**第6章选课学习**

**1 模块需求分析**

**1.1 模块介绍**

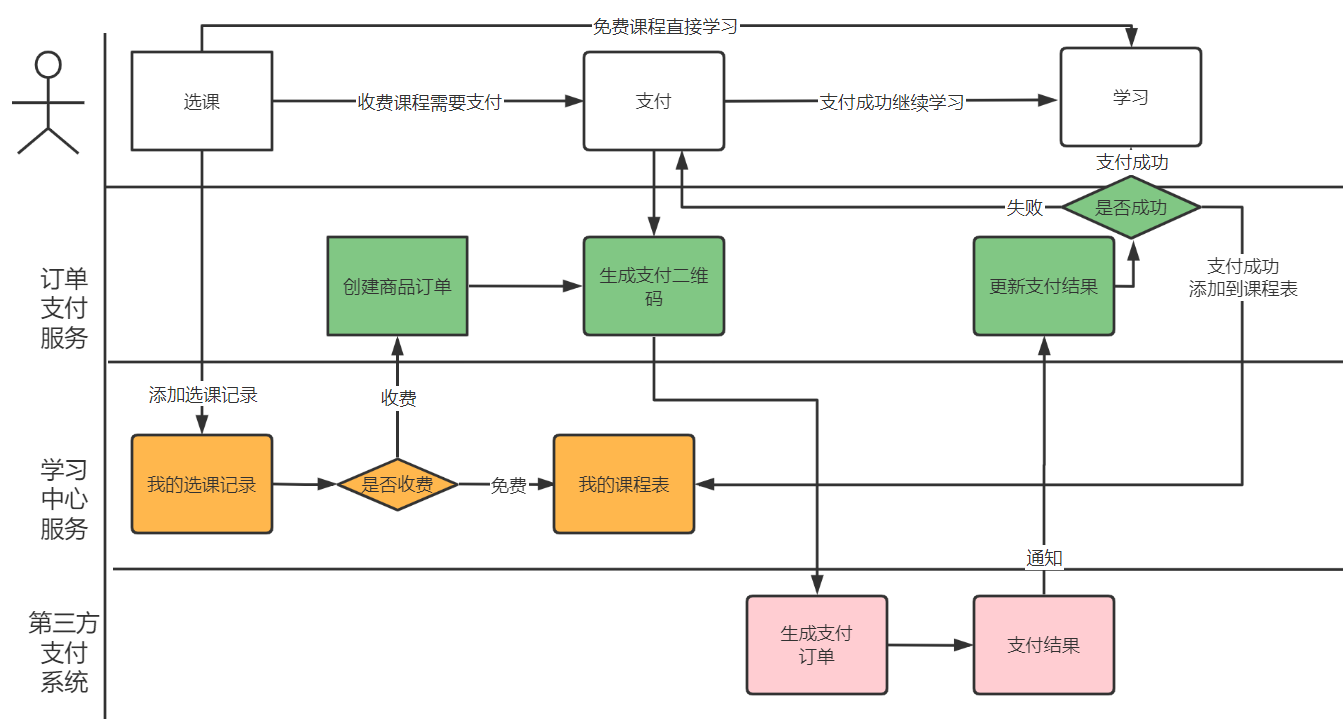
本模块实现了学生选课、下单支付、学习的整体流程。

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：

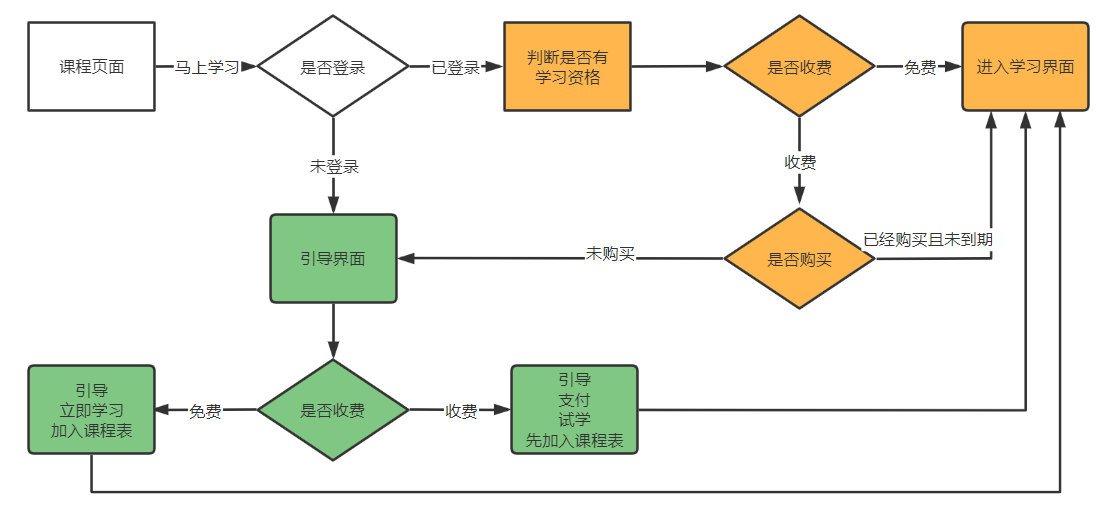


**1.2 业务流程**

**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：



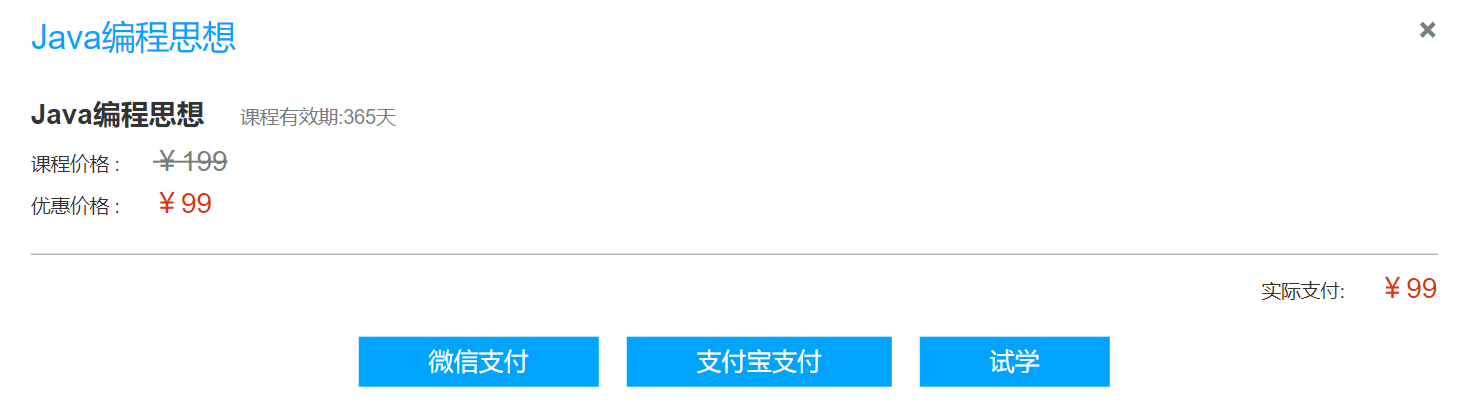
1、进入课程详情点击马上学习



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。

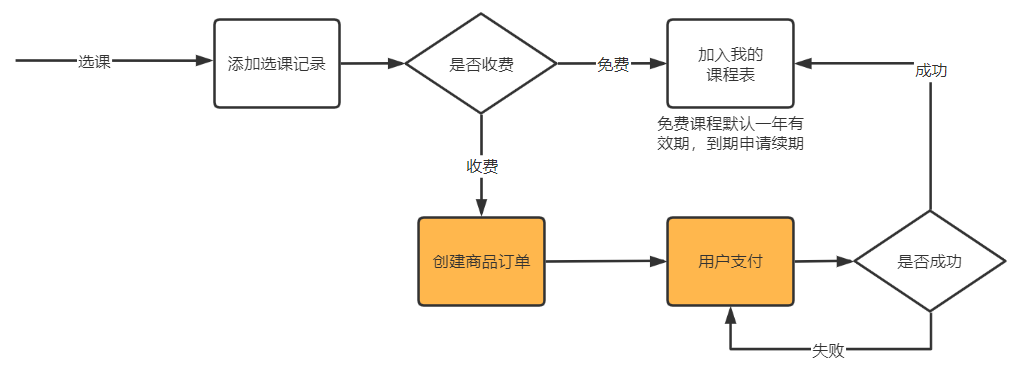


**1.2.2 选课流程**

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。

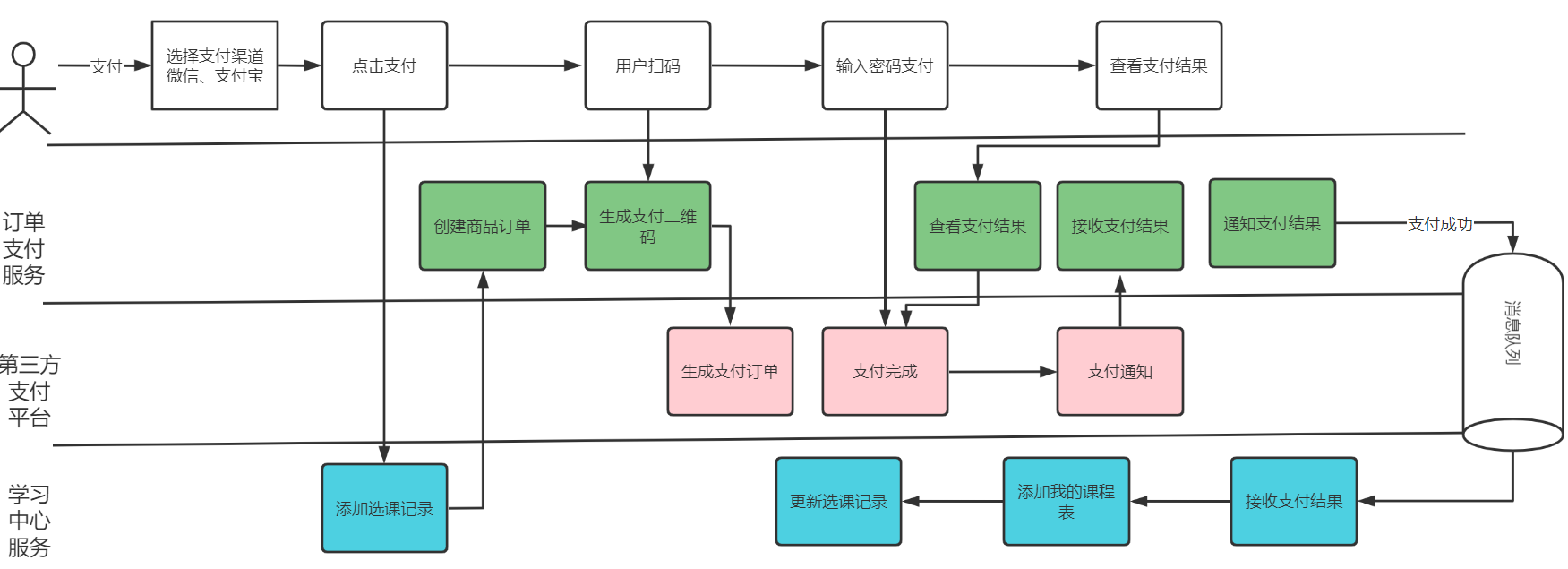
流程如下：



**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。

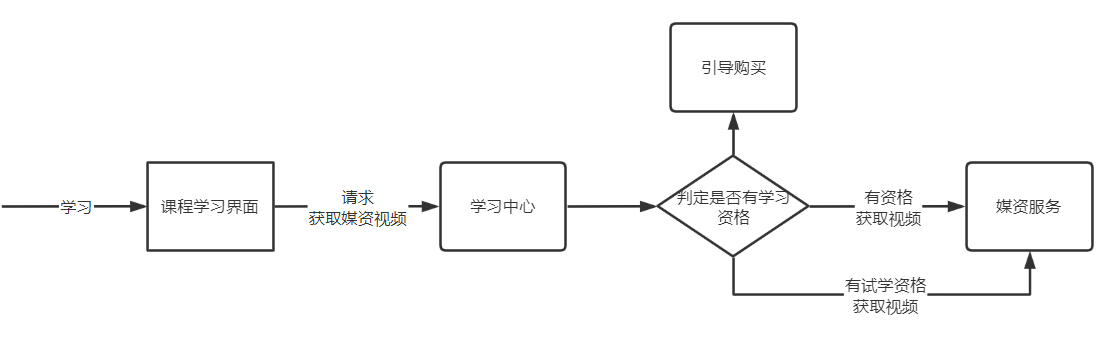
流程如下：



**1.2.4 在线学习**

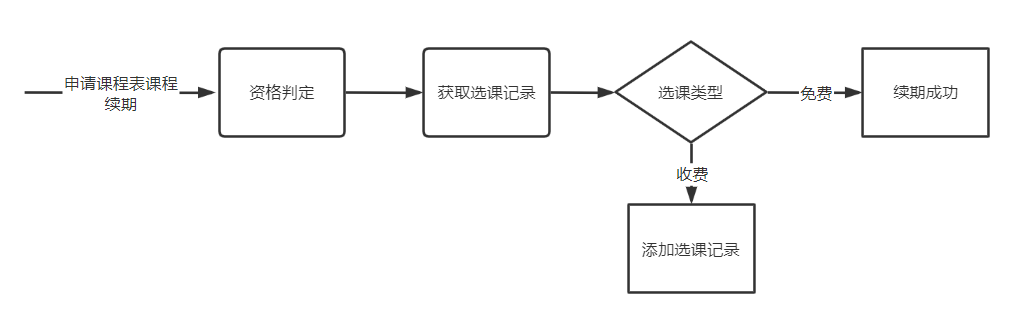
选课成功用户使可以在线学习，对于免费课程无需选课即可在线学习。

