软件设计文档

项目组成员: 杨霁晗 15331353

杨伟铭 15331364

杨涵 15331351

陈志扬 15331046

颜林屹 15331347

指导老师：王青

一、问题陈述

我们从大学课堂的实际需求出发，我们观察到许多课程都有组队需求，而这对于老师和同学来说这都是一个稍微麻烦的事情，这往往需要较长的时间，特别是针对公选课这种同学之间往往都互相不认识的课程，组队对于只有一个人和较为害羞的同学往往是一件困难的事情。另一方面，老师也难以知道同学的组队进展情况，因为其难以可视化。因此我们从这个问题出发，计划设计一个专为大学课堂组队的程序。

每当学期开学，教师就可以通过我们的程序创建对应的课程并加上必要的限制条件，同时教师也能删除课程，进入教师主页能查看已创建课程信息，还可以进入具体的课程，查看当前课程的组队情况，包括成员学号、队员信息、队伍人数等信息。

学生进入主页后能够查看已加入的课程信息，还可以搜索想要加入的课程，在搜索时添加必要的条件，就能获得系统给出的符合条件的课程列表。学生选上教师的课程后，就能加入老师创建的课程，并能在课程里面创建队伍和发布相关信息，要是学生不知道要加入哪支队，系统还提供了快速匹配的功能，能够让学生随机加入到有空位的队伍。创建新队伍的学生将会作为此队的队长，加入已有队伍的学生作为队员，队长有权利移除队员，他也可以选择退出队伍和解散队伍，退出队伍就需要指定另外一个人当队长。当然，学生都能自主选择退出相应的课程。当一门课程到教师规定的截止时间后，教师有权利获取并打印课程组队信息，从而知道哪些学生是一队，便于老师给每个队伍的最终课程成绩评分。

管理员能够在后台通过数据库维护以上相关的课程信息和用户信息，用户信

息包括学生和老师的个人信息，如昵称、性别和与其相关的课程等，课程信息包括各个队伍编号、队员信息和队伍人数等。中山大学 WeTeam 能够访问这个数据库的开放 SQL 接口，能够及时有效地获取这个系统上的数据，并能够实时更新这个系统上的课程信息和用户信息。

二、技术选型

微信小程序是一种全新的连接用户与服务的方式，它可以在微信内被便捷地获取和传播，同时具有出色的使用体验。因此我们决定开发 WeTeam 小程序以实现相应的功能。WeTeam 小程序项目中分为前端、后端两个部分，因此我们将分别陈述前、后端的技术选型。

2.1 前端技术选型

前端部分，我们需要设计小程序的页面，包括页面的内容、样式、交互等。微信小程序的代码构成分为四部分：JSON（配置）、WXML（模板）、WXSS（样式）、JS（逻辑交互），无论是从标签、样式集合还是从语法规范的角度上，都与传统 WEB 页面的代码构成（HTML+CSS+JS+JSON）相近。在开发工具上，我们使用微信 web 开发者工具完成前端内容的开发。另外，微信小程序开发中不需要引入框架，因此我们的前端技术选型中没有使用框架。

2.2 后端技术选型

后端部分，我们需要建立数据库用于存储小程序中产生的数据，例如用户数据、用户创建的队伍数据等，另外就是对于前端的请求，修改和查

询数据库，并返回所需数据。在具体代码编写阶段，我们主要使用 Python 语言，并且结合利用了如下工具&库完成了后端的设计：

· Flask：一个使用 Python 编写的轻量级服务器框架，使用该框架可以快速搭建一个服务器所需的功能框架。我们对比了 Django，我们由于功能比较简单，数据之间关系也并不复杂，因此 Django 虽然功能更加全面，但是相比而言，Flask 更适用于我们这种轻量级的程序。

· Sqlite：一个实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。相比于 MySQL，我们选择 Sqlite 的原因只是因为我之前选用过 sqlite 数据库，相比而言更熟悉，其实两者都是 SQL 数据库，使用上并无明显的区别。而 SQL 型数据库相较于 NoSQL 数据库，比如 MongoDB，其优势主要在于其是关系型数据库，而且我们在数据库系统一门课程中有较多的学习，相比而言用起来也会更了解。

· SQLAlchemy：Python 编程语言下的一款插件，提供了 SQL 工具包以及对象关系映射工具。Flask 框架提供了对于 SQLAlchemy 的支持，将其作为其中的一个插件，它的好处在于无法重复的写许多 SQL 语句，只需要调用一些更语义化的函数，并添加一些简单的限定，就可以完成 SQL 型数据库的各种操作。

三、架构设计

1．架构综述

我们的整个项目使用的是 MVC 架构，即 Model-View-Controller 三个部分。MVC 架构将界面、模型、业务逻辑三个部分分离开来，提高代码

的可复用性。这种分层的、模块化的架构有利于整个项目的开发，使得因为我们可以再一个时间内专门关注一个方面。比如：我们可以在不依赖业务逻辑的情况下而完成视图层的设计，同时这种架构也更有利于分组开发，不同的组员可以同时进行视图层、控制层和模型的开发。

2. 架构分层描述

2.1 模型层

我们的项目的数据模型分为两个部分：

1) 用户的数据模型，包括用户的类别，用户加入的课程信息，用户创建的课程信息等

2) 课程信息，包括课程中组队的情况，队伍的信息，已经其他课程的基本信息

2.2 视图层

负责与用户进行交互的界面。界面设计人员进行图形化界面的设计。具体包括以下页面：

1) 主页：

创建课程、加入课程、显示已加入课程

2) 课程信息页面：

显示一个已创建的课程，包括课程的基本信息，已经存在的可视化组队情况。

3) 队伍信息页面：

一个队伍的信息页面，队员可以查看队伍的基本信息，已经退出队伍，队长可以踢出队员，进行这些基本操作。

4）创建课程页面：

创建课程所需要的信息填充页面，包括上课时间，组队起止时间，组队人数限制，还有一些课程说明信息等

5）课程搜索页面

主要用于学生搜索想要加入的课程，系统会给出满足条件的课程列表以供学生进行选择。

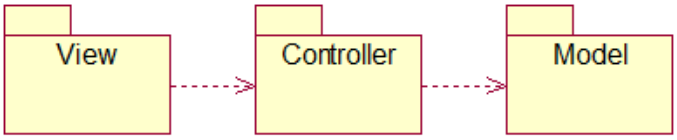
2.3 控制器层

控制器层负责转发请求，处理事件（用户的行为或是 Model 的改变）并作出响应。对于我们的项目来说主要包括以下几个部分：

1) 在主页需要根据用户信息进行渲染，老师在主页进行创建课程或者学生在主页进行加入课程，或是点击进入一门已加入的课程，并且对于已加入的课程，需要根据不同用户来在 Model 载入数据进行个性化显示。当点击一门已加入课程或加入一门课程的时候，需要跳转到课程信息页面
2) 在课程信息中，需要根据 Model 中的课程信息数据来加载课程页面，并渲染出队伍信息，区别不同用户可以进行的不同操作（比如课程的创建者能够对于课程的一些信息进行修改）已经渲染出的界面。当用户点击一个队伍的时候，能够跳转到队伍信息页面。

