# 3.系统实现

要实现一个学校考勤系统，并通过配置文件对系统运行进行设置，首先需要定义配置文件的格式，然后通过解析这个配置文件来获取系统设置。

## 3.1 配置文件示例

### 3.1.1 API端口配置及分析

|  |
| --- |
| from [flask](https://github.com/pallets/flask) import Flask  from flask\_cors import CORS  from [routes.student\_routes](https://gitee.com/not-much/software-engineering-homework/blob/master/routes/student_routes.py) import student\_routes  from [routes.teacher\_routes](https://gitee.com/not-much/software-engineering-homework/blob/master/routes/teacher_routes.py) import teacher\_routes  app = Flask(\_\_name\_\_)  CORS(app)  # 注册蓝图  app.register\_blueprint(student\_routes)  app.register\_blueprint(teacher\_routes)  if \_\_name\_\_ == '\_\_main\_\_':  print("Flask app is running on http://0.0.0.0:5000")  app.run(host='0.0.0.0', port=5000) |

**（1）导入必要的模块和蓝图：**

Flask：Flask框架的核心类，用于创建应用实例。

CORS：用于处理跨域资源共享（CORS）的扩展。

student\_routes 和 teacher\_routes：两个蓝图（Blueprint），分别包含了学生和教师相关的路由。

**（2）创建Flask应用实例：**

app = Flask(\_\_name\_\_)：创建一个Flask应用实例。

**（3）启用CORS支持：**

CORS(app)：为应用启用CORS支持，允许跨域请求。

**（4）注册蓝图：**

app.register\_blueprint(student\_routes)：将学生相关的路由蓝图注册到应用中。

app.register\_blueprint(teacher\_routes)：将教师相关的路由蓝图注册到应用中。

**（5）运行应用：**

if \_\_name\_\_ == '\_\_main\_\_':：确保只有在直接运行此脚本时才会启动应用。

print("Flask app is running on http://0.0.0.0:5000")：打印应用运行的URL。

app.run(host='0.0.0.0', port=5000)：启动Flask应用，监听所有网络接口的5000端口。

### 3.1.2数据库配置及代码解析

|  |
| --- |
| import [pandas](https://pypi.org/project/pandas) as pd  from [sqlalchemy](https://pypi.org/project/SQLAlchemy) import create\_engine  # 读取文件的名称  file\_name\_1 = '学生考勤系统数据库用例表.xlsx'  file\_name\_2 = ''  # 将要存入数据库中的表名称  table\_name\_1 = ''  table\_name\_2 = ''  table\_name\_3 = ''  file\_path = file\_name\_1  df = pd.read\_excel(file\_path)  print(df)  # 连接到远程数据库  db\_user = 'ubuntu'  db\_password = 'i3^N9g&uz'  db\_host = '服务器地址'  db\_port = 端口  db\_name = 'db'  # 连接引擎  engine = create\_engine(f'mysql+pymysql://{db\_user}:{db\_password}@{db\_host}:{db\_port}/{db\_name}')  print(engine)  # 将数据存入数据库  df.to\_sql(table\_name\_1, engine, index=True, if\_exists='replace')  print(f'Data has been successfully loaded into the {table\_name\_1} table in the remote database.') |

**（1）定义文件名和表名：**

file\_name\_1：Excel文件的名称。

table\_name\_1、table\_name\_2、table\_name\_3：将要存入数据库中的表名称。这些变量目前是空的，需要填充具体的表名。

**（2）读取Excel文件：**

file\_path = file\_name\_1：设置文件路径。

df = pd.read\_excel(file\_path)：使用Pandas读取Excel文件并将其加载到一个DataFrame中。

print(df)：打印DataFrame，查看读取的数据。

**（3）连接到远程数据库：**

db\_user、db\_password、db\_host、db\_port、db\_name：数据库连接所需的参数。

engine = create\_engine(f'mysql+pymysql://{db\_user}:{db\_password}@{db\_host}:{db\_port}/{db\_name}')：使用SQLAlchemy创建一个数据库连接引擎。

print(engine)：打印引擎对象，确认连接是否成功。

**（4）将数据存入数据库：**

df.to\_sql(table\_name\_1, engine, index=True, if\_exists='replace')：将DataFrame中的数据存入指定的数据库表中。index=True表示将DataFrame的索引也存入数据库，if\_exists='replace'表示如果表已经存在，则替换它。

print(f'Data has been successfully loaded into the {table\_name\_1} table in the remote database.')：打印成功消息，确认数据已成功加载到数据库中。