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：

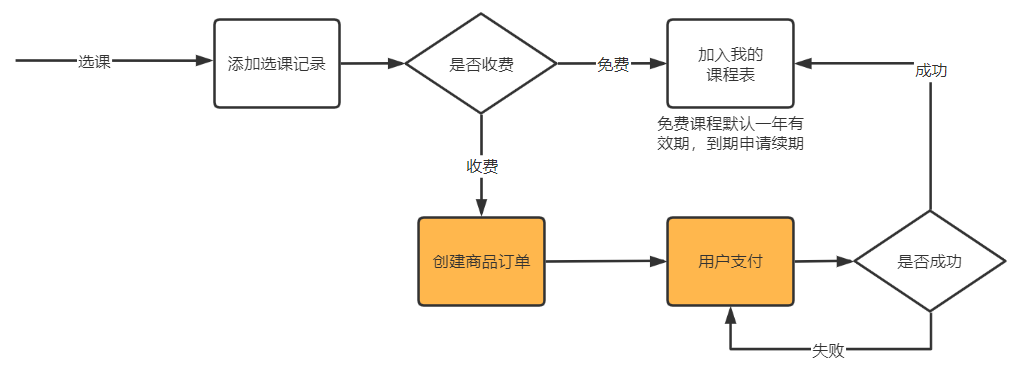


**2 添加选课**

**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：



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。

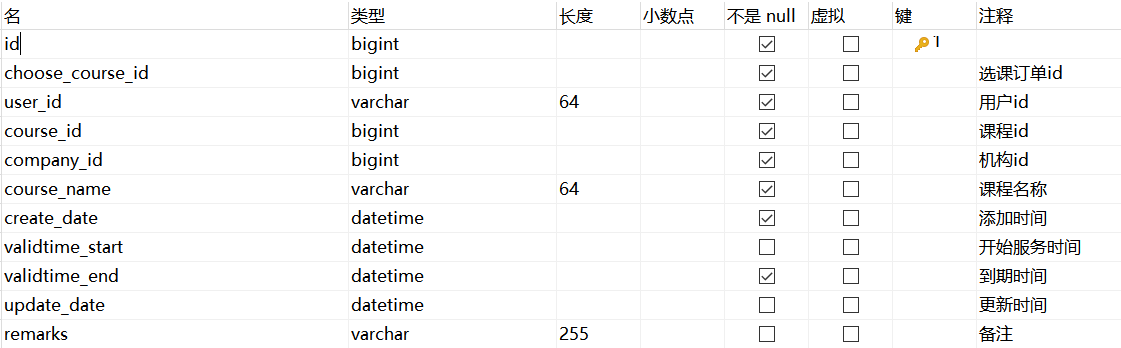
收费课程的选课记录需要支付成功后选课状态为成功。

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。



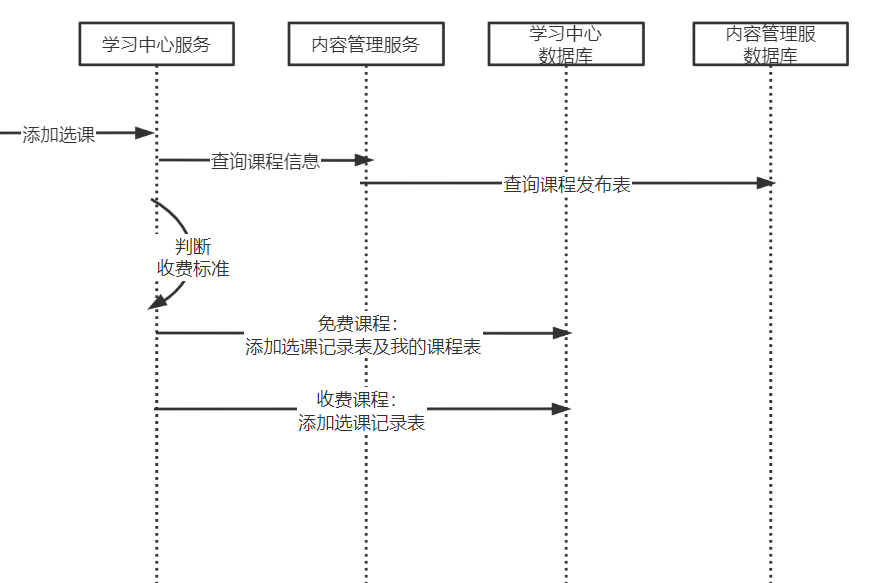
**2.1.2 执行流程**

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。

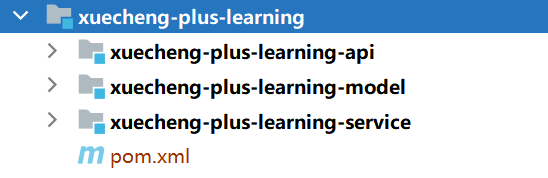
这里首先实现将免费课程加入我的课程表，执行流程如下：



**2.2 接口开发**

**2.2.1 创建学习中心工程**

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。

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=UTC&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 |

