第 6 章选课学习 v3.1

1 模块需求分析

1.1 模块介绍

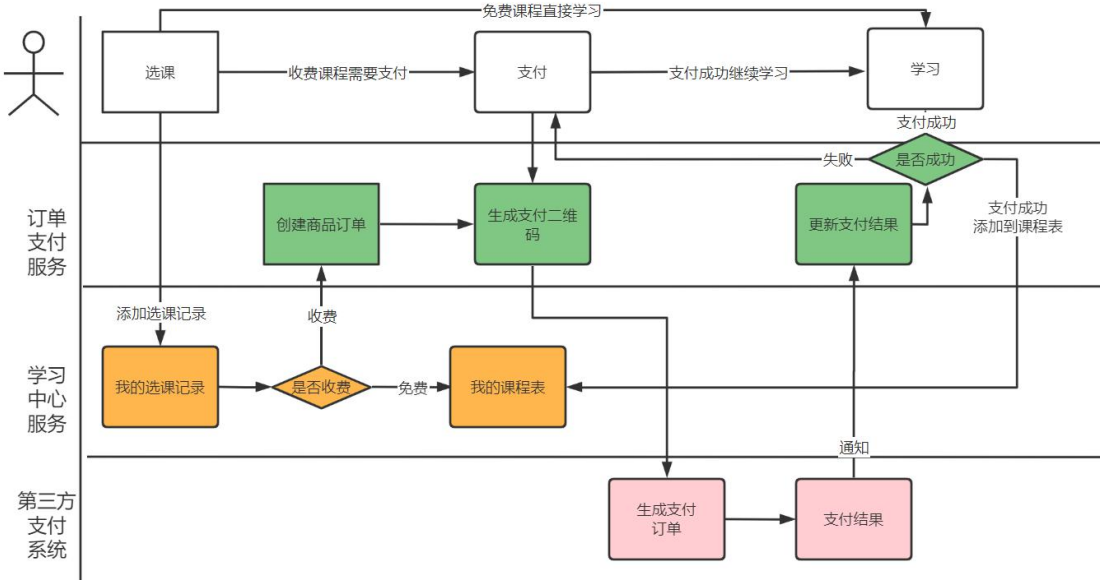
本模块实现了学生选课、下单支付、学习的整体流程。

网站的课程有免费和收费两种，对于免费课程学生选课后可直接学习，对于收费课程学生需要下单且支付成功方可选课、学习。

选课：是将课程加入我的课程表的过程。

我的课程表：记录我在网站学习的课程，我的课程表中有免费课程和收费课程两种，对于免费课程可直接添加到我的课程表，对于收费课程需要下单、支付成功后自动加入我的课程表。

模块整体流程如下：



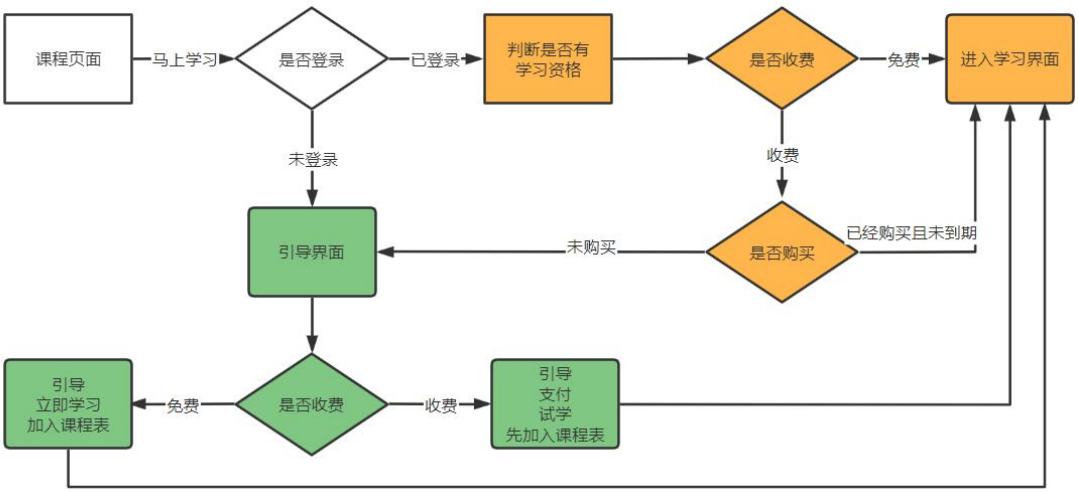
1.2 业务流程

1.2.1 学习引导

用户通过搜索课程、课程推荐等信息进入课程详情页面，点击“马上学习”引导进入学

习界面去学习。

流程如下:



1、进入课程详情点击马上学习

2、课程免费时引导加入我的课程表、或进入学习界面。

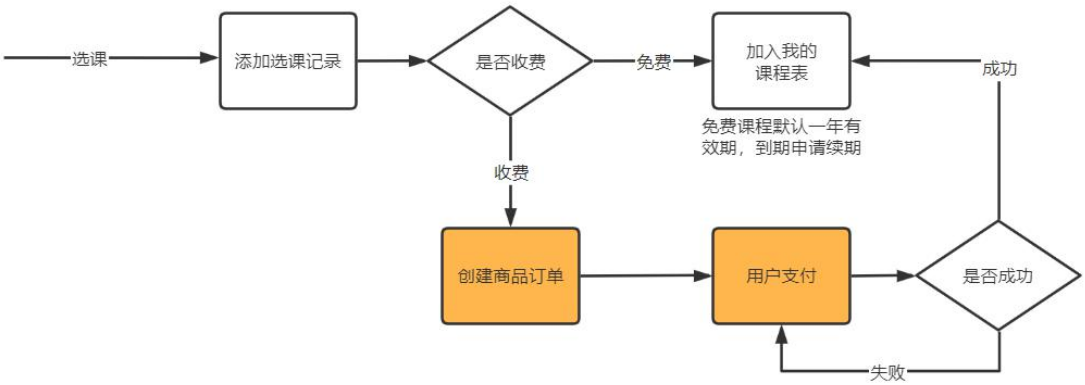
3、课程收费时引导去支付、或试学。

1.2.2 选课流程

选课是将课程加入我的课程表的过程。

对免费课程选课后可直接加入我的课程表，对收费课程选课后需要下单支付成功系统自动加入我的课程表。

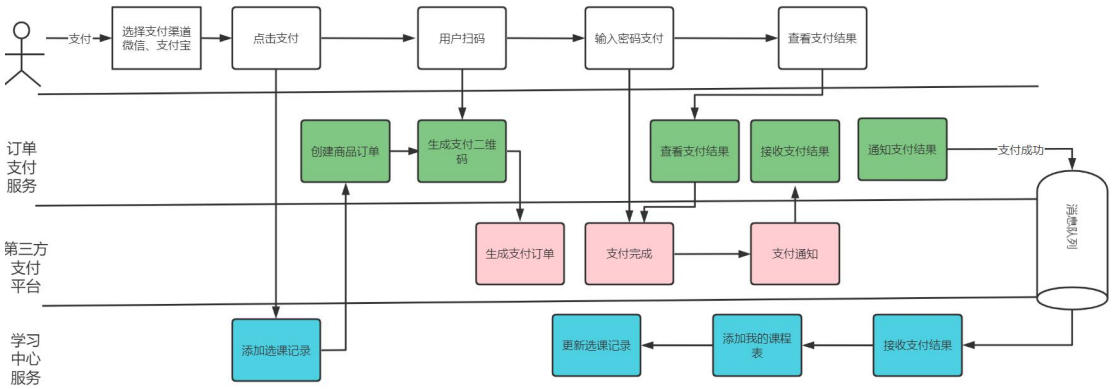
流程如下:



1.2.3 支付流程

本项目与第三方支付平台对接完成支付操作。

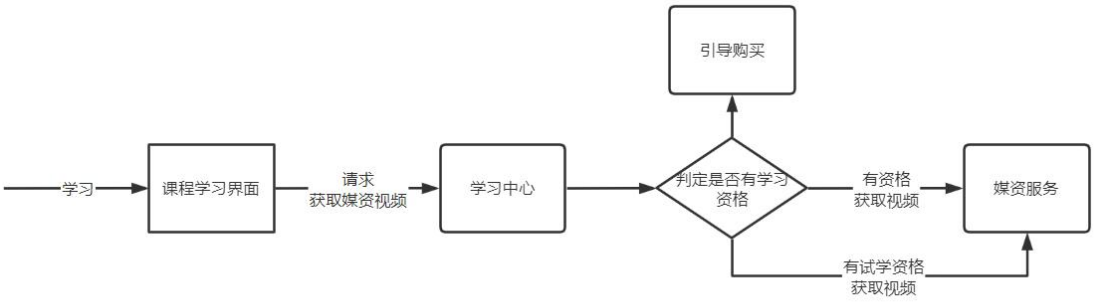
流程如下:



1.2.4 在线学习

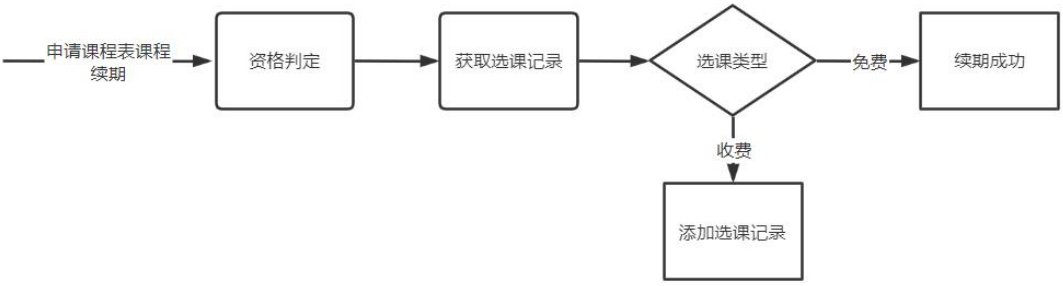
选课成功用户可以在线学习，对于免费课程无需选课即可在线学习。

流程如下:



1.2.5 免费课程续期

免费课程加入我的课程表默认为 1 年有效期，到期用户可申请续期，流程如下:

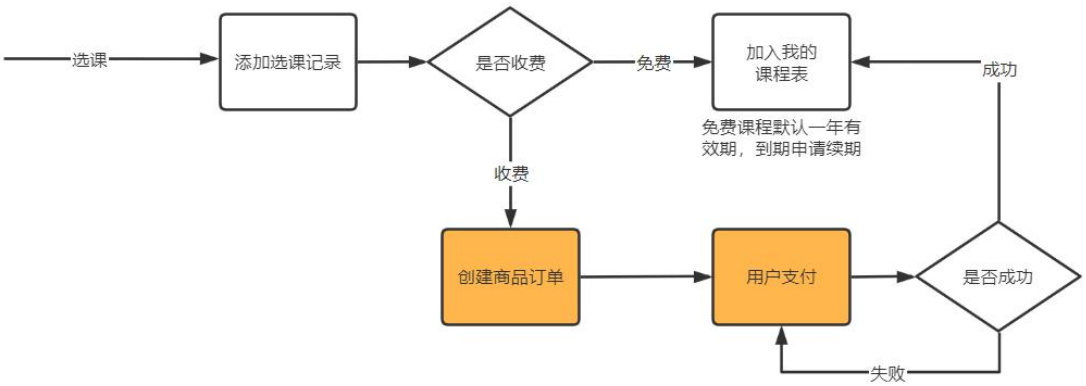


2 添加选课

2.1 需求分析

2.1.1 数据模型

选课是将课程加入我的课程表的过程，根据选课的业务流程进行详细分析，业务流程如下:



选课信息存入选课记录表，免费课程被选课除了进入选课记录表同时进入我的课程表，收费课程进入选课记录表后需要经过下单、支付成功才可以进入我的课程表。

我的课程表记录了用户学习的课程，包括免费课程、收费课程（已经支付）。

1、选课记录表

当用户将课程添加到课程表时需要先创建选课记录。

结构如下：

名	类型	长度	小数点	不是 null	虚拟	键	注释
id	bigint			☑	☐	🔑 1	主键
course_id	bigint			☑	☐		课程id
course_name	varchar	32		☑	☐		课程名称
user_id	varchar	32		☑	☐		用户id
company_id	bigint			☑	☐		机构id
order_type	varchar	32		☑	☐		选课类型
create_date	datetime			☑	☐		添加时间
course_price	float	10	2	☑	☐		课程价格
valid_days	int			☑	☐		课程有效期(天)
status	varchar	32		☑	☐		选课状态
validtime_start	datetime			☑	☐		开始服务时间
validtime_end	datetime			☑	☐		结束服务时间
remarks	varchar	255		☐	☐		备注

选课类型：免费课程、收费课程。

选课状态：选课成功、待支付、选课删除。

对于免费课程：课程价格为 0，有效期默认 365，开始服务时间为选课时间，结束服务时间为选课时间加 1 年后的时间，选课状态为选课成功。

对于收费课程：按课程的现价、有效期确定开始服务时间、结束服务时间，选课状态为待支付。

收费课程的选课记录需要支付成功后选课状态为成功。

2、我的课程表

我的课程表中记录了用户选课成功的课程，所以我的课程表的数据来源于选课记录表。

对于免费课程创建选课记录后同时向我的课程表添加记录。

对于收费课程创建选课记录后需要下单支付成功后自动向我的课程表添加记录。

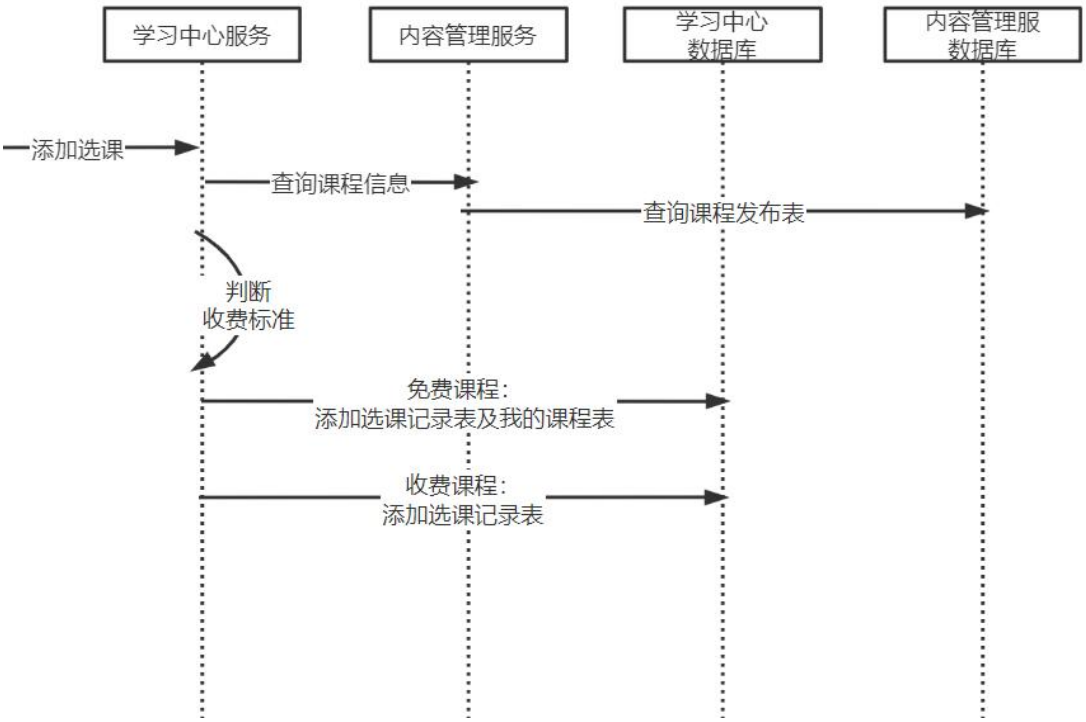
名	类型	长度	小数点	不是 null	虚拟	键	注释
id	bigint			☑	☐	🔑 1	
choose_course_id	bigint			☑	☐		选课订单id
user_id	varchar	64		☑	☐		用户id
course_id	bigint			☑	☐		课程id
company_id	bigint			☑	☐		机构id
course_name	varchar	64		☑	☐		课程名称
create_date	datetime			☑	☐		添加时间
validtime_start	datetime			☐	☐		开始服务时间
validtime_end	datetime			☑	☐		到期时间
update_date	datetime			☐	☐		更新时间
remarks	varchar	255		☐	☐		备注

2.1.2 执行流程

在学习引导处，可以直接将免费课程加入我的课程表，如下图：

对于收费课程先创建选课记录表，支付成功后，收到支付结果由系统自动加入我的课程表。

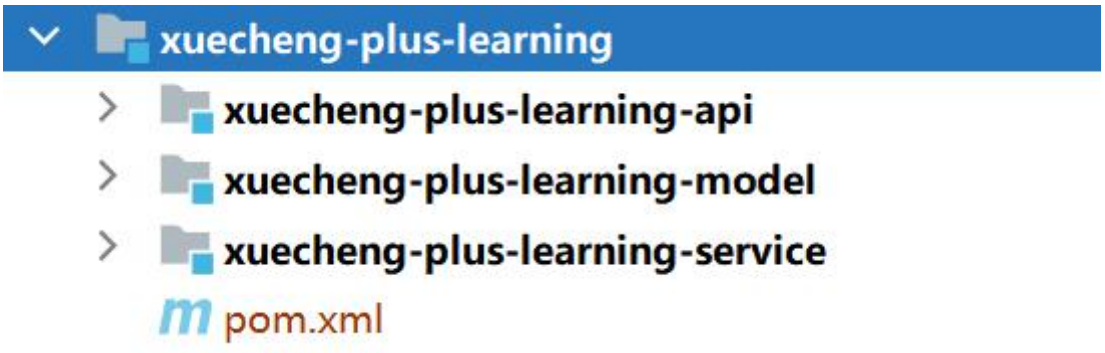
执行流程如下：



2.2 接口开发

2.2.1 部署学习中心工程

从课程资料拷贝学习中心服务工程到自己的工程目录，结构如下：



注意去修改 nacos 的命名空间。

创建数据库 xc_learning，并导入数据

修改数据库的连接，改成自己的数据库。

nacos 配置文件：learning-api-dev.yaml

```yaml
server:
```

```
  servlet:
    context-path: /learning
  port: 63020
```

learning-service-dev.yaml

```
YAML
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://192.168.101.65:3306/xc1010_learning?serverTimezone=UT
C&userUnicode=true&useSSL=false&
    username: root
    password: mysql
```

2.2.2 添加查询课程接口

内容管理服务提供查询课程信息接口，此接口从课程发布表查询。

此接口主要提供其它微服务远程调用，所以此接口不用授权，本项目标记此类接口统一以 /r 开头。

在课程发布 controller 类中定义课程发布信息查询接口。

```
Java
@ApiOperation("查询课程发布信息")
@ResponseBody
@GetMapping("/r/coursepublish/{courseId}")
public CoursePublish getCoursepublish(@PathVariable("courseId")
Long courseId) {
    CoursePublish coursePublish =
coursePublishService.getCoursePublish(courseId);
     return coursePublish;
}
```

Service 如下：

如果课程发布状态正常则正常返回，否则返回空。

```
Java
```

```
public CoursePublish getCoursePublish(Long courseId){
    CoursePublish coursePublish =
coursePublishMapper.selectById(courseId);
    return coursePublish ;
}
```

测试：

启动内容管理服务，使用 httpclient 测试

```
Plain Text
### 查询课程发布信息
GET {{content_host}}/content/r/coursepublish/2
```

由于是在网关处进行令牌校验，所以在微服务处不再校验令牌的合法性，修改内容管理 content-api 工程的 ResouceServerConfig 类，屏蔽 authenticated()。

```
Java
 @Override
 public void configure(HttpSecurity http) throws Exception {
  http.csrf().disable()
          .authorizeRequests()
//          .antMatchers("/r/**","/course/**").authenticated()//所有/r/**的请求必须认证通过
          .anyRequest().permitAll()
  ;
 }
```

2.2.3 测试查询课程信息接口

学生中心服务远程调用内容管理服务的查询课程发布信息接口。

导入的学习中心工程已经存在 ContentServiceClient 接口，通过此接口远程调用课程查询接口。

编写测试接口

```
Java
package com.xuecheng.learning;

import com.xuecheng.content.model.po.CoursePublish;
import com.xuecheng.learning.feignclient.ContentServiceClient;
import org.junit.jupiter.api.Assertions;
```

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

/**
 * @description Feign 接口测试类
 * @author Mr.M
 * @date 2022/10/24 17:15
 * @version 1.0
 */
 @SpringBootTest
public class FeignClientTest {

  @Autowired
 ContentServiceClient contentServiceClient;

  @Test
  public void testContentServiceClient(){
   CoursePublish coursepublish =
contentServiceClient.getCoursepublish(18L);
   Assertions.assertNotNull(coursepublish);
  }
}
```

在进行 feign 远程调用时会将字符串转成 LocalDateTime，在 CoursePublish 类中 LocalDateTime 的属性上边添加如下代码:

```
Java
@JsonFormat(shape = JsonFormat.Shape.STRING,pattern = "yyyy-MM-dd
HH:mm:ss")
```

2.2.5 添加选课接口

2.2.5.1 接口分析

本接口支持免费课程选课、收费课程选课。

免费课程选课：添加选课记录、添加我的课程表。

收费课程选课：添加选课记录。

2.2.5.2 接口定义

1、请求参数：课程 id、当前用户 id

2、响应结果：选课记录信息、学习资格

学习资格：[{"code":"702001","desc":"正常学习"},{"code":"702002","desc":"没有选课或选课后没有支付"},{"code":"702003","desc":"已过期需要申请续期或重新支付"}]

接口定义如下：

```
Java
package com.xuecheng.learning.api;

import com.xuecheng.base.execption.XueChengPlusException;
import com.xuecheng.base.model.RestResponse;
import com.xuecheng.base.model.XcUser;
import com.xuecheng.learning.model.dto.XcChooseCourseDto;
import com.xuecheng.learning.service.MyCourseTablesService;
import com.xuecheng.learning.util.SecurityUtil;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @author Mr.M
 * @version 1.0
 * @description 我的课程表接口
 * @date 2022/10/2 14:52
 */
@Api(value = "我的课程表接口", tags = "我的课程表接口")
@Slf4j
@RestController
public class MyCourseTablesController {


    @ApiOperation("添加选课")
     @PostMapping("/choosecourse/{courseId}")
    public XcChooseCourseDto
addChooseCourse(@PathVariable("courseId") Long courseId)  {
```

```
    }

}
```

Service 接口定义:

```
Java
package com.xuecheng.learning.service;

import com.xuecheng.learning.model.dto.XcChooseCourseDto;
import com.xuecheng.learning.model.po.XcChooseCourse;
import com.xuecheng.learning.model.po.XcCourseTables;

/**
 * @description 我的课程表 service 接口
 * @author Mr.M
 * @date 2022/10/2 16:07
 * @version 1.0
 */
public interface MyCourseTablesService {

    /**
 * @description 添加选课
 * @param userId 用户 id
 * @param courseId 课程 id
 * @return com.xuecheng.learning.model.dto.XcChooseCourseDto
 * @author Mr.M
 * @date 2022/10/24 17:33
*/
 public XcChooseCourseDto addChooseCourse(String userId, Long
courseId);

