource):

@allow\_cross\_domain

def post(self):#请求方式为POST

usertoken = request.form["usertoken\_str"]

postid = request.form["post\_id"]

po = Postorml()

Infoa = po.postdelete(usertoken,postid)

return jsonify(Infoa)

class Mypostlist(restful.Resource):

@allow\_cross\_domain

def get(self): #请求方式为GET

usertoken = request.args.get("usertoken\_str")

startposi = request.args.get("startposi")

pagesize = request.args.get("pagesize")

categoryname = request.args.get("category\_name")

if not all([startposi,pagesize,usertoken]):

return jsonify(Info(False,'参数不足', None ).tojson())

po = Postorml() #类实例化的对象

Infoa = po.mypostlist(startposi,pagesize,usertoken,categoryname)

return jsonify(Infoa)

class Posttop(restful.Resource):

@allow\_cross\_domain

def post(self):#请求方式为POST

usertoken = request.form["usertoken\_str"]

postid = request.form["post\_id"]

categoryname = request.form["category\_name"]

po = Postorml()

Infoa = po.posttop(usertoken,postid,categoryname)

return jsonify(Infoa)

class Posttopcancel(restful.Resource):

@allow\_cross\_domain

def post(self):#请求方式为POST

usertoken = request.form["usertoken\_str"]

postid = request.form["post\_id"]

po = Postorml()

Infoa = po.posttopcancel(usertoken,postid)

return jsonify(Infoa)

class Postiddetail(restful.Resource):

'''

请求方式: GET

请求参数:

usertoken\_str:usertoken

post\_id:1

返回值:

True,返回成功,{"post\_id":1,

"post\_userid":1,

"post\_username":"张三",

"post\_userhead":"http://192.168.1.100:8080/logo/1.png",

"post\_title":"校内信息头条",

"post\_content":"校内信息头条的内容",

"post\_time":"2016/10/10 17:30",

"post\_pics":[http://192.168.1.100:8080/pic/1.png, http://192.168.1.100:8080/pic/2.png],

"post\_comments":[{"comment\_id":1,

"comment\_userid":2,

"comment\_username":"李四",

"comment\_userhead":"http://192.168.1.100:8080/logo/1.png",

"comment\_text":"评论内容",

"comment\_datetime":"2016/10/10 17:30",

"comment\_parentid":0,

"comment\_parentname":""},

{"comment\_id":2,

"comment\_userid":2,

"comment\_username":"李四",

"comment\_userhead":"http://192.168.1.100:8080/logo/1.png",

"comment\_text":"评论内容",

"comment\_datetime":"2016/10/10 17:30",

"comment\_parentid":1,

"comment\_parentname":"李四"},

{"comment\_id":3,

"comment\_userid":2,

"comment\_username":"李四",

"comment\_userhead":"http://192.168.1.100:8080/logo/1.png",

"comment\_text":"评论内容",

"comment\_datetime":"2016/10/10 17:30",

"comment\_parentid":2,"comment\_parentname":"王五"}];

False,数据库错误,None

'''

@allow\_cross\_domain

def get(self): #请求方式为GET

postid = request.args.get("post\_id")

po = Postorml() #类实例化的对象

Infoa = po.postiddetail(postid)

return jsonify(Infoa)

class Postcomment(restful.Resource):

@allow\_cross\_domain

def post(self):#请求方式为POST

usertoken = request.form["usertoken\_str"]

postid = request.form["post\_id"]

commentstr = request.form["comment\_str"]

po = Postorml()

Infoa = po.postcomment(usertoken,postid,commentstr)

return jsonify(Infoa)

class Postreply(restful.Resource):

@allow\_cross\_domain

def post(self):#请求方式为POST

usertoken = request.form["usertoken\_str"]

comment\_id = request.form["comment\_id"]

reply\_str = request.form["reply\_str"]

po = Postorml()

Infoa = po.postreply(usertoken,comment\_id,reply\_str)

return jsonify(Infoa)

class Pictureremove(restful.Resource):

@allow\_cross\_domain

def get(self):

usertoken = request.args.get("usertoken\_str")

pic\_address = request.args.get("pic\_address")

if not all([usertoken,pic\_address]):

return jsonify(Info(False,"参数不足",None).tojson())

mo = Postorml()