**2.2.3 远程调用查询课程信息接口**

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。

在学生中心service工程添加Feign接口：

|  |
| --- |
| Java package com.xuecheng.learning.feignclient;  import com.xuecheng.content.model.po.CoursePublish; import org.springframework.cloud.openfeign.FeignClient; import org.springframework.web.bind.annotation.\*;  /\*\*  \* @description 内容管理服务远程接口  \* @author Mr.M  \* @date 2022/9/20 20:29  \* @version 1.0  \*/ @FeignClient(value = "content-api", fallbackFactory = ContentServiceClientFactory.class) @RequestMapping("/content") public interface ContentServiceClient {   @ResponseBody  @GetMapping("/r/coursepublish/{courseId}")  public CoursePublish getCoursepublish(@PathVariable("courseId") Long courseId); } |

编写降级方法

|  |
| --- |
| Java package com.xuecheng.learning.feignclient;  import com.xuecheng.content.model.po.CoursePublish; import feign.hystrix.FallbackFactory; import lombok.extern.slf4j.Slf4j; import org.springframework.stereotype.Component;  /\*\*  \* @description 内容管理服务远程接口降级类  \* @author Mr.M  \* @date 2022/10/24 17:13  \* @version 1.0  \*/ @Slf4j @Component public class ContentServiceClientFactory implements FallbackFactory<ContentServiceClient> {  @Override  public ContentServiceClient create(Throwable throwable) {   return new ContentServiceClient(){   @Override  public CoursePublish getCoursepublish(Long courseId) {  *log*.error("调用内容管理服务接口熔断:{}",throwable.getMessage());  return null;  }  };  } } |

启动类添加@EnableFeignClients(basePackages={"com.xuecheng.\*.feignclient"})

|  |
| --- |
| Java package com.xuecheng;   import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.openfeign.EnableFeignClients;  @EnableFeignClients(basePackages={"com.xuecheng.\*.feignclient"}) @SpringBootApplication public class LearningApplication {  public static void main(String[] args) {  SpringApplication.*run*(LearningApplication.class, args);  } } |

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);  } } |

**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;   @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();  if("201000".equals(charge)){//课程免费  //添加免费课程  XcChooseCourse xcChooseCourse = currentProxy.addFreeCoruse(userId, coursepublish);  }else{  //添加收费课程  XcChooseCourse xcChooseCourse = currentProxy.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 @Transactional public XcChooseCourse addFreeCoruse(String userId, CoursePublish coursepublish) {    return null; }  //添加收费课程 @Transactional public XcChooseCourse addChargeCoruse(String userId,CoursePublish coursepublish){   return null; }  } |

**2.2.5.3 添加免费课程**

|  |
| --- |
| Java @Transactional //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);  //添加到我的课程表    return xcChooseCourse;  } |

**2.2.5.4 添加到我的课程表**

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 \*/ @Transactional 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.getValidtimeStart());  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 @Transactional //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(coursepublish.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 getLeanringStatus(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 getLeanringStatus(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;  }  } |

**2.2.5.5 接口完善**

1、完善Service接口

|  |
| --- |
| Java @Override public XcChooseCourseDto addChooseCourse(String userId,Long courseId) {  //查询课程信息  CoursePublish coursepublish = contentServiceClient.getCoursepublish(courseId);  if(coursepublish==null){  XueChengPlusException.*cast*("课程信息不存在");  }  Long id = coursepublish.getId();  if(id==null){  XueChengPlusException.*cast*(CommonError.*UNKOWN\_ERROR*);  }  //课程收费标准  String charge = coursepublish.getCharge();   XcChooseCourse xcChooseCourse = null;  if(charge.equals("201001")){  //添加收费课程  xcChooseCourse= myCourseTablesService.addChargeCoruse(userId,coursepublish);  }else{  //添加免费课程  xcChooseCourse= myCourseTablesService.addFreeCoruse(userId,coursepublish);  }   XcChooseCourseDto xcChooseCourseDto = new XcChooseCourseDto();  BeanUtils.*copyProperties*(xcChooseCourse,xcChooseCourseDto);  //获取学习资格  XcCourseTablesDto xcCourseTablesDto = getLeanringStatus(userId, courseId);  xcChooseCourseDto.setLearnStatus(xcCourseTablesDto.getLearnStatus());  return xcChooseCourseDto; } |

2、完善controller

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

**2.2.5 接口测试**

1、准备测试环境

保证课程发布有已发布的课程

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。

使用httpclient

|  |
| --- |
| Bash ### 添加选课 POST {{learning\_host}}/learning/choosecourse/2 *Authorization*: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..b77ZreiNlPoN-\_dnAWxuBfH32tPIoRwg2ePgKn\_aZ8c |

**2.3 前后端联调**

**2.3.1 查询学习资格**

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。



这里需要查询用户的学习资格，下边编写查询学习资格的接口

|  |
| --- |
| Java @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);  } |