3) 对于队伍信息页面，区别队长和队员能够看到的页面，已经进行的操作，需要根据 Model 中的课程信息数据进行渲染。队长和队员作为队员均可以进行操作，但是种类有所不同，当做出操作时，需要相应改变 Model 的课程信息的数据。

3．MVC 架构图划分



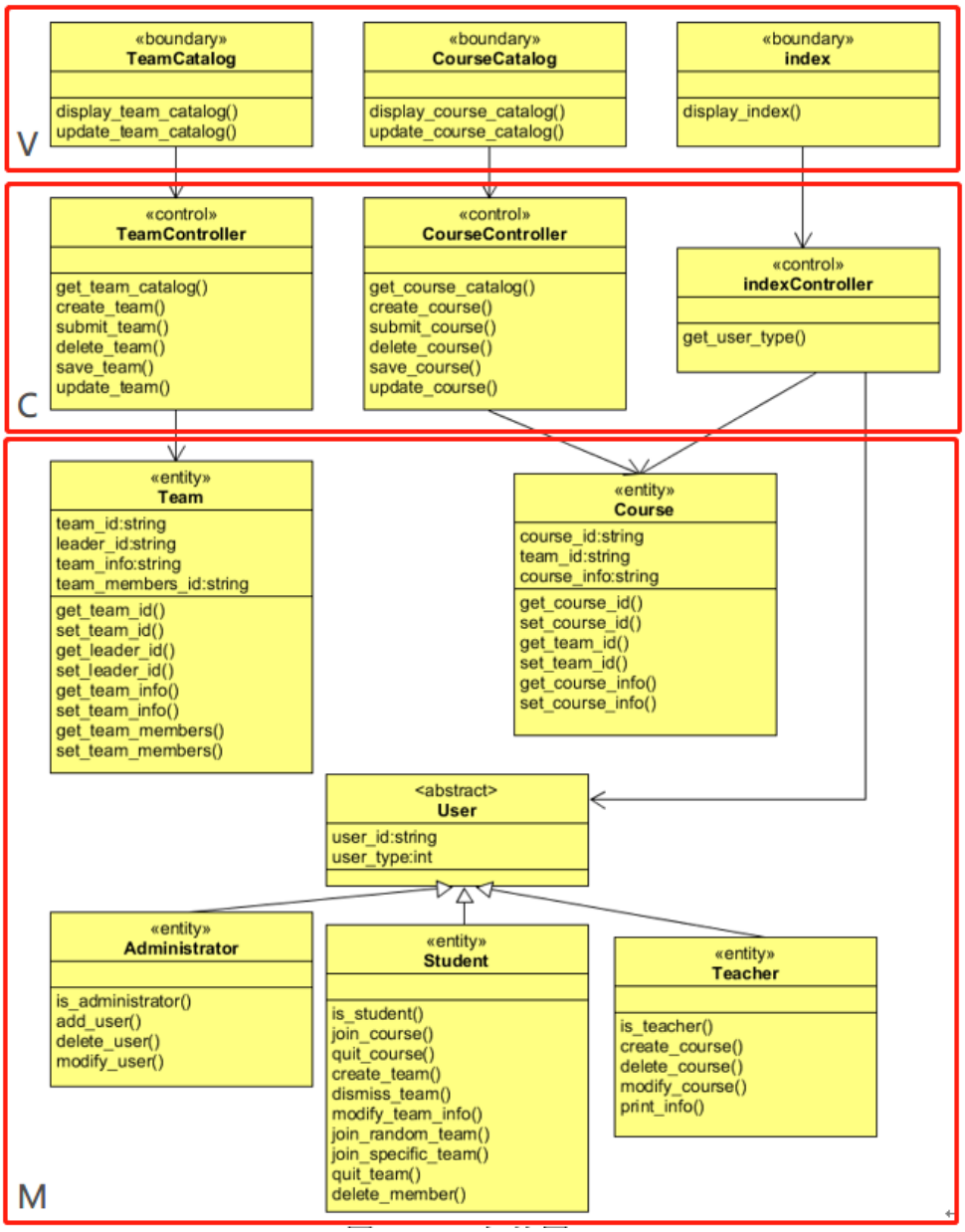


图 1 MVC 架构图

四、用例分析

为了更准确地展示出小程序的功能，我们绘制了小程序的用例设计图：

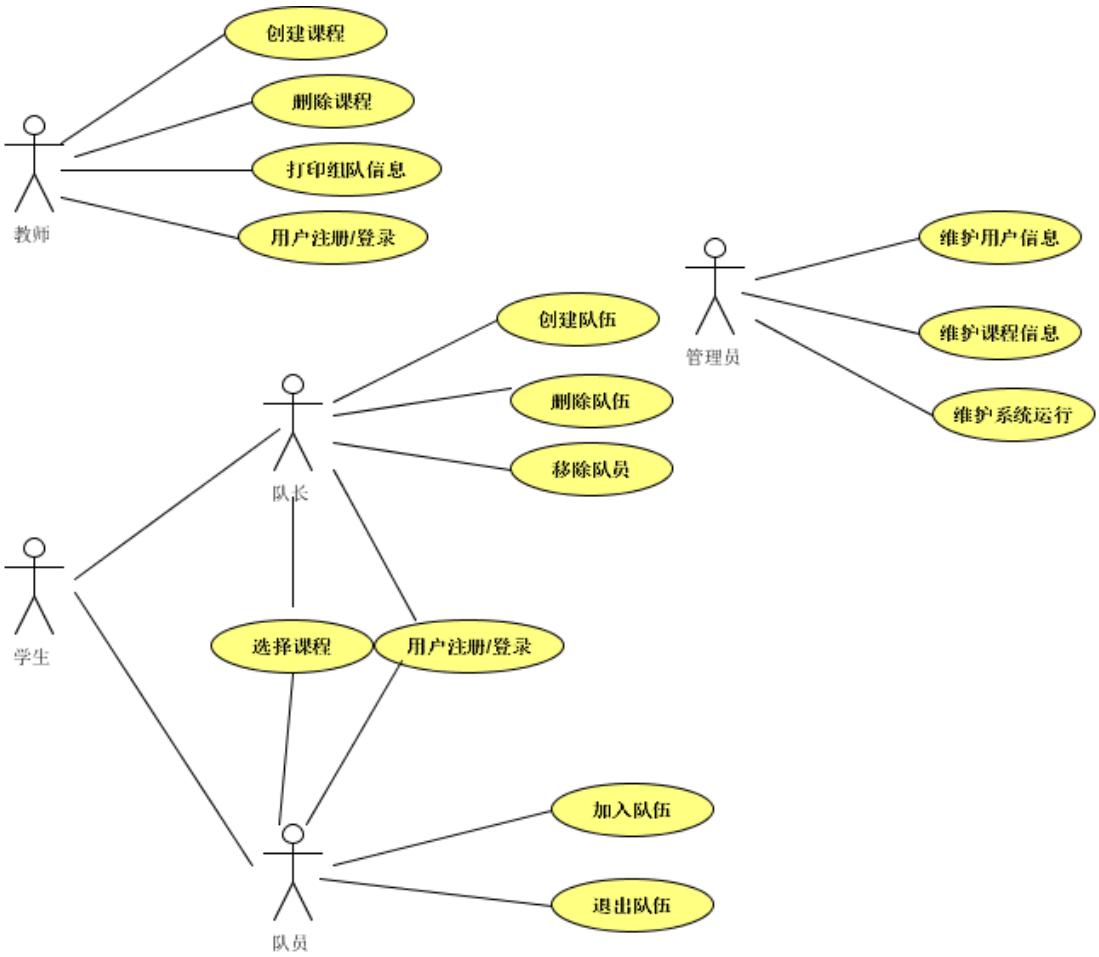


图 2 用例图

对于上面的用例图，具体的用例分析如下：

- **创建课程**

（1）简要描述

教师需要输入课程相关信息，包括课程名、上课时间、组队开始时间、组队截止时间、队伍人数区间、课程信息等来创建一门课程，学生此后将加入该课程进行组队。

（2）参与者

教师。

（3）场景描述

经过系统认证的教师可以登录系统并创建课程，从头至尾输入合法的课程名、上课时间、课程组队起止时间、最小/最大组队人数、课程说明，完成上述过程后完成课程的创建。学生可以在此后加入该课程进行组队。

（4）前置条件

教师信息已经预存在数据库中，通过系统校验。该老师没有创建过同一上课时间和同一课程名字的课程。

（5）后置条件

系统通过数据库的开放 SQL 接口查询教师所创建的课程是否符合课程相关信息，系统对此给出确认结果。

（6）事件流

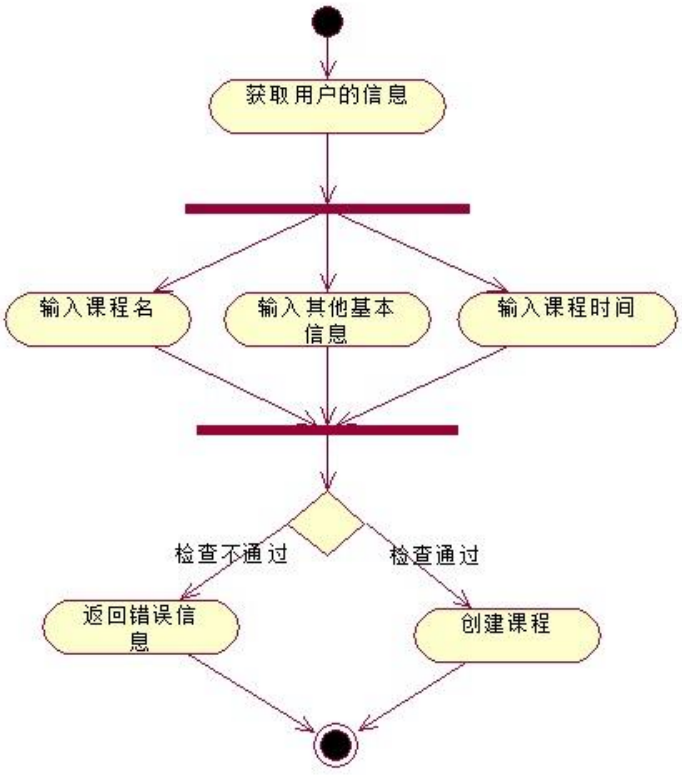


图 3 创建课程的活动图

i.基本事件流

本用例开始于教师创建课程。