}
```

Service 接口执行流程

```
Java
package com.xuecheng.learning.service.impl;

import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.xuecheng.base.execption.XueChengPlusException;
import com.xuecheng.content.model.po.CoursePublish;
import com.xuecheng.learning.feignclient.ContentServiceClient;
```

```
import com.xuecheng.learning.mapper.XcChooseCourseMapper;
import com.xuecheng.learning.mapper.XcCourseTablesMapper;
import com.xuecheng.learning.model.dto.XcChooseCourseDto;
import com.xuecheng.learning.model.po.XcChooseCourse;
import com.xuecheng.learning.model.po.XcCourseTables;
import com.xuecheng.learning.service.MyCourseTablesService;
import lombok.extern.slf4j.Slf4j;
import org.apache.ibatis.executor.statement.StatementUtil;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;

/**
 * @author Mr.M
 * @version 1.0
 * @description TODO
 * @date 2022/10/2 16:12
 */
@Slf4j
@Service
public class MyCourseTablesServiceImpl implements
MyCourseTablesService {

    @Autowired
    XcChooseCourseMapper xcChooseCourseMapper;

    @Autowired
    XcCourseTablesMapper xcCourseTablesMapper;

    @Autowired
    ContentServiceClient contentServiceClient;

    @Autowired
    MyCourseTablesService myCourseTablesService;

    @Autowired
    MyCourseTablesServiceImpl currentProxy;

    @Transactional
    @Override
    public XcChooseCourseDto addChooseCourse(String userId,Long
courseId) {
```

```
        //查询课程信息
        CoursePublish coursepublish =
contentServiceClient.getCoursepublish(courseId);
        //课程收费标准
        String charge = coursepublish.getCharge();
        //选课记录
        XcChooseCourse chooseCourse = null;
        if("201000".equals(charge)){//课程免费
            //添加免费课程
            chooseCourse  = addFreeCoruse(userId, coursepublish);
            //添加到我的课程表
            XcCourseTables xcCourseTables =
addCourseTabls(chooseCourse);
        }else{
            //添加收费课程
            chooseCourse  = addChargeCoruse(userId, coursepublish);
        }
        //获取学习资格
        ...
        return null;
    }



//添加免费课程,免费课程加入选课记录表、我的课程表
public XcChooseCourse addFreeCoruse(String userId, CoursePublish
coursepublish) {

        return null;
}

//添加收费课程
public XcChooseCourse addChargeCoruse(String userId,CoursePublish
coursepublish){

    return null;
}
//添加到我的课程表
public XcCourseTables addCourseTabls(XcChooseCourse
xcChooseCourse){
   return null;
}
```

2.2.5.3 添加免费课程

```
Java

//添加免费课程,免费课程加入选课记录表、我的课程表
public XcChooseCourse addFreeCoruse(String userId, CoursePublish
coursepublish) {
    //查询选课记录表是否存在免费的且选课成功的订单
    LambdaQueryWrapper<XcChooseCourse> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper = queryWrapper.eq(XcChooseCourse::getUserId,
userId)
            .eq(XcChooseCourse::getCourseId, coursepublish.getId())
            .eq(XcChooseCourse::getOrderType, "700001")//免费课程
            .eq(XcChooseCourse::getStatus, "701001");//选课成功
    List<XcChooseCourse> xcChooseCourses =
xcChooseCourseMapper.selectList(queryWrapper);
    if (xcChooseCourses != null && xcChooseCourses.size()>0) {
        return xcChooseCourses.get(0);
    }
    //添加选课记录信息
    XcChooseCourse xcChooseCourse = new XcChooseCourse();
    xcChooseCourse.setCourseId(coursepublish.getId());
    xcChooseCourse.setCourseName(coursepublish.getName());
    xcChooseCourse.setCoursePrice(0f);//免费课程价格为 0
    xcChooseCourse.setUserId(userId);
    xcChooseCourse.setCompanyId(coursepublish.getCompanyId());
    xcChooseCourse.setOrderType("700001");//免费课程
    xcChooseCourse.setCreateDate(LocalDateTime.now());
    xcChooseCourse.setStatus("701001");//选课成功

    xcChooseCourse.setValidDays(365);//免费课程默认 365
    xcChooseCourse.setValidtimeStart(LocalDateTime.now());

xcChooseCourse.setValidtimeEnd(LocalDateTime.now().plusDays(365));
    xcChooseCourseMapper.insert(xcChooseCourse);

    return xcChooseCourse;

}
```

2.2.5.4 添加我的课程表

我的课程表的记录来源于选课记录，选课记录成功将课程信息添加到我的课程表。

如果我的课程表已存在课程可能已经过期，如果有新的选课记录则需要更新我的课程表中的现有信息。

```
Java
/**
 * @description 添加到我的课程表
 * @param xcChooseCourse 选课记录
 * @return com.xuecheng.learning.model.po.XcCourseTables
 * @author Mr.M
 * @date 2022/10/3 11:24
*/
public XcCourseTables addCourseTabls(XcChooseCourse
xcChooseCourse){
    //选课记录完成且未过期可以添加课程到课程表
    String status = xcChooseCourse.getStatus();
    if (!"701001".equals(status)){
        XueChengPlusException.cast("选课未成功，无法添加到课程表");
    }
    //查询我的课程表
    XcCourseTables xcCourseTables =
getXcCourseTables(xcChooseCourse.getUserId(),
xcChooseCourse.getCourseId());
    if(xcCourseTables!=null){
         return xcCourseTables;
    }
    XcCourseTables xcCourseTablesNew = new XcCourseTables();
    xcCourseTablesNew.setChooseCourseId(xcChooseCourse.getId());
    xcCourseTablesNew.setUserId(xcChooseCourse.getUserId());
    xcCourseTablesNew.setCourseId(xcChooseCourse.getCourseId());
    xcCourseTablesNew.setCompanyId(xcChooseCourse.getCompanyId());

xcCourseTablesNew.setCourseName(xcChooseCourse.getCourseName());
    xcCourseTablesNew.setCreateDate(LocalDateTime.now());

xcCourseTablesNew.setValidtimeStart(xcChooseCourse.getValidtimeSta
rt());

xcCourseTablesNew.setValidtimeEnd(xcChooseCourse.getValidtimeEnd()
);
```

```
xcCourseTablesNew.setCourseType(xcChooseCourse.getOrderType());
    xcCourseTablesMapper.insert(xcCourseTablesNew);

    return xcCourseTablesNew;

}

/**
 * @description 根据课程和用户查询我的课程表中某一门课程
 * @param userId
 * @param courseId
 * @return com.xuecheng.learning.model.po.XcCourseTables
 * @author Mr.M
 * @date 2022/10/2 17:07
*/
public XcCourseTables getXcCourseTables(String userId,Long
courseId){
    XcCourseTables xcCourseTables =
xcCourseTablesMapper.selectOne(new
LambdaQueryWrapper<XcCourseTables>().eq(XcCourseTables::getUserId,
userId).eq(XcCourseTables::getCourseId, courseId));
    return xcCourseTables;

}
```

2.2.5.5 添加收费课程

```
Java

//添加收费课程
public XcChooseCourse addChargeCoruse(String userId,CoursePublish
coursepublish){

    //如果存在待支付交易记录直接返回
    LambdaQueryWrapper<XcChooseCourse> queryWrapper = new
LambdaQueryWrapper<>();
    queryWrapper = queryWrapper.eq(XcChooseCourse::getUserId,
userId)
            .eq(XcChooseCourse::getCourseId, coursepublish.getId())
            .eq(XcChooseCourse::getOrderType, "700002")//收费订单
            .eq(XcChooseCourse::getStatus, "701002");//待支付
    List<XcChooseCourse> xcChooseCourses =
```

```
xcChooseCourseMapper.selectList(queryWrapper);
    if (xcChooseCourses != null && xcChooseCourses.size()>0) {
        return xcChooseCourses.get(0);
    }

    XcChooseCourse xcChooseCourse = new XcChooseCourse();
    xcChooseCourse.setCourseId(coursepublish.getId());
    xcChooseCourse.setCourseName(coursepublish.getName());
    xcChooseCourse.setCoursePrice(coursepublish.getPrice());
    xcChooseCourse.setUserId(userId);
    xcChooseCourse.setCompanyId(coursepublish.getCompanyId());
    xcChooseCourse.setOrderType("700002");//收费课程
    xcChooseCourse.setCreateDate(LocalDateTime.now());
    xcChooseCourse.setStatus("701002");//待支付

    xcChooseCourse.setValidDays(coursepublish.getValidDays());
    xcChooseCourse.setValidtimeStart(LocalDateTime.now());

xcChooseCourse.setValidtimeEnd(LocalDateTime.now().plusDays(course
publish.getValidDays()));
    xcChooseCourseMapper.insert(xcChooseCourse);
    return xcChooseCourse;
}
```

2.2.5.6 获取学习资格

定义获取学习资格接口

```
Java
public interface MyCourseTablesService {

    public XcChooseCourseDto addChooseCourse(String userId, Long
courseId);
    /**
     * @description 判断学习资格
     * @param userId
     * @param courseId
     * @return XcCourseTablesDto 学习资格状态
[{"code":"702001","desc":"正常学习"},{"code":"702002","desc":"没有
选课或选课后没有支付"},{"code":"702003","desc":"已过期需要申请续期或重
新支付"}]
     * @author Mr.M
     * @date 2022/10/3 7:37
```

```
     */
    public XcCourseTablesDto getLearningStatus(String userId, Long
courseId);
}
```

接口实现如下:

```
Java
/**
 * @description 判断学习资格
 * @param userId
 * @param courseId
 * @return XcCourseTablesDto 学习资格状态
[{"code":"702001","desc":"正常学习"},{"code":"702002","desc":"没有
选课或选课后没有支付"},{"code":"702003","desc":"已过期需要申请续期或重
新支付"}]
 * @author Mr.M
 * @date 2022/10/3 7:37
*/
public XcCourseTablesDto getLearningStatus(String userId, Long
courseId){
    //查询我的课程表
    XcCourseTables xcCourseTables = getXcCourseTables(userId,
courseId);
    if(xcCourseTables==null){
        XcCourseTablesDto xcCourseTablesDto = new
XcCourseTablesDto();
        //没有选课或选课后没有支付
        xcCourseTablesDto.setLearnStatus("702002");
        return xcCourseTablesDto;
    }
    XcCourseTablesDto xcCourseTablesDto = new XcCourseTablesDto();
    BeanUtils.copyProperties(xcCourseTables,xcCourseTablesDto);
    //是否过期,true 过期，false 未过期
    boolean isExpires =
xcCourseTables.getValidtimeEnd().isBefore(LocalDateTime.now());
    if(!isExpires){
        //正常学习
        xcCourseTablesDto.setLearnStatus("702001");
       return xcCourseTablesDto;

    }else{
        //已过期
```

```
        xcCourseTablesDto.setLearnStatus("702003");
        return xcCourseTablesDto;
    }

}
```

2.2.5.7 service 接口完善

完善 Service 接口

```
Java
@Transactional
@Override
public XcChooseCourseDto addChooseCourse(String userId, Long
courseId) {
    //查询课程信息
    CoursePublish coursepublish =
contentServiceClient.getCoursepublish(courseId);
    //课程收费标准
    String charge = coursepublish.getCharge();
    //选课记录
    XcChooseCourse chooseCourse = null;
    if ("201000".equals(charge)) {//课程免费
        //添加免费课程
        chooseCourse = addFreeCoruse(userId, coursepublish);
        //添加到我的课程表
        XcCourseTables xcCourseTables =
addCourseTabls(chooseCourse);
    } else {
        //添加收费课程
        chooseCourse = addChargeCoruse(userId, coursepublish);
    }
    XcChooseCourseDto xcChooseCourseDto = new XcChooseCourseDto();
    BeanUtils.copyProperties(chooseCourse,xcChooseCourseDto);
    //获取学习资格
    XcCourseTablesDto xcCourseTablesDto =
getLearningStatus(userId, courseId);

xcChooseCourseDto.setLearnStatus(xcCourseTablesDto.getLearnStatus(
));
    return xcChooseCourseDto;
```

```
}
```

2.2.5.8 完善 controller

```
Java
@Autowired
MyCourseTablesService courseTablesService;

@ApiOperation("添加选课")
@PostMapping("/choosecourse/{courseId}")
public XcChooseCourseDto addChooseCourse(@PathVariable("courseId")
Long courseId) {
    //登录用户
    SecurityUtil.XcUser user = SecurityUtil.getUser();
    if(user == null){
        XueChengPlusException.cast("请登录后继续选课");
    }
    String userId = user.getId();
    return  courseTablesService.addChooseCourse(userId, courseId);

}

@ApiOperation("查询学习资格")
@PostMapping("/choosecourse/learnstatus/{courseId}")
public XcCourseTablesDto getLearnstatus(@PathVariable("courseId")
Long courseId) {
    //登录用户
    SecurityUtil.XcUser user = SecurityUtil.getUser();
    if(user == null){
        XueChengPlusException.cast("请登录后继续选课");
    }
    String userId = user.getId();
    return  courseTablesService.getLeanringStatus(userId,
courseId);

}
```

2.3 接口测试

2.3.1 单元测试

1、准备测试环境

发布两门课程，一门为免费，一门为收费。

小技巧：可以更改课程发布表已有课程的收费标准进行测试。

2、测试添加免费课程

成功：选课记录表一条记录、我的课程表一条记录。

3、测试添加收费课程

成功：选课记录表一条记录

4、重复添加选课

重复添加相同的课程，观察是否存在异常。

5、生成令牌，为方便生成令牌暂时将 PasswordAuthServiceImpl 类中的验证码屏蔽

6、使用 httpclient 测试如下：

```
Bash
### 添加选课
POST {{learning_host}}/learning/choosecourse/2
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsieHVlY2hlbmctcGx1c
yJdLCJ1c2VyX25hbWUiOiJ7XCJiaXJ0aGRheVwiOlwiMjAyMi0wOS0yOFQxOToyODo
0NlwiLFwiY3JlYXRlVGltZVwiOlwiMjAyMi0wOS0yOFQwODozMjowM1wiLFwiaWRcI
jpcIjUwXCIsXCJuYW1lXCI6XCLlrabnlJ8xXCIsXCJuaWNrbmFtZVwiOlwi5aSn5rC
054mbXCIsXCJwZXJtaXNzaW9uc1wiOltcInhjX3N5c21hbmFnZXJcIixcInhjX3N5c
21hbmFnZXJfdXNlclwiLFwieGNfc3lzbWFuYWdlcl91c2VyX2FkZFwiLFwieGNfc3l
zbWFuYWdlcl91c2VyX2VkaXRcIixcInhjX3N5c21hbmFnZXJfdXNlcl92aWV3XCIsX
CJ4Y19zeXNtYW5hZ2VyX3VzZXJfZGVsZXRlXCIsXCJ4Y19zeXNtYW5hZ2VyX2RvY1w
iLFwieGNfc3lzbWFuYWdlcl9sb2dcIixcInhjX3RlYWNobWFuYWdlcl9jb3Vyc2VcI
ixcInhjX3RlYWNobWFuYWdlcl9jb3Vyc2VfYWRkXCIsXCJ4Y190ZWFjaG1hbmFnZXJ
fY291cnNlX2Jhc2VcIixcInhjX3N5c21hbmFnZXJfY29tcGFueVwiLFwieGNfdGVhY
2htYW5hZ2VyX2NvdXJzZV9saXN0XCJdLFwic2V4XCI6XCIxXCIsXCJzdGF0dXNcIjp
cIjFcIixcInVzZXJuYW1lXCI6XCJzdHUxXCIsXCJ1c2VycGljXCI6XCJodHRwOi8vZ
mlsZS54dWVjaGVuZy1wbHVzLmNvbS9kZGRmXCIsXCJ1dHlwZVwiOlwiMTAxMDAxXCJ
9Iiwic2NvcGUiOlsiYWxsIl0sImV4cCI6MTY2NzI5OTQwNiwiYXV0aG9yaXRpZXMiO
lsieGNfc3lzbWFuYWdlcl9kb2MiLCJ4Y19zeXNtYW5hZ2VyX3VzZXJfdmlldyIsInh
jX3RlYWNobWFuYWdlcl9jb3Vyc2UiLCJ4Y19zeXNtYW5hZ2VyX3VzZXJfYWRkIiwie
GNfc3lzbWFuYWdlcl9jb21wYW55IiwieGNfc3lzbWFuYWdlcl91c2VyX2RlbGV0ZSI
sInhjX3N5c21hbmFnZXJfdXNlciIsInhjX3RlYWNobWFuYWdlcl9jb3Vyc2VfYmFzZ
SIsInhjX3RlYWNobWFuYWdlcl9jb3Vyc2VfbGlzdCIsInhjX3N5c21hbmFnZXIiLCJ
4Y19zeXNtYW5hZ2VyX2xvZyIsInhjX3N5c21hbmFnZXJfdXNlcl9lZGl0IiwieGNfd
GVhY2htYW5hZ2VyX2NvdXJzZV9hZGQiXSwianRpIjoiOTYyOTYzMWQtYjRiMC00NTl
kLTgzYzktM2Q4MmRiNmI4NDEzIiwiY2xpZW50X2lkIjoiWGNXZWJBcHAifQ.b77Zre
```