### 3.1.3创建增删改查功能

|  |
| --- |
| import [numpy](https://pypi.org/project/numpy/" \l "files) as np  # 创建数组  def create\_array(size):  array = np.zeros(size)  for i in range(len(array)):  array[i] = np.inf  return array  # 插入值  def append\_element(array, value):  position = search\_element(array, np.inf)[0]  array[position] = value  array = sort\_element(array)  return array  # 删除某值，输入下标索引  def delete\_element(array, index):  array = np.delete(array, index)  array = sort\_element(array)  return array  # 修改具体的值  def update\_element(array, index, new\_value):  array[index] = new\_value  array = sort\_element(array)  return array  # 查找具体的值索引  def search\_element(array, value):  indices = np.where(array == value)[0]  return indices if indices.size > 0 else None  # 排序算法  def sort\_element(array):  return np.sort(array) |

## 3.2功能实现及代码解释

### 3.2.1考勤信息统计表

|  |
| --- |
| from [utils.database\_manager](https://pypi.org/project/utils) import DatabaseManager  from [sqlalchemy](https://pypi.org/project/SQLAlchemy) import Column, String, Integer, DateTime, Time, text  from [sqlalchemy.ext.declarative](https://pypi.org/project/SQLAlchemy) import declarative\_base  from [sqlalchemy.orm](https://pypi.org/project/SQLAlchemy) import sessionmaker  from [datetime](https://docs.python.org/3/library/datetime.html) import datetime  Base = declarative\_base()  class AttendanceRecord(Base):  \_\_tablename\_\_ = 'attendance\_information'  stu\_id = Column(String(length=20), primary\_key=True)  course\_id = Column(String, primary\_key=True)  course\_no = Column(Integer, primary\_key=True)  teacher\_id = Column(String)  date = Column(DateTime, default=datetime.now())  status = Column(Integer)  signin\_time = Column(Time)  signout\_time = Column(Time)  reason = Column(String)  def \_\_init\_\_(self, stu\_id, course\_id, course\_no, teacher\_id, date, status, signin\_time=None, signout\_time=None, reason=None):  def \_\_str\_\_(self):  returnf"AttendanceRecord(stu\_id={self.stu\_id}, course\_id={self.course\_id}, " \  f"course\_no={self.course\_no}, teacher\_id={self.teacher\_id}, "  f"date={self.date}, status={self.status}, " \  f"signin\_time={self.signin\_time}, signout\_time={self.signout\_time}), reason={self.reason}"  class AttendanceManager:  def \_\_init\_\_(self, table\_name):  self.table\_name = table\_name  self.db\_manager = DatabaseManager(table\_name)  def add\_attendance\_record(self, attendance\_record):  '''  添加考勤记录  :param attendance\_record:  :return: console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  existing\_attendance\_record = session.query(AttendanceRecord).filter\_by(  stu\_id=attendance\_record.stu\_id,  course\_id=attendance\_record.course\_id,  course\_no=attendance\_record.course\_no  ).first()  if existing\_attendance\_record:  print(f"This attendance record with stu\_id && course\_id && course\_no"  f"{attendance\_record.stu\_id} && {attendance\_record.course\_id} && {attendance\_record.course\_no} "  f"already exists in the database.")  else:  session.add(attendance\_record)  session.commit()  print(f"This attendance record with stu\_id && course\_id && date"  f"{attendance\_record.stu\_id} && {attendance\_record.course\_id} && {attendance\_record.course\_no} "  f"{attendance\_record.teacher\_id} && {attendance\_record.date} && {attendance\_record.status} "  f"{attendance\_record.signin\_time} && {attendance\_record.signout\_time}"  f"added to the database.")  def view\_all\_attendance\_records(self)：  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  all\_attendance\_records = session.query(AttendanceRecord).all()  for attendance\_record\_item in all\_attendance\_records:  print(attendance\_record\_item)  def search\_attendance\_record(self, stu\_id, course\_id, course\_no):  """  查看某个学生是否签到了  :param stu\_id: 学生的ID  :param course\_id: 课程的ID  :param course\_no: 课程的课次  :return: Attendance record if found, else None  """  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.query(AttendanceRecord).filter\_by(  stu\_id=stu\_id,  course\_id=course\_id,  course\_no=course\_no  ).first()  # if result.status == 1:  # print(f'{result.stu\_id}已签到，签到时间为{result.signin\_time}')  # else:  # print(f"{result.stu\_id}未签到")  return result  def execute\_sql\_query(self, sql\_query):  '''  执行任意sql语句  :param sql\_query: 需要在attendance\_information表中执行的sql语句  :return: 查询的结果  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.execute(text(sql\_query))  # 检查语句是更新还是删除  if sql\_query.strip().upper().startswith("UPDATE") or sql\_query.strip().upper().startswith("DELETE"):  row\_count = result.rowcount  print(f"{row\_count} row(s) affected.")  session.commit() # Commit the transaction  return row\_count  # 对于其他类型的查询（SELECT、INSERT 等），返回获取的结果  return result.fetchall() |