a） 系统验证登录的用户为教师

b） 获取用户信息

c） 已被系统认证的教师进入创建课程界面，填写课程的各种信息

d） 系统检查用户填写的信息

e） 创建课程

ii.后备事件流

A1. 创建课程的信息不合法

✧ 系统显示错误信息

A2. 创建课程与数据库中的已有课程重复

✧ 系统显示错误信息

- **打印组队信息**

（1）简要描述

教师可以打印课程组队信息，从而得知学生组队情况。

（2）参与者

教师。

（3）场景描述

经过系统认证的教师创建课程，进入课程详情页面后，可以点击打印课程组队信息按钮，小程序将展示出所有队伍的情况，包括每支队伍的学生学号、队伍已有人数、空位人数等具体组队信息。

（4）前置条件

教师信息已经预存在数据库中，通过系统校验。

（5）后置条件

教师打印课程组队信息后，系统将展示出所有队伍的信息或者提示错误。

（6）事件流

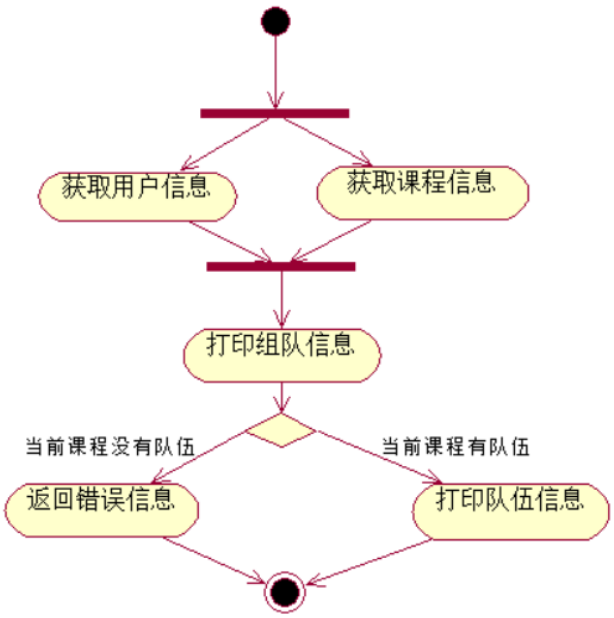


图 4 打印组队信息的活动图

i.基本事件流

本用例开始于教师打印组队信息。

a） 系统验证登录的用户为教师

b） 教师可以打印组队信息

ii.后备事件流

A1.当前课程没有学生创建的队伍

✧ 系统显示该课程没有队伍

- **选择课程**

（1）简要描述

学生可以加入老师所创建的课程。

（2）参与者

学生

（3）场景描述

经过系统认证的教师创建课程后，选上该课程的学生就可以搜索课程名称，小程序将展示该课程的所有信息，包括上课时间、任课老师、组队起止时间，然后学生可点击该课程确认加入，进而完成组队，在加入课程后也可以选择退出课程。