```
iNlPoN-_dnAWxuBfH32tPIoRwg2ePgKn_aZ8c
```

2.3.2 前后端联调

测试流程:

1、启动认证服务、网关服务、验证码服务、学习中心服务、内容管理服务。

2、发布一门免费课程、一门收费课程

3、进入课程详情界面，点击“马上学习”

4、报名成功，自动跳转到学习界面。

5、观察选课记录表、我的课程表数据是否正确。

对于免费课程在课程详情页面点击“马上学习”，通过引导界面添加选课。

1、进入课程详情点击马上学习

2、课程免费时引导加入我的课程表、或进入学习界面。

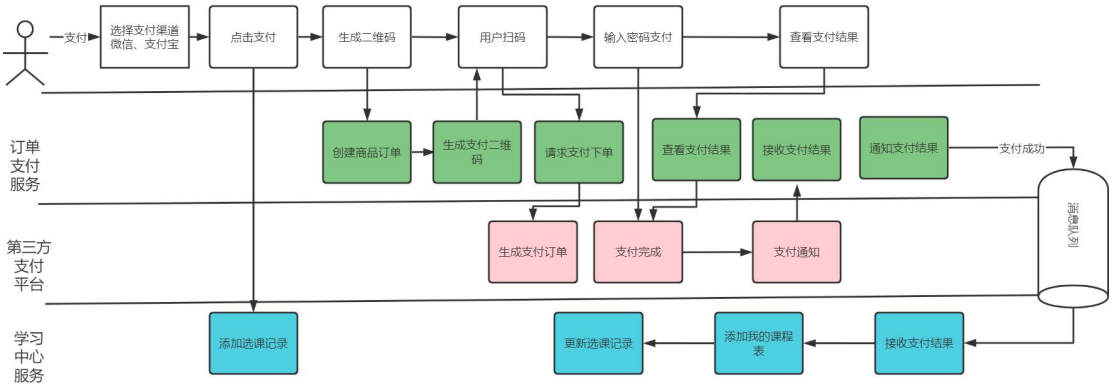
3 支付

3.1 需求分析

3.1.1 执行流程

用户去学习收费课程时引导其去支付，如下图:

当用户点击“微信支付”或支付宝支付时执行流程如下:



1、请求学习中心服务创建选课记录

2、请求订单服务创建商品订单、生成支付二维码。

3、用户扫码请求订单支付服务，订单支付服务请求第三方支付平台生成支付订单。

4、前端唤起支付客户端，用户输入密码完成支付。

5、第三方支付平台支付完成发起支付通知。

6、订单支付服务接收第三方支付通知结果。

7、用户在前端查询支付结果，请求订单支付服务查询支付结果。

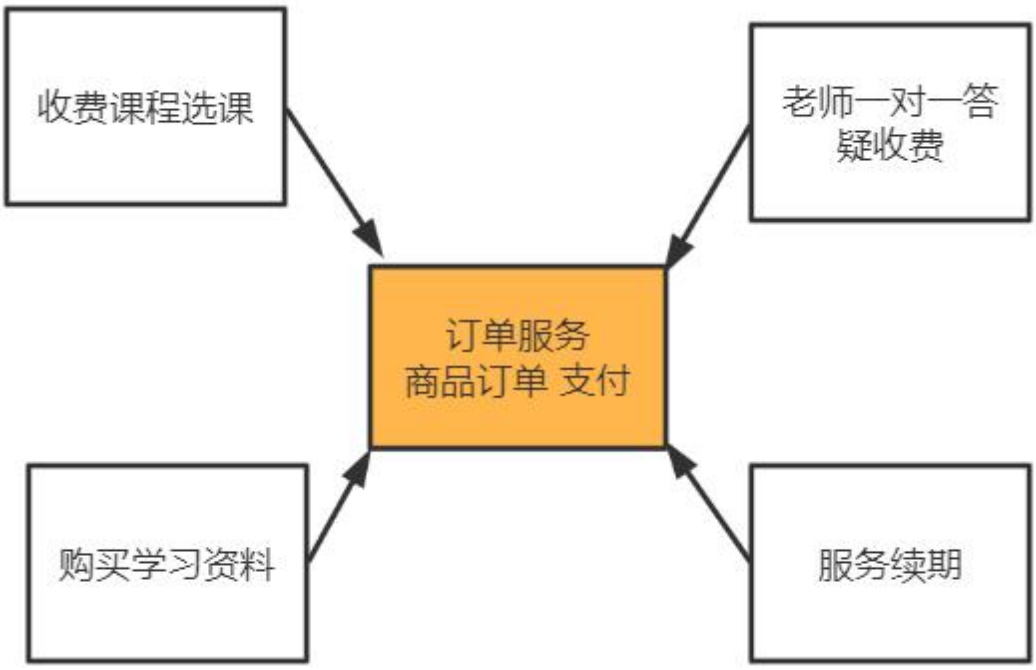
8、订单支付服务向学习中心服务通知支付结果。

9、学习中心服务收到支付结果，如果支付成功则更新选课记录，并添加到我的课程表。

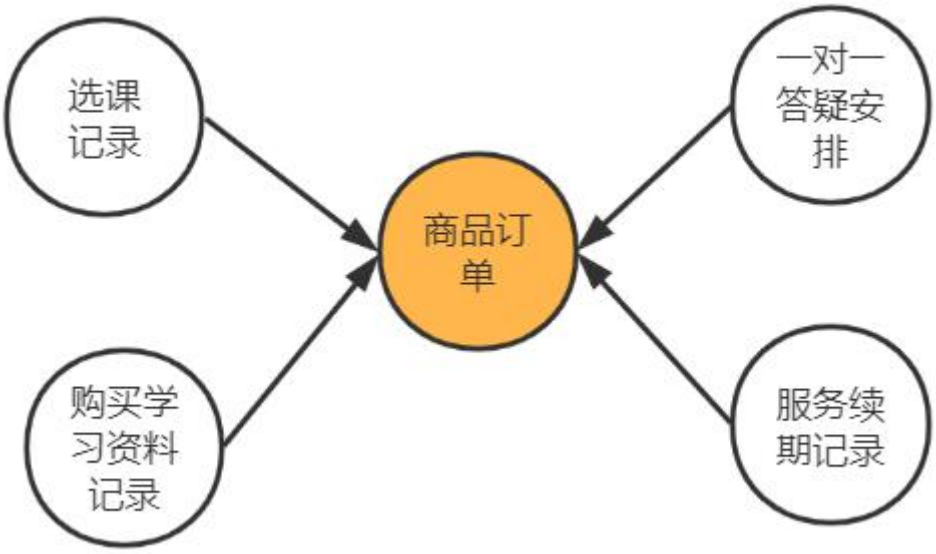
3.1.2 通用订单服务设计

在本项目中不仅选课需要下单、购买学习资料、老师一对一答疑等所以收费项目都需要下单支付。

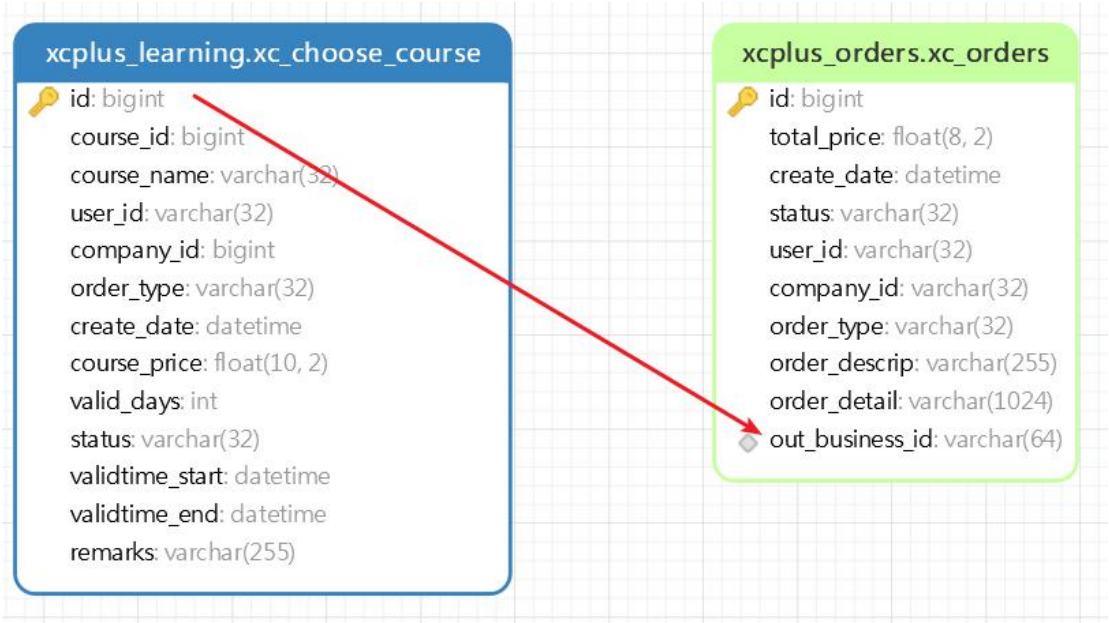
所以本项目设计通用的订单服务，通用的订单服务承接各业务模块的收费支付需求，当用户需要交费时统一生成商品订单并进行支付。



所有收费业务最终转换为商品订单记录在订单服务的商品订单表。



以选课为例，选课记录表的 ID 记录在商品订单表的 out_business_id 字段。



3.2 支付接口调研

3.2.1 微信支付接口调研

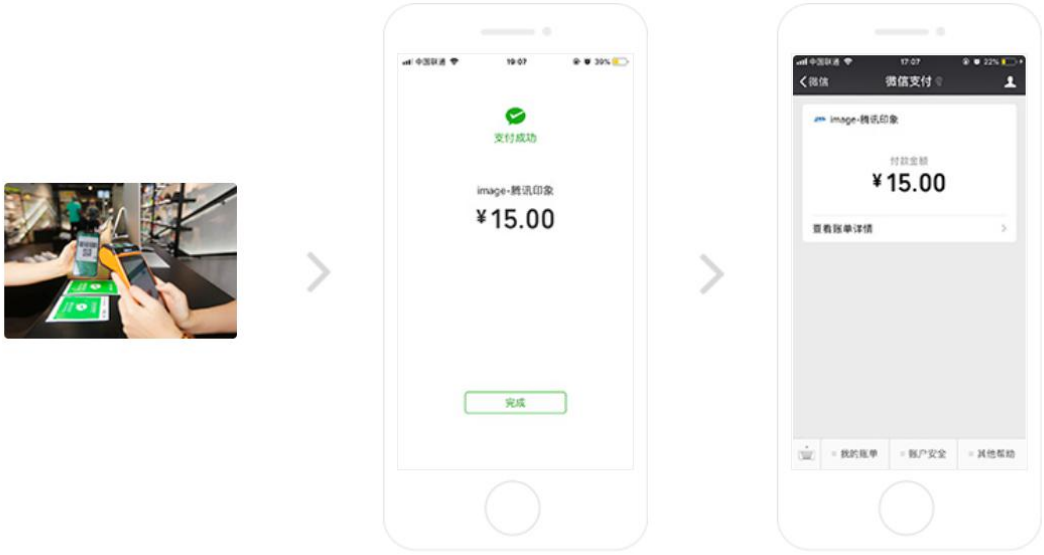
一般情况下，一个网站要支持在线支付功能通常接入第三方支付平台，比如：微信支付、支付宝、其它的聚合支付平台。

本项目的需求实现手机扫码支付，现在对微信、支付宝的支付接口进行调研。

微信目前提供的支付方式如下：

地址：https://pay.weixin.qq.com/static/product/product_index.shtml

1、付款码支付是指用户展示微信钱包内的“付款码”给商户系统扫描后直接完成支付，适用于线下场所面对面收银的场景，例如商超、便利店、餐饮、医院、学校、电影院和旅游景区等具有明确经营地址的实体场所。



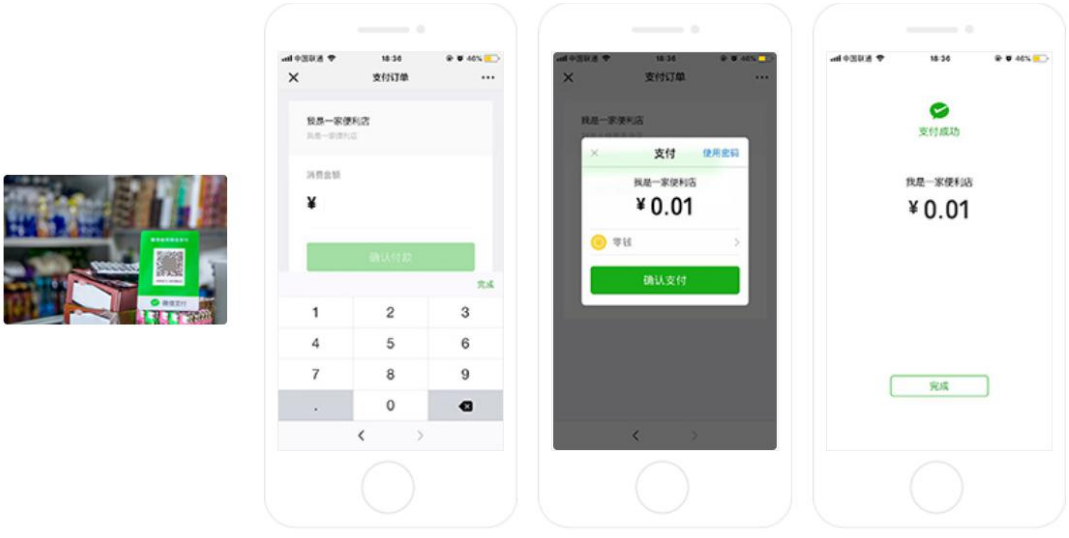
商家扫描付款码　　支付成功　　用户收到账单消息

2、JSAPI 支付是指商户通过调用微信支付提供的 JSAPI 接口，在支付场景中调起微信支付模块完成收款

线下场所：调用接口生成二维码，用户扫描二维码后在微信浏览器中打开页面后完成支付

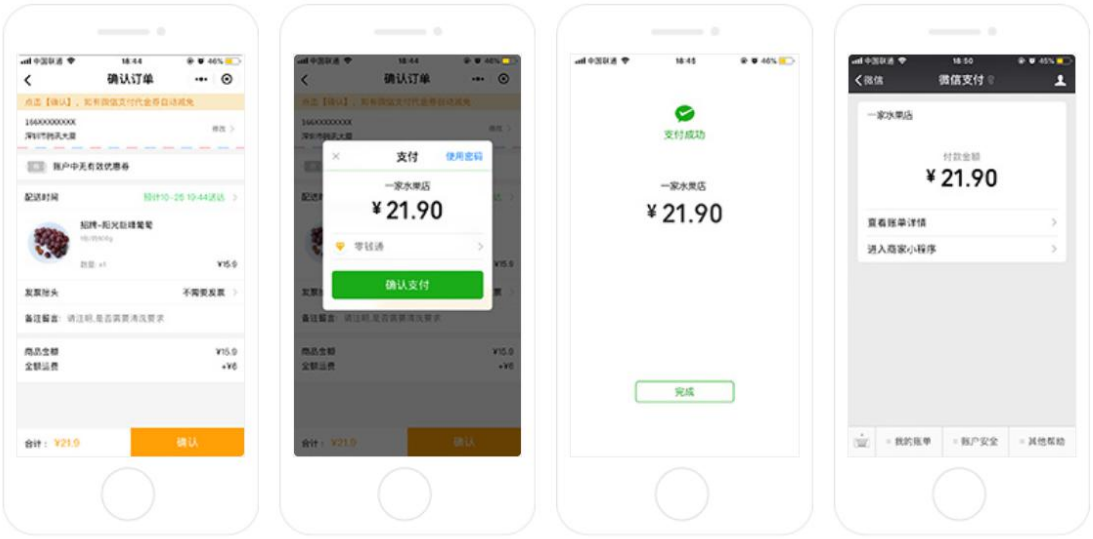
公众号场景：用户在微信公众账号内进入商家公众号，打开某个主页面，完成支付

PC 网站场景：在网站中展示二维码，用户扫描二维码后在微信浏览器中打开页面后完成支付



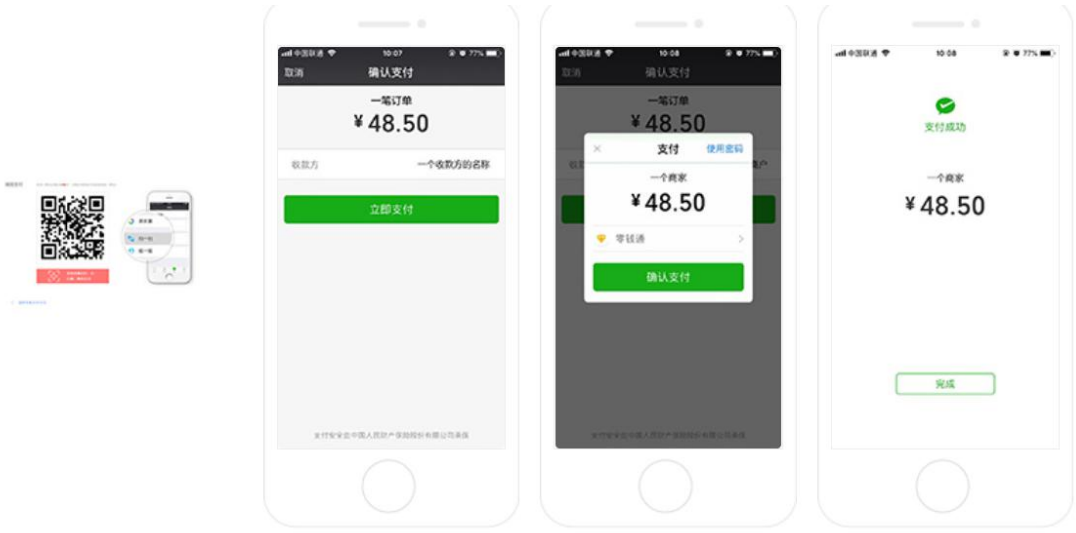
用户扫描二维码　　输入金额　　确认支付　　支付成功

3、小程序支付是指商户通过调用微信支付小程序支付接口，在微信小程序平台内实现支付功能；用户打开商家助手小程序下单，输入支付密码并完成支付后，返回商家小程序。



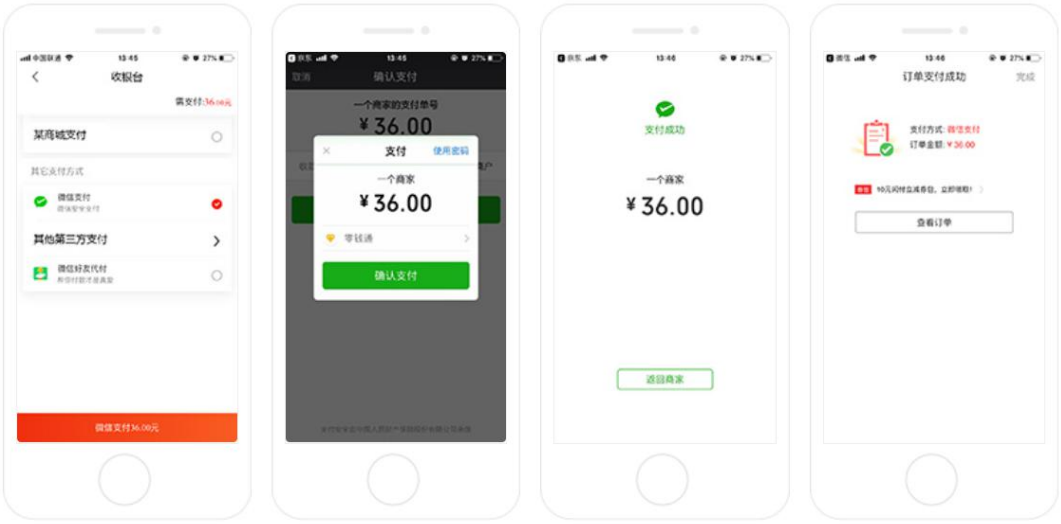
用户在商家助手小程序下单　　确认支付　　支付成功　　用户收到账单消息

4、Native 支付是指商户系统按微信支付协议生成支付二维码，用户再用微信“扫一扫”完成支付的模式。该模式适用于 PC 网站、实体店单品或订单、媒体广告支付等场景。



用户扫描二维码　　确认商家指定金额　　确认支付　　支付成功

5、APP 支付是指商户通过在移动端应用 APP 中集成开放 SDK 调起微信支付模块来完成支付。适用于在移动端 APP 中集成微信支付功能的场景。



商户APP中下单　　确认支付　　支付成功　　返回商户APP界面

6、刷脸支付是指用户在刷脸设备前通过摄像头刷脸、识别身份后进行的一种支付方式，安全便捷。适用于线下实体场所的收银场景，如商超、餐饮、便利店、医院、学校等。

以上接口 native 和 JSAPI 都可以实现 pc 网站实现扫码支付，两者区别是什么？怎么选择？

JSAPI 除了在 pc 网站扫码支付还可以实现公众号页面内支付，可以实现在手机端 H5 页面唤起微信客户端完成支付。

本项目选择 JSAPI 支付接口。

接口文档：https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_1_1.shtml

如何开通 JSAPI 支付接口?

以企业身份注册微信公众号 https://mp.weixin.qq.com/

登录公众号，点击左侧菜单“微信支付”开通微信支付，如下：

需要提供营业执照、身份证等信息。

点击申请接入，需要注册微信商户号。

接入微信支付

注册微信商户号的过程请参考官方文档，本文档略。参考地址如下：

https://pay.weixin.qq.com/index.php/apply/applyment_home/guide_normal#none

开通微信支付后即可在微信商户平台（pay.weixin.qq.com）开通 JSAPI 支付。

登录商品平台，进入产品中心，开通 JSAPI 支付：

注意：JSAPI 支付方式需要在公众号配置回调域名，此域名为已经备案的外网域名。

最后在公众号开发信息中获取：开发者 id、开发者密码。

3.2.2 支付宝接口调研

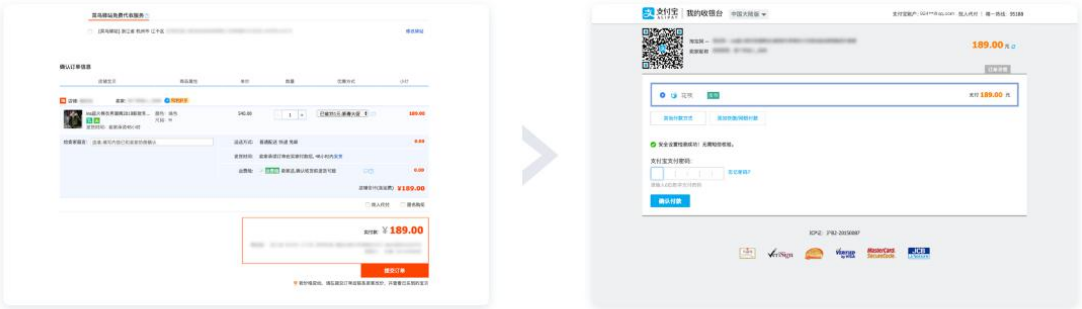
支付宝支付产品如下：

文档：https://b.alipay.com/signing/productSetV2.htm

与本项目需求相关的接口：电脑网站支付、手机网站支付。

1、电脑网站支付

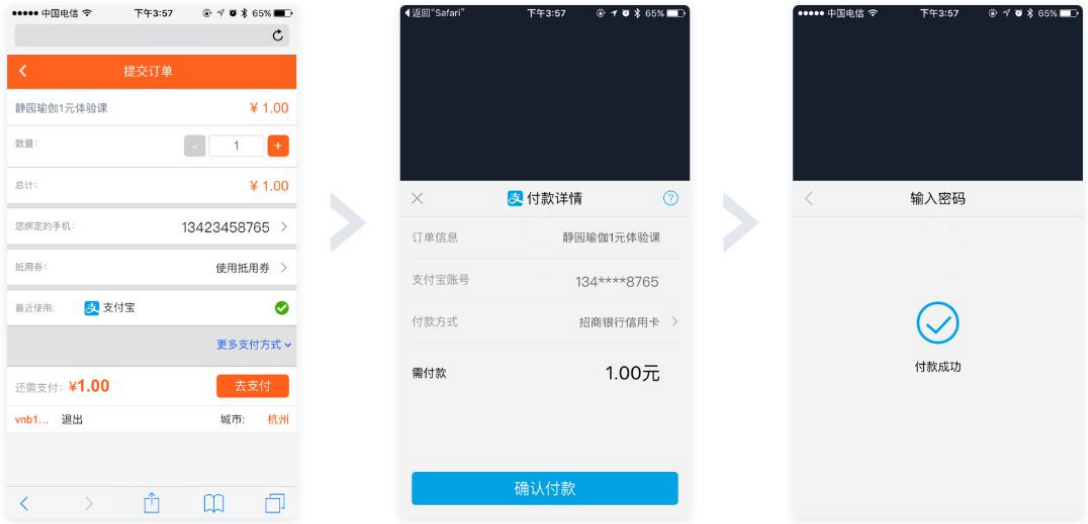
PC 网站轻松收款，资金马上到账：用户在商家 PC 网站消费，自动跳转支付宝 PC 网站收银台完成付款。 交易资金直接打入商家支付宝账户，实时到账。



交易下单　　选择支付宝付款

2、手机网站支付

用户在商家手机网站消费，通过浏览器自动跳转支付宝 APP 或支付宝网页完成付款。轻松实现和 APP 支付相同的支付体验。



交易下单　　选择支付宝付款　　用户完成支付

对比两种支付方式：手机网站支付方式可以在 H5 网页唤起支付宝，手机扫码支付可以使用手机网站支付方式来完成，相比电脑网站支付形式更灵活。

本项目选择手机网站支付方式。

文档：https://opendocs.alipay.com/open/02ivbt

如何开通支付宝手机网站支付接口？

进入网址：
https://b.alipay.com/signing/productDetailV2.htm?productId=I1011000290000001001

点击：立即开通

上传营业执照等资料，提交审核，根据提示进行开通。

3.3 准备开发环境

3.3.1 支付宝开发环境

第三方支付接口流程大同小异，考虑开发及教学的方便性，支付宝提供支付宝沙箱环境开发支付接口，在教学中接入支付宝手机网站支付接口。

1、配置沙箱环境

沙箱环境是支付宝开放平台为开发者提供的与生产环境完全隔离的联调测试环境，开发者在沙箱环境中完成的接口调用不会对生产环境中的数据造成任何影响。

接入手机网站支付需要具备如下条件：

- 申请前必须拥有经过实名认证的支付宝账户；
- 企业或个体工商户可申请；
- 需提供真实有效的营业执照，且支付宝账户名称需与营业执照主体一致；
- 网站能正常访问且页面显示完整，网站需要明确经营内容且有完整的商品信息；
- 网站必须通过 ICP 备案。如为个体工商户，网站备案主体需要与支付宝账户主体名称一致；
- 如为个体工商户，则团购不开放，且古玩、珠宝等奢侈品、投资类行业无法申请本产品。

详细参见：https://docs.open.alipay.com/203

本文档使用支付宝沙箱进行开发测试，这里主要介绍支付宝沙箱环境配置。

详细参见：https://docs.open.alipay.com/200/105311/

2、模拟器

下载模拟器：http://mumu.163.com/

安装模拟器，安装在没有空格和中文的目录。

安装成功，启动模拟器

下一步在模拟器安装支付宝：

选择课程资料中支付宝安装包 wallet_101521226_client_release_201812261416.apk（沙箱版本）

安装成功后支付宝客户端的快捷方式出现在桌面上。

使用沙箱环境的买家账号登录沙箱版本的支付宝。

查看沙箱环境的账号：

3.3.2 创建订单服务

拷贝课程资料目录下的订单服务工程 xuecheng-plus-orders 到自己的工程目录。

创建 xc_orders 数据库，并导入 xcplus_orders.sql

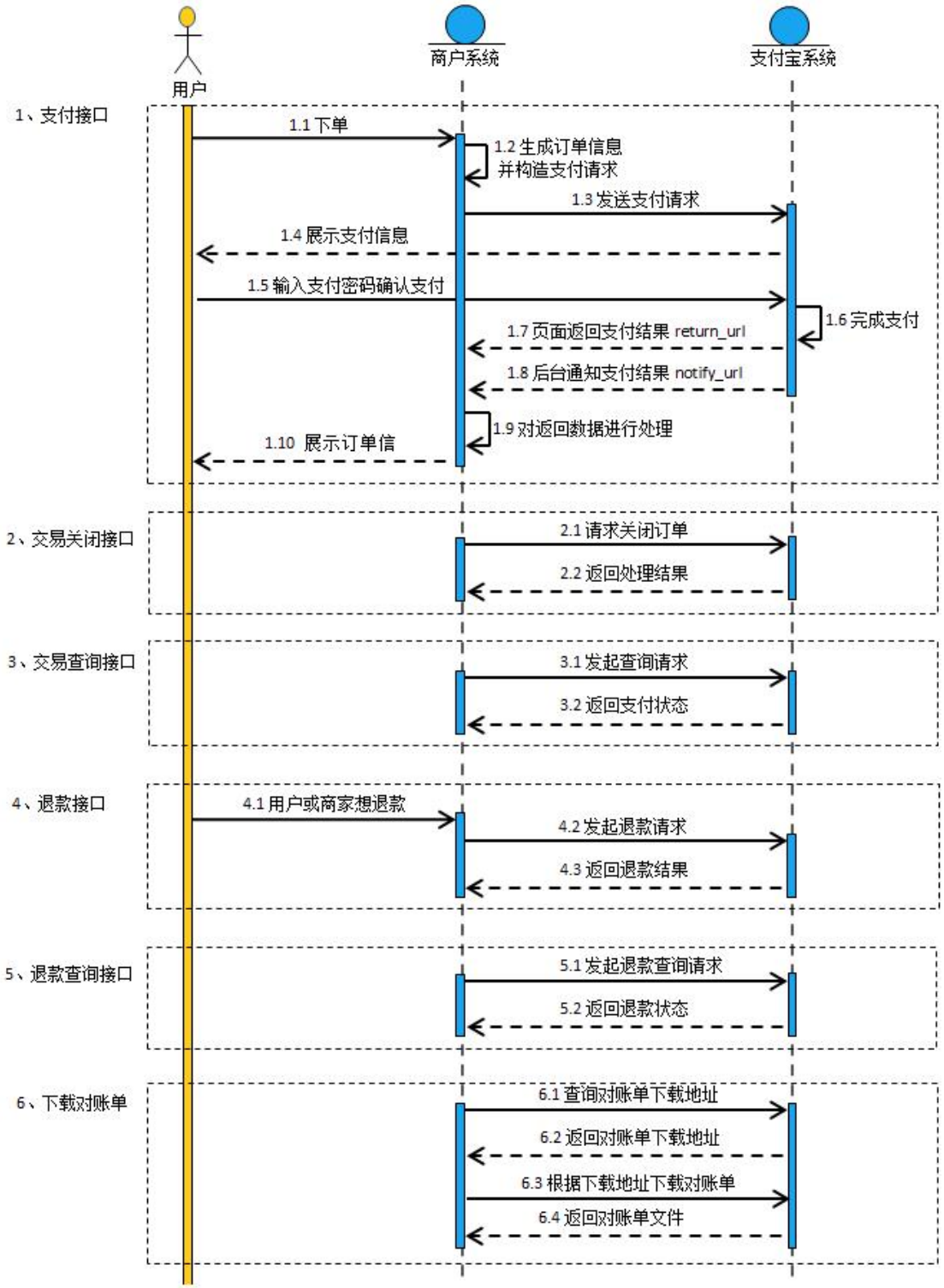
修改 nacos 中 orders-service-dev.yaml 的数据库连接参数。

3.4 支付接口测试

3.4.1 阅读接口定义

手机网站支付接入流程详细参见：https://docs.open.alipay.com/203/105285/

1、接口交互流程如下：



1）用户在商户的 H5 网站下单支付后，商户系统按照手机网站支付接口 alipay.trade.wap.payAPI 的参数规范生成订单数据

2）前端页面通过 Form 表单的形式请求到支付宝。此时支付宝会自动将页面跳转至支付宝 H5 收银台页面，如果用户手机上安装了支付宝 APP，则自动唤起支付宝 APP。

3）输入支付密码完成支付。

4）用户在支付宝 APP 或 H5 收银台完成支付后，会根据商户在手机网站支付 API 中

传入的前台回跳地址 return_url 自动跳转回商户页面，同时在 URL 请求中以 Query String 的形式附带上支付结果参数，详细回跳参数见“手机网站支付接口 alipay.trade.wap.pay”前台回跳参数。

5）支付宝还会根据原始支付 API 中传入的异步通知地址 notify_url，通过 POST 请求的形式将支付结果作为参数通知到商户系统，详情见支付结果异步通知。

2、接口定义

文档：https://opendocs.alipay.com/open/203/107090

接口定义：外部商户请求支付宝创建订单并支付

公共参数

请求地址:

开发中使用沙箱地址：https://openapi.alipaydev.com/gateway.do

请求参数:

详细查阅 https://opendocs.alipay.com/open/203/107090

一部分由 sdk 设置，一部分需要编写程序时指定。

app_id	String	是	32	支付宝分配给开发者的应用 ID	2014072300007148
method	String	是	128	接口名称	alipay.trade.wap.pay
format	String	否	40	仅支持 JSON	JSON
return_url	String	否	256	HTTP/HTTPS 开头字符串	https://m.alipay.com/Gk8NF23
charset	String	是	10	请求使用的编码格式，如 utf-8,gbk,gb2312等	utf-8
sign_type	String	是	10	商户生成签名字符串所使用的签名算法类型，目前支持RSA2和RSA，推荐使用RSA2	RSA2
sign	String	是	256	商户请求参数的签名串，详见签名	详见示例
timestamp	String	是	19	发送请求的时	2014-07-24 03:

body	String	否	128	对一笔交易的具体描述信息。如果是多种商品，请将商品描述字符串累加传给body。	Iphone6 16G
subject	String	是	256	商品的标题/交易标题/订单标题/订单关键字等。	大乐透
out_trade_no	String	是	64	商户网站唯一订单号	70501111111S001111119
total_amount	Price	是	9	订单总金额，单位为元，精确到小数点后两位，取值范围[0.01,100000000]	9.00
product_code	String	是	64	销售产品码，商家和支付宝签约的产品码。该产品请填写固定值：QUICK_WAP_WAY	QUICK_WAP_WAY

其它扩展参数参见接口文档。

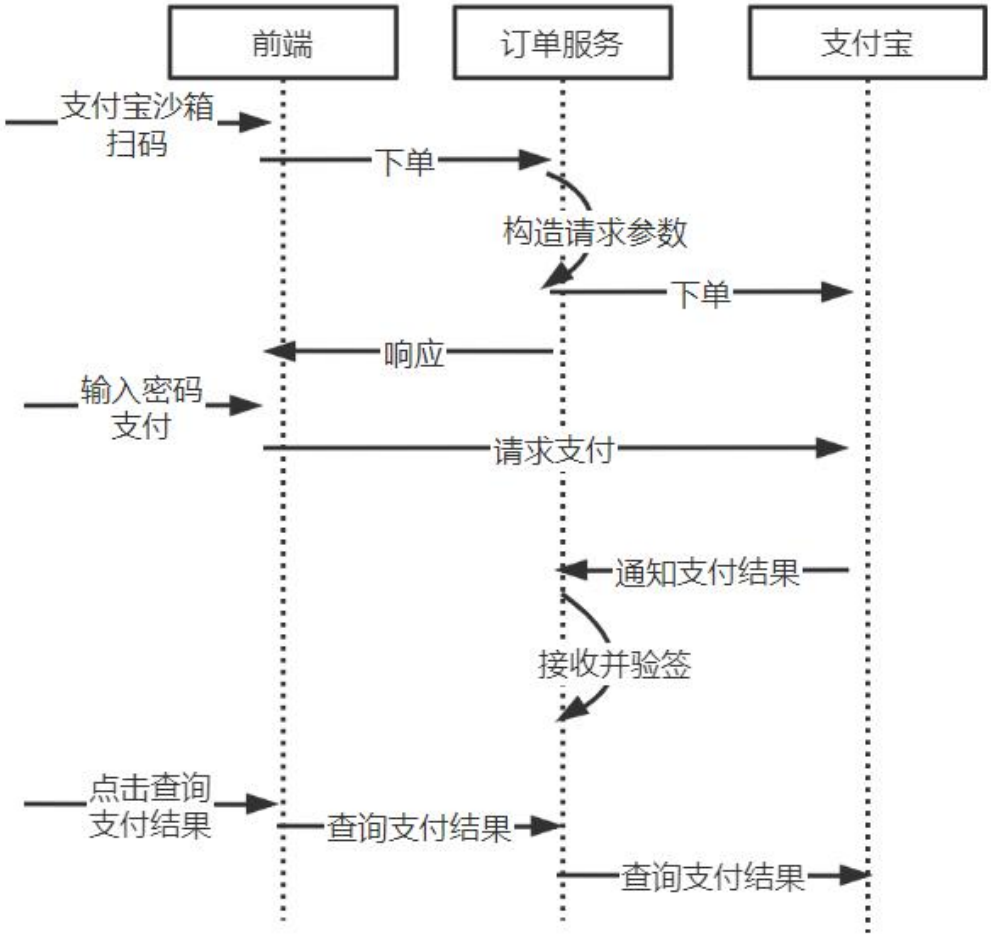
3、示例代码