### 3.2.2 课程表功能

|  |
| --- |
| class ClassScheduleManager:  def \_\_init\_\_(self, table\_name):  self.table\_name = table\_name  self.db\_manager = DatabaseManager(table\_name)  def add\_course(self, schedule):  '''  往class\_schedule表里面添加记录  :param schedule:  :return:  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  existing\_class\_schedule = session.query(ClassSchedule).filter\_by(schedule\_id=schedule.schedule\_id).first()  if existing\_class\_schedule:  print(f"Schedule with ID {schedule.schedule\_id} already exists in the database.")  else:  session.add(schedule)  session.commit()  print(f"Schedule with ID {schedule.schedule\_id} added to the database.")  def view\_all\_courses(self):  '''  查看class\_schedule表中全部的信息  :return: class\_schedule表中全部的信息  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  all\_schedule = session.query(ClassSchedule).all()  for schedule in all\_schedule:  print(schedule)  def execute\_sql\_query(self, sql\_query):  '''  执行任意sql语句  :param sql\_query: 需要在course表中执行的sql语句  :return: 查询的结果  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.execute(text(sql\_query))  return result.fetchall() |

### 3.2.3 选课功能

|  |
| --- |
| class CourseSelectionManager:  def \_\_init\_\_(self, table\_name):  self.table\_name = table\_name  self.db\_manager = DatabaseManager(table\_name)  def add\_course\_selection\_record(self, course\_selection\_record):  '''  添加学生选课记录  :param course\_selection\_record:  :return:console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  existing\_course\_selection\_record = session.query(CourseSelectionRecord).filter\_by(  course\_id=course\_selection\_record.course\_id,  student\_id=course\_selection\_record.student\_id  ).first()  if existing\_course\_selection\_record:  print(f"This course\_selection\_record with course\_id && student\_id"  f"{course\_selection\_record.course\_id} &&"  f"{course\_selection\_record.student\_id} "  f"already exists in the database.")  else:  session.add(course\_selection\_record) # Corrected this line  session.commit()  print(f"This course\_selection\_record with course\_id && student\_id"  f"{course\_selection\_record.course\_id} &&"  f"{course\_selection\_record.student\_id} "  f"added to the database.")  def view\_all\_courses\_selection\_record(self):  """  查看选课表的全部信息  :return:course\_selection的全部记录信息  """  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  all\_course\_selection\_record = session.query(CourseSelectionRecord).all()  for course\_selection\_record\_item in all\_course\_selection\_record:  print(course\_selection\_record\_item)  def execute\_sql\_query(self, sql\_query):  '''  执行任意sql语句  :param sql\_query: 需要在course\_selection表中执行的sql语句  :return: 查询的结果  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.execute(text(sql\_query))  return result.fetchall() |