**2.3.2 测试**

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

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。

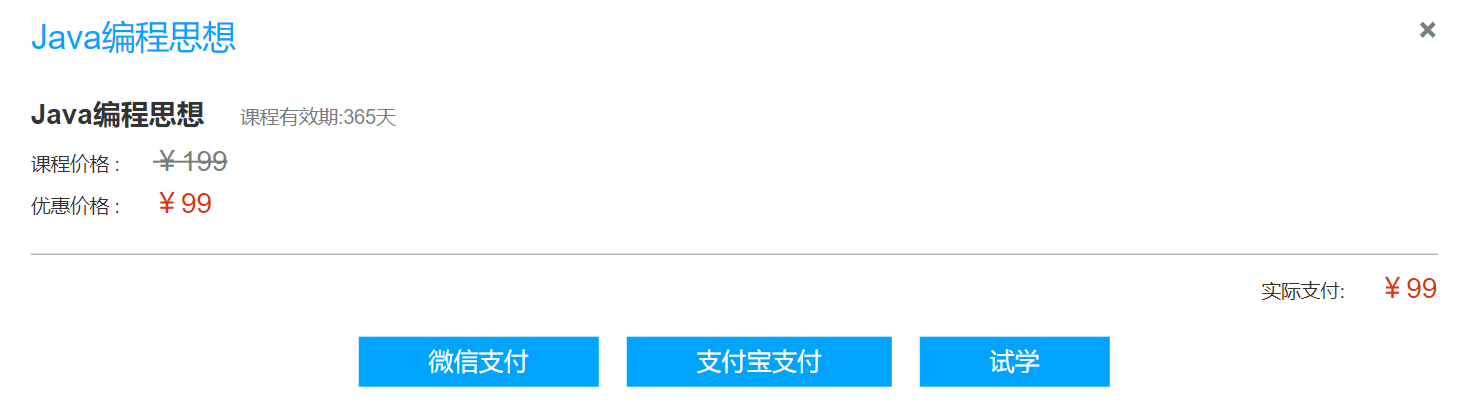
5、观察选课记录表、我的课程表数据是否正确。

**3 支付**

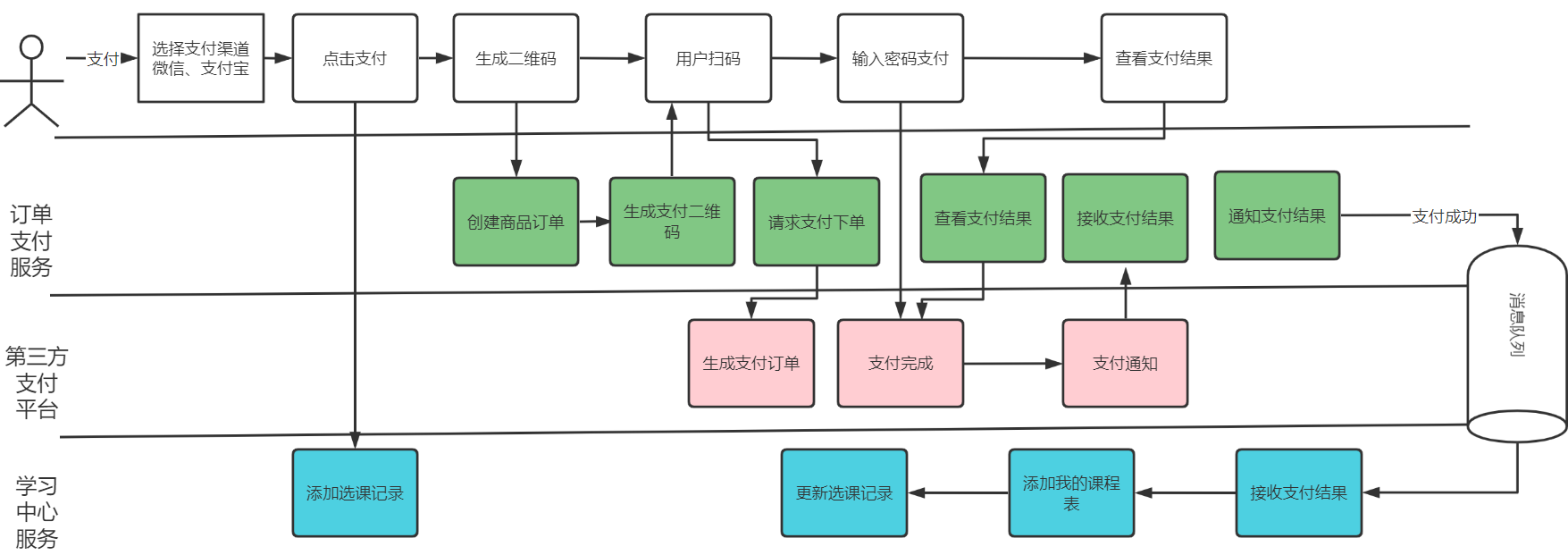
**3.1 需求分析**

**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，如果订单支付服务还没有收到支付结果则请求学习中心查询支付结果。

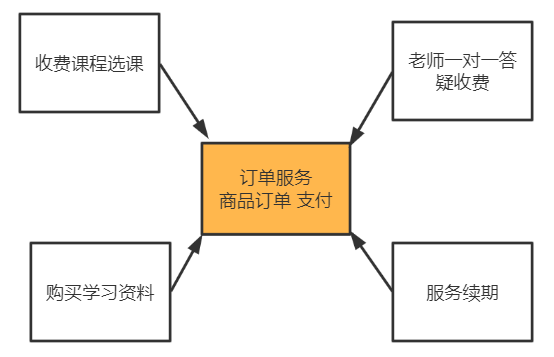
8、订单支付服务向学习中心服务通知支付结果。

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。

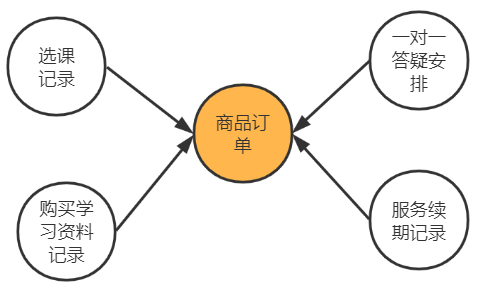
**3.1.2 通用订单服务设计**

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。

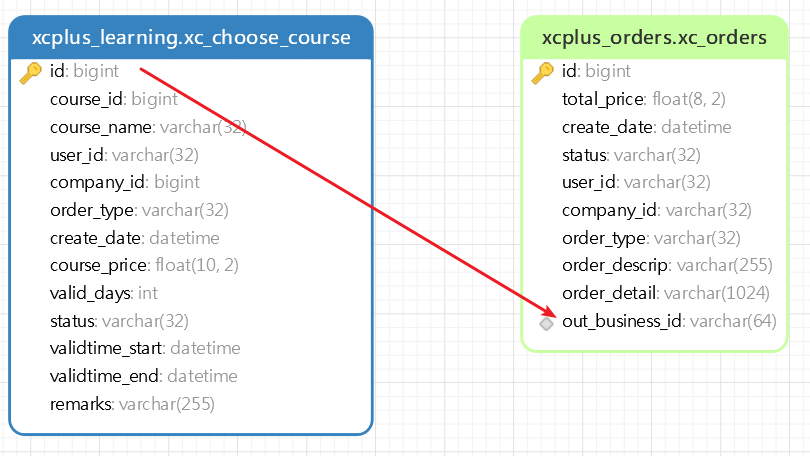
所以本项目设计通用的订单服务，通用的订单服务承接各业务模块的收费支付需求，当用户需要交费时统一生成商品订单并进行支付。



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。

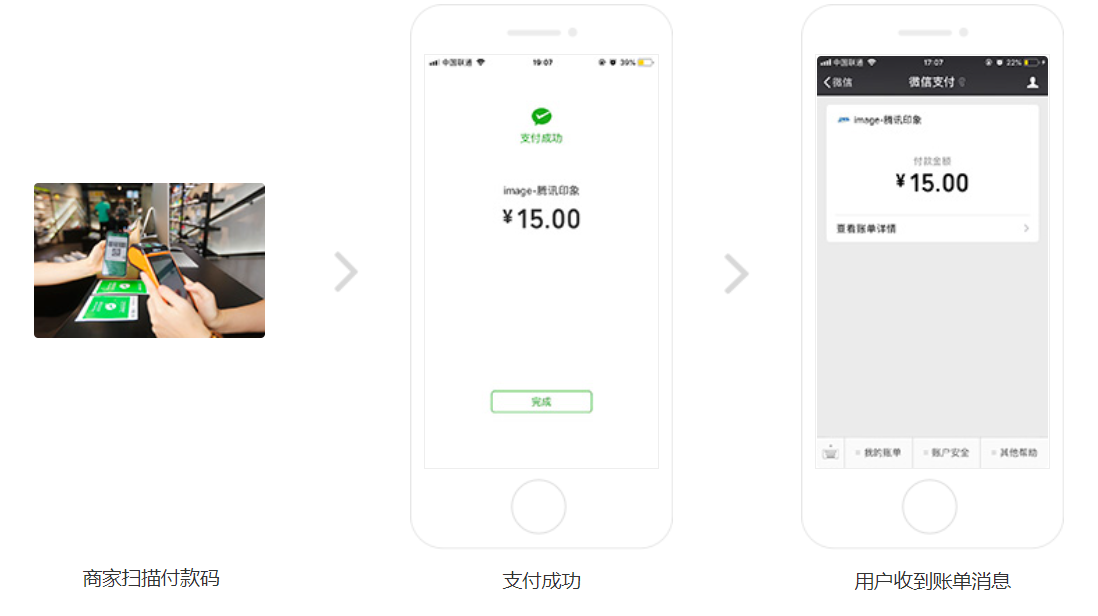
本项目的需求实现手机扫码支付，现在对微信、支付宝的支付接口进行调研。

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。



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

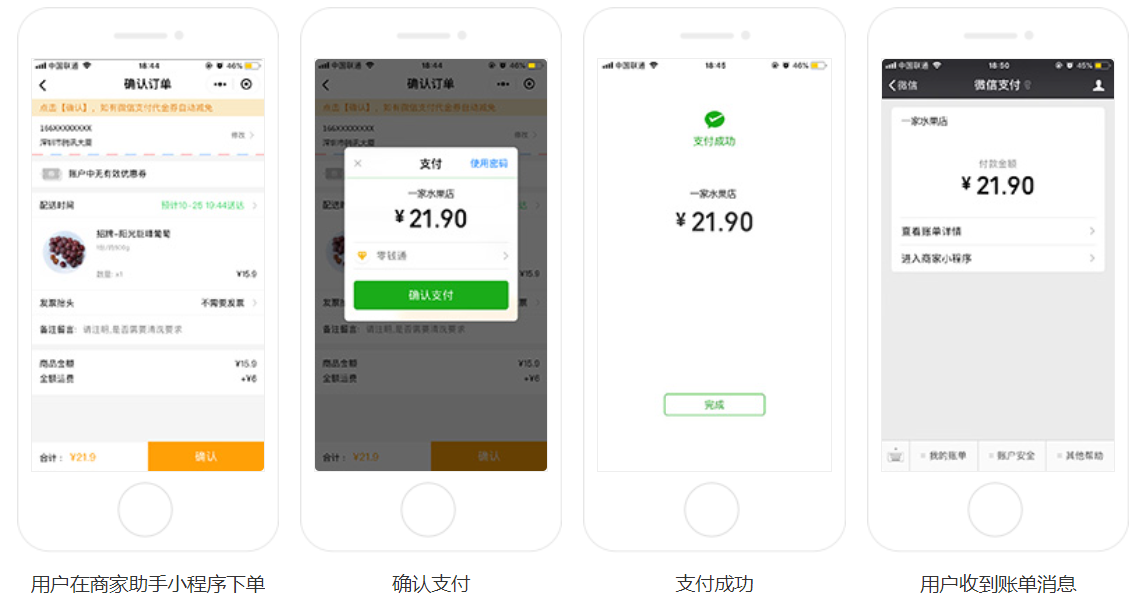
线下场所：调用接口生成二维码，用户扫描二维码后在微信浏览器中打开页面后完成支付

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



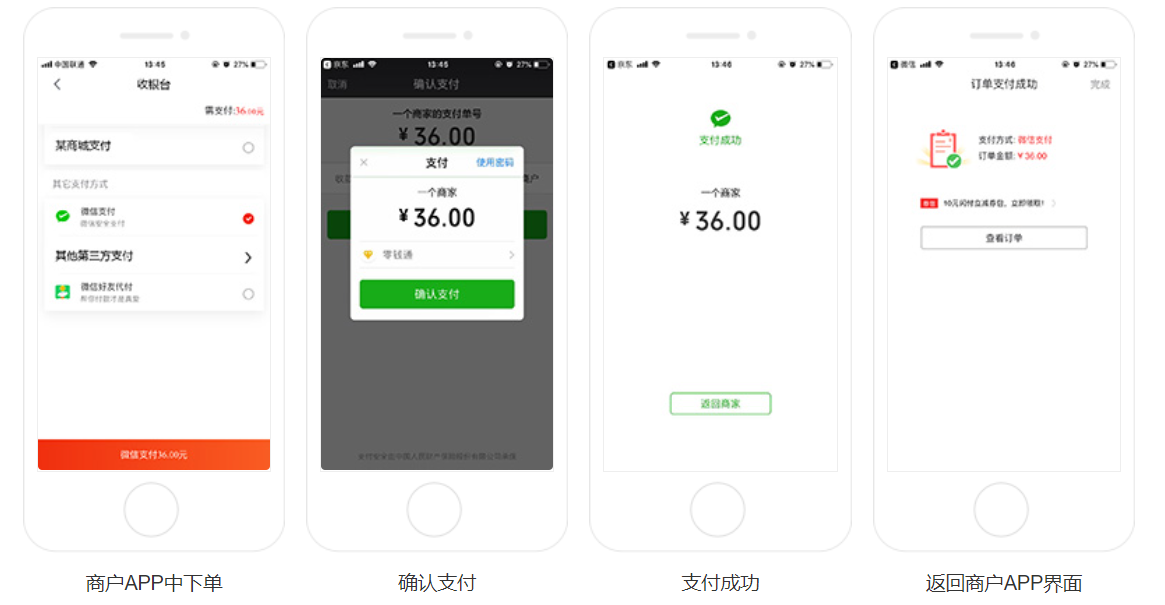
3、小程序支付是指商户通过调用微信支付小程序支付接口，在微信小程序平台内实现支付功能；用户打开商家助手小程序下单，输入支付密码并完成支付后，返回商家小程序。