```
Java
public void doPost(HttpServletRequest httpRequest,
                   HttpServletResponse httpResponse) throws
ServletException, IOException {
    AlipayClient alipayClient = ... //获得初始化的 AlipayClient
    AlipayTradeWapPayRequest alipayRequest = new
AlipayTradeWapPayRequest();//创建 API 对应的 request

alipayRequest.setReturnUrl("http://domain.com/CallBack/return_url.
jsp");

alipayRequest.setNotifyUrl("http://domain.com/CallBack/notify_url.
jsp");//在公共参数中设置回跳和通知地址
    alipayRequest.setBizContent("{" +
            "    \"out_trade_no\":\"20150320010101002\"," +
            "    \"total_amount\":88.88," +
```

```
            "    \"subject\":\"Iphone6 16G\"," +
            "    \"product_code\":\"QUICK_WAP_WAY\"" +
            "  }");//填充业务参数
    String form =
alipayClient.pageExecute(alipayRequest).getBody(); //调用 SDK 生成表
单
    httpResponse.setContentType("text/html;charset=" +
AlipayServiceEnvConstants.CHARSET);
    httpResponse.getWriter().write(form);//直接将完整的表单 html 输出
到页面
    httpResponse.getWriter().flush();
}
```

3.4.2 下单执行流程

根据接口描述，支付宝下单接口的执行流程如下：



3.4.3 支付接口测试

3.4.3.1 编写下单代码

根据接口流程，首先在订单服务编写测试类请求支付宝下单的接口。

在订单服务 api 工程添加依赖:

```XML
<!-- 支付宝 SDK -->
<dependency>
    <groupId>com.alipay.sdk</groupId>
    <artifactId>alipay-sdk-java</artifactId>
    <version>3.7.73.ALL</version>
</dependency>

<!-- 支付宝 SDK 依赖的日志 -->
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.2</version>
</dependency>
```

下载示例代码 https://opendocs.alipay.com/open/203/105910

拷贝示例代码，修改、测试。

拷贝 AlipayConfig.java 到订单服务的 service 工程。

```Java
package com.xuecheng.orders.api;

import com.alipay.api.AlipayApiException;
import com.alipay.api.AlipayClient;
import com.alipay.api.DefaultAlipayClient;
import com.alipay.api.request.AlipayTradeWapPayRequest;
import com.xuecheng.orders.config.AlipayConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * @author Mr.M
```

```
 * @version 1.0
 * @description 测试支付宝接口
 * @date 2022/10/20 22:19
 */
@Controller
public class PayTestController {

    @Value("${pay.alipay.APP_ID}")
    String APP_ID;
    @Value("${pay.alipay.APP_PRIVATE_KEY}")
    String APP_PRIVATE_KEY;

    @Value("${pay.alipay.ALIPAY_PUBLIC_KEY}")
    String ALIPAY_PUBLIC_KEY;



    @RequestMapping("/alipaytest")
    public void doPost(HttpServletRequest httpRequest,
                       HttpServletResponse httpResponse) throws
ServletException, IOException, AlipayApiException {
        AlipayClient alipayClient = new
DefaultAlipayClient(AlipayConfig.URL, APP_ID, APP_PRIVATE_KEY,
AlipayConfig.FORMAT, AlipayConfig.CHARSET,
ALIPAY_PUBLIC_KEY,AlipayConfig.SIGNTYPE);
        //获得初始化的 AlipayClient
        AlipayTradeWapPayRequest alipayRequest = new
AlipayTradeWapPayRequest();//创建 API 对应的 request
//
alipayRequest.setReturnUrl("http://domain.com/CallBack/return_url.
jsp");
//
alipayRequest.setNotifyUrl("http://domain.com/CallBack/notify_url.
jsp");//在公共参数中设置回跳和通知地址
        alipayRequest.setBizContent("{" +
                "    \"out_trade_no\":\"202210100010101002\"," +
                "    \"total_amount\":0.1," +
                "    \"subject\":\"Iphone6 16G\"," +
                "    \"product_code\":\"QUICK_WAP_WAY\"" +
                "  }");//填充业务参数
        String form =
alipayClient.pageExecute(alipayRequest).getBody(); //调用 SDK 生成表
单
        httpResponse.setContentType("text/html;charset=" +
```

```
AlipayConfig.CHARSET);
        httpResponse.getWriter().write(form);//直接将完整的表单 html
输出到页面
        httpResponse.getWriter().flush();
    }

}
```

3.4.3.2 生成二维码

用户在前端使用支付宝沙箱通过扫码请求下单接口，我们需要生成订单服务的下单接口的二维码。

ZXing 是一个开源的类库，是用 Java 编写的多格式的 1D / 2D 条码图像处理库，使用 ZXing 可以生成、识别 QR Code（二维码）。常用的二维码处理库还有 zbar，近几年已经不再更新代码，下边介绍 ZXing 生成二维码的方法。

1）引入依赖

在 base 工程 pom.xml 中添加依赖:

```
XML
        <!-- 二维码生成&识别组件 -->
        <dependency>
            <groupId>com.google.zxing</groupId>
            <artifactId>core</artifactId>
            <version>3.3.3</version>
        </dependency>

        <dependency>
            <groupId>com.google.zxing</groupId>
            <artifactId>javase</artifactId>
            <version>3.3.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>
```

2）生成二维码方法

拷贝课程资料中 utils 下的 QRCodeUtil.java 到 base 工程 util 包下。

测试根据内容生成二维码方法，在 QRCodeUtil 中添加 main 方法如下：

```Java
    public static void main(String[] args) throws IOException {
        QRCodeUtil qrCodeUtil = new QRCodeUtil();

System.out.println(qrCodeUtil.createQRCode("http://www.itcast.cn/"
, 200, 200));
    }
```

运行 main 方法输入二维码图片的 base64 串，如下：

```Bash

AAABQElEQVR42u2YPZKDMAyF5aFIuUfIUThafDSOwhEoUzC8fZKMySSbrVI8ZuICBX
8uIvtZPxjeDfuSf8liPi7LFSgrzRTvV3XCKawXYLptFobviz6ZzB2xEfTjhyS9OwXB
3A7jbMSngLOQ0I4v2AZf96wqTWJ9+9/dYEHSx2RYqfg/oqUgiX3nFBVfcCepcSbiJP
67iwZ1G+5+Am7kyTzW9OcW/kRAX+QJ953+uCl8zO5PV5UsaffUp8rqP5+jiySJU8jt
NxcNrysetCNK6A/V4lEQeU+xa0eZREE1tOTpFYod0VKXsKCqvRqMkW5pkza8Ggy3Wg
EuTvZcz0dcUBc+9MneL1DqkXjQz0eaZA1LqVtmzcMffTKPiPwz1mh2zkGyNwtT9kgu
TVI7LWv6ul7DCpOjX9iaGV66HDny/ZL1WfILfc/hMHLUpekAAAAASUVORK5CYII=
```

将 base64 串复制到浏览器地址后回车将展示一个二维码，用户用手机扫此二维码将请求至 http://www.itcast.cn/。

3.4.3.3 接口测试

1、生成订单服务下单接口的二维码

修改二维码生成的代码如下:

```
Java
public static void main(String[] args) throws IOException {
      QRCodeUtil qrCodeUtil = new QRCodeUtil();

System.out.println(qrCodeUtil.createQRCode("http://localhost:63030
/orders/alipaytest", 200, 200));

}
```

注意：http://localhost:63030 地址用模拟器无法访问，进入 cmd 命令状态，输入命令 ipconfig -all 查看本地网卡分配的局域网 ip 地址，将上边的地址修改如下:

```
Plain Text
http://192.168.101.1:63030/orders/alipaytest
```

运行 main 方法，复制输出到控制台的 base64 串。