### 3.2.4课程信息表

|  |
| --- |
| class CourseManager:  def \_\_init\_\_(self, table\_name):  self.table\_name = table\_name  self.db\_manager = DatabaseManager(table\_name)  def add\_course(self, course):  '''  :param course:  :return:  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  existing\_course = session.query(Course).filter\_by(course\_id=course.course\_id).first()  if existing\_course:  print(f"Student with ID {course.course\_id} already exists in the database.")  else:  session.add(course)  session.commit()  print(f"Student with ID {course.course\_id} added to the database.")  def view\_all\_courses(self):  '''  查看student表中全部的信息  :return: student表中全部的学生信息  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  all\_course = session.query(Course).all()  for course in all\_course:  print(course)  def execute\_sql\_query(self, sql\_query):  '''  执行任意sql语句  :param sql\_query: 需要在course表中执行的sql语句  :return: 查询的结果  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.execute(text(sql\_query))  return result.fetchall() |

### 3.2.5 发布考勤信息

|  |
| --- |
| class PostAttendanceManager:  def \_\_init\_\_(self, table\_name):  self.table\_name = table\_name  self.db\_manager = DatabaseManager(table\_name)  def get\_max\_index(self):  '''  获取当前表中的最大索引  :return: 所有记录中索引的最大值  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  # 使用 func.max 获取 attendance\_id 列的最大值  result = session.query(func.max(PostAttendanceRecord.attendance\_id)).scalar()  # 如果没有记录，返回默认值 0  return result if result is not None else 0  def post\_attendance(self, post\_attendance\_record):  '''  发布考勤  :param post\_attendance\_record: 发布考勤的记录  :return: console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  existing\_post\_attendance\_record = session.query(PostAttendanceRecord).filter\_by(  code=post\_attendance\_record.code,  ).first()  if existing\_post\_attendance\_record:  print(f"This post attendance record with code '{post\_attendance\_record.code}' already exists in the database.")  return False  else:  session.add(post\_attendance\_record)  session.commit()  print(f"This post attendance record with attendance\_id"  f"{post\_attendance\_record.attendance\_id} && {post\_attendance\_record.course\_id} &&{post\_attendance\_record.course\_name}"  f"{post\_attendance\_record.course\_no} && {post\_attendance\_record.attendance\_start\_time}"  f"{post\_attendance\_record.attendance\_end\_time} && {post\_attendance\_record.code}"  f"added to the database.")  return True  def view\_all\_post\_attendance\_records(self):  """  查看发布考勤表的全部信息  :return: post\_attendance\_information的全部记录信息  """  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  all\_post\_attendance\_records = session.query(PostAttendanceRecord).all()  for post\_attendance\_record\_item in all\_post\_attendance\_records:  print(post\_attendance\_record\_item)  def search\_post\_attendance\_record(self, attendance\_id):  """  查看某个考勤是否发布了  :param attendance\_id: 考勤号  :return: post attendance record if found, else None  """  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.query(PostAttendanceRecord).filter\_by(  attendance\_id=attendance\_id,  ).first()  if result != None:  print(f'{result.attendance\_id}已存在')  else:  print(f"{attendance\_id}查找不到")  def execute\_sql\_query(self, sql\_query):  '''  执行任意sql语句  :param sql\_query: 需要在attendance\_information表中执行的sql语句  :return: 查询的结果  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.execute(text(sql\_query))  return result.fetchall() |

### 3.2.6学生个人信息

|  |
| --- |
| class StudentManager:  def \_\_init\_\_(self, table\_name):  '''创建一个StudentManager对象时需要传入表的名称（student\_information），用于连接数据库中的相关表  :param table\_name:  '''  self.table\_name = table\_name  self.db\_manager = DatabaseManager(table\_name)  def add\_student(self, student):  '''  插入到student表需要传入一个Student对象  :param student:  :return: console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  # Check if the student already exists  existing\_student = session.query(Student).filter\_by(stu\_id=student.stu\_id).first()  if existing\_student:  print(f"Student with ID {student.stu\_id} already exists in the database.")  else:  session.add(student)  session.commit()  print(f"Student with ID {student.stu\_id} added to the database.")  def view\_all\_students(self):  '''  查看student表中全部的信息  :return: student表中全部的学生信息  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  all\_students = session.query(Student).all()  for student in all\_students:  print(student);  return all\_students  def delete\_student(self, student\_id):  '''  删除student表中stu\_id为student\_id的记录  :param student\_id: 需要删除的学生的id号  :return: console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  student\_to\_delete = session.query(Student).filter\_by(stu\_id=student\_id).first()  if student\_to\_delete:  session.delete(student\_to\_delete)  session.commit()  print(f"Student with ID {student\_id} deleted from the database.")  else:  print(f"Student with ID {student\_id} not found in the database.")  def search\_student(self, student\_id):  '''  查询某个学生的记录，以stu\_id为查询条件  :param student\_id: 需要查询的学生id  :return: 某个学生记录的全部信息  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  student = session.query(Student).filter\_by(stu\_id=student\_id).first()  if student:  return student  else:  print(f"Student with ID {student\_id} not found in the database.")  return None  def modify\_student(self, student\_id, new\_data):  '''  修改某个学生记录的某些属性列  :param student\_id:需要修改的学生记录的id号  :param new\_data:需要修改的属性列的字典  :return:console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  student\_to\_modify = session.query(Student).filter\_by(stu\_id=student\_id).first()  if student\_to\_modify:  for key, value in new\_data.items():  setattr(student\_to\_modify, key, value)  session.commit()  print(f"Student with ID {student\_id} modified in the database.")  else:  print(f"Student with ID {student\_id} not found in the database.")  def execute\_sql\_query(self, sql\_query):  '''  执行任意sql语句  :param sql\_query: 需要在student\_information表中执行的sql语句  :return: 查询的结果  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.execute(text(sql\_query))  return result.fetchall() |