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。



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。



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。



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：

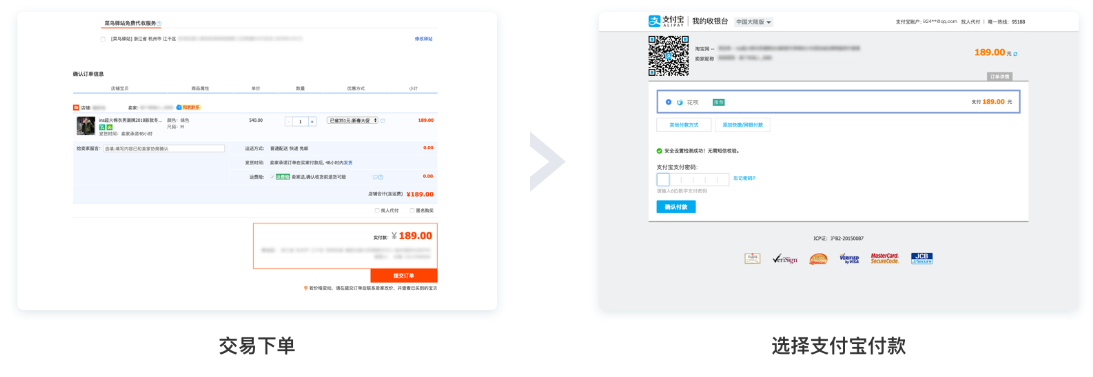
文档：https://b.alipay.com/signing/productSetV2.htm



与本项目需求相关的接口：电脑网站支付、手机网站支付。

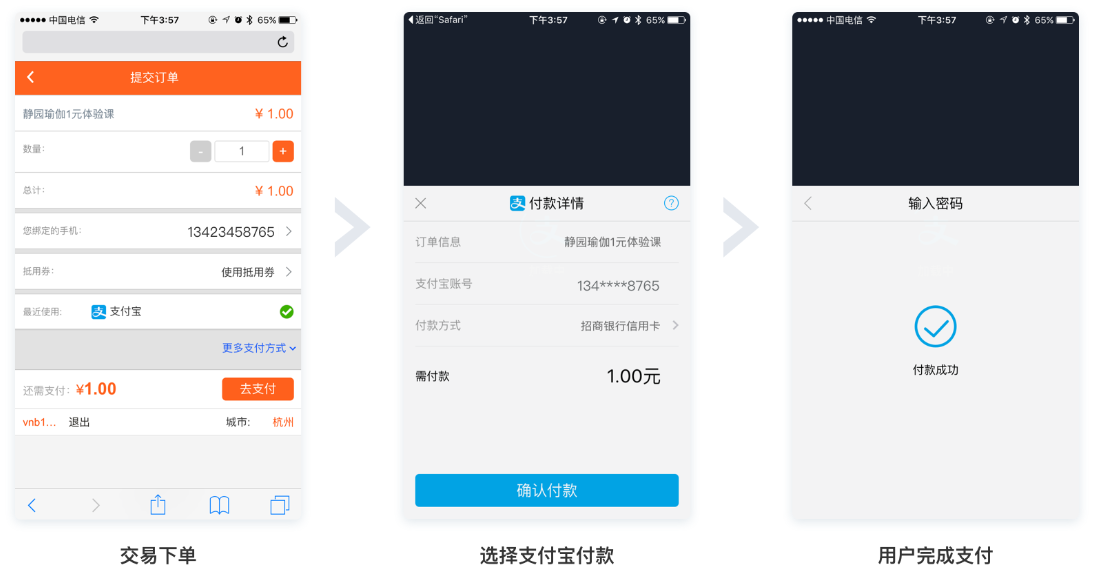
1、电脑网站支付

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。



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。



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：

选择资料文件夹提供的支付宝安装包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。

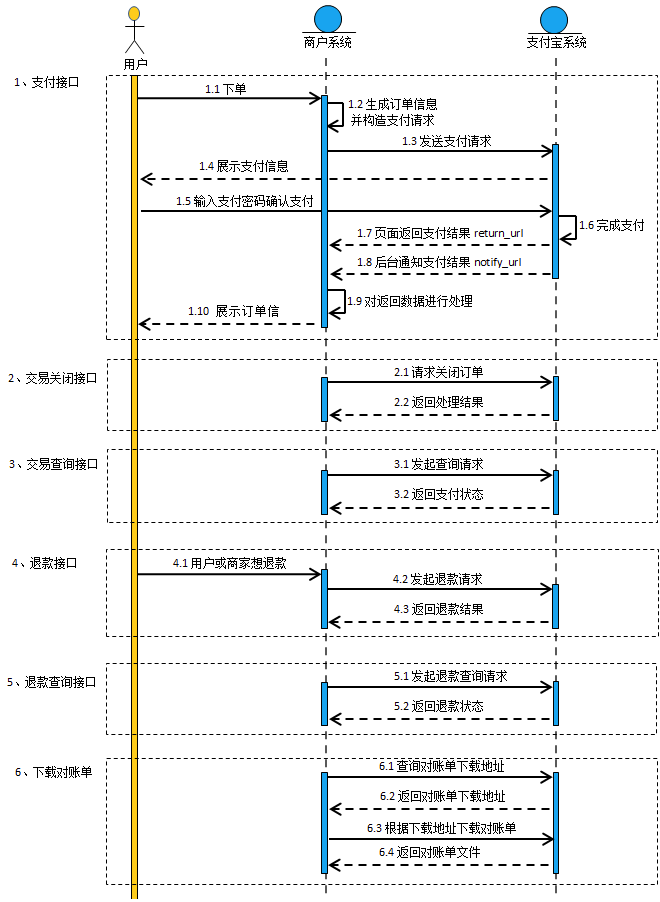


**3.4 支付接口测试**

**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.pay](https://docs.open.alipay.com/203/107090)API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](https://docs.open.alipay.com/203/107090#s2)。

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](https://docs.open.alipay.com/203/105286)。

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

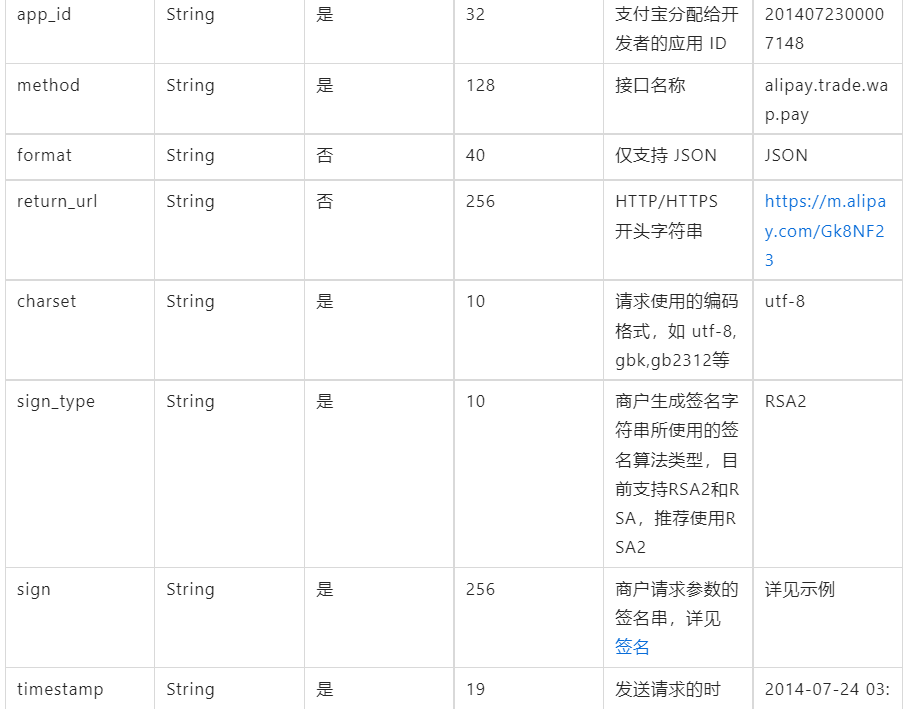
**请求地址**：

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。









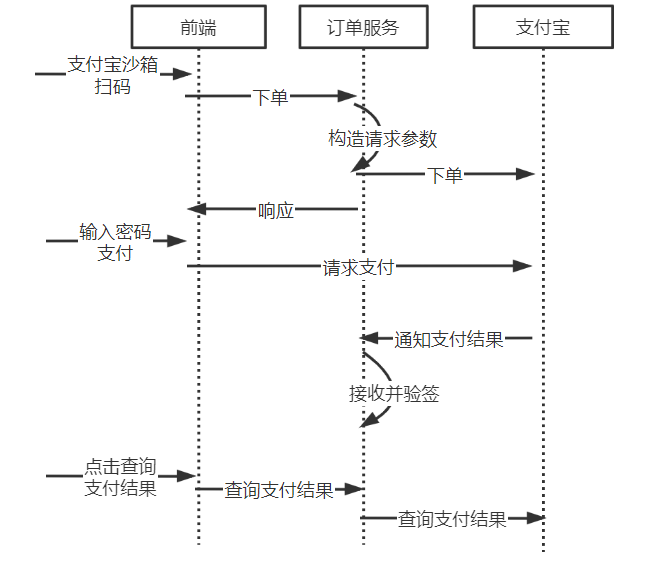
其它扩展参数参见接口文档。

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

也可以从课程资料目录获取alipay.trade.wap.pay-java-utf-8.zip。

拷贝示例代码，修改、测试。

|  |
| --- |
| 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

拷贝课程资料下项目工程目录的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 data:image/png;base64, |