（4）前置条件

学生经过系统认证，并已选上该教师所创建的课程。

（5）后置条件

系统通过数据库的开放 SQL 接口查询是否存在该课程，从而向学生做出反馈，若存在，则返回课程信息，否则返回不存在该课程的信息。

（6）事件流

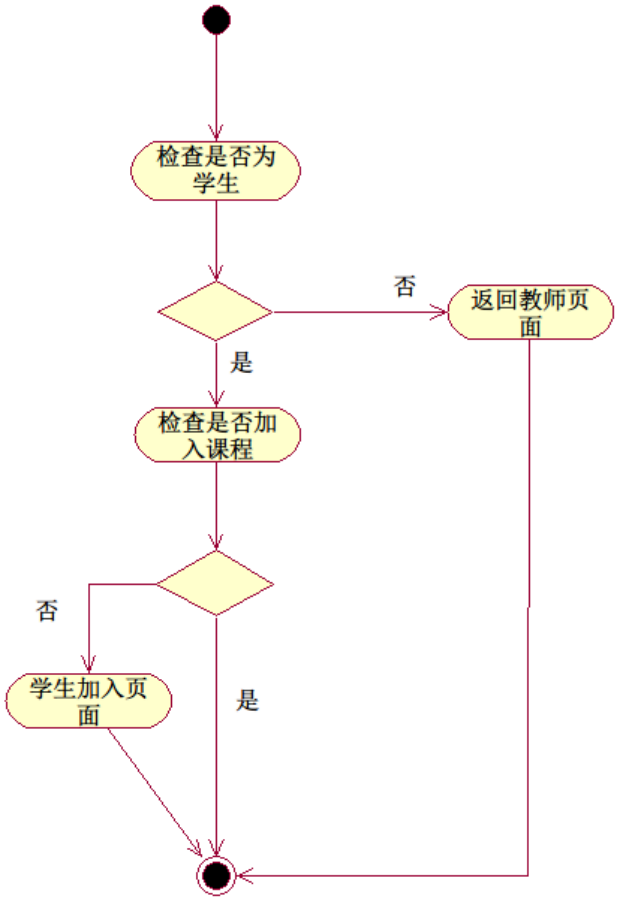


图 5 选择课程的活动图

i.基本事件流

本用例开始于学生选择课程。

a） 系统验证登录的用户为学生

b） 系统验证该学生是否已加入该课程

A1.是否加入该课程

A2.是否退出该课程

ii.后备事件流

A1.用户不是学生

✧ 系统跳转到教师页面

A2.学生未加入该课程

- 跳转到加入该课程的界面

A3.学生已加入该课程

- 跳转到退出该课程的界面

- **创建队伍**

（1）简要描述

学生加入老师所创建的课程后，并且创建一支新队伍，此时该学生即默认为队长。

（2）参与者

学生

（3）场景描述

学生加入老师所创建的课程后，学生可以创建一支新队伍，填写好队伍信息以及队伍最大人数即可创建，默认情况下该学生即为队长，队长有权力移除队员或者解散队伍。

（4）前置条件

学生经过系统认证，并已选上该教师所创建的课程，学生未加入其他队伍。

（5）后置条件

系统通过数据库的开放 SQL 接口查询该学生是否已选上该课程，从而向学生做出反馈。学生加入课程并填写完整信息后完成一支新队伍的创建后，系统返回创建成功信息。

（6）事件流

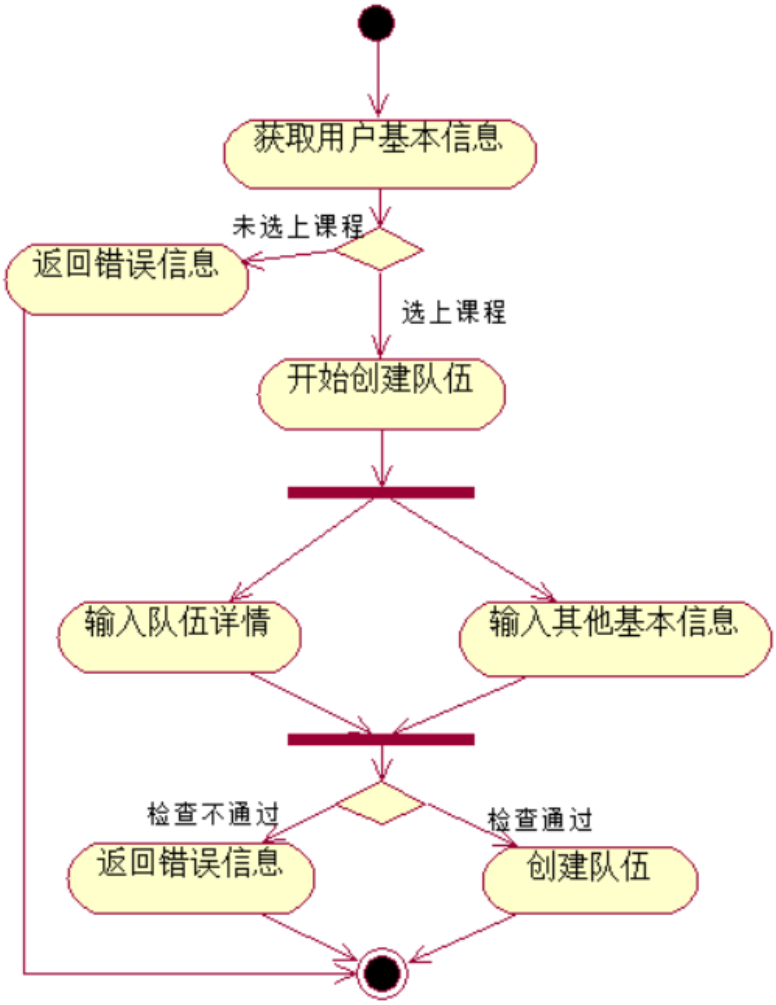


图 6 创建队伍的活动图

i.基本事件流

本用例开始于学生创建队伍。

a) 系统验证登录的用户为学生

b) 系统验证该学生是否已选上该课程

c) 学生加入该课程

d) 学生创建队伍

e) 默认该学生为队长

ii.后备事件流

A1.用户不是学生

- 系统跳转到教师页面

A2.学生未选上该课程

- 系统显示该学生未选上该课程

A3.学生未创建队伍

• 加入队伍

（1）简要描述

学生加入老师所创建的课程后，可以选择加入一支队伍，成为队员。

（2）参与者

学生

（3）场景描述

学生加入老师所创建的课程后，学生可以浏览该课程的所有队伍，并查看所有队伍的队伍详情，当学生选定加入一支已存在的队伍，学生成为这支队伍的队员，不能再加入该课程的其它队伍，队员可以选择退出队伍，。