```Bash

AAABxUlEQVR42u2YO46EMBBEGxE45AjcBC6GhCUuNtzER3DoANFbZT4zu9KmNavVOL
Awj6BV/TXmvy37kL9Oopl1sS+DWV/M2oRzKyWL+95tfXDfenyzJK/vlCTZ0G1WGpzD
DlKA9ST2YbfJrMnz2wiE8WQw8A2E/lkc6kQ66afnBKTGKCwaru179ApIXVBnzNb7gy
+/ZbCAQJgB5zLCyrD6ZnSSlPAxm1VNyphh28NmKUFI7F3EQ55qrXJfL3VUBOGZJyss
F74C45tWSjbEBW2DJslG94QPtQTNYsyzH9WSrgmXOiqCCBlgFmIUhMLwKCVcSFHkxs
aagcY1vmSwgkRDZE5WO4YzT6tYSuIprNTEzskBedq5lNS4wEs2TOiEbT1tUxGslaW6
OoX98+5ZKhLpmhYFi8LsTNY7T1WE7apNzE5UCgiDD2cpca+T612v0zNGZYQWMVDhJG
7h2dE1BOoMSIujUjQcZUYxOWcXBodzeuKmJdddhtNTqZNcc1cxDTnuMmwbxr7dup5w
jl8S44J9O75Mdkpyzo1hr/eqV9tUBE+8waBy80p1T3YictxpDyexcQUXk+Muw9m5Ro
hZnWKU5PMX55+RLyKnzJvqtaeFAAAAAElFTkSuQmCC
```

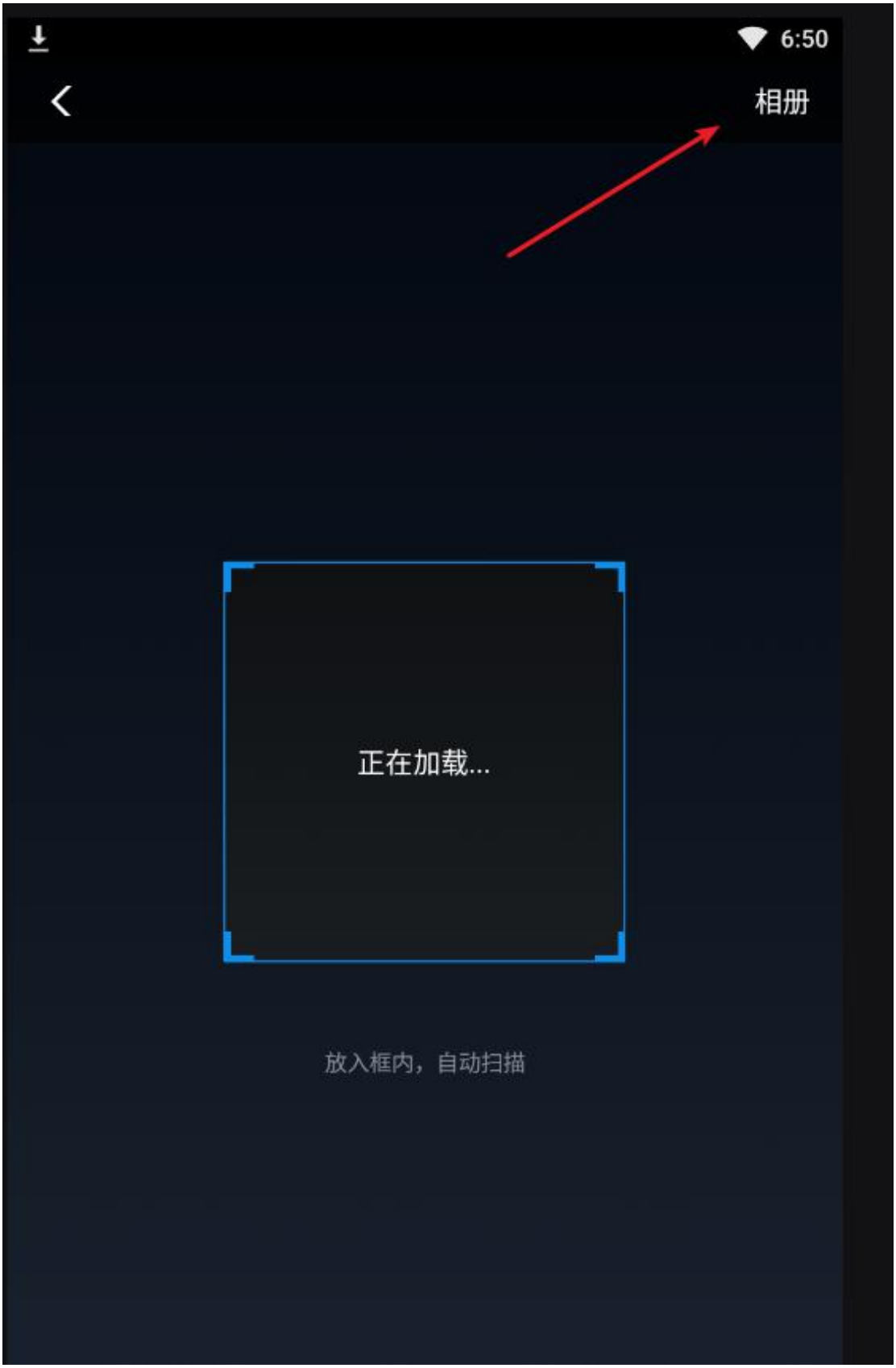
打开模拟器，在模拟器中打开浏览器，将 base64 串复制到浏览器的地址栏。

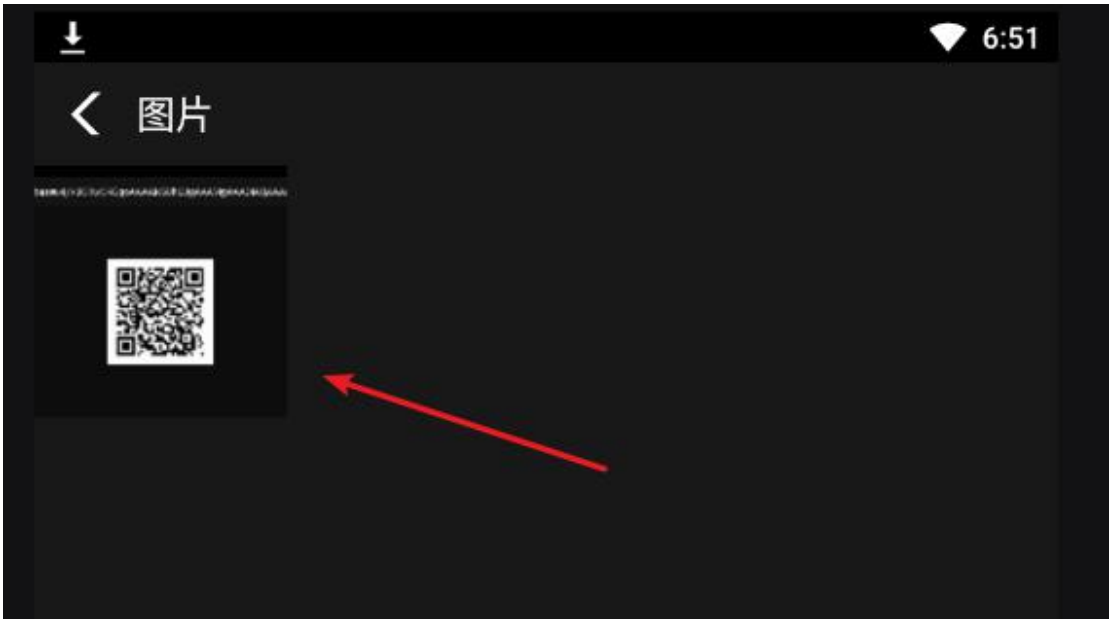
使用截屏工具进行截屏，稍后使用支付宝沙箱客户端扫此图片。

2、启动订单服务

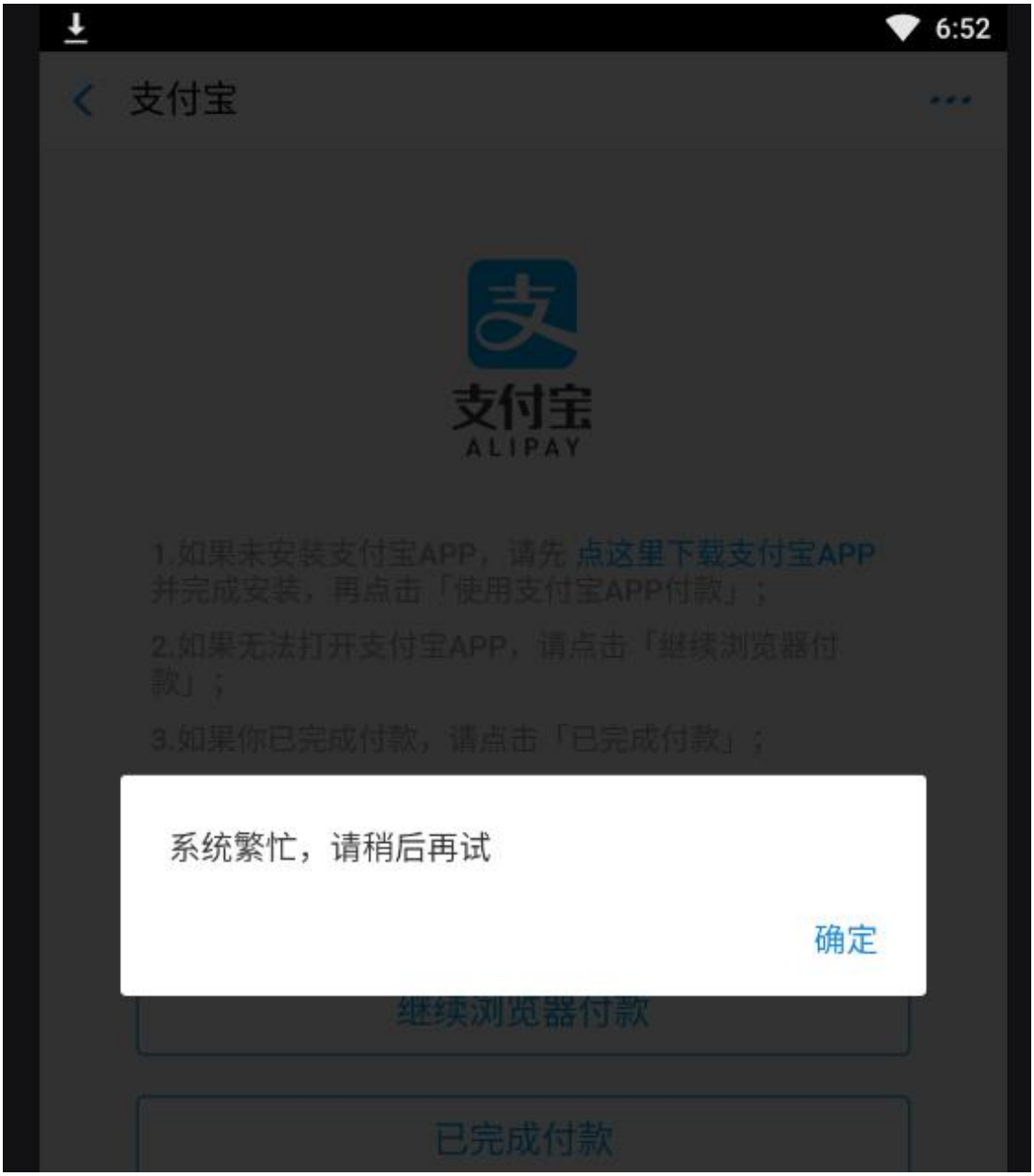
3、打开模拟器，在模拟器中打开支付宝沙箱客户端，并使用沙箱客户端账号密码登录。

点击扫一扫选择相册中刚才截屏的二维码


6:50
相册
正在加载...
放入框内，自动扫描



扫码后如果提示系统繁忙再重试



如果提示请求勿重复提交则需要修改下单测试代码中指定的 out_trade_no 商品订单号，订单号在每个商户是唯一的，每次支付前修改 out_trade_no 为一个没有使用过的订单号。

修改订单号后重启订单服务，再使用沙箱支付宝客户端扫码

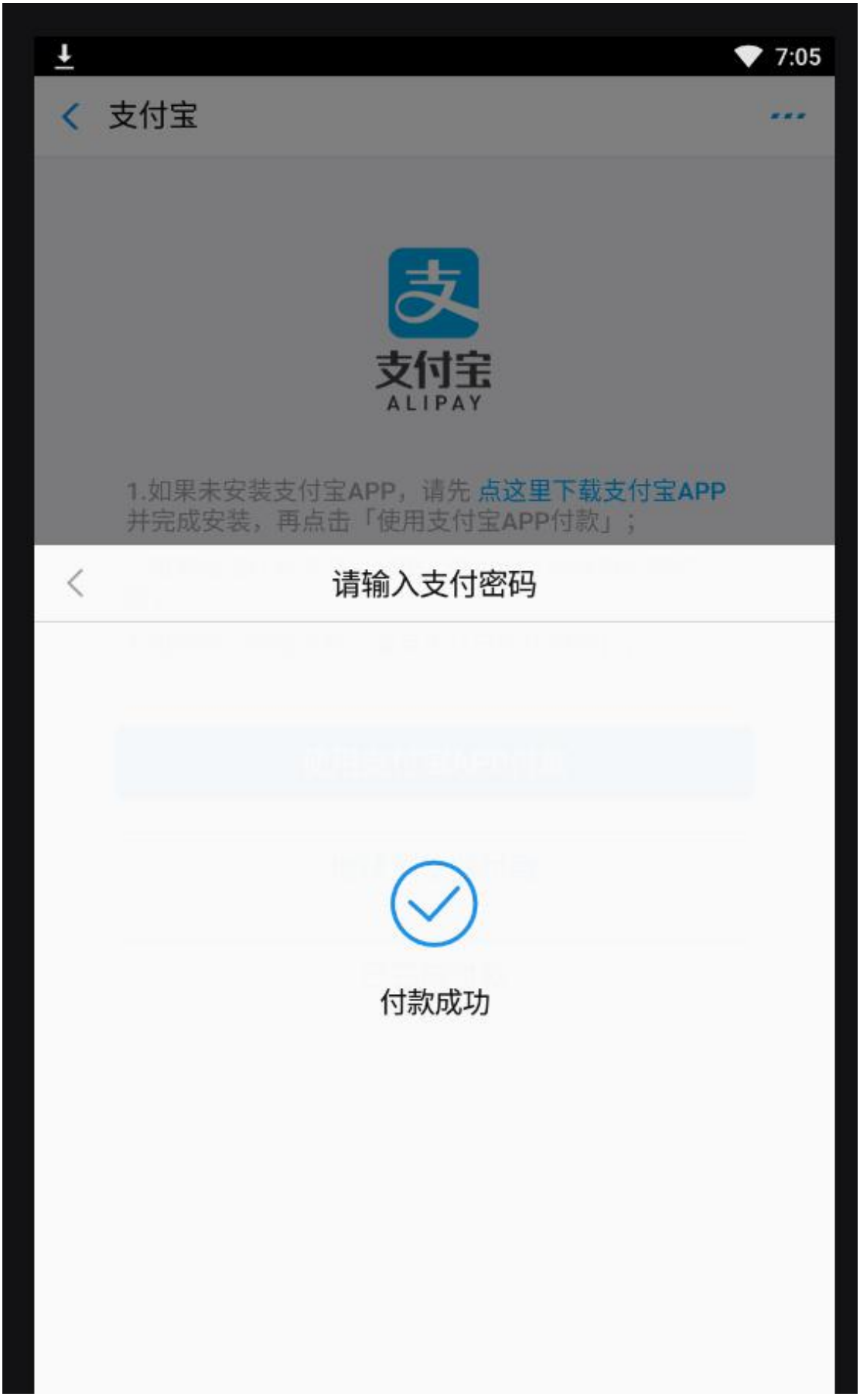

6:58
支付宝
支付宝
ALIPAY
1.如果未安装支付宝APP，请先 点这里下载支付宝APP
并完成安装，再点击「使用支付宝APP付款」；
确认付款
¥1.30
订单信息
Iphone7 16G
付款方式
账户余额
立即付款

输入支付密码进行支付。

支付密码为沙箱账号的支付密码，此支付所扣款为沙箱账号的虚拟货币余额。


6:59
支付宝
支付宝
ALIPAY
1.如果未安装支付宝APP，请先 点这里下载支付宝APP
并完成安装，再点击「使用支付宝APP付款」；
请输入支付密码
忘记密码?
1
2
3
4
5
6
7
8
9
0

支付成功界面:


7:05
支付宝
支付宝
ALIPAY
1.如果未安装支付宝APP，请先 点这里下载支付宝APP
并完成安装，再点击「使用支付宝APP付款」；
请输入支付密码
付款成功

3.4.4 支付结果查询接口

支付完成可以调用第三方支付平台的支付结果查询接口 查询支付结果。

文档：https://opendocs.alipay.com/open/02ivbt

示例代码:

```
Java
AlipayClient alipayClient = new
DefaultAlipayClient("https://openapi.alipay.com/gateway.do","app_i
d","your private_key","json","GBK","alipay_public_key","RSA2");
AlipayTradeQueryRequest request = new AlipayTradeQueryRequest();
JSONObject bizContent = new JSONObject();
bizContent.put("out_trade_no", "20150320010101001");
//bizContent.put("trade_no", "2014112611001004680073956707");
request.setBizContent(bizContent.toString());
AlipayTradeQueryResponse response = alipayClient.execute(request);
if(response.isSuccess()){
    System.out.println("调用成功");
} else {
    System.out.println("调用失败");
}
```

刚才订单付款成功，可以使用 out_trade_no 商品订单号或支付宝的交易流水号 trade_no 去查询支付结果。

out_trade_no 商品订单号： 是在下单请求时指定的商品订单号。

支付宝的交易流水号 trade_no：是支付完成后支付宝通知支付结果时发送的 trade_no

我们使用 out_trade_no 商品订单号去查询，代码如下:

```
Java
package com.xuecheng.orders.api;

import com.alibaba.fastjson.JSONObject;
import com.alipay.api.AlipayApiException;
import com.alipay.api.AlipayClient;
import com.alipay.api.DefaultAlipayClient;
import com.alipay.api.request.AlipayTradeQueryRequest;
import com.alipay.api.response.AlipayTradeQueryResponse;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.test.context.SpringBootTest;
```

```
/**
 * @author Mr.M
 * @version 1.0
 * @description 支付宝查询接口
 * @date 2022/10/4 17:18
 */
@SpringBootTest
public class AliPayTest {

    @Value("${pay.alipay.APP_ID}")
    String APP_ID;
    @Value("${pay.alipay.APP_PRIVATE_KEY}")
    String APP_PRIVATE_KEY;

    @Value("${pay.alipay.ALIPAY_PUBLIC_KEY}")
    String ALIPAY_PUBLIC_KEY;

@Test
public void queryPayResult() throws AlipayApiException {
    AlipayClient alipayClient = new
DefaultAlipayClient(AlipayConfig.URL, APP_ID, APP_PRIVATE_KEY,
"json", AlipayConfig.CHARSET, ALIPAY_PUBLIC_KEY,
AlipayConfig.SIGNTYPE); //获得初始化的 AlipayClient
    AlipayTradeQueryRequest request = new
AlipayTradeQueryRequest();
    JSONObject bizContent = new JSONObject();
    bizContent.put("out_trade_no", "202210100010101002");
    //bizContent.put("trade_no", "2014112611001004680073956707");
    request.setBizContent(bizContent.toString());
    AlipayTradeQueryResponse response =
alipayClient.execute(request);
    if (response.isSuccess()) {
        System.out.println("调用成功");
        String resultJson = response.getBody();
        //转 map
        Map resultMap = JSON.parseObject(resultJson, Map.class);
        Map alipay_trade_query_response = (Map)
resultMap.get("alipay_trade_query_response");
        //支付结果
        String trade_status = (String)
alipay_trade_query_response.get("trade_status");
        System.out.println(trade_status);
    } else {
        System.out.println("调用失败");
```

```
    }
}
}
```

运行代码，输出如下：

```
Plain Text
调用成功
TRADE_SUCCESS
```

输出结果即是调用支付宝查询接口查询到的支付结果


```
{
   "alipay_trade_query_response" : {
      "buyer_logon_id" : "jfw***@sandbox.com",
      "buyer_pay_amount" : "0.00",
      "buyer_user_id" : "2088102180322760",
      "buyer_user_type" : "PRIVATE",
      "code" : "10000",
      "invoice_amount" : "0.00",
      "msg" : "Success",
      "out_trade_no" : "20220520010101026",
      "point_amount" : "0.00",
      "receipt_amount" : "0.00",
      "send_pay_date" : "2022-10-04 07:03:20",
      "total_amount" : "1.30",
      "trade_no" : "2022100422001422760505740639",
      "trade_status" : "TRADE_SUCCESS"
   },
   "sign" : "h+86hYXaQCuNOcfi/ak/XZnX14ZhmUJ3OvU+dwjdQEfe3x7oPEtJh1I5mBDbvJ24CybpJ
}
```


参考文档 https://opendocs.alipay.com/open/02ivbt 查阅每个参数的意义。

我们主要需要下边的参数：

"out_trade_no" : "20220520010101026",

"trade_no":"2022100422001422760505740639" ： 支付宝交易流水号

"total_amount" : "1.30"

"trade_status" : "TRADE_SUCCESS"： 交易状态

交易状态类型：

交易状态：WAIT_BUYER_PAY（交易创建，等待买家付款）

TRADE_CLOSED（未付款交易超时关闭，或支付完成后全额退款）

TRADE_SUCCESS（交易支付成功）

TRADE_FINISHED（交易结束，不可退款）

3.4.5 支付结果通知接口

3.4.5.1 准备环境

对于手机网站支付产生的交易，支付宝会通知商户支付结果，有两种通知方式，通过return_url、notify_url 进行通知，使用 return_url 不能保证通知到位，推荐使用notify_url 完成支付结构通知。

return_url	支付成功后点击完成会自动跳转回商家页面地址， 同时在 URL 地址上附带支付结果参数，回跳参数可查看本文 **附录** > 前台回跳参数说明。在 iOS 系统中，唤起支付宝客户端支付完成后，不会自动回到浏览器或商家 App。用户可手工切回到浏览器或商家 App。
notify_url	异步通知地址，用于接收支付宝推送给商户的支付/退款成功的消息。

具体的使用方法是在调用下单接口的 API 中传入的异步通知地址 notify_url，通过POST 请求的形式将支付结果作为参数通知到商户系统。详情可查看 支付宝异步通知说明 。

文档：https://opendocs.alipay.com/open/203/105286

根据下单执行流程，订单服务收到支付结果需要对内容进行验签，验签过程如下：

1. 在通知返回参数列表中，除去 sign、sign_type 两个参数外，凡是通知返回回来的参数皆是待验签的参数。将剩下参数进行 url_decode，然后进行字典排序，组成字符串，得到待签名字符串； 生活号异步通知组成的待验签串里需要保留 sign_type 参数。

2. 将签名参数（sign）使用 base64 解码为字节码串；

3. 使用 RSA 的验签方法，通过签名字符串、签名参数（经过 base64 解码）及支付宝公钥验证签名。

4. 验证签名正确后，必须再严格按照如下描述校验通知数据的正确性。

在上述验证通过后，商户必须根据支付宝不同类型的业务通知，正确的进行不同的业务处理，并且过滤重复的通知结果数据。

通过验证 out_trade_no、total_amount、appid 参数的正确性判断通知请求的合法性。

验证的过程可以参考 sdk demo 代码， 下载 sdk demo 代码，https://opendocs.alipay.com/open/203/105910

开放平台服务端 SDK

为了帮助开发者调用开放接口，我们提供了开放平台服务端SDK，包含JAVA、PHP和.NET三语言版本封装了签名&验签、HTTP接口请求等基础功能。 详见下载和使用教程。

手机网站支付 Demo

Demo版本	运行环境	下载链接
JAVA版	jdk1.5及以上	点击下载
PHP版	php5.5及以上	点击下载
.NET版	.NET 2010及以上	点击下载

参考 demo 中的 alipay.trade.wap.pay-java-utf-8\WebContent\ notify_url.jsp

另外，支付宝通知订单服务的地址必须为外网域名且备案通过可以正常访问。

此接口仍然使用内网穿透技术。

3.4.5.2 编写测试代码

1、在下单请求时设置通知地址 request.setNotifyUrl("商户自己的 notify_url 地址");

```
Java
@GetMapping("/alipaytest")
    public void alipaytest(HttpServletRequest httpRequest,
                           HttpServletResponse httpResponse) throws
ServletException, IOException {
        //构造 sdk 的客户端对象
        AlipayClient alipayClient = new
DefaultAlipayClient(serverUrl, APP_ID, APP_PRIVATE_KEY, "json",
CHARSET, ALIPAY_PUBLIC_KEY, sign_type); //获得初始化的 AlipayClient
        AlipayTradeWapPayRequest alipayRequest = new
AlipayTradeWapPayRequest();//创建 API 对应的 request
//
alipayRequest.setReturnUrl("http://domain.com/CallBack/return_url.
jsp");
        alipayRequest.setNotifyUrl("http://tjxt-user-
t.itheima.net/xuecheng/orders/paynotify");//在公共参数中设置回跳和通
知地址
        .....
```

2、编写接收通知接口，接收参数并验签

参考课程资料下的 alipay.trade.wap.pay-java-utf-8\WebContent\notify_url.jsp

代码如下:

```
Java
//接收通知
@PostMapping("/paynotify")
public void paynotify(HttpServletRequest
request,HttpServletResponse response) throws
UnsupportedEncodingException, AlipayApiException {
    Map<String,String> params = new HashMap<String,String>();
    Map requestParams = request.getParameterMap();
    for (Iterator iter = requestParams.keySet().iterator();
iter.hasNext();) {
        String name = (String) iter.next();
        String[] values = (String[]) requestParams.get(name);
        String valueStr = "";
        for (int i = 0; i < values.length; i++) {
            valueStr = (i == values.length - 1) ? valueStr +
values[i]
                    : valueStr + values[i] + ",";
        }
        //乱码解决，这段代码在出现乱码时使用。如果 mysign 和 sign 不相等
也可以使用这段代码转化
        //valueStr = new String(valueStr.getBytes("ISO-8859-1"),
"gbk");
        params.put(name, valueStr);
    }


    //获取支付宝的通知返回参数，可参考技术文档中页面跳转同步通知参数列表
(以上仅供参考)//
    //计算得出通知验证结果
    //boolean AlipaySignature.rsaCheckV1(Map<String, String>
params, String publicKey, String charset, String sign_type)
    boolean verify_result = AlipaySignature.rsaCheckV1(params,
ALIPAY_PUBLIC_KEY, AlipayConfig.CHARSET, "RSA2");

    if(verify_result) {//验证成功

//////////////////////////////////////////////////////////////////
//////////////////////
```

```
        //请在这里加上商户的业务逻辑程序代码

         //商户订单号
        String out_trade_no = new
String(request.getParameter("out_trade_no").getBytes("ISO-8859-
1"),"UTF-8");
        //支付宝交易号

        String trade_no = new
String(request.getParameter("trade_no").getBytes("ISO-8859-
1"),"UTF-8");

        //交易状态
        String trade_status = new
String(request.getParameter("trade_status").getBytes("ISO-8859-
1"),"UTF-8");


        //—请根据您的业务逻辑来编写程序（以下代码仅作参考）—

        if (trade_status.equals("TRADE_FINISHED")) {//交易结束
            //判断该笔订单是否在商户网站中已经做过处理
            //如果没有做过处理，根据订单号（out_trade_no）在商户网站的
订单系统中查到该笔订单的详细，并执行商户的业务程序
            //请务必判断请求时的 total_fee、seller_id 与通知时获取的
total_fee、seller_id 为一致的
            //如果有做过处理，不执行商户的业务程序

            //注意：
            //如果签约的是可退款协议，退款日期超过可退款期限后（如三个月
可退款），支付宝系统发送该交易状态通知
            //如果没有签约可退款协议，那么付款完成后，支付宝系统发送该交
易状态通知。
        } else if (trade_status.equals("TRADE_SUCCESS")) {//交易成
功
            System.out.println(trade_status);
            //判断该笔订单是否在商户网站中已经做过处理
            //如果没有做过处理，根据订单号（out_trade_no）在商户网站的
订单系统中查到该笔订单的详细，并执行商户的业务程序
            //请务必判断请求时的 total_fee、seller_id 与通知时获取的
total_fee、seller_id 为一致的
            //如果有做过处理，不执行商户的业务程序
```

```
            //注意：
            //如果签约的是可退款协议，那么付款完成后，支付宝系统发送该交易状态通知。
        }
        response.getWriter().write("success");
    }else{
        response.getWriter().write("fail");
    }