### 3.2.7 老师个人信息查询

|  |
| --- |
| class TeacherManager:  def \_\_init\_\_(self, table\_name):  '''  创建一个TeacherManager对象时需要传入表的名称（teacher\_information），用于连接数据库中的相关表  :param table\_name:  '''  self.table\_name = table\_name  self.db\_manager = DatabaseManager(table\_name)  def add\_teacher(self, teacher):  '''  插入到teacher表需要传入一个teacher对象  :param teacher:  :return: console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  # Check if the teacher already exists  existing\_teacher = session.query(Teacher).filter\_by(teacher\_id=teacher.teacher\_id).first()  if existing\_teacher:  print(f"teacher with ID {teacher.teacher\_id} already exists in the database.")  else:  session.add(teacher)  session.commit()  print(f"teacher with ID {teacher.teacher\_id} added to the database.")  def view\_all\_teachers(self):  '''  查看teacher表中全部的信息  :return: teacher表中全部的教师信息  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  all\_teachers = session.query(Teacher).all()  for teacher in all\_teachers:  print(teacher)  def delete\_teacher(self, teacher\_id):  '''  删除teacher表中teacher\_id为teacher\_id的记录  :param teacher\_id: 需要删除的老师的id号  :return: console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  teacher\_to\_delete = session.query(Teacher).filter\_by(teacher\_id=teacher\_id).first()  if teacher\_to\_delete:  session.delete(teacher\_to\_delete)  session.commit()  print(f"teacher with ID {teacher\_id} deleted from the database.")  else:  print(f"teacher with ID {teacher\_id} not found in the database.")  def search\_teacher(self, teacher\_id):  '''  查询某个老师的记录，以teacher\_id为查询条件  :param teacher\_id: 需要查询的老师id  :return: 某个老师记录的全部信息  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  teacher = session.query(Teacher).filter\_by(teacher\_id=teacher\_id).first()  if teacher:  print(teacher)  return teacher  else:  print(f"teacher with ID {teacher\_id} not found in the database.")  return None  def modify\_teacher(self, teacher\_id, new\_data):  '''  修改某个老师记录的某些属性列  :param teacher\_id:需要修改的老师记录的id号  :param new\_data:需要修改的属性列的字典  :return:console.log  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  teacher\_to\_modify = session.query(Teacher).filter\_by(teacher\_id=teacher\_id).first()  if teacher\_to\_modify:  for key, value in new\_data.items():  setattr(teacher\_to\_modify, key, value)  session.commit()  print(f"teacher with ID {teacher\_id} modified in the database.")  else:  print(f"teacher with ID {teacher\_id} not found in the database.")  def execute\_sql\_query(self, sql\_query):  '''  执行任意sql语句  :param sql\_query: 需要在teacher\_information表中执行的sql语句  :return: 查询的结果  '''  Session = sessionmaker(bind=self.db\_manager.engine)  session = Session()  result = session.execute(text(sql\_query))  return result.fetchall() |