（4）前置条件

学生经过系统认证，并已加入该教师所创建的课程，学生未加入该课程的任一队伍，可以选择一支存在且未满人的队伍加入。

（5）后置条件

系统通过数据库的开放 SQL 接口查询该学生是否已选上该课程，从而向学生做出反馈。学生加入课程和一支队伍后，成为队员，系统对此给出确认结果。

（6）事件流

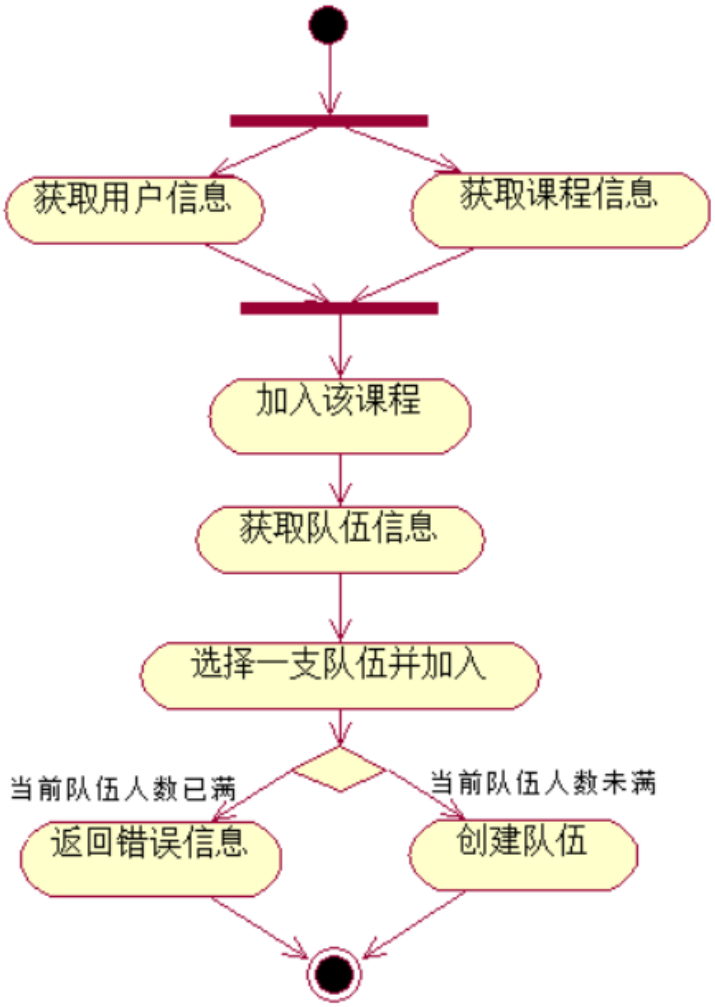


图 7 加入队伍的活动图

i.基本事件流

本用例开始于学生加入队伍。

a) 系统验证登录的用户为学生

b) 获取用户信息以及获取课程信息

c) 学生点击加入该课程

d) 系统验证该学生是否已选上该课程

e) 已选上该课程的学生可跳到队伍信息界面

f) 学生选择一支队伍并加入

ii.后备事件流

A1.用户不是学生

- ✧ 系统跳转到教师页面

A2.学生未选上该课程

- ✧ 系统显示该学生未选上该课程

五、模块划分

根据实际需求，我们将小程序分为了三个模块：

（1） 用户模块：User (抽象类) 、Administrator (管理员用户) 、Student（学生用户）、Teacher（教师用户）

（2） 课程模块：Course（课程）

（3） 队伍模块：Team（队伍）

模块划分图如下：

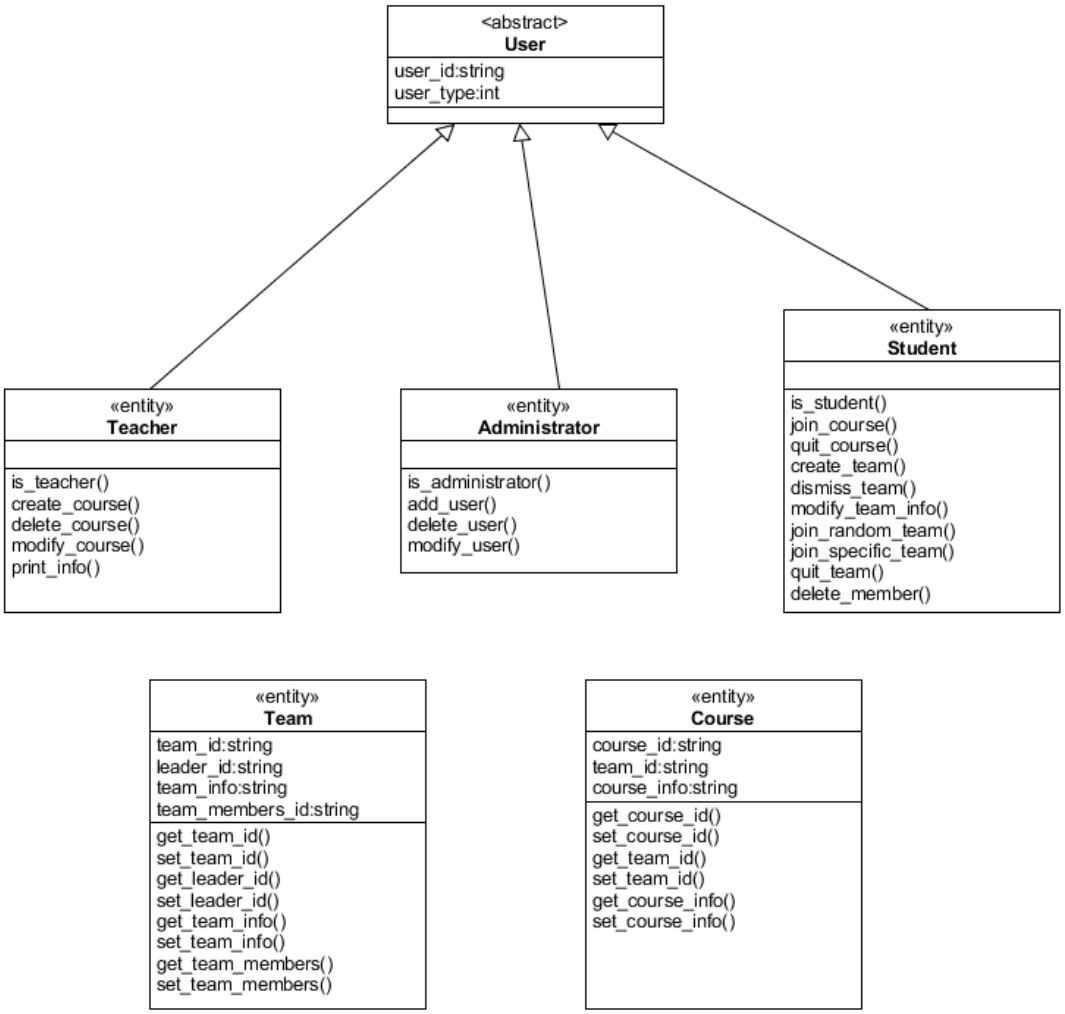

<abstract>
User
user_id:string
user_type:int
«entity»
Teacher
is_teacher()
create_course()
delete_course()
modify_course()
print_info()
«entity»
Administrator
is_administrator()
add_user()
delete_user()
modify_user()
«entity»
Student
is_student()
join_course()
quit_course()
create_team()
dismiss_team()
modify_team_info()
join_random_team()
join_specific_team()
quit_team()
delete_member()
«entity»
Team
team_id:string
leader_id:string
team_info:string
team_members_id:string
get_team_id()
set_team_id()
get_leader_id()
set_leader_id()
get_team_info()
set_team_info()
get_team_members()
set_team_members()
«entity»
Course
course_id:string
team_id:string
course_info:string
get_course_id()
set_course_id()
get_team_id()
set_team_id()
get_course_info()
set_course_info()