}
```

3.4.5.3 通知接口测试

1、重启订单服务，并在接收通知接口中打上断点

2、配置内网穿透的本地端口为订单服务端口，启动内网穿透客户端。

3、打开模拟器、支付宝沙箱，扫码、支付。

4、观察接收订单支付数据等是否正常。

3.5 生成支付二维码

3.5.1 需求分析

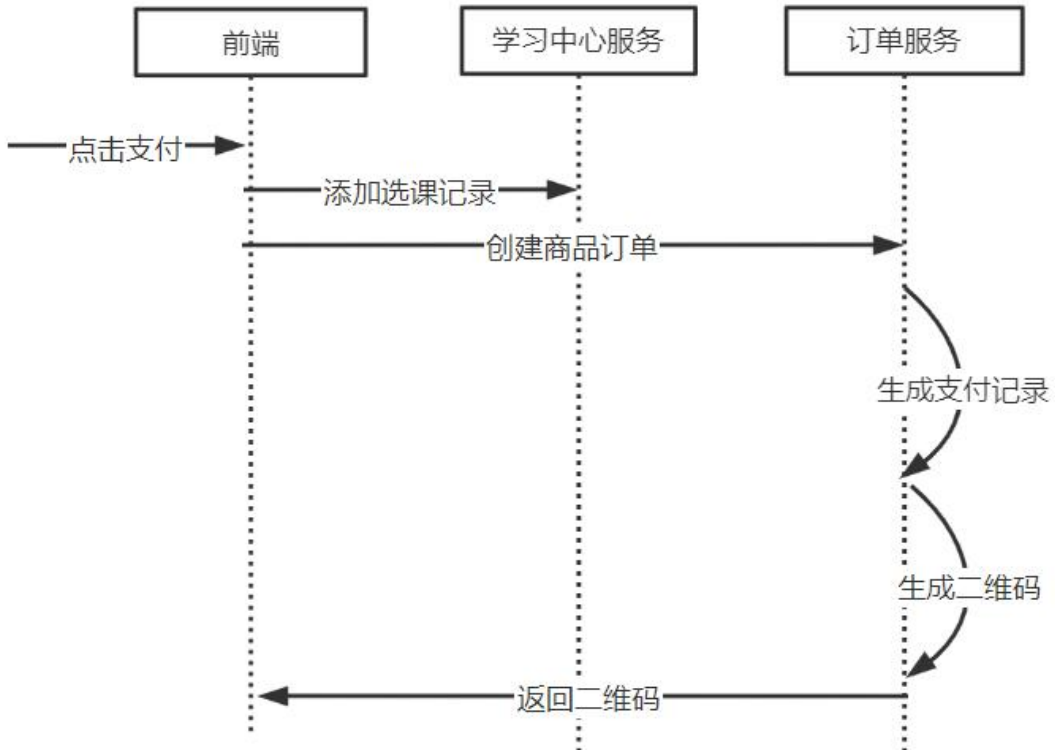
3.5.1.1 执行流程

再次打开课程支付引导界面，点击“支付宝支付”按钮系统该如何处理？

点击“支付宝支付”此时打开支付二维码，用户扫码支付。

所以首先需要生成支付二维码，用户扫描二维码开始请求支付宝下单，在向支付宝下单前需要添加选课记录、创建商品订单、生成支付交易记录。

生成二维码执行流程如下:



执行流程:

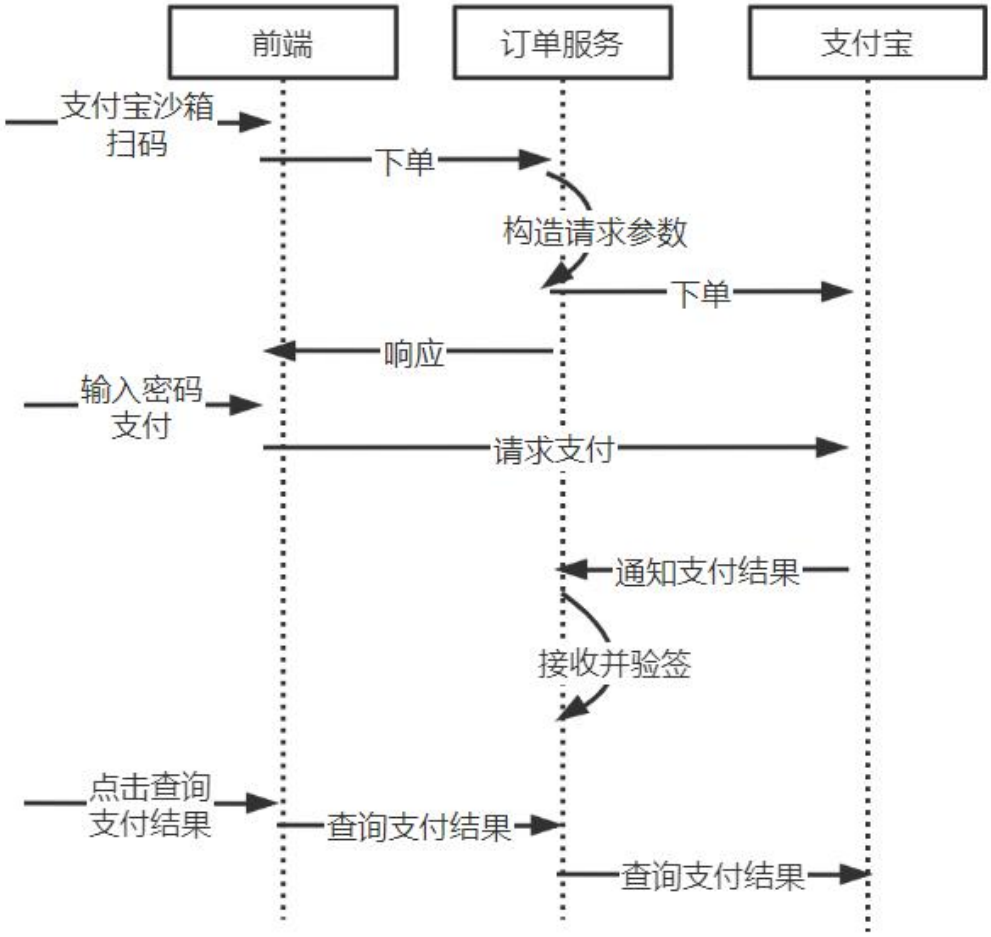
1、前端调用学习中心服务的添加选课接口。

2、添加选课成功请求订单服务生成支付二维码接口。

3、生成二维码接口：创建商品订单、生成支付交易记录、生成二维码。

4、将二维码返回到前端，用户扫码。

用户扫码支付流程如下:



执行流程:

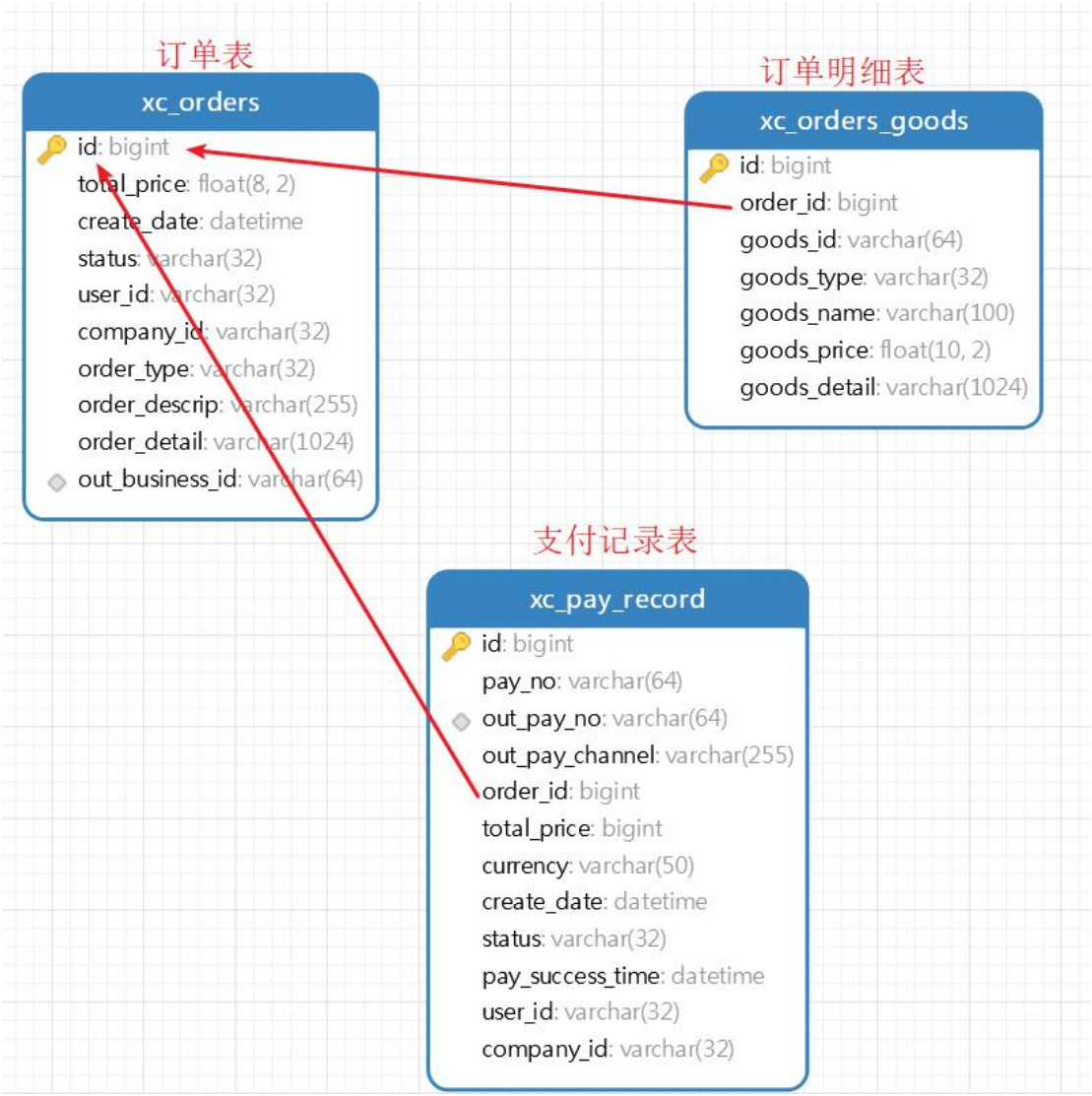
1、用户输入支付密码，支付成功。

2、接收第三方平台通知的支付结果。

3、根据支付结果更新支付交易记录的支付状态为支付成功。

3.5.1.2 数据模型

订单支付模式的核心由三张表组成：订单表、订单明细表、支付交易记录表。



订单表：记录订单信息

名	类型	长度	小数点	不是 null	虚拟	键	注释
id	bigint			☑	☐	1	订单号
total_price	float	8	2	☑	☐		总价
create_date	datetime			☑	☐		创建时间
status	varchar	32		☑	☐		交易状态
user_id	varchar	32		☑	☐		用户id
order_type	varchar	32		☑	☐		订单类型
order_descrip	varchar	255		☑	☐		订单描述
order_detail	varchar	1024		☑	☐		订单明细json
out_business_id	varchar	64		☑	☐		外部系统业务id

订单明细表记录订单的详细信息

名	类型	长度	小数点	不是 null	虚拟	键	注释
id	bigint			☑	☐	1	
order_id	bigint			☑	☐		订单号
goods_id	varchar	64		☑	☐		商品id
goods_type	varchar	32		☐	☐		商品类型
goods_name	varchar	100		☑	☐		商品名称
goods_price	float	10	2	☑	☐		商品交易价，单位分
goods_detail	varchar	1024		☐	☐		商品详情json

支付交易记录表记录每次支付的交易明细

名	类型	长度	小数点	不是 null	虚拟	键	注释
id	bigint			☑	☐	1	支付记录号
pay_no	varchar	64		☐	☐		本系统支付交易号
out_pay_no	varchar	64		☑	☐		第三方支付交易流水号
out_pay_channel	varchar	255		☑	☐		第三方支付渠道编号
order_id	bigint			☑	☐		商品订单号
total_price	bigint			☑	☐		订单总价单位分
currency	varchar	50		☑	☐		币种CNY
create_date	datetime			☑	☐		创建时间
status	varchar	32		☑	☐		支付状态
pay_success_time	datetime			☐	☐		支付成功时间
user_id	varchar	32		☑	☐		用户id

订单号注意唯一性、安全性、尽量短等特点，生成方案常用的如下：

1、时间戳+随机数

年月日时分秒毫秒+随机数

2、高并发场景

年月日时分秒毫秒+随机数+redis 自增序列

3、订单号中加上业务标识

订单号加上业务标识方便客服，比如：第 10 位是业务类型，第 11 位是用户类型等。

4、雪花算法

雪花算法是推特内部使用的分布式环境下的唯一 ID 生成算法，它基于时间戳生成，保证有序递增，加以入计算机硬件等元素，可以满足高并发环境下 ID 不重复。

本项目订单号生成采用雪花算法。

3.5.2 接口定义

在订单服务中定义生成支付二维码接口。

请求：订单信息

```
Java
package com.xuecheng.orders.model.dto;

import com.xuecheng.orders.model.po.XcOrders;
import lombok.Data;
import lombok.ToString;

/**
 * @author Mr.M
```

```
 * @version 1.0
 * @description 创建商品订单
 * @date 2022/10/4 10:21
 */
@Data
@ToString
public class AddOrderDto  {

    /**
     * 总价
     */
    private Float totalPrice;

    /**
     * 订单类型
     */
    private String orderType;

    /**
     * 订单名称
     */
    private String orderName;
    /**
     * 订单描述
     */
    private String orderDescrip;

    /**
     * 订单明细 json，不可为空
     *
[{"goodsId":"","goodsType":"","goodsName":"","goodsPrice":"","goodsDetail":""},{...}]
     */
    private String orderDetail;

    /**
     * 外部系统业务 id
     */
    private String outBusinessId;

}
```

响应：支付交易记录信息及二维码信息

```Java
@Data
@ToString
public class PayRecordDto extends XcPayRecord {

    //二维码
    private String qrcode;

}
```

接口定义如下:

```Java
@Api(value = "订单支付接口", tags = "订单支付接口")
@Slf4j
@Controller
public class OrderController {

    @ApiOperation("生成支付二维码")
    @PostMapping("/generatepaycode")
    @ResponseBody
    public PayRecordDto generatePayCode(@RequestBody AddOrderDto
addOrderDto) {

            return null;
    }

}
```

用户扫码请求下单，定义下单接口如下:

```Java
@ApiOperation("扫码下单接口")
@GetMapping("/requestpay")
public void requestpay(String payNo,HttpServletResponse
httpResponse) throws IOException {

}
```

3.5.3 接口实现

3.5.3.1 保存商品订单

定义保存订单信息接口

```
Java
public interface OrderService {


   /**
 * @description 创建商品订单
 * @param addOrderDto 订单信息
 * @return PayRecordDto 支付交易记录(包括二维码)
 * @author Mr.M
 * @date 2022/10/4 11:02
*/
public PayRecordDto createOrder(String userId,AddOrderDto
addOrderDto);
```

在保存订单接口中需要完成创建商品订单、创建支付交易记录，接口实现方法如下:

```
Java
@Slf4j
@Service
public class OrderServiceImpl implements OrderService {
    @Autowired
    XcOrdersMapper ordersMapper;
    @Autowired
    XcOrdersGoodsMapper ordersGoodsMapper;

    @Autowired
    XcPayRecordMapper payRecordMapper;

    @Transactional
    @Override
    public PayRecordDto createOrder(String userId, AddOrderDto
addOrderDto) {

        //添加商品订单

        //添加支付交易记录

        //生成二维码

        return null;
```

```
    }
}
```

编写创建商品订单方法，商品订单的数据来源于选课记录，在订单表需要存入选课记录的 ID，这里需要作好幂等处理。

```Java
@Transactional
public XcOrders saveXcOrders(String userId,AddOrderDto
addOrderDto){
    //幂等性处理
    XcOrders order =
getOrderByBusinessId(addOrderDto.getOutBusinessId());
    if(order!=null){
        return order;
    }
    order = new XcOrders();
    //生成订单号
    long orderId = IdWorkerUtils.getInstance().nextId();
    order.setId(orderId);
    order.setTotalPrice(addOrderDto.getTotalPrice());
    order.setCreateDate(LocalDateTime.now());
    order.setStatus("600001");//未支付
    order.setUserId(userId);
    order.setOrderType(addOrderDto.getOrderType());
    order.setOrderName(addOrderDto.getOrderName());
    order.setOrderDetail(addOrderDto.getOrderDetail());
    order.setOrderDescrip(addOrderDto.getOrderDescrip());
    order.setOutBusinessId(addOrderDto.getOutBusinessId());//选课记
录 id
    ordersMapper.insert(order);
    String orderDetailJson = addOrderDto.getOrderDetail();
    List<XcOrdersGoods> xcOrdersGoodsList =
JSON.parseArray(orderDetailJson, XcOrdersGoods.class);
    xcOrdersGoodsList.forEach(goods->{
        XcOrdersGoods xcOrdersGoods = new XcOrdersGoods();
        BeanUtils.copyProperties(goods,xcOrdersGoods);
        xcOrdersGoods.setOrderId(orderId);//订单号
        ordersGoodsMapper.insert(xcOrdersGoods);
    });
    return order;
}
```

```
//根据业务 id 查询订单
public XcOrders getOrderByBusinessId(String businessId) {
    XcOrders orders = ordersMapper.selectOne(new
LambdaQueryWrapper<XcOrders>().eq(XcOrders::getOutBusinessId,
businessId));
    return orders;
}
```

3.5.3.2 创建支付交易记录

为什么创建支付交易记录?

在请求微信或支付宝下单接口时需要传入 商品订单号，在与第三方支付平台对接时发现，当用户支付失败或因为其它原因最终该订单没有支付成功，此时再次调用第三方支付平台的下单接口发现报错“订单号已存在”，此时如果我们传入一个没有使用过的订单号就可以解决问题，但是商品订单已经创建，因为没有支付成功重新创建一个新订单是不合理的。

解决以上问题的方案是:

1、用户每次发起都创建一个新的支付交易记录 ，此交易记录与商品订单关联。

2、将支付交易记录的流水号传给第三方支付系统下单接口，这样就即使没有支付成功就不会出现上边的问题。

3、需要提醒用户不要重复支付。

编写创建支付交易记录的方法:

```
Java
```

```
public XcPayRecord createPayRecord(XcOrders orders){
    if(order==null){
       XueChengPlusException.cast("订单不存在");
    }
    if(orders.getStatus().equals("600002")){
        XueChengPlusException.cast("订单已支付");
    }
    XcPayRecord payRecord = new XcPayRecord();
    //生成支付交易流水号
    long payNo = IdWorkerUtils.getInstance().nextId();
    payRecord.setPayNo(payNo);
    payRecord.setOrderId(orders.getId());//商品订单号
    payRecord.setOrderName(orders.getOrderName());
    payRecord.setTotalPrice(orders.getTotalPrice());
    payRecord.setCurrency("CNY");
    payRecord.setCreateDate(LocalDateTime.now());
    payRecord.setStatus("601001");//未支付
    payRecord.setUserId(orders.getUserId());
    payRecordMapper.insert(payRecord);
    return payRecord;

}
```

3.5.3.3 生成支付二维码

1、在 nacos 中 orders-service-dev.yaml 配置二维码的 url

```
Java
pay:
 qrcodeurl: http://192.168.101.1/api/orders/requestpay?payNo=%s
```

2、完善创建订单 service 方法:

```
Java
@Value("${pay.qrcodeurl}")
String qrcodeurl;

@Transactional
@Override
public PayRecordDto createOrder(String userId, AddOrderDto
addOrderDto) {
    //创建商品订单
    XcOrders orders = saveXcOrders(userId, addOrderDto);
```

```
    if(orders==null){
        XueChengPlusException.cast("订单创建失败");
    }
    if(orders.getStatus().equals("600002")){
        XueChengPlusException.cast("订单已支付");
    }
    //生成支付记录
    XcPayRecord payRecord = createPayRecord(orders);
    //生成二维码
    String qrCode = null;
    try {
        //url 要可以被模拟器访问到，url 为下单接口(稍后定义)
        String url = String.format(qrcodeurl,
payRecord.getPayNo());
        qrCode = new QRCodeUtil().createQRCode(url, 200, 200);
    } catch (IOException e) {
        XueChengPlusException.cast("生成二维码出错");
    }
    PayRecordDto payRecordDto = new PayRecordDto();
    BeanUtils.copyProperties(payRecord,payRecordDto);
    payRecordDto.setQrcode(qrCode);

    return payRecordDto;
}
```

3.5.3.4 生成二维码接口完善

完善生成支付二维码 controller 接口

```
Java
@Autowired
OrderService orderService;

@ApiOperation("生成支付二维码")
@PostMapping("/generatepaycode")
@ResponseBody
public PayRecordDto generatePayCode(@RequestBody AddOrderDto
addOrderDto) {
    //登录用户
    SecurityUtil.XcUser user = SecurityUtil.getUser();
    if(user == null){
        XueChengPlusException.cast("请登录后继续选课");
    }
```

```
   return orderService.createOrder(user.getId(), addOrderDto);

}
```

3.5.3.5 扫码下单接口完善

生成了支付二维码，用户扫码请求第三方支付平台下单、支付。

1、定义查询支付交易记录的 Service 接口与实现方法

```
Java
/**
 * @description 查询支付交易记录
 * @param payNo  交易记录号
 * @return com.xuecheng.orders.model.po.XcPayRecord
 * @author Mr.M
 * @date 2022/10/20 23:38
*/
public XcPayRecord getPayRecordByPayno(String payNo);
```

实现如下:

```
Java
@Override
public XcPayRecord getPayRecordByPayno(String payNo) {
    XcPayRecord xcPayRecord = payRecordMapper.selectOne(new
LambdaQueryWrapper<XcPayRecord>().eq(XcPayRecord::getPayNo,
payNo));
    return xcPayRecord;
}
```

2 定义下单接口如下:

```
Java
@Value("${pay.alipay.APP_ID}")
String APP_ID;
@Value("${pay.alipay.APP_PRIVATE_KEY}")
String APP_PRIVATE_KEY;

@Value("${pay.alipay.ALIPAY_PUBLIC_KEY}")
String ALIPAY_PUBLIC_KEY;
```

```
    @ApiOperation("扫码下单接口")
    @GetMapping("/requestpay")
    public void requestpay(String payNo,HttpServletResponse
httpResponse) throws IOException {
        //如果 payNo 不存在则提示重新发起支付
        XcPayRecord payRecord =
orderService.getPayRecordByPayno(payNo);
        if(payRecord == null){
            XueChengPlusException.cast("请重新点击支付获取二维码");
        }
        //支付状态
        String status = payRecord.getStatus();
        if("601002".equals(status)){
            XueChengPlusException.cast("订单已支付，请勿重复支付。");
        }
        //构造 sdk 的客户端对象
        AlipayClient client = new
DefaultAlipayClient(AlipayConfig.URL, APP_ID, APP_PRIVATE_KEY,
AlipayConfig.FORMAT, AlipayConfig.CHARSET, ALIPAY_PUBLIC_KEY,
AlipayConfig.SIGNTYPE);//获得初始化的 AlipayClient
        AlipayTradeWapPayRequest alipayRequest = new
AlipayTradeWapPayRequest();//创建 API 对应的 request
//
alipayRequest.setReturnUrl("http://domain.com/CallBack/return_url.
jsp");
//       alipayRequest.setNotifyUrl("http://tjxt-user-
t.itheima.net/xuecheng/orders/paynotify");//在公共参数中设置回跳和通
知地址
        alipayRequest.setBizContent("{" +
                " \"out_trade_no\":\""+payRecord.getPayNo()+"\"," +
                "
\"total_amount\":\""+payRecord.getTotalPrice()+"\"," +
                " \"subject\":\""+payRecord.getOrderName()+"\"," +
                " \"product_code\":\"QUICK_WAP_PAY\"" +
                " }");//填充业务参数
        String form = "";
        try {
            //请求支付宝下单接口,发起 http 请求
            form = client.pageExecute(alipayRequest).getBody(); //
调用 SDK 生成表单
        } catch (AlipayApiException e) {
            e.printStackTrace();
        }
```

```
        httpResponse.setContentType("text/html;charset=" +
AlipayConfig.CHARSET);
        httpResponse.getWriter().write(form);//直接将完整的表单 html
输出到页面
        httpResponse.getWriter().flush();
        httpResponse.getWriter().close();
    }
```

3.5.3 支付测试

测试准备:

1、启动网关服务、认证服务、验证码服务、学习中心服务、订单服务、内容管理服务。

2、发布一门收费课程。

3、使用资料目录中的新模板 course_template.ftl

测试流程:

1、进入收费课程详细页面，点击马上学习。

2、跟踪浏览器及微服务，观察选课记录是否创建成功、商品订单是否创建成功、支付交易记录是否创建成功。

3、观察生成二维码是否成功

4、使用模拟器扫码测试，是否可以正常支付。

如果报订单参数异常报如下错误，需要检查请求支付宝的下单数据是否正确。


2:23
支付宝
支付宝
ALIPAY
1.如果未安装支付宝APP，请先 点这里下载支付宝APP 并完成安装，再点击「使用支付宝APP付款」；
2.如果无法打开支付宝APP，请点击「继续浏览器付款」；
3.如果你已完成付款，请点击「已完成付款」；
订单参数异常，请重新下单后再发起付款。
(ALIN42273)
确定
继续浏览器付款
已完成付款

3.6 查询支付结果

3.6.1 接口定义

根据前边我们调研的获取支付结果的接口，包括：主动查询支付结果、被动接收支付结果。

这里先实现主动查询支付结果，当支付完成用户点击“支付结果”将请求第三方支付平台查询支付结果。

在 OrderController 类中定义接口如下：