## 3.3 前端页面代码实现及文法分析

### 3.3.1总页面（以登陆页面为例）

|  |
| --- |
| // pages/login/index.js  import { M  } from '../../utils/M.js';  const m = new M();  let app = getApp();  let that;  Page({  //页面的初始数据   data: {                role: ['学生', '教师'],                roleIndex: 0 },         role(e) {                that.setData({                       roleIndex: e.detail.value});},     //生命周期函数--监听页面加载         onLoad: function (options) {                that = this;                that.setData({                       width: app.globalData.windowWidth,                       height: app.globalData.windowHeight}); },        //生命周期函数--监听页面显示         onShow: function () {                let loginStatus = wx.getStorageSync('s');                if (loginStatus) {                       wx.reLaunch({                              url: '../index/index',});                       return;} },         login(e) {                if (e.detail.value.userName == "") {                       m.showTost('请输入学号');                       return;} else if (e.detail.value.userPwd == "") {                       m.showTost('请输入姓名');                       return;                } else {                       // m.showLoading('正在登录');                       if (e.detail.value.role == 0) {                              //测试代码                              // console.log(typeof(e.detail.value.userName))                              // console.log(typeof(e.detail.value.userPwd))                              m.student\_signup(                                     e.detail.value.userName,                                     e.detail.value.userPwd)                       } else if (e.detail.value.role == 1) { //测试                              m.teacher\_signup(                                     e.detail.value.userName,                                     e.detail.value.userPwd)}}},         get\_account: function (event) {                const usrname = event.detail.value                this.setData({                       usrname: usrname})                app.globalData.name = usrname},         get\_password: function (event) {                const usrid = event.detail.value                this.setData({                       usrid: usrid })                app.globalData.id = usrid},}) |

**对上述代码进行文法分析：**

这段代码是一个微信小程序的登录页面逻辑。下面是对代码的文法分析：

导入模块：

import { M } from '../../utils/M.js';

这行代码导入了一个名为 M 的模块，该模块位于 ../../utils/M.js 路径下。

实例化对象：

const m = new M();

这行代码创建了一个 M 类的实例，并将其赋值给变量 m。

获取全局应用实例：

let app = getApp();

这行代码获取了小程序的全局应用实例，并将其赋值给变量 app。

定义页面：

Page({

这行代码开始定义一个页面，Page 是微信小程序提供的用于定义页面的函数。

页面的初始数据：

data: {

role: ['学生', '教师'],

roleIndex: 0},

这部分定义了页面的初始数据，包括一个角色数组 role 和一个角色索引 roleIndex。

处理角色选择：

role(e) {

that.setData({

roleIndex: e.detail.value });},

这个函数处理角色选择事件，更新 roleIndex 的值。

页面加载生命周期函数：

|  |
| --- |
| onLoad: function (options) {  that = this;  that.setData({  width: app.globalData.windowWidth,  height: app.globalData.windowHeight  });}, |

这个函数在页面加载时调用，设置页面的宽度和高度。

页面显示生命周期函数：

|  |
| --- |
| onShow: function () {  let loginStatus = wx.getStorageSync('s');  if (loginStatus) {  wx.reLaunch({  url: '../index/index',  });  return;  }  }, |

这个函数在页面显示时调用，检查登录状态并重定向到首页。

登录处理函数：

|  |
| --- |
| login(e) {  if (e.detail.value.userName == "") {  m.showTost('请输入学号');  return;  } else if (e.detail.value.userPwd == "") {  m.showTost('请输入姓名');  return;  } else {  if (e.detail.value.role == 0) {  m.student\_signup(  e.detail.value.userName,  e.detail.value.userPwd  );  } else if (e.detail.value.role == 1) {  m.teacher\_signup(  e.detail.value.userName,  e.detail.value.userPwd);}}}, |

这个函数处理登录逻辑，根据角色调用不同的注册函数。

获取账号和密码：

|  |
| --- |
| get\_account: function (event) {  const usrname = event.detail.value;  this.setData({  usrname: usrname  });  app.globalData.name = usrname;  },get\_password: function (event) {  const usrid = event.detail.value;  this.setData({  usrid: usrid});  app.globalData.id = usrid;}, |

这两个函数分别处理获取账号和密码的逻辑，并更新全局数据。

微信小程序页面的逻辑部分，主要功能是处理学生课程选择和请假申请。下面是对代码的文法分析：

获取全局应用实例：

var app = getApp();