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 |

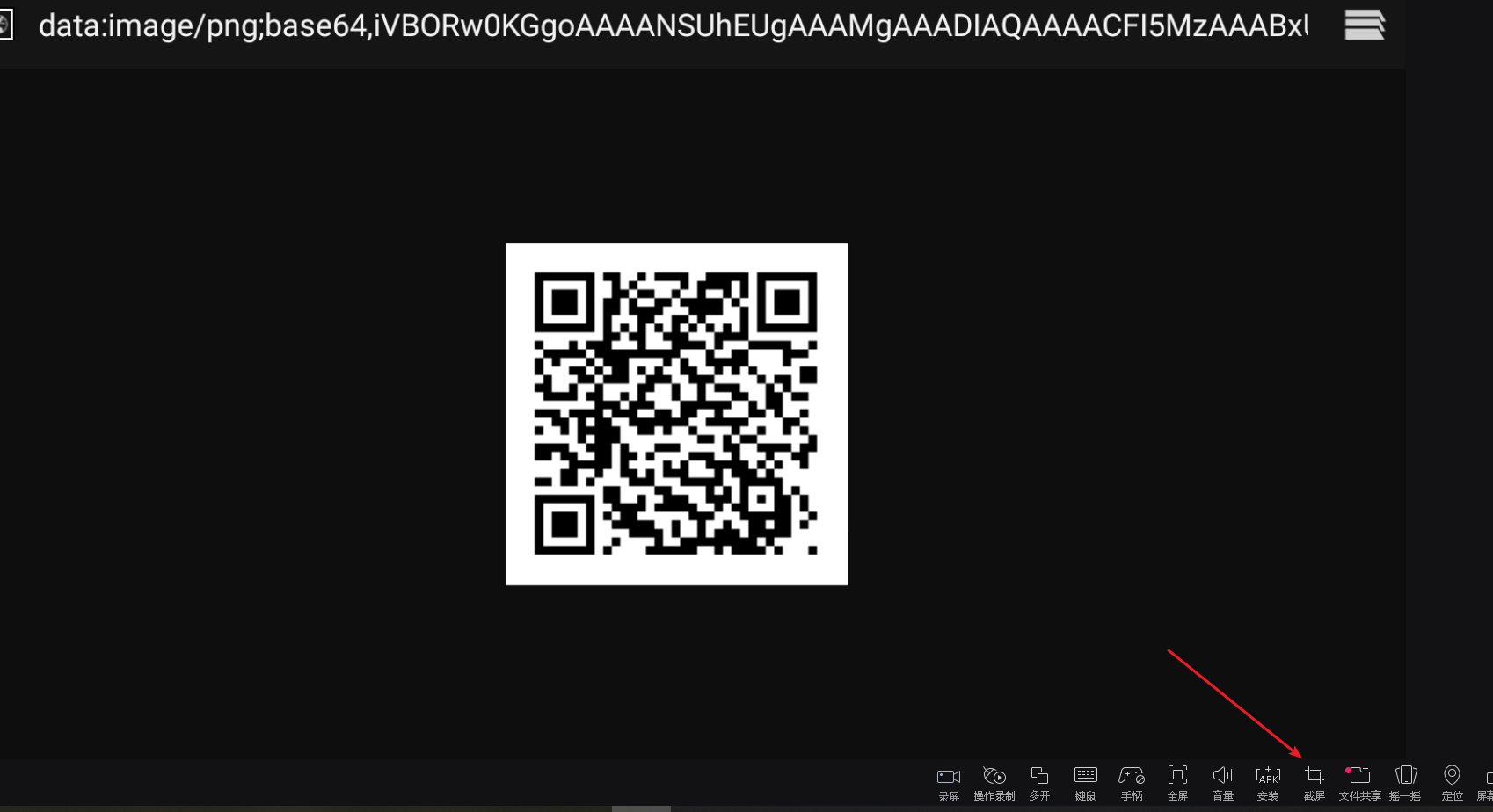
运行main方法，复制输出到控制台的base64串。

|  |
| --- |
| Bash data:image/png;base64, |

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。

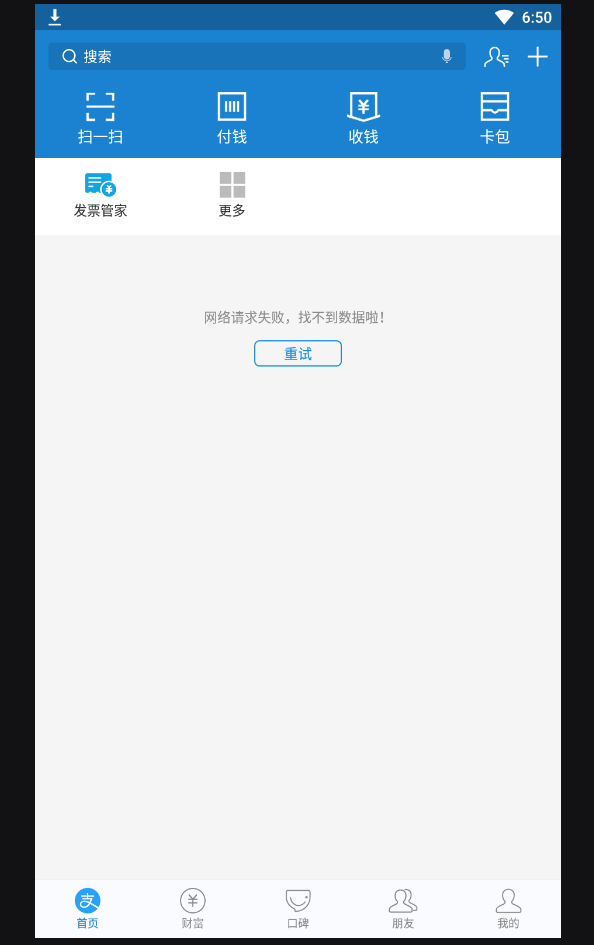


使用截屏工具进行截屏，稍后使用支付宝沙箱客户端扫此图片。

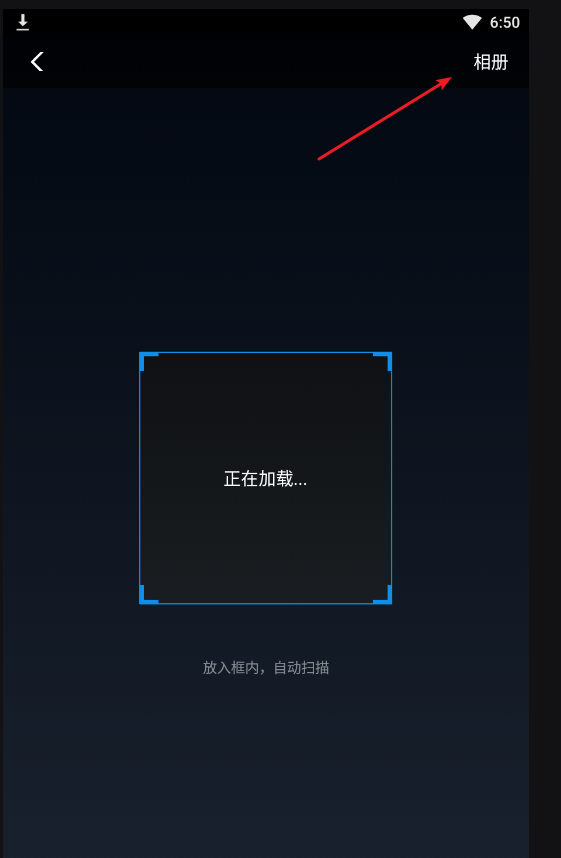


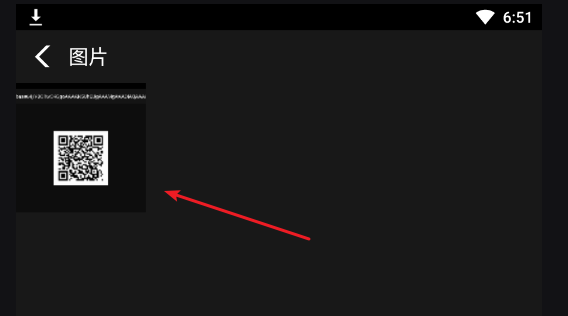
2、启动订单服务

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。

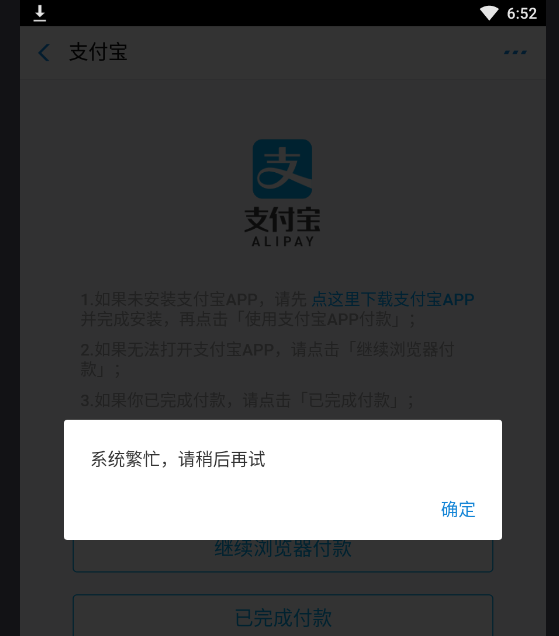


点击扫一扫选择相册中刚才截屏的二维码

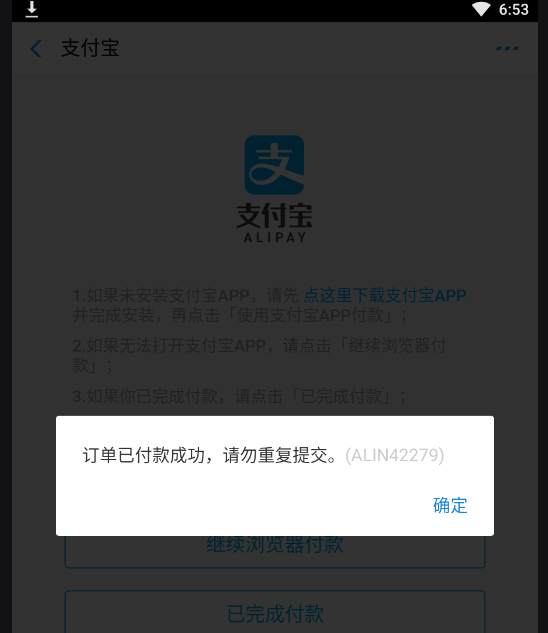




扫码后如果提示系统繁忙再重试



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。

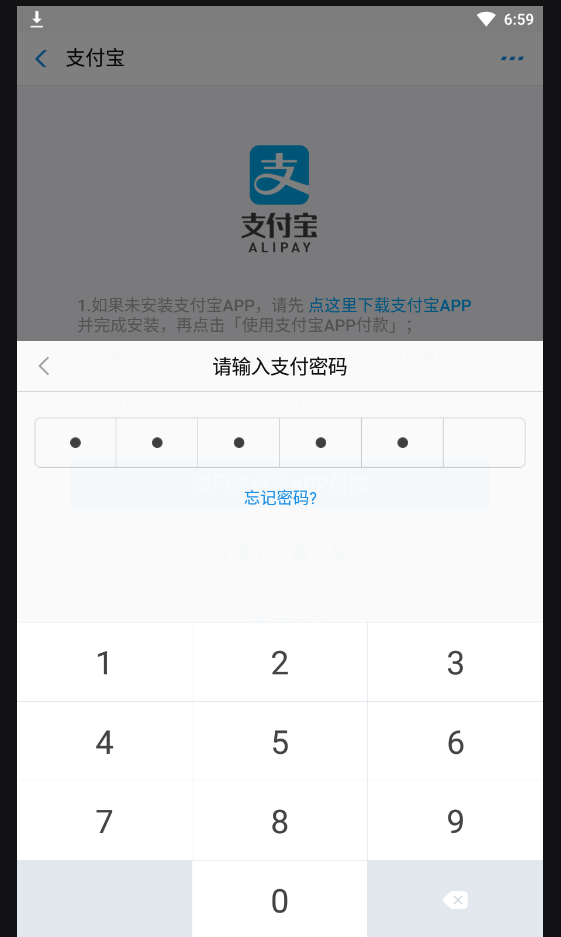


修改订单号后重启订单服务，再使用沙箱支付宝客户端扫码

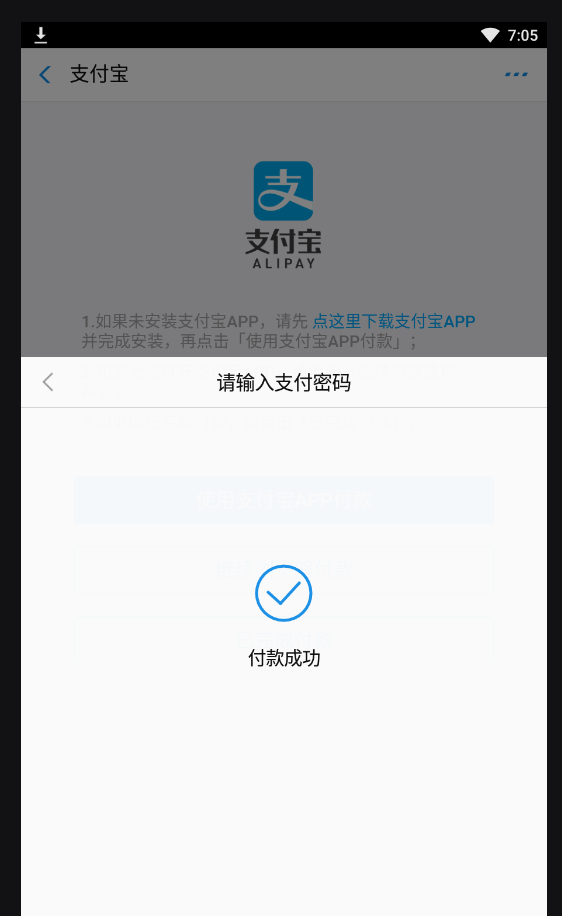


输入支付密码进行支付。

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。



支付成功界面：



**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\_id","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.U*RL,* APP\_ID, APP\_PRIVATE\_KEY, "json", AlipayConfig.C*HARSET,* ALIPAY\_PUBLIC\_KEY, AlipayConfig.S*IGNTYPE)*; //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.o*ut.*println("调用成功");  System.o*ut.*println(response.getBody());  } else {  System.o*ut.*println("调用失败");  } } } |

运行代码，输出如下：

|  |
| --- |
| Plain Text 调用成功 |

输出结果即是调用支付宝查询接口查询到的支付结果



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。详情可查看 [支付宝异步通知说明](https://opendocs.alipay.com/support/01raw4) 。

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



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) 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);  }   //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");   //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*, *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   //——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。  }  }   } |