```
Java
@ApiOperation("查询支付结果")
@GetMapping("/payresult")
@ResponseBody
public PayRecordDto payresult(String payNo) throws IOException {

    //查询支付结果

    return null;

}
```

3.6.2 接口实现

3.6.2.1 service 总体接口

1、定义查询支付结果的 service

```
Java
/**
 * 请求支付宝查询支付结果
 * @param payNo 支付记录 id
 * @return 支付记录信息
 */
public PayRecordDto queryPayResult(String payNo);
```

2、service 实现如下：

```
Java
@Override
public PayRecordDto queryPayResult(String payNo){
```

```
    XcPayRecord payRecord = getPayRecordByPayno(payNo);
    if (payRecord == null) {
        XueChengPlusException.cast("请重新点击支付获取二维码");
    }
    //支付状态
    String status = payRecord.getStatus();
    //如果支付成功直接返回
    if ("601002".equals(status)) {
        PayRecordDto payRecordDto = new PayRecordDto();
        BeanUtils.copyProperties(payRecord, payRecordDto);
        return payRecordDto;
    }
    //从支付宝查询支付结果
    PayStatusDto payStatusDto = queryPayResultFromAlipay(payNo);
    //保存支付结果
    currentProxy.saveAliPayStatus( payStatusDto);
    //重新查询支付记录
    payRecord = getPayRecordByPayno(payNo);
    PayRecordDto payRecordDto = new PayRecordDto();
    BeanUtils.copyProperties(payRecord, payRecordDto);
    return payRecordDto;

}

/**
 * 请求支付宝查询支付结果
 * @param payNo 支付交易号
 * @return 支付结果
 */
public PayStatusDto queryPayResultFromAlipay(String payNo){

}

/**
 * @description 保存支付宝支付结果
 * @param payStatusDto  支付结果信息
 * @return void
 * @author Mr.M
 * @date 2022/10/4 16:52
 */
public void saveAliPayStatus(PayStatusDto payStatusDto) ;
```

3.6.2.2 查询支付结果

定义从支付宝查询支付结果的方法

```
Java
/**
 * 请求支付宝查询支付结果
 * @param payNo 支付交易号
 * @return 支付结果
 */
public PayStatusDto queryPayResultFromAlipay(String payNo) {

    //========请求支付宝查询支付结果=============
    AlipayClient alipayClient = new
DefaultAlipayClient(AlipayConfig.URL, APP_ID, APP_PRIVATE_KEY,
"json", AlipayConfig.CHARSET, ALIPAY_PUBLIC_KEY,
AlipayConfig.SIGNTYPE); //获得初始化的 AlipayClient
    AlipayTradeQueryRequest request = new
AlipayTradeQueryRequest();
    JSONObject bizContent = new JSONObject();
    bizContent.put("out_trade_no", payNo);
    request.setBizContent(bizContent.toString());
    AlipayTradeQueryResponse response = null;
    try {
        response = alipayClient.execute(request);
        if (!response.isSuccess()) {
            XueChengPlusException.cast("请求支付查询查询失败");
        }
    } catch (AlipayApiException e) {
        log.error("请求支付宝查询支付结果异常:{}", e.toString(), e);
        XueChengPlusException.cast("请求支付查询查询失败");
    }

    //获取支付结果
    String resultJson = response.getBody();
    //转 map
    Map resultMap = JSON.parseObject(resultJson, Map.class);
    Map alipay_trade_query_response = (Map)
resultMap.get("alipay_trade_query_response");
    //支付结果
    String trade_status = (String)
alipay_trade_query_response.get("trade_status");
    String total_amount = (String)
```

```
alipay_trade_query_response.get("total_amount");
    String trade_no = (String)
alipay_trade_query_response.get("trade_no");
    //保存支付结果
    PayStatusDto payStatusDto = new PayStatusDto();
    payStatusDto.setOut_trade_no(payNo);
    payStatusDto.setTrade_status(trade_status);
    payStatusDto.setApp_id(APP_ID);
    payStatusDto.setTrade_no(trade_no);
    payStatusDto.setTotal_amount(total_amount);
    return payStatusDto;

}
```

3.6.2.3 保存支付结果

1、定义保存支付结果的接口

```
Java
/**
 * @description 保存支付宝支付结果
 * @param payStatusDto  支付结果信息
 * @return void
 * @author Mr.M
 * @date 2022/10/4 16:52
 */
public void saveAliPayStatus(PayStatusDto payStatusDto) ;
```

2、编写接口实现

```
Java
@Transactional
@Override
public void saveAliPayStatus(PayStatusDto payStatusDto) {
    //支付流水号
    String payNo = payStatusDto.getOut_trade_no();
    XcPayRecord payRecord = getPayRecordByPayno(payNo);
    if (payRecord == null) {
        XueChengPlusException.cast("支付记录找不到");
    }
    //支付结果
    String trade_status = payStatusDto.getTrade_status();
```

```
    log.debug("收到支付结果:{},支付记录:{}}",
payStatusDto.toString(),payRecord.toString());
    if (trade_status.equals("TRADE_SUCCESS")) {

        //支付金额变为分
        Float totalPrice = payRecord.getTotalPrice() * 100;
        Float total_amount =
Float.parseFloat(payStatusDto.getTotal_amount()) * 100;
        //校验是否一致
        if (!payStatusDto.getApp_id().equals(APP_ID) ||
totalPrice.intValue() != total_amount.intValue()) {
            //校验失败
            log.info("校验支付结果失败,支付记
录:{},APP_ID:{},totalPrice:{}" ,payRecord.toString(),payStatusDto.
getApp_id(),total_amount.intValue());
            XueChengPlusException.cast("校验支付结果失败");
        }
        log.debug("更新支付结果,支付交易流水号:{},支付结果:{}", payNo,
trade_status);
        XcPayRecord payRecord_u = new XcPayRecord();
        payRecord_u.setStatus("601002");//支付成功
        payRecord_u.setOutPayChannel("Alipay");
        payRecord_u.setOutPayNo(payStatusDto.getTrade_no());//支付
宝交易号
        payRecord_u.setPaySuccessTime(LocalDateTime.now());//通知时
间
        int update1 = payRecordMapper.update(payRecord_u, new
LambdaQueryWrapper<XcPayRecord>().eq(XcPayRecord::getPayNo,
payNo));
        if (update1 > 0) {
            log.info("更新支付记录状态成功:{}",
payRecord_u.toString());
        } else {
            log.info("更新支付记录状态失败:{}",
payRecord_u.toString());
            XueChengPlusException.cast("更新支付记录状态失败");
        }
        //关联的订单号
        Long orderId = payRecord.getOrderId();
        XcOrders orders = ordersMapper.selectById(orderId);
        if (orders == null) {
            log.info("根据支付记录[{}}]找不到订单",
```

```
payRecord_u.toString());
            XueChengPlusException.cast("根据支付记录找不到订单");
        }
        XcOrders order_u = new XcOrders();
        order_u.setStatus("600002");//支付成功
        int update = ordersMapper.update(order_u, new
LambdaQueryWrapper<XcOrders>().eq(XcOrders::getId, orderId));
        if (update > 0) {
            log.info("更新订单表状态成功,订单号:{}", orderId);
        } else {
            log.info("更新订单表状态失败,订单号:{}", orderId);
            XueChengPlusException.cast("更新订单表状态失败");
        }
    }

}
```

3.6.3 接口测试

1、完善接口

```
Java
@ApiOperation("查询支付结果")
@GetMapping("/payresult")
@ResponseBody
public PayRecordDto payresult(String payNo) throws IOException {
    //调用支付宝接口查询
    PayRecordDto payRecordDto =
orderService.queryPayResult(payNo);
    return payRecordDto;
}
```

2、测试流程

完成生成支付二维码

用支付宝扫码但不支付

使用 httpclient 请求查询支付结果，查询失败

```
Java
```

```
### 查询支付结果
GET {{orders_host}}/orders/payresult?payNo=1628648111951941632
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsieHVlY2hlbmctcGx1c
yJdLCJ1c2VyX25hbWUiOiJ7XCJiaXJ0aGRheVwiOlwiMjAyMi0wOS0yOFQxOToyODo
0NlwiLFwiY3JlYXRlVGltZVwiOlwiMjAyMi0wOS0yOFQwODozMjowM1wiLFwiaWRcI
jpcIjUwXCIsXCJuYW1lXCI6XCLlrabnlJ8xXCIsXCJuaWNrbmFtZVwiOlwi5aSn5rC
054mbXCIsXCJwZXJtaXNzaW9uc1wiOltdLFwic2V4XCI6XCIxXCIsXCJzdGF0dXNcI
jpcIjFcIixcInVzZXJuYW1lXCI6XCJzdHUxXCIsXCJ1c2VycGljXCI6XCJodHRwOi8
vZmlsZS41MXh1ZWNoZW5nLmNuL2RkZGZcIixcInV0eXBlXCI6XCIxMDEwMDFcIn0iL
CJzY29wZSI6WyJhbGwiXSwiZXhwIjoxNjc3MTQwMDI1LCJhdXRob3JpdGllcyI6WyJ
wMSJdLCJqdGkiOiJmYThiMmY1OS03ZTQ5LTRmODUtOTBlMC05NzYwNjlkYjE3ODIiL
CJjbGllbnRfaWQiOiJYY1dlYkFwcCJ9.89sp5lPdFafZ_HdGhe8Cpv0anMJC3vT4Pt
aGMHgCkr8
```

用支付宝扫码再次完成支付

使用 httpclient 请求查询支付结果，支付结果为成功，并更新支付记录状态和订单状态。

3、使用前后端联调

拷贝资料目录中 LocalDateTimeConfig.java 到 base 工程下，处理 long 转 string 精度损失的问题。

使用最新的门户代码 xc-ui-pc-static-portal.zip

3.7 接收支付通知

3.7.1 接口定义

支付完成后第三方支付系统会主动通知支付结果，要实现主动通知需要在请求支付系统下单时传入 NotifyUrl，这里有两个 url：NotifyUrl 和 ReturnUrl，ReturnUrl 是支付完成后支付系统携带支付结果重定向到 ReturnUrl 地址，NotifyUrl 是支付完成后支付系统在后台定时去通知，使用 NotifyUrl 比使用 ReturnUrl 有保证。

根据接口描述：https://opendocs.alipay.com/open/203/105286 的内容下边在订单服务定义接收支付结果通知的接口。

首先在下单时指定 NotifyUrl:

```
Java
```

```
alipayRequest.setNotifyUrl("http://tjxt-user-
t.itheima.net/xuecheng/orders/receivenotify");
```

接收支付结果通知接口如下:

```
Java
@ApiOperation("接收支付结果通知")
@PostMapping("/receivenotify")
public void receivenotify(HttpServletRequest
request,HttpServletResponse out) throws
UnsupportedEncodingException, AlipayApiException {
    Map<String,String> params = new HashMap<String,String>();
    Map requestParams = request.getParameterMap();
    for (Iterator iter = requestParams.keySet().iterator();
iter.hasNext();) {
        String name = (String) iter.next();
        String[] values = (String[]) requestParams.get(name);
        String valueStr = "";
        for (int i = 0; i < values.length; i++) {
            valueStr = (i == values.length - 1) ? valueStr +
values[i]
                    : valueStr + values[i] + ",";
        }
        params.put(name, valueStr);
    }

    //验签
    boolean verify_result = AlipaySignature.rsaCheckV1(params,
ALIPAY_PUBLIC_KEY, AlipayConfig.CHARSET, "RSA2");

    if(verify_result) {//验证成功

        //商户订单号
        String out_trade_no = new
String(request.getParameter("out_trade_no").getBytes("ISO-8859-
1"),"UTF-8");
        //支付宝交易号
        String trade_no = new
String(request.getParameter("trade_no").getBytes("ISO-8859-
1"),"UTF-8");
        //交易状态
        String trade_status = new
String(request.getParameter("trade_status").getBytes("ISO-8859-
1"),"UTF-8");
```

```
        //appid
        String app_id = new
String(request.getParameter("app_id").getBytes("ISO-8859-1"),"UTF-
8");
        //total_amount
        String total_amount = new
String(request.getParameter("total_amount").getBytes("ISO-8859-
1"),"UTF-8");

        //交易成功处理
        if (trade_status.equals("TRADE_SUCCESS")) {

           //处理逻辑。。。

        }
    }


}
```

3.7.2 接口实现

完善 contorller 接口

```
Java
@ApiOperation("接收支付结果通知")
@PostMapping("/receivenotify")
public void receivenotify(HttpServletRequest request) throws
UnsupportedEncodingException, AlipayApiException {
    Map<String,String> params = new HashMap<String,String>();
    Map requestParams = request.getParameterMap();
    for (Iterator iter = requestParams.keySet().iterator();
iter.hasNext();) {
        String name = (String) iter.next();
        String[] values = (String[]) requestParams.get(name);
        String valueStr = "";
        for (int i = 0; i < values.length; i++) {
            valueStr = (i == values.length - 1) ? valueStr +
values[i]
                    : valueStr + values[i] + ",";
        }
        params.put(name, valueStr);
    }
```

```
    //验签
    boolean verify_result = AlipaySignature.rsaCheckV1(params,
ALIPAY_PUBLIC_KEY, AlipayConfig.CHARSET, "RSA2");

    if(verify_result) {//验证成功

        //商户订单号
        String out_trade_no = new
String(request.getParameter("out_trade_no").getBytes("ISO-8859-
1"),"UTF-8");
        //支付宝交易号
        String trade_no = new
String(request.getParameter("trade_no").getBytes("ISO-8859-
1"),"UTF-8");
        //交易状态
        String trade_status = new
String(request.getParameter("trade_status").getBytes("ISO-8859-
1"),"UTF-8");
        //appid
        String app_id = new
String(request.getParameter("app_id").getBytes("ISO-8859-1"),"UTF-
8");
        //total_amount
        String total_amount = new
String(request.getParameter("total_amount").getBytes("ISO-8859-
1"),"UTF-8");

        //交易成功处理
        if (trade_status.equals("TRADE_SUCCESS")) {

            PayStatusDto payStatusDto = new PayStatusDto();
            payStatusDto.setOut_trade_no(out_trade_no);
            payStatusDto.setTrade_status(trade_status);
            payStatusDto.setApp_id(app_id);
            payStatusDto.setTrade_no(trade_no);
            payStatusDto.setTotal_amount(total_amount);

           //处理逻辑。。。
            orderService.saveAliPayStatus(payStatusDto);
        }
    }
```

```
}
```

3.7.4 接口测试

测试准备:

1、启动网关服务、认证服务、验证码服务、学习中心服务、内容管理服务。

2、发布一门收费课程。

测试流程:

1、对选课进行支付

2、支付成功跟踪 service 方法的日志，支付成功需要更新支付交易表记录的状态、通知时间、支付宝交易号、支付渠道(Alipay)

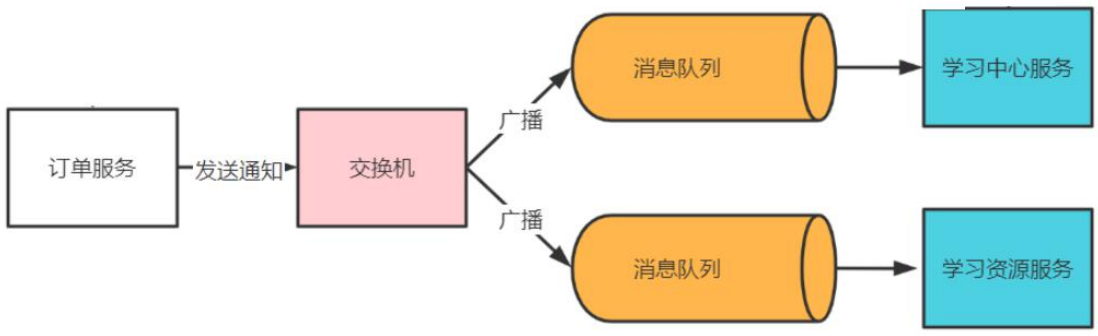
支付成功更新订单表的状态为空。

4 支付通知

4.1 需求分析

订单服务作为通用服务在订单支付成功后需要将支付结果异步通知给其它微服务。

下图使用了消息队列完成支付结果通知:



学习中心服务：对收费课程选课需要支付，与订单服务对接完成支付。

学习资源服务：对收费的学习资料需要购买后下载，与订单服务对接完成支付。

订单服务完成支付后将支付结果发给每一个与订单服务对接的微服务，订单服务将消息发给交换机，由交换机广播消息，每个订阅消息的微服务都可以接收到支付结果.

微服务收到支付结果根据订单的类型去更新自己的业务数据。

4.2 技术方案

使用消息队列进行异步通知需要保证消息的可靠性，即生产端将消息成功通知到消费端。

消息从生产端发送到消费端经历了如下过程:

1、消息发送到交换机

2、消息由交换机发送到队列

3、消息者收到消息进行处理

保证消息的可靠性需要保证以上过程的可靠性，本项目使用 RabbitMQ 可以通过如下方面保证消息的可靠性。

1、生产者确认机制

发送消息前使用数据库事务将消息保证到数据库表中

成功发送到交换机将消息从数据库中删除

2、mq 持久化

mq 收到消息进行持久化，当 mq 重启即使消息没有消费完也不会丢失。

需要配置交换机持久化、队列持久化、发送消息时设置持久化。

3、消费者确认机制

消费者消费成功自动发送 ack，否则重试消费。

4.3 发送支付结果

4.3.1 订单服务集成 MQ

订单服务通过消息队列将支付结果发给学习中心服务，消息队列采用发布订阅模式。

1、订单服务创建支付结果通知交换机。

2、学习中心服务绑定队列到交换机。

项目使用 RabbitMQ 作为消息队列，在课前下发的虚拟上已经安装了 RabbitMQ.

执行 docker start rabbitmq 启动 RabbitMQ。访问：http://192.168.101.65:15672/

账户密码：guest/guest

交换机为 Fanout 广播模式。

首先需要在学习中心服务和订单服务工程配置连接消息队列。

1、首先在订单服务添加消息队列依赖

```XML
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

2、在 nacos 配置 rabbitmq-dev.yaml 为通用配置文件

```YAML
spring:
  rabbitmq:
    host: 192.168.101.65
    port: 5672
    username: guest
    password: guest
    virtual-host: /
    publisher-confirm-type: correlated #correlated 异步回调，定义 ConfirmCallback，MQ 返回结果时会回调这个 ConfirmCallback
    publisher-returns: false #开启 publish-return 功能，同样是基于 callback 机制，需要定义 ReturnCallback
    template:
      mandatory: false #定义消息路由失败时的策略。true，则调用 ReturnCallback；false：则直接丢弃消息
    listener:
      simple:
        acknowledge-mode: none #出现异常时返回 unack，消息回滚到 mq；没有异常，返回 ack ,manual:手动控制,none:丢弃消息，不回滚到 mq
        retry:
          enabled: true #开启消费者失败重试
          initial-interval: 1000ms #初识的失败等待时长为 1 秒
          multiplier: 1 #失败的等待时长倍数，下次等待时长 = multiplier * last-interval
          max-attempts: 3 #最大重试次数
          stateless: true #true 无状态；false 有状态。如果业务中包含事务，这里改为 false
```

3、在订单服务接口工程引入 rabbitmq-dev.yaml 配置文件

```YAML
shared-configs:
  - data-id: rabbitmq-${spring.profiles.active}.yaml
    group: xuecheng-plus-common
    refresh: true
```

4、在订单服务 service 工程编写 MQ 配置类，配置交换机

```Java
package com.xuecheng.orders.config;

import com.alibaba.fastjson.JSON;
import com.xuecheng.messagesdk.model.po.MqMessage;
import com.xuecheng.messagesdk.service.MqMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.core.*;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @author Mr.M
 * @version 1.0
 * @description TODO
 * @date 2023/2/23 16:59
 */
@Slf4j
@Configuration
public class PayNotifyConfig implements ApplicationContextAware {

    //交换机
    public static final String PAYNOTIFY_EXCHANGE_FANOUT =
"paynotify_exchange_fanout";
    //支付结果通知消息类型
    public static final String MESSAGE_TYPE = "payresult_notify";
    //支付通知队列
    public static final String PAYNOTIFY_QUEUE =
"paynotify_queue";
```

```
    //声明交换机，且持久化
    @Bean(PAYNOTIFY_EXCHANGE_FANOUT)
    public FanoutExchange paynotify_exchange_fanout() {
        // 三个参数：交换机名称、是否持久化、当没有 queue 与其绑定时是否
自动删除
        return new FanoutExchange(PAYNOTIFY_EXCHANGE_FANOUT, true,
false);
    }
    //支付通知队列,且持久化
    @Bean(PAYNOTIFY_QUEUE)
    public Queue course_publish_queue() {
        return QueueBuilder.durable(PAYNOTIFY_QUEUE).build();
    }

    //交换机和支付通知队列绑定
    @Bean
    public Binding
binding_course_publish_queue(@Qualifier(PAYNOTIFY_QUEUE) Queue
queue, @Qualifier(PAYNOTIFY_EXCHANGE_FANOUT) FanoutExchange
exchange) {
        return BindingBuilder.bind(queue).to(exchange);
    }

    @Override
    public void setApplicationContext(ApplicationContext
applicationContext) throws BeansException {
        // 获取 RabbitTemplate
        RabbitTemplate rabbitTemplate =
applicationContext.getBean(RabbitTemplate.class);
        //消息处理 service
        MqMessageService mqMessageService =
applicationContext.getBean(MqMessageService.class);
        // 设置 ReturnCallback
        rabbitTemplate.setReturnCallback((message, replyCode,
replyText, exchange, routingKey) -> {
            // 投递失败，记录日志
            log.info("消息发送失败，应答码{}，原因{}，交换机{}，路由键
{},消息{}",
                    replyCode, replyText, exchange, routingKey,
message.toString());
            MqMessage mqMessage =
JSON.parseObject(message.toString(), MqMessage.class);
```

```
            //将消息再添加到消息表

mqMessageService.addMessage(mqMessage.getMessageType(),mqMessage.g
etBusinessKey1(),mqMessage.getBusinessKey2(),mqMessage.getBusiness
Key3());

        });
    }
}
```

重启订单服务，登录 rabbitmq，查看交换机自动创建成功

查看队列自动成功

4.3.2 发送支付结果

在 OrderService 中定义接口

```
Java
/**
 * 发送通知结果
 * @param message
 */
public void notifyPayResult(MqMessage message);
```