这行代码获取了小程序的全局应用实例，并将其赋值给变量 `app`。

定义页面：

Page({

这行代码开始定义一个页面，`Page` 是微信小程序提供的用于定义页面的函数。

页面的初始数据：

|  |
| --- |
| data: {  week: ['3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18'],  weekIndex: 0,  stuId: '',  courseData: [],  courseNames: [],  selectedCourse: '',  contactInfo: '',  expOverview: ''}, |

这部分定义了页面的初始数据，包括周数、周索引、学生ID、课程数据、课程名称、选中的课程、联系信息和实验概述。

处理周选择：

|  |
| --- |
| weekchoose(e) {  this.setData({  weekIndex: e.detail.value  });  }, |

这个函数处理周选择事件，更新 `weekIndex` 的值。

处理课程选择：

|  |
| --- |
| classchoose(e) {  this.setData({  selectedCourse: this.data.courseNames[e.detail.value]  });  }, |

这个函数处理课程选择事件，更新 `selectedCourse` 的值。

处理文本域输入：

|  |
| --- |
| handleTextAreaInput: function (e) {  this.setData({  expOverview: e.detail.value  });  console.log(this.data.expOverview);  }, |

这个函数处理文本域输入事件，更新 `expOverview` 的值并打印到控制台。

页面加载生命周期函数：

|  |
| --- |
| onLoad: function () {  app.editTabBar();  const that = this;  wx.getStorage({  key: 'stuId',  success: function (res) {  console.log(res.data);  that.setData({  stuId: res.data  });  console.log(that.data.stuId);  that.get\_stu\_class(that.data.stuId);  }  });  }, |

这个函数在页面加载时调用，获取缓存的学生ID并调用 `get\_stu\_class` 函数获取课程信息。

获取课程信息：

|  |
| --- |
| get\_stu\_class(stuId) {  var that = this;  var apiUrl = 'http://主机IP:5000/student\_manager/view\_student\_courses';  console.log(stuId);  wx.request({  url: apiUrl,  method: 'GET',  header: {  'app': 'wx-app'  },  data: {  'student\_id': stuId,  'semester': '2023',  'week\_no': 5  },  success: function (res) {  if (res.statusCode === 200) {  var result = res.data;  that.setData({  courseData: result.class\_schedule\_records  }, () => {  that.fetchCourseData();  });  } else {  console.error('Error:', res.statusCode, res.data);  }  },  fail: function (error) {  console.error('Request failed:', error);  }  });  }, |

这个函数通过请求接口获取学生的课程信息，并调用 `fetchCourseData` 函数处理课程数据。

处理课程数据：

|  |
| --- |
| fetchCourseData: function () {  var that = this;  var names = that.data.courseData.map(function (course) {  return course.course\_name;  });  var uniqueNames = [...new Set(names)];  that.setData({  courseNames: uniqueNames,  selectedCourse: uniqueNames[0] || ''  });  console.log(that.data.courseNames);  }, |

这个函数处理课程数据，提取课程名称并去重，然后更新 `courseNames` 和 `selectedCourse`。提交表单\*\*：

|  |
| --- |
| submitForm: function (e) {  var apiUrl = 'http://主机IP:5000/student\_manager/absence\_on\_leave';  const formData = e.detail.value;  const requestData = {  'student\_id': '2021611001',  'course\_id': 'c1',  'teacher\_id': 'T001',  'course\_number': '10'  };  wx.request({  url: apiUrl,  method: 'POST',  header: {  'app': 'wx-app',  'Content-Type': 'application/x-www-form-urlencoded'  },  data: requestData,  success: function (res) {  switch (res.statusCode) {  case 200:  wx.showToast({  title: '提交成功',  icon: 'success',  duration: 2000  });  break;  case 410:  wx.showToast({  title: '课程号输入错误',  icon: 'none',  duration: 2000  });  break;  case 412:  wx.showToast({  title: '课次输入有误',  icon: 'none',  duration: 2000  });  break;  case 420:  wx.showToast({  title: '重复申请',  icon: 'none',  duration: 2000  });  break;  case 400:  wx.showToast({  title: '请求头错误',  icon: 'none',  duration: 2000  });  break;  case 500:  wx.showToast({  title: '非法请求',  icon: 'none',  duration: 2000  });  break;  default:  wx.showToast({  title: '未知错误',  icon: 'none',  duration: 2000  });  break;  }  },  fail: function (error) {  wx.showToast({  title: '请求失败',  icon: 'none',  duration: 2000  });  }  });  }, |