图 8 模块划分图

根据模块划分的情况，并结合模块之间的关系，得到领域模型图如下：

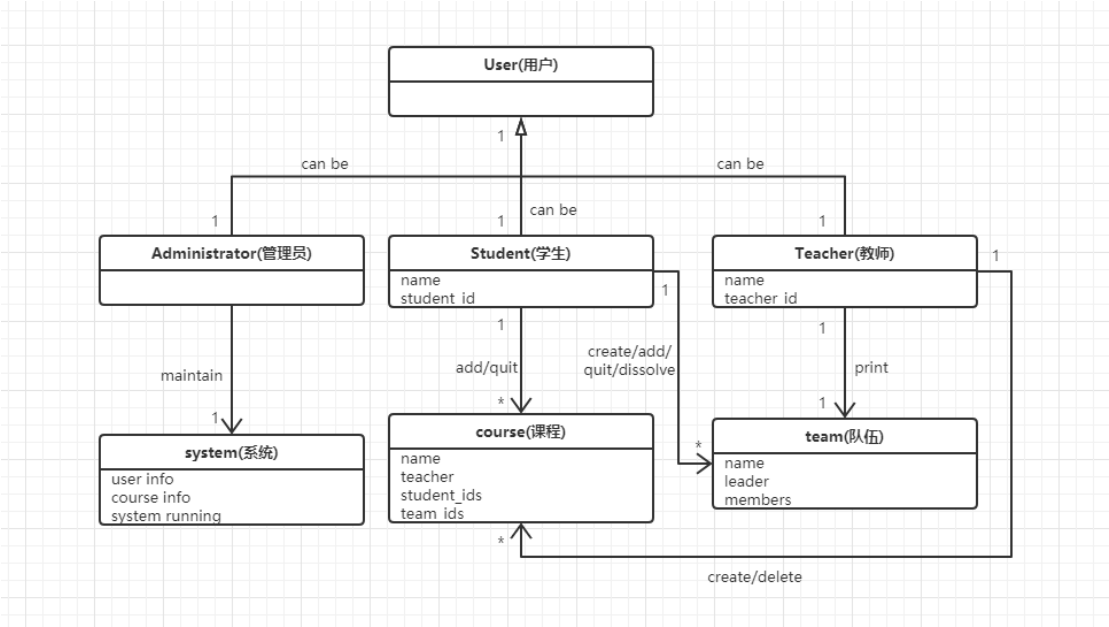


图 9 领域模型图

六、软件设计技术

项目中所使用的软件设计技术最主要的是 Object-Oriented Programming 即面向对象编程。我们所熟知的面向对象编程有三大基本的特征：封装，继承和多态。

（1） 封装：封装可以使类具有独立性和隔离性；保证类的高内聚。只暴露给类的外部或者子类必须的属性和操作。类的封装的实现依赖类的修饰符，在本项目中主要是对每个 page 进行封装，page 内的 data 封装在这个页面使用的需要的变量。并在 Page 中封装可以在该 Page 进行操作的函数，比如点击操作，刷新页面操作等等。

```
data: {
    hidden_modal: true,
    newteam_team_info: '',
    newteam_max_num: '',
    passed_course_id: '',
    student_info: {
        student_id: '',
        username: '',
        attended_course_ids: ''
    },
    course_info: {
        teacher_id: '',
        course_info: '',
        teacher_name: '',
        course_id: '',
        name: '',
        course_time: '',
        start_time: '',
        end_time: '',
        max_team: '',
        min_team: '',
        team_ids: '',
        student_ids: ''
    },
    my_team: {
        team_id: '',
        team_leader: '',
        team_info: '',
        team_members_id: '',
        team_members: []
    },
    team_list: []
},
```

图 10 student_course.js 部分代码片段

图 10 是在 student_course.js 的页面对 data 进行封装，可以看到内部的变量又可以进一步封装为了一个小的结构体，如 student_info 包含了学生学号，用户名以及参加的课程 id 集合。再如 course_info 则封装了这个课程的必要属性，例如教师的 id，课程的信息，课程 id，课程名称等等。

```
// 创建队伍点击事件
create_team: function () {
    var that = this
    // 如果创建队伍的时间不在start_time和end_time之间则不能创建队伍
    var time = util.formatTime(new Date())
    var date_list = time.split(' ')[0].split('/')
    var date_time = date_list[0] + '-' + date_list[1] + '-' + date_list[2]

    // 如果自身已经加入了队伍，则不可以创建队伍
    wx.request({
        url: 'http://jihanyang.cn:8080/get_course',
        data: {
            course_id: that.data.course_info.course_id
        },
        method: 'GET',
        header: {
            'content-type': 'application/json'
        },
        success: function (res) {
            console.log(res.data)
```

图 11 student_course.js 部分代码片段

图 11 呈现的是 student_course.js 内部的一个函数，也是在这个页面内部封装的一个函数，主要是实现对页面点击“创建页面”按钮所触发的事件。当点击事件触发时，该函数则会被立即执行，这也是属于面向对象编程的类内部函数的封装。其它所有的页面都有用到跟 student_course.js 页面相同的对 Page 内的数据以及操作进行封装，所以不需要分别列举。