编写接口实现方法:

```
Java
@Override
public void notifyPayResult(MqMessage message) {

    //1、消息体，转 json
    String msg = JSON.toJSONString(message);
    //设置消息持久化
    Message msgObj =
MessageBuilder.withBody(msg.getBytes(StandardCharsets.UTF_8))
            .setDeliveryMode(MessageDeliveryMode.PERSISTENT)
            .build();
    // 2.全局唯一的消息 ID，需要封装到 CorrelationData 中
    CorrelationData correlationData = new
```

```
CorrelationData(message.getId().toString());
    // 3.添加 callback
    correlationData.getFuture().addCallback(
            result -> {
                if(result.isAck()){
                    // 3.1.ack，消息成功
                    log.debug("通知支付结果消息发送成功, ID:{}",
correlationData.getId());
                    //删除消息表中的记录
                    mqMessageService.completed(message.getId());
                }else{
                    // 3.2.nack，消息失败
                    log.error("通知支付结果消息发送失败, ID:{}, 原因
{}",correlationData.getId(), result.getReason());
                }
            },
            ex -> log.error("消息发送异常, ID:{}, 原因
{}",correlationData.getId(),ex.getMessage())
    );
    // 发送消息

rabbitTemplate.convertAndSend(PayNotifyConfig.PAYNOTIFY_EXCHANGE_F
ANOUT, "", msgObj,correlationData);

}
```

订单服务收到第三方平台的支付结果时，在 saveAliPayStatus 方法中添加代码，向数据库消息表添加消息并进行发送消息，如下所示：

```
Java
@Transactional
@Override
public void saveAliPayStatus(PayStatusDto payStatusDto) {
        .......
        //保存消息记录,参数 1：支付结果通知类型，2: 业务 id，3:业务类型
        MqMessage mqMessage =
mqMessageService.addMessage("payresult_notify",
orders.getOutBusinessId(), orders.getOrderType(), null);
        //通知消息
        notifyPayResult(mqMessage);
    }
}
```

配置交换机和队列

在 order-service 工程配置

消息发送方法

```Java
/**
 * 发送通知结果
 * @param message
 */
public void notifyPayResult(MqMessage message);
```

4.4 接收支付结果

4.4.1 学习中心服务集成 MQ

1、在学习中心服务添加消息队列依赖

```XML
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

2、在学习中心服务接口工程引入 rabbitmq-dev.yaml 配置文件

```YAML
shared-configs:
  - data-id: rabbitmq-${spring.profiles.active}.yaml
    group: xuecheng-plus-common
```

```
    refresh: true
```

3、添加配置类

```
Java
package com.xuecheng.learning.config;

import com.alibaba.fastjson.JSON;
import com.xuecheng.messagesdk.model.po.MqMessage;
import com.xuecheng.messagesdk.service.MqMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.core.*;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @author Mr.M
 * @version 1.0
 * @description TODO
 * @date 2023/2/23 16:59
 */
@Slf4j
@Configuration
public class PayNotifyConfig {

    //交换机
    public static final String PAYNOTIFY_EXCHANGE_FANOUT =
"paynotify_exchange_fanout";
    //支付结果通知消息类型
    public static final String MESSAGE_TYPE = "payresult_notify";
    //支付通知队列
    public static final String PAYNOTIFY_QUEUE =
"paynotify_queue";

    //声明交换机，且持久化
    @Bean(PAYNOTIFY_EXCHANGE_FANOUT)
    public FanoutExchange paynotify_exchange_fanout() {
        // 三个参数：交换机名称、是否持久化、当没有 queue 与其绑定时是否
自动删除
```

```
        return new FanoutExchange(PAYNOTIFY_EXCHANGE_FANOUT, true,
false);
    }
    //支付通知队列,且持久化
    @Bean(PAYNOTIFY_QUEUE)
    public Queue course_publish_queue() {
        return QueueBuilder.durable(PAYNOTIFY_QUEUE).build();
    }

    //交换机和支付通知队列绑定
    @Bean
    public Binding
binding_course_publish_queue(@Qualifier(PAYNOTIFY_QUEUE) Queue
queue, @Qualifier(PAYNOTIFY_EXCHANGE_FANOUT) FanoutExchange
exchange) {
        return BindingBuilder.bind(queue).to(exchange);
    }

}
```

4.4.2 接收支付结果

监听 MQ，接收支付结果，定义 ReceivePayNotifyService 类如下：

```
Java
package com.xuecheng.learning.service.impl;

import com.alibaba.fastjson.JSON;
import com.rabbitmq.client.Channel;
import com.xuecheng.base.exception.XueChengPlusException;
import com.xuecheng.learning.config.PayNotifyConfig;
import com.xuecheng.learning.service.MyCourseTablesService;
import com.xuecheng.messagesdk.model.po.MqMessage;
import com.xuecheng.messagesdk.service.MqMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.io.IOException;

/**
```

```
 * @author Mr.M
 * @version 1.0
 * @description 接收支付结果
 * @date 2023/2/23 19:04
 */
@Slf4j
@Service
public class ReceivePayNotifyService {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Autowired
    MqMessageService mqMessageService;

    @Autowired
    MyCourseTablesService myCourseTablesService;


    //监听消息队列接收支付结果通知
    @RabbitListener(queues = PayNotifyConfig.PAYNOTIFY_QUEUE)
    public void receive(Message message, Channel channel) {
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        //获取消息
        MqMessage mqMessage = JSON.parseObject(message.getBody(),
MqMessage.class);
        log.debug("学习中心服务接收支付结果:{}", mqMessage);

        //消息类型
        String messageType = mqMessage.getMessageType();
        //订单类型,60201 表示购买课程
        String businessKey2 = mqMessage.getBusinessKey2();
        //这里只处理支付结果通知
        if (PayNotifyConfig.MESSAGE_TYPE.equals(messageType) &&
"60201".equals(businessKey2)) {
            //选课记录 id
            String choosecourseId = mqMessage.getBusinessKey1();
            //添加选课
            boolean b =
```

```
myCourseTablesService.saveChooseCourseStauts(choosecourseId);
            if(!b){
                //添加选课失败，抛出异常，消息重回队列
                XueChengPlusException.cast("收到支付结果，添加选课失败
");
            }
        }


    }


}
```

4.5 通知支付结果测试

测试准备:

1、找一门已发布的收费课程。

2、如果在我的课程表存储则删除。

3、删除此课程的选课记录及订单信息。

测试流程:

1、进入课程详细页面，点击马上学习，生成二维码进行支付。

2、支付完成点击“支付完成”，观察订单服务控制台是否发送消息。

3、观察学习中心服务控制台是否接收到消息。

4、观察数据库中的消息表的相应记录是否已删除。

消费重试测试:

1、在学习中心服务接收支付结果方法中制造异常。

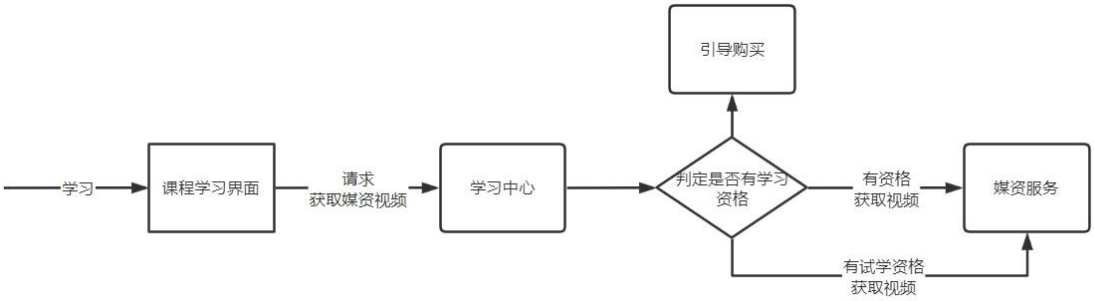
2、重新执行上边的测试流程，观察是否消费重试。

4 在线学习

4.1 需求分析

用户通过课程详情界面点击马上学习　进入 视频播放界面进行视频点播。

获取视频资源时进行学习资格校验，如下图：



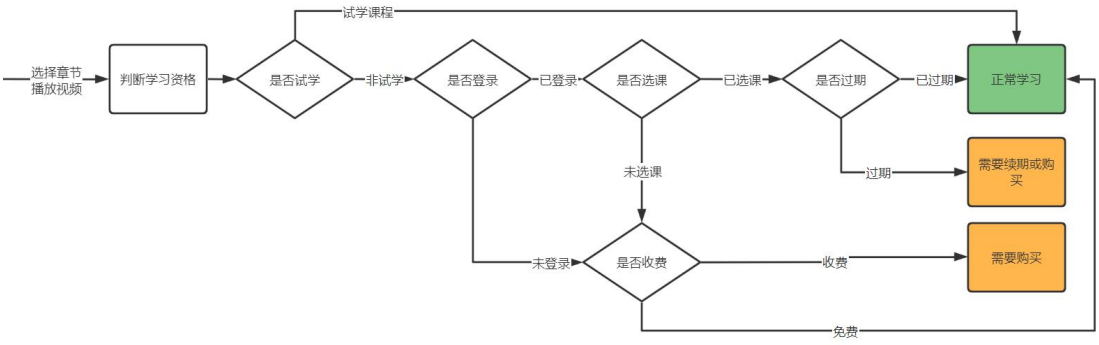
拥有学习资格则继续播放视频，不具有学习资格则引导去购买、续期等操作。

如何判断是否拥有学习资格?

首先判断是否为试学视频，如果为试学视频则可以正常学习。

如果为非试学课程首先判断用户是否登录，如果已登录则判断是否选课，如果已经选课且没有过期可以正常学习。

详细流程如下图：



4.2 查询课程信息

在视频点播页面需要查询课程信息，课程上线后也需要访问/api/content/course/whole/{courseId}

课程预览时请求获取课程的接口为：/open/content/course/whole/{courseId}

在 nginx 中进行配置：

/open、/api 在 nginx 的配置如下：(已经配置的不要重复配置)

```
Plain Text
       #api
       location /api/ {
              proxy_pass http://gatewayserver/;
       }
       #openapi
       location /open/content/ {
              proxy_pass http://gatewayserver/content/open/;
       }
       location /open/media/ {
              proxy_pass http://gatewayserver/media/open/;
       }
```

下边实现/api/content/course/whole/{courseId} 获取课程发布信息接口。

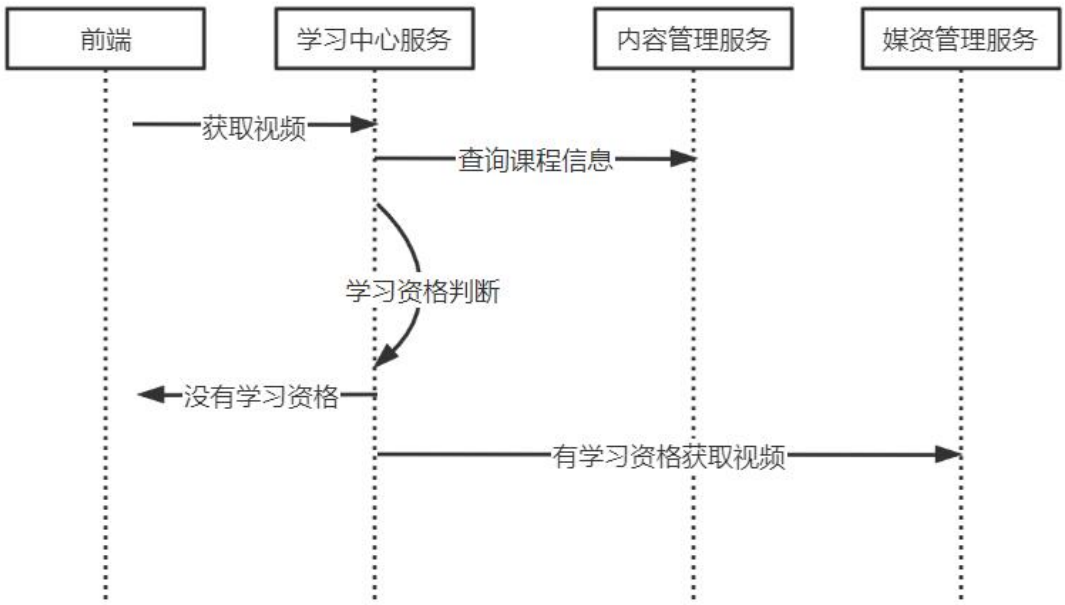
进入内容管理服务 api 工程 CoursePublishController 类，定义查询课程预览信息接口如下：

```
Java
@ApiOperation("获取课程发布信息")
@ResponseBody
@GetMapping("/course/whole/{courseId}")
public CoursePreviewDto getCoursePublish(@PathVariable("courseId")
Long courseId) {
    //查询课程发布信息
    CoursePublish coursePublish =
coursePublishService.getCoursePublish(courseId);
    if (coursePublish == null) {
       return new CoursePreviewDto();
     }

    //课程基本信息
    CourseBaseInfoDto courseBase = new CourseBaseInfoDto();
    BeanUtils.copyProperties(coursePublish, courseBase);
    //课程计划
    List<TeachplanDto> teachplans =
JSON.parseArray(coursePublish.getTeachplan(), TeachplanDto.class);
    CoursePreviewDto coursePreviewInfo = new CoursePreviewDto();
    coursePreviewInfo.setCourseBase(courseBase);
    coursePreviewInfo.setTeachplans(teachplans);
    return coursePreviewInfo;
}
```

重启内容管理服务，进入学习界面查看课程计划、课程名称等信息是否显示正常。

4.3 获取视频

4.3.1 需求分析



4.3.2 接口定义

```
Java
package com.xuecheng.learning.api;

import com.xuecheng.base.execption.XueChengPlusException;
import com.xuecheng.base.model.RestResponse;
import com.xuecheng.base.model.XcUser;
import com.xuecheng.learning.model.dto.XcChooseCourseDto;
import com.xuecheng.learning.model.dto.XcCourseTablesDto;
import com.xuecheng.learning.service.LearningService;
import com.xuecheng.learning.service.MyCourseTablesService;
import com.xuecheng.learning.util.SecurityUtil;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @author Mr.M
 * @version 1.0
 * @description 学习过程管理接口
 * @date 2022/10/2 14:52
 */
@Api(value = "学习过程管理接口", tags = "学习过程管理接口")
@Slf4j
@RestController
public class MyLearningController {

    @Autowired
    LearningService learningService;


    @ApiOperation("获取视频")

@GetMapping("/open/learn/getvideo/{courseId}/{teachplanId}/{mediaI
d}")
    public RestResponse<String> getvideo(@PathVariable("courseId")
Long courseId,@PathVariable("courseId") Long teachplanId,
@PathVariable("mediaId") String mediaId) {
        //登录用户
        XcUser user = SecurityUtil.getUser();
        String userId = null;
        if(user != null){
            userId = user.getId();
        }
        //获取视频


    }

}
```

定义 service 接口

```
Java
```

```
package com.xuecheng.learning.service;

import com.xuecheng.base.model.RestResponse;
import com.xuecheng.learning.model.dto.XcChooseCourseDto;
import com.xuecheng.learning.model.dto.XcCourseTablesDto;

/**
 * @description 学习过程管理 service 接口
 * @author Mr.M
 * @date 2022/10/2 16:07
 * @version 1.0
 */
public interface LearningService {

/**
 * @description 获取教学视频
 * @param courseId 课程 id
 * @param teachplanId 课程计划 id
 * @param mediaId 视频文件 id
 * @return com.xuecheng.base.model.RestResponse<java.lang.String>
 * @author Mr.M
 * @date 2022/10/5 9:08
*/
public RestResponse<String> getVideo(String userId,Long
courseId,Long teachplanId,String mediaId);
}
```

4.3.3 获取视频远程接口

在学习中心服务 service 工程中定义媒资管理 Feignclient

```
Java
package com.xuecheng.learning.feignclient;

import com.xuecheng.base.model.RestResponse;
import com.xuecheng.content.model.po.CoursePublish;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

/**
 * @description 媒资管理服务远程接口
```

```
 * @author Mr.M
 * @date 2022/9/20 20:29
 * @version 1.0
 */
 @FeignClient(value = "media-api",fallbackFactory =
MediaServiceClientFallbackFactory.class)
 @RequestMapping("/media")
public interface MediaServiceClient {

 @GetMapping("/open/preview/{mediaId}")
 public RestResponse<String>
getPlayUrlByMediaId(@PathVariable("mediaId") String mediaId);

 }
```

FeignClient 接口的降级类:

```
Java

 package com.xuecheng.learning.feignclient;

import com.xuecheng.base.model.RestResponse;
import com.xuecheng.content.model.po.CoursePublish;
import feign.hystrix.FallbackFactory;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;

/**
 * @author Mr.M
 * @version 1.0
 * @description TODO
 * @date 2022/10/3 8:03
 */
@Slf4j
@Component
public class MediaServiceClientFallbackFactory implements
FallbackFactory<MediaServiceClient> {
    @Override
    public MediaServiceClient create(Throwable throwable) {
        return new MediaServiceClient() {
            @Override
            public RestResponse<String> getPlayUrlByMediaId(String
mediaId) {
```

```
                log.error("远程调用媒资管理服务熔断异常：
{}",throwable.getMessage());
                return null;
            }
        };
    }
}
```

4.3.5 学习资格校验

编写获取视频的接口实现方法：

```
Java
@Override
public RestResponse<String> getVideo(String userId,Long
courseId,Long teachplanId, String mediaId) {
    //查询课程信息
    CoursePublish coursepublish =
contentServiceClient.getCoursepublish(courseId);
    if(coursepublish==null){
        XueChengPlusException.cast("课程信息不存在");
    }
    //校验学习资格

    //如果登录
    if(StringUtils.isNotEmpty(userId)){

        //判断是否选课，根据选课情况判断学习资格
        XcCourseTablesDto xcCourseTablesDto =
myCourseTablesService.getLeanringStatus(userId, courseId);
        //学习资格状态 [{"code":"702001","desc":"正常学习
"},{"code":"702002","desc":"没有选课或选课后没有支付
"},{"code":"702003","desc":"已过期需要申请续期或重新支付"}]
        String learnStatus = xcCourseTablesDto.getLearnStatus();
        if(learnStatus.equals("702001")){
            return mediaServiceClient.getPlayUrlByMediaId(mediaId);
        }else if(learnStatus.equals("702003")){
            RestResponse.validfail("您的选课已过期需要申请续期或重新支
付");
        }
    }
```

```
    //未登录或未选课判断是否收费
    String charge = coursepublish.getCharge();
    if(charge.equals("201000")){//免费可以正常学习
        return mediaServiceClient.getPlayUrlByMediaId(mediaId);
    }

    return RestResponse.validfail("请购买课程后继续学习");


}
```

4.3.6 测试

1、完善接口

```
Java
@ApiOperation("获取视频")
@GetMapping("/open/learn/getvideo/{courseId}/{teachplanId}/{mediaI
d}")
public RestResponse<String> getvideo(@PathVariable("courseId")
Long courseId, @PathVariable("courseId") Long teachplanId,
@PathVariable("mediaId") String mediaId) {

    //登录用户
    SecurityUtil.XcUser user = SecurityUtil.getUser();
    String userId = null;
    if (user != null) {
        userId = user.getId();
    }
    //获取视频
    return learningService.getVideo(userId, courseId, teachplanId,
mediaId);

}
```

2、测试准备

选课成功一门课程。

没有选课的免费课程、收费课程各一门，其中收费课程具有试学课程。

3、测试项目

1）选课成功的课程是否可以正常获取视频

2）免费课程没有选课是否可以正常学习

可修改选课记录表中的课程 id 为不存在进行测试，测试完再恢复原样。

3）收费课程没有选课是否可以正常学习

可修改选课记录表中的课程 id 为不存在进行测试，测试完再恢复原样。

4.4 我的课表

4.4.1 需求分析

4.4.1.1 业务流程

登录网站，点击“我的学习”进入个人中心，

个人中心首页显示我的课程表：

我的课表中显示了选课成功的免费课程、收费课程。最近学习课程显示了当前用户最近学习的课程信息。

点击继续学习进入当前学习章节的视频继续学习。

点击课程评价进入课程评价界面。

4.4.1.2 配置 nginx

在 nginx 配置用户中心 server ,如下:

```
XML
    server {
        listen       80;
        server_name  ucenter.51xuecheng.cn;
        #charset koi8-r;
        ssi on;
        ssi_silent_errors on;
        #access_log  logs/host.access.log  main;
        location / {
            alias   D:/itcast2022/xc_edu3.0/code_1/xc-ui-pc-static-
portal/ucenter/;
            index  index.html index.htm;
        }
        location /include {
            proxy_pass   http://127.0.0.1;
        }
        location /img/ {
            proxy_pass   http://127.0.0.1/static/img/;
        }
        location /api/ {
                proxy_pass http://gatewayserver/;
        }
   }
```

4.4.2 接口定义

在 MyCourseTablesController 中定义我的课程表接口:

```
Java
@ApiOperation("我的课程表")
@GetMapping("/mycoursetable")
public PageResult<XcCourseTables>
mycoursetable(MyCourseTableParams params) {


}
```

4.4.3 接口开发

4.4.3.1DAO

使用自动生成的 mapper 即可实现分页查询。

4.4.3.2 Service

在 service 中定义我的课程表接口：

```
Java
/**
 * @description 我的课程表
 * @param params
 * @return
com.xuecheng.base.model.PageResult<com.xuecheng.learning.model.po.
XcCourseTables>
 * @author Mr.M
 * @date 2022/10/27 9:24
*/
public PageResult<XcCourseTables>
mycourestabls(MyCourseTableParams params);
```

编写接口实现：

```
Java
public PageResult<XcCourseTables>
mycourestabls( MyCourseTableParams params){
    //页码
    long pageNo = params.getPage();
    //每页记录数,固定为 4
    long pageSize = 4;
    //分页条件
    Page<XcCourseTables> page = new Page<>(pageNo, pageSize);
    //根据用户 id 查询
    String userId = params.getUserId();
    LambdaQueryWrapper<XcCourseTables> lambdaQueryWrapper = new
LambdaQueryWrapper<XcCourseTables>().eq(XcCourseTables::getUserId,
userId);

    //分页查询
    Page<XcCourseTables> pageResult =
courseTablesMapper.selectPage(page, lambdaQueryWrapper);
    List<XcCourseTables> records = pageResult.getRecords();
    //记录总数
    long total = pageResult.getTotal();
```

```
    PageResult<XcCourseTables> courseTablesResult = new
PageResult<>(records, total, pageNo, pageSize);
    return courseTablesResult;

}
```

完善接口：

```
Java
@ApiOperation("我的课程表")
@GetMapping("/mycoursetable")
public PageResult<XcCourseTables>
mycoursetable(MyCourseTableParams params) {
 //登录用户
 SecurityUtil.XcUser user = SecurityUtil.getUser();
 if(user == null){
  XueChengPlusException.cast("请登录后继续选课");
 }
 String userId = user.getId();
//设置当前的登录用户
 params.setUserId(userId);

  return myCourseTablesService.mycourestabls(params);
}
```

4.4.4 接口测试

登录网站，点击“我的学习”进入个人中心，查看我的课程表中课程是否是当前用户所选课程。

个人中心

梦醒时分

- 我的课程 >
- 我的订单 >
- 我的收藏 >
- 修改信息 >
- 退出 >

最近学习课程

HOT 继续学习 程序语言设计 学习中

正在学习 使用对象 有效日期: 2017.06.05 - 2018.06.05

1/4 已完成部分 进度25%

继续学习

课程评价

全部课表

按学习时间进行排序 | 按加入时间进行排序 全部 | 付费 | 即将过期 | 失效

HOT 继续学习 程序语言设计 学习中

正在学习 使用对象

有效日期: 2017.06.05 - 2018.06.05

1/4 已完成部分 进度 25%

继续学习 课程评价

HOT 继续学习 程序语言设计 学习中

正在学习 使用对象

有效日期: 2017.06.05 - 2018.06.05

1/4 已完成部分 进度 25%

继续学习 课程评价