这个函数处理表单提交，构造请求数据并发送请求，根据响应状态码显示不同的提示信息。

### 3.3.3 教师页面（以请假审批为例）

代码是一个微信小程序页面的逻辑部分，主要功能是处理教师审批学生请假申请。下面是对代码的文法分析：

获取全局应用实例：

var app = getApp();

这行代码获取了小程序的全局应用实例，并将其赋值给变量 `app`。

定义页面：

Page({

这行代码开始定义一个页面，`Page` 是微信小程序提供的用于定义页面的函数。

页面的初始数据：

data: {

leaveApplications: []

},

这部分定义了页面的初始数据，包括一个空数组 `leaveApplications`，用于存储请假申请列表。

页面加载生命周期函数：

onLoad: function () {

app.editTabBar1();

this.getLeaveApplications();

},

这个函数在页面加载时调用，显示自定义的底部导航并调用 `getLeaveApplications` 函数获取请假申请列表。

获取请假申请列表：

|  |
| --- |
| getLeaveApplications: function () {  var that = this;  var apiUrl = 'http://主机IP:5000/teacher\_manager/get\_leave\_requests';  wx.request({  url: apiUrl,  header: {  'app': 'wx-app'  },  data: {  'teacher\_id': 'T001'  },  method: 'GET',  success: function (res) {  that.setData({  leaveApplications: res.data.leave\_requests  });  console.log('获取请假申请列表成功', that.data.leaveApplications);  },  fail: function (err) {  console.error('获取请假申请列表失败', err);  }  });  }, |

这个函数通过请求接口获取请假申请列表，并更新页面数据。

审批通过请假申请：

|  |
| --- |
| approveLeave: function (e) {  var that = this;  var index = e.currentTarget.dataset.index;  var application = this.data.leaveApplications[index];  var student\_id = application.studentId;  var course\_id = application.courseCode;  var course\_no = application.courseWeek;  console.log('通过请假申请：', this.data.leaveApplications[index]);  var apiUrl = 'http://主机IP:5000/teacher\_manager/review\_leave\_request';  wx.request({  url: apiUrl,  method: 'POST',  header: {  'app': 'wx-app',  'Content-Type': 'application/x-www-form-urlencoded'  },  data: {  'student\_id': student\_id,  'course\_id': course\_id,  'course\_no': course\_no,  'is\_reviewed': true  },  success: function (res) {  console.log('通过申请成功', res);  wx.showToast({  title: '通过申请成功',  icon: 'none'  });  that.getLeaveApplications();  },  fail: function (err) {  console.error('通过申请失败', err);  wx.showToast({  title: '通过申请失败',  icon: 'none'  });  }});}, |

这个函数处理审批通过请假申请的逻辑，发送请求并更新请假申请列表。

审批拒绝请假申请：

|  |
| --- |
| rejectLeave: function (e) {  var that = this;  var index = e.currentTarget.dataset.index;  var application = this.data.leaveApplications[index];  var student\_id = application.studentId;  var course\_id = application.courseCode;  var course\_no = application.courseWeek;  console.log('拒绝请假申请：', this.data.leaveApplications[index]);  var apiUrl = 'http://主机IP:5000/teacher\_manager/review\_leave\_request';  wx.request({  url: apiUrl,  method: 'POST',  header: {  'app': 'wx-app',  'Content-Type': 'application/x-www-form-urlencoded'  },  data: {  'student\_id': student\_id,  'course\_id': course\_id,  'course\_no': course\_no,  'is\_reviewed': false  },  success: function (res) {  console.log('拒绝申请成功', res);  wx.showToast({  title: '拒绝申请成功',  icon: 'none'  });  that.getLeaveApplications();  },  fail: function (err) {  console.error('拒绝申请失败', err);  wx.showToast({  title: '拒绝申请失败',  icon: 'none'  });  }  });  }, |

这个函数处理审批拒绝请假申请的逻辑，发送请求并更新请假申请列表。