另外，在后端每个数据库的表都很自然的被封装成一个类，继承自 Flask 框架中 SQLAlchemy 插件的数据库类：

```python
class User(db.Model):
    """用户表"""
    __tablename__ = 'users'
    user_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    student_id = db.Column(db.String(20), unique=True, nullable=False)
    is_teacher = db.Column(db.Boolean, nullable=False)
    # 存 dict 形如 id1@id2
    attended_course_ids = db.Column(db.Text, nullable=False)
    username = db.Column(db.String(20), nullable=False)
    profile_photo = db.Column(db.Text)

    def __init__(self, student_id, name, is_teacher, profile_photo, attended_course_ids):
        self.student_id = student_id
        self.username = name
        self.is_teacher = is_teacher
        self.profile_photo = profile_photo
        self.attended_course_ids = attended_course_ids

    def add_user(self):
        """如果成功，那么就返回success，如果失败，返回失败原因"""
        if User.query.filter(self.student_id == User.student_id).first() is None:
            db.session.add(self)
            db.session.commit()
            return '%s' % json.dumps(self.__json__()), 200
        else:
            return 'Already have this student_id', 400

    def get_course_ids(self):
        """以list的形式返回team_members_id"""
        return self.attended_course_ids.split('@')

    def __json__(self):
        info = {
            "user_id": self.user_id,
            "student_id": self.student_id,
            "username": self.username,
            "is_teacher": self.is_teacher,
            "profile_photo": str(self.profile_photo),
            "attended_course_ids": str(self.attended_course_ids)
        }
```

```python
class Team(db.Model):
    """队伍表"""
    __tablename__ = 'teams'
    team_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    course_id = db.Column(db.Integer)
    # 学号
    leader_id = db.Column(db.Integer, nullable=False)
    team_info = db.Column(db.String(100), nullable=False)
    # list
    team_members_id = db.Column(db.Text, nullable=False)
    max_team = db.Column(db.Integer, nullable=False)
    available_team = db.Column(db.Integer, nullable=False)

    def __init__(self, course_id, leader_id, team_info, max_team,
                 available_team, team_members_id):
        self.course_id = course_id
        self.leader_id = leader_id
        self.team_info = team_info
        self.max_team = max_team
        self.available_team = available_team
        self.team_members_id = team_members_id

    def __json__(self):
        """以JSON的形式返回当前用户的数据"""
        info = {
            "team_id": self.team_id,
            "course_id": self.course_id,
            "leader_id": self.leader_id,
            "team_info": self.team_info,
            "team_members_id": self.team_members_id,
            "max_team": self.max_team,
            "available_team": self.available_team
        }
        return info

    def get_members_id(self):
        """以list的形式返回team_members_id"""
        return str(self.team_members_id).split('@')

    def delete_team(self, course):
        """删除指定的队伍"""
        # 得到student_ids
        student_ids_dict = eval(course.student_ids)
        team_member = self.get_members_id()
        # 将组内每个成员的组队信息更改
        for member in team_member:
            student_ids_dict[member] = 0
        # 将course的team_ids进行更改
        new_team_ids = course.get_team_ids()
        new_team_ids.remove(str(self.team_id))
        if new_team_ids is None:
```

```python
class Course(db.Model):
    """课程表"""
    __tablename__ = 'course'
    course_id = db.Column(db.Integer, primary_key=True, autoincrement=Tr
    teacher_id = db.Column(db.Integer)
    team_ids = db.Column(db.Text, nullable=False)
    student_ids = db.Column(db.Text, nullable=False)
    course_info = db.Column(db.String(200), nullable=False)
    name = db.Column(db.String(100), nullable=False)
    course_time = db.Column(db.String(100), nullable=False)
    start_time = db.Column(db.String(100), nullable=False)
    end_time = db.Column(db.String(100), nullable=False)
    max_team = db.Column(db.Integer, nullable=False)
    min_team = db.Column(db.Integer, nullable=False)

    def __init__(self, teacher_id, course_info, name, course_time, start_
                 end_time, max_team, min_team, student_ids, team_ids):
        self.teacher_id = teacher_id
        self.course_info = course_info
        self.name = name
        self.course_time = course_time
        self.start_time = start_time
        self.end_time = end_time
        self.max_team = max_team
        self.min_team = min_team
        self.team_ids = team_ids
        self.student_ids = student_ids

    def __json__(self):
        """以json的格式返回课程信息"""
        info = {
            "course_id": self.course_id,
            "teacher_id": self.teacher_id,
            "team_ids": self.team_ids,
            "course_info": self.course_info,
            "name": self.name,
            "course_time": self.course_time,
            "start_time": self.start_time,
            "end_time": self.end_time,
            "max_team": self.max_team,
            "min_team": self.min_team,
            "student_ids": self.student_ids
        }
        return info

    def get_team_ids(self):
        """以list的形式返回team_ids"""
        return str(self.team_ids).split('@')
```

图 12 后端数据库表设计代码片段

Session 也被单独封装起来

```python
class ThirdSessionKey(db.Model):
    """third session key"""
    __tablename__ = 'thirdsessionkeys'
    third_session_key_id = db.Column(db.Integer, primary_key=True, autoincrement=True
    third_session_key = db.Column(db.String(50), nullable=False)
    session_key = db.Column(db.String(50), nullable=False)
    openid = db.Column(db.String(50), nullable=False)

    def __init__(self, third_session_key=None, session_key=None, openid=None):
        self.appid = 'wx3b06cde5635b391d'
        self.secret = '559c9a13441e8e2a8e970860a37170f8'
        #self.appid = 'wx4f4bc4dec97d474b'
        #self.session_key = 'tiihtNczf5v6AKRyjwEUhQ=='
        self.third_session_key = third_session_key
        self.session_key = session_key
        self.openid = openid

    # decrypt encrypted_data and add third_session_key to it
    def get_third_session_key(self, js_code):
        session_key_and_openid = self.jscode2session(js_code)
        self.session_key = session_key_and_openid.get('session_key')
```

图 13 Session 设计代码片段

另外就是，对于每一种请求，都会封装一个函数来返回一个 response

```python
@app.route('/course_modify_student', methods=['POST'])
def course_modify_student():
    """对于课程增删学生"""
    # 从url中取出所需参数
    course_id = request.values.get('course_id')
    student_ids = request.values.get('student_ids')
    course = Course.query.filter(course_id == Course.course_id).first()
    # 确定存在该课程
    if course is None:
        return 'Cannot find such a course', 400
    else:
        course.student_ids = student_ids
        db.session.add(course)
        db.session.commit()
        return '%s' % json.dumps(course.__json__()), 200


@app.route('/modify_team_ids', methods=['POST'])
def modify_team_ids():
    """更改课程中的队伍列表"""
    # 从url中取出所需参数
    course_id = request.values.get('course_id')
    team_ids = request.values.get('team_ids')
    course = Course.query.filter(course_id == Course.course_id).first()
    # 确定存在该课程
    if course is None:
        return 'Cannot find such a course', 400
    else:
        course.team_ids = team_ids
        db.session.add(course)
        db.session.commit()
        return '%s' % json.dumps(course.__json__()), 200
```

图 14 请求响应设计代码片段

（2） 结构与逻辑分离思想

由于小程序的特性，每一个页面都遵循着经典的结构与逻辑分离的思想，即 HTML/CSS/JavaScript 的形式构建页面。但应用到小程序之后就变成了小程序内部的形式 WXML/WXSS/JavaScript 形式。其中 WXML 和 WXSS 与 HTML 和 CSS 相对应。除了标签的名称改变之外，其余的语法和结构都是一样的。实际上 WXML 和 WXSS 也是对 HTML 和 CSS 进行了相应的封装之后使其对开发者更加友好。