Infoa = mo.pictureremove(usertoken,pic\_address)

result = jsonify(Infoa)

return result

**//有关文件传输的service文件fileservice.py**

from flask.ext import restful

from flask import request

from flask import jsonify

from flask import Response

from tools.info import Info

from tools.crossdomain import allow\_cross\_domain

import os

import json

import time

from tools.generatecode import GenerateCode

from conf import staticserver, staticport

from tools.cutimage import ImageCut

from PIL import Image

class Webpicupload(restful.Resource):

'''

路径: /v1/manifest/web/pics/

请求方式: OPTION POST

请求参数:

files[]: image

返回值:

True, http://192.168.1.100:8030/pics/1.png, None

False, 图片过小（不小于480px \* 360px）, None

False, 图片过大（不大于2400px \* 1800px）, None

'''

@allow\_cross\_domain

def options(self):

return jsonify(Info(True, None, None).tojson())

@allow\_cross\_domain

def post(self):

file = request.files["files[]"]

img = Image.open(file)

width = list(img.size)[1]

length = list(img.size)[0]

if width < 480 or length < 640 :

return jsonify(Info(False,"图片过小（不小于480px \* 360px）",None).tojson())

if width > 2400 or length > 1800 :

return jsonify(Info(False,"图片过大（不大于2400px \* 1800px）",None).tojson())

nowtime = time.strftime('%Y-%m-%d',time.localtime())

if os.path.exists('/home/fris/公共的/CampusPost/picture/temp/'+nowtime):

newfaddr = "/home/fris/公共的/CampusPost/picture/temp/" + nowtime + '/' + file.filename

img.save(newfaddr)

else:

os.mkdir('/home/fris/公共的/CampusPost/picture/temp/'+nowtime)

newfaddr = "/home/fris/公共的/CampusPost/picture/temp/" + nowtime + '/' + file.filename

img.save(newfaddr)

return jsonify(Info(True, "http://" + str(staticserver) + ":" + str(staticport) + "/temp/" + nowtime +'/' + file.filename, None).tojson())

class PostImageCutService(restful.Resource):

'''

路径: /v1/file/imagecut/

请求方式: GET

请求参数:

filename: filename

x1: x1

y1: y1

x2: x2

y2: y2

cw: cw

ch: ch

返回值:

True, http://192.168.1.100:8030/logo/1.png, None

False, 数据库错误, None

'''

@allow\_cross\_domain

def get(self):

x1 = request.args.get("x1")

y1 = request.args.get("y1")

x2 = request.args.get("x2")

y2 = request.args.get("y2")

cw = request.args.get("cw")

ch = request.args.get("ch")

nowtime = time.strftime('%Y-%m-%d',time.localtime())

filename = "/home/fris/公共的/CampusPost/picture/temp/" + nowtime + '/' + request.args.get("filename").split("/")[-1]

ci = ImageCut()

faddr = "http://" + str(staticserver) + ":" + str(

staticport) + "/pics/" + ci.cutpics(filename, x1, x2, y1, y2, cw, ch).split("/")[-1]

print(faddr)

return jsonify(Info(True, faddr, None).tojson())

**//所有API文件v1.py**

from dwsl import userservice

from dwsl import postservice

from dwsl import fileservice

routedict = {

userservice.CheckUserExist: "/v1/user/signup/", #检查用户名是否被占用

userservice.AddUser: "/v1/user/signup/", #用户注册

userservice.CheckLogin: "/v1/user/login/", #用户登录

userservice.TokenDelete: "/v1/token/delete/", #删除usertoken

postservice.Postlist: "/v1/post/list/", #返回帖子列表

postservice.Postappend:"/v1/post/append/", #发布帖子

postservice.Postdelete:"/v1/post/delete/", #删除帖子、

postservice.Mypostlist: "/v1/post/mypostlist/", #返回我发布的帖子

postservice.Posttop: "/v1/post/top/", #帖子置顶

postservice.Posttopcancel: "/v1/post/top/cancel/", #取消置顶帖子

postservice.Postiddetail: "/v1/post/id/detail/",#根据帖子编号获取帖子详情

postservice.Postcomment: "/v1/post/comment/", #评论帖子

postservice.Postreply: "/v1/post/reply/", #回复帖子

fileservice.Webpicupload: "/v1/post/web/pics/", # 网站端上传图片

fileservice.PostImageCutService: "/v1/file/post/imagecut/", #帖子图片截取

postservice.Pictureremove: "/v1/picture/remove/", # 根据图片地址删除图片

}

# 文件下面的某个类与寻找路径一一对应，这是一个字典。

**//创建时间计算辅助文件timetools.py**

import time

import datetime

class Timetools:

'''

日期转换工具

'''

def \_\_init\_\_(self):

self.timeformatter = '%Y-%m-%d %H:%M:%S' # 时间格式字符串

self.dateformatter = '%Y年%m月%d日' # 日期格式字符串

self.maxdatespan = 50 # 大于100天差距，显示当时的日期

def timetostr(self, ttime):

'''

把一个time类的对象转成字符串

'''

return time.strftime(self.timeformatter, ttime)

def strtotime(self, tstr):

'''

把一个日期字符串转成time类的对象

'''

return time.strptime(tstr, self.timeformatter)

def timetodatetime(self, ti):

'''

把一个time类的对象转成datetime类的对象

'''

return datetime.datetime(

ti.tm\_year, ti.tm\_mon, ti.tm\_mday, ti.tm\_hour, ti.tm\_min, ti.tm\_sec)

def timestrtodatestr(self, tistr):

'''

用日期格式字符串格式化一个time类对象的字符串形式

'''

ts = self.strtotime(tistr)

return time.strftime(self.dateformatter, ts)

def sectoother(self, secs):

'''

一天之内，秒转成小时和分钟

'''

if secs / 3600 > 1:

return str(int(secs / 3600)) + "小时前"

elif secs / 60 > 1:

return str(int(secs / 60)) + "分钟前"

else:

return str(int(secs)) + "秒前"

def timetonow(self, tstr):

'''

计算某个日期字符串距离当前日期的时间差，返回可读格式

'''

now = time.localtime()

last = self.strtotime(tstr)

dnow = self.timetodatetime(now)

dlast = self.timetodatetime(last)

span = dnow - dlast

if span.days > self.maxdatespan:

return self.timestrtodatestr(tstr)

elif span.days > 0:

return str(span.days) + "天前"

else:

return self.sectoother(int(span.seconds))

if \_\_name\_\_ == "\_\_main\_\_":

th = Timetools()

tstr = th.timetostr(time.localtime())

print(tstr)

tdate = th.strtotime(tstr)

print(tdate)

tstrlist = ["2016-01-08 19:50:00",

"2016-05-05 19:50:00",

"2016-05-09 10:50:00",

"2016-05-10 10:50:00",

"2016-05-10 19:20:00",

"2016-05-10 20:00:00",

"2016-05-31 22:36:35"

]

tstrlist = [th.timetonow(x) for x in tstrlist]

print(tstrlist)

**//创建图片剪切文件cutimage**

from PIL import Image

import uuid

import os

import time

class ImageCut:

def cutpics(self, filename, x1, x2, y1, y2, cw, ch):

newfname = str(uuid.uuid1()) + ".jpg"

img = Image.open(filename)

img = img.resize((360, 480), Image.ANTIALIAS)

x1 = int(x1)

x2 = int(x2)

y1 = int(y1)

y2 = int(y2)

cw = int(cw)

ch = int(ch)

region = (x1, y1, x2, y2)

cropimg = img.crop(region)

newfaddr = "/home/fris/公共的/CampusPost/picture/pics/" + newfname

cropimg.save(newfaddr)

os.remove(filename)

return newfaddr

**//创建http协议成功连接的装饰器crossdomain.py**

from functools import wraps

from flask import make\_response

def allow\_cross\_domain(fun):

@wraps(fun)

def wrapper\_fun(\*args, \*\*kwargs):

rst = make\_response(fun(\*args, \*\*kwargs))

rst.headers['Access-Control-Allow-Origin'] = '\*'

rst.headers['Access-Control-Allow-Methods'] = 'PUT,GET,POST,DELETE'

allow\_headers = "Referer,Accept,Origin,User-Agent"

rst.headers['Access-Control-Allow-Headers'] = allow\_headers

return rst

return wrapper\_fun

1. [↑](#footnote-ref-0)