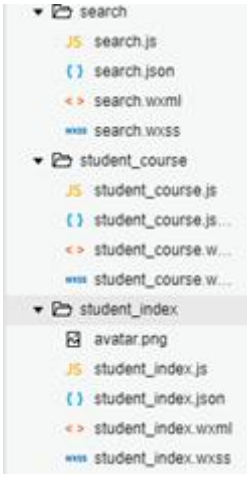


图 15 部分页面内部结构图

图 15 显示了每个页面都是由对应的 wxml,wxss 和 javascript 一起组装构成。

（3）单一职责原则

单一职责原则使得类的复杂性降低，实现什么职责都有清晰明确的定义。使代码的可读性提高，复杂度降低。同时也使可维护性提高。变更引起的风险降低。因为需求变更增加是无法避免的。如果接口的单一职责得到实现，则一个接口的修改只对应相应的类有影响，对其他的接口无影响，这对系统的扩展性都有非常大的帮助。因此在软件开发的过程中，应该保证做到接口的单一职责，且类的设计尽量做到只有一个原因引起变化。

```
16    modify_info: function() {
17      var temp = this.data.course_id;
18      wx.navigateTo({
19        url: "../course_info/course_info?course_id=" + temp,
20      })
21    },
22
23    check_team: function(e) {
24      var temp = e.currentTarget.dataset.id;
25      wx.navigateTo({
26        url: '../teacher_team_info/teacher_team_info?team_id=' + temp,
27      })
28    },
29
30    deleteCourse: function() {
31      var that = this;
32      wx.showModal({
33        title: '警告',
34        content: '确认要删除课程? ',
35        success: function (res) {
36          if (res.confirm) {
37            wx.request({
38              url: 'http://jihanyang.cn:8080/delete_course',
39              method: 'POST',
40              header: {
41                "Content-Type": "application/x-www-form-urlencoded"
42              },
43              data: {
```

图 16 teacher_course.js 部分代码片段

图 16 体现了单一职责的原则，teacher_course.js 内部的 modify_info，check_team 以及 deleteCourse 函数是相互独立的，每个接口都对应着该页面的一个功能。如果需要对该页面进行修改，则对其中一个函数进行修改不需要对其它的函数和接口进行修改。但对于某些页面，这种单一职责原则是因页面和环境而变的，如 student_course 则无可避免因更改需求而修改多个相应的函数接口。

七、测试说明以及相关测试用例

为了提升小程序的可维护性，我们对小程序的部分功能进行了测试。我们主要采用了黑盒测试中的等价类测试法，设计了相关的测试用例进行分析。

5.1 用户注册/登录

这一过程中，我们需要测试小程序能否正常实现用户的登录功能。

输入数据	有效等价类	无效等价类
用户名	① 长度大于 0 小于等于 20 的字符串	② 空字符串 ③ 长度大于 20 的字符串
学工号	④ 长度大于 0 小于等于 20 的数字字符串	⑤ 空字符串 ⑥ 非数字字符串 ⑦ 长度大于 20 的字符串
密码	⑧ 学工号+010	⑨ 空字符串 ⑩ 不是学工号+010

覆盖有效等价类的测试用例：

用户名	学工号	密码	覆盖等价类
WeTeam	15331000	15331000010	①、④、⑧
中山大学	12345678	12345678010	①、④、⑧

覆盖无效等价类的测试用例：

用户名	学工号	密码	覆盖等价类
	15331001	15331001010	②
abcdefghijklmnopqrstuvwxyz	15331002	15331002010	③
Test1		010	⑤
Test1	isNaN	isNaN010	⑥
Test1	1234567890 123456789000	1234567890 123456789000010	⑦
Test1	15331003		⑨
Test1	15331003	15331003	⑩
Test1	15331003	15331003000	⑩
Test1	15331003	15331000010	⑪

5.2 创建队伍

这部分要求输入数据队伍信息、队伍人数，其中队伍人数的限制是根据教师创建课程时要求的队伍人数来的，即假设教师创建课程时队伍人数设置为 3-7 人，则创建队伍时队伍人数应该在这个区间内。另外，还必须保证当前处于组队时间范围内，这个组队时间范围同样由教师创建课程时设置，为组队开始日期到组队

截止日期。

输入数据	描述	测试用例说明	测试数据	期望结果	选取理由
队伍信息	字符串（不为空)	① 空字符串 ② 非空字符串	① “” ② 大佬带队	① 输入无效 ② 输入有效	① 空字符串 ② 类型与长度均符合
队伍人数	整数 (在课程要求的组队人数区间中)	③ 少于要求人数 ④ 多于要求人数	③ 2 ④ 8 ⑤ 5	③ 输入无效 ④ 输入无效 ⑤ 输入有效	③ 小于 3 ④ 大于 7 ⑤ 在 [3,7] 区间内

5.3 创建课程

教师创建课程时，需要输入课程名字、上课时间、组队时间、组队人数限制、课程组队说明，其中上课时间可以在一周的任意一天上课（无论周六周日），起始节数必须小于终止节数（即每次课不得少于 2 节），组队开始时间必须小于组队截止时间（即必须保证有至少一天的组队时间）。

输入数据	描述	测试用例说明	测试数据	期望结果
课程名字	非空字符串	① 空字符串	“”	输入无效

			② 非空字符串	系统分析与设计	输入有效
上课时间	星期几	从滚动选择器控件选择（一、二、三、四、五、六、日）	① 为空 ② 星期几随便设置都行	“” 星期一、二、…、六、日	输入无效 输入有效
	起始节数	从滚动选择器控件选择（1到11）	① 为空 ② 第1节 ③ 第11节 ④ 其他节	“” 1 11 2-10	输入无效 输入有效（判断） 输入无效 输入有效（判断）
	终止节数	从滚动选择器控件选择（1到11），必须比起始节数大	① 为空 ② 第1节 ③ 第11节 ④ 其他节	“” 1 11 2-10	输入无效 输入无效 输入有效（判断） 输入有效（判断）
组队时间	开始时间	从滚动选择器控件选择	① 为空 ② 之前日期 ③ 当前日期	“” 2018-06-10 2018-06-12	输入无效 输入有效 输入有效

			④ 往后日期	2018-06-15	输入有效
	截止时间	从滚动选择器控件选择，必须在开始时间后面	① 为空 ② 之前日期 ③ 当前日期 ④ 往后日期	“” 2018-06-11 2018-06-13 2018-06-16	输入无效 输入有效（判断） 输入有效（判断） 输入有效（判断）
组队人数限制	最小人数	正整数	① 为空 ② 非正整数 ③ 字符串 ④ 正整数	“” -5、1.8、0 Aaa 3	输入无效 输入无效 输入无效 输入有效
	最大人数	正整数	① 为空 ② 非正整数 ③ 字符串 ④ 正整数	“” -8、3.5、0 Bbb 7	输入无效 输入无效 输入无效 输入有效
课程组队说明		字符串 (长度不能超过200)	① 为空 ② 字符串长度超过200 ③ 字符串长度在200以内	“” aaaa....aaaa（超过200个) 必修课	输入无效 输入无效 输入有效

5.4 其他用例的测试案例

由于其他测试案例不包含数据的输入，测试案例设计难以用文档表示，测试时更多的是界面交互、后台数据库的操作，因此这部分留做展示。