**3.4.5.3 通知接口测试**

1、重启订单服务，并在下单接口中打上断点

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。

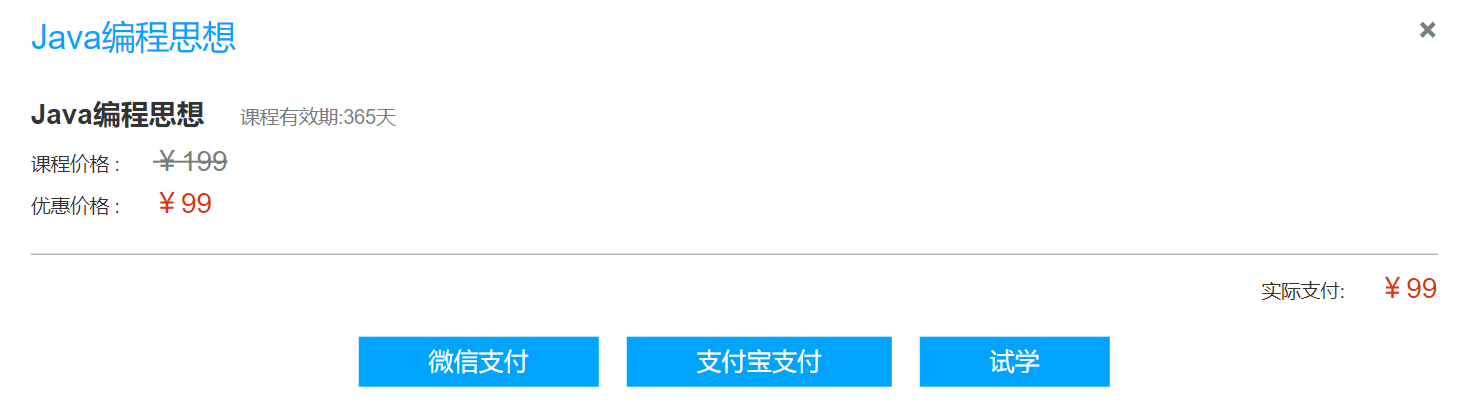
4、观察下单接口断点处的执行情况、接收数据等是否正常。

**3.5 生成支付二维码**

**3.5.1 需求分析**

**3.5.1.1 执行流程**

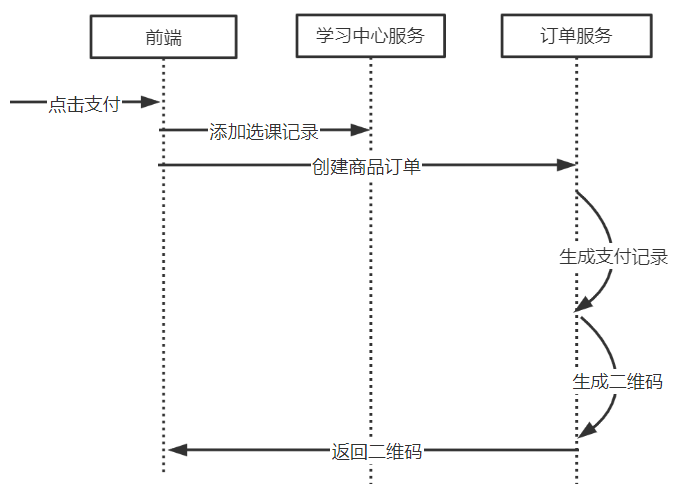
再次打开课程支付引导界面，点击“支付宝支付”按钮系统该如何处理？



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：

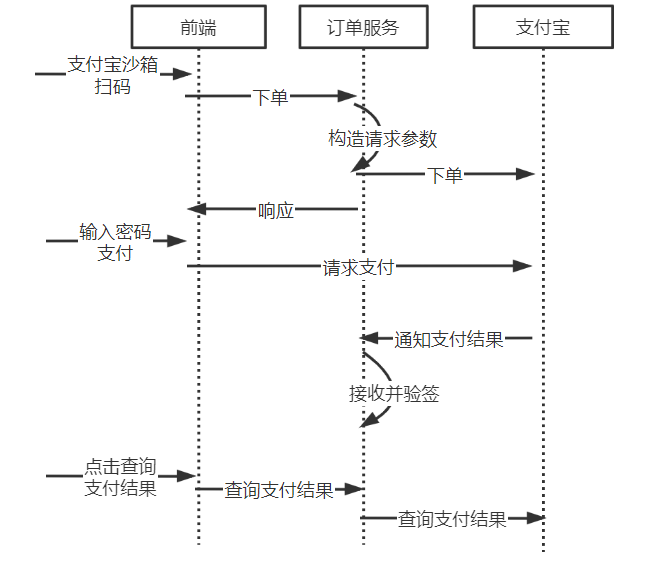
1、前端调用学习中心服务的添加选课接口。

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。

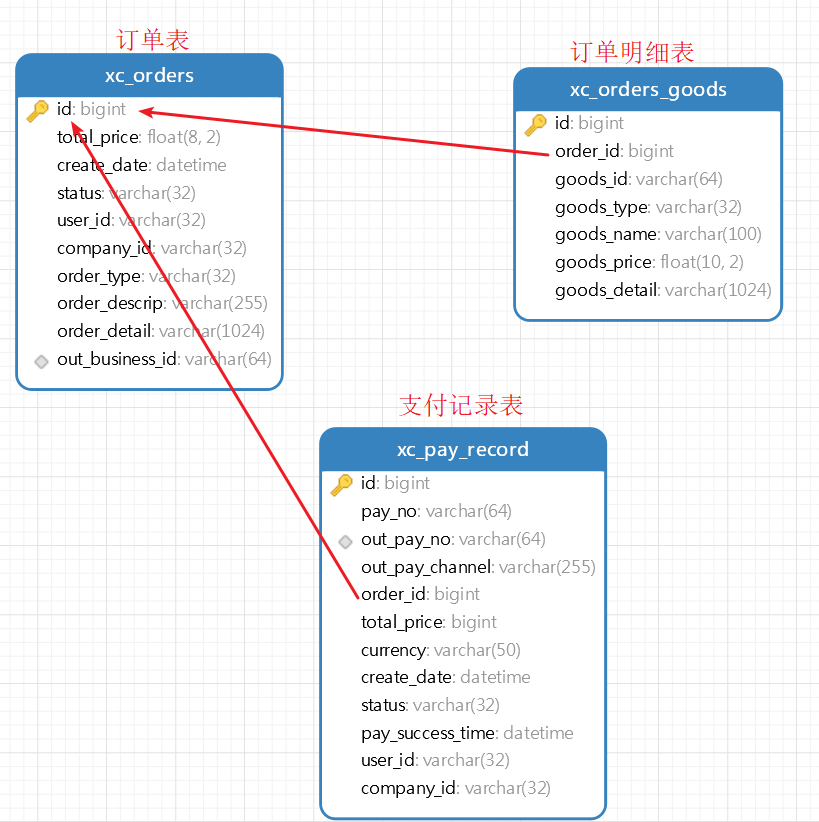
4、将二维码返回到前端，用户扫码。

用户扫码支付流程如下：

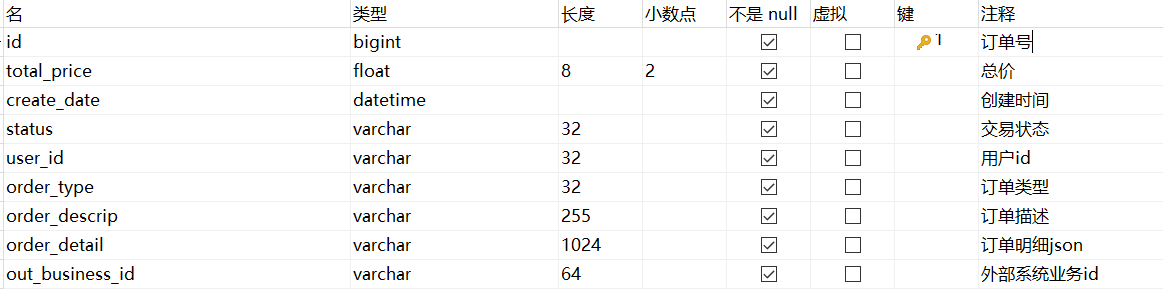


**3.5.1.2 数据模型**

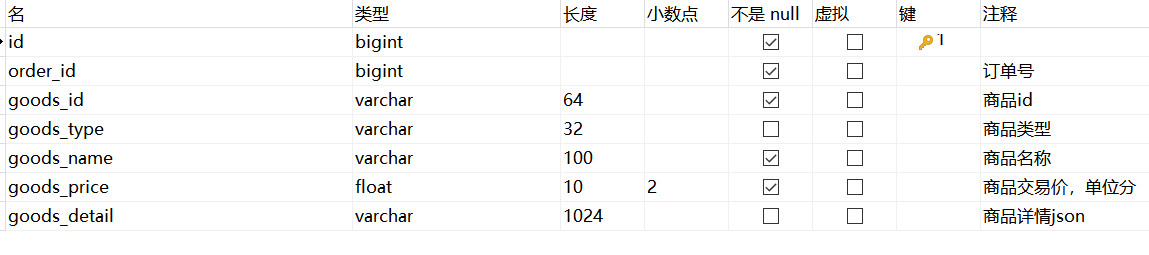
订单支付模式的核心由三张表组成：订单表、订单明细表、支付交易记录表。



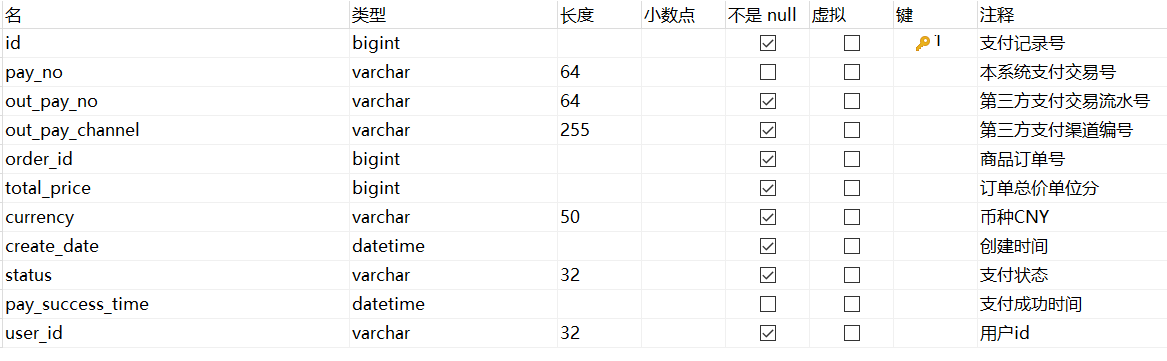
订单表：记录订单信息



订单明细表记录订单的详细信息



支付交易记录表记录与支付平台的交易明细



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) {  } |

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 集成认证授权**

订单服务也需要获取用户的身份信息，需要在订单服务中集成认证授权

集成过程参考内容管理服务。

1、添加依赖

2、配置资源服务、令牌配置。

3、拷贝SecurityUtil

**3.5.3.2 保存商品订单**

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。

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做创建商品订单、创建支付交易记录

|  |
| --- |
| Java @Transactional @Override public PayRecordDto createOrder(String userId,AddOrderDto addOrderDto) {  //添加商品订单   //添加支付交易记录  } |

编写创建商品订单方法：

|  |
| --- |
| 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？

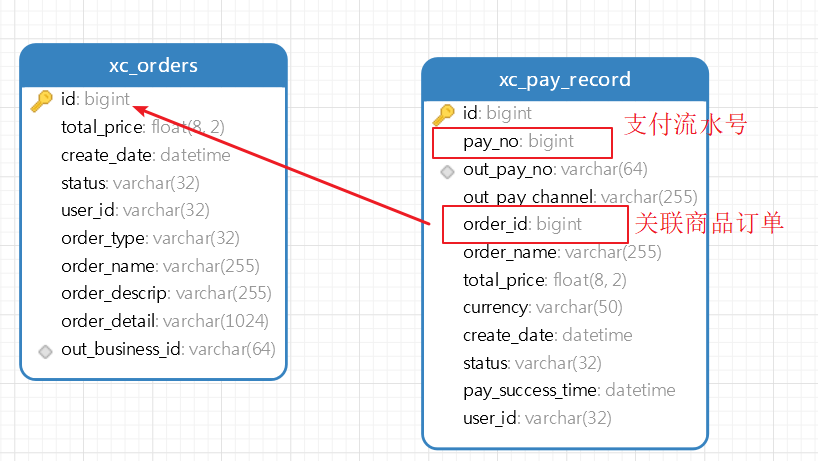
在请求微信或支付宝下单接口时需要传入 商品订单号，在与第三方支付平台对接时发现，当用户支付失败或因为其它原因最终该订单没有支付成功，此时再次调用第三方支付平台的下单接口发现报错“订单号已存在”，此时如果我们传入一个没有使用过的订单号就可以解决问题，但是商品订单已经创建，因为没有支付成功重新创建一个新订单是不合理的。

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){   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**

完善创建订单service方法

|  |
| --- |
| Java @Transactional @Override public PayRecordDto createOrder(String userId, AddOrderDto addOrderDto) {  //创建商品订单  XcOrders orders = saveXcOrders(userId, addOrderDto);  //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)  qrCode = new QRCodeUtil().createQRCode("http://192.168.101.1/api/orders/requestpay?payNo="+payRecord.getPayNo(), 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 @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**

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 @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");  }  //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  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 = alipayClient.pageExecute(alipayRequest).getBody(); //调用SDK生成表单  } catch (AlipayApiException e) {  e.printStackTrace();  }  httpResponse.setContentType("text/html;charset=" + *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 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、订单服务、内容管理服务。

2、发布一门收费课程。

测试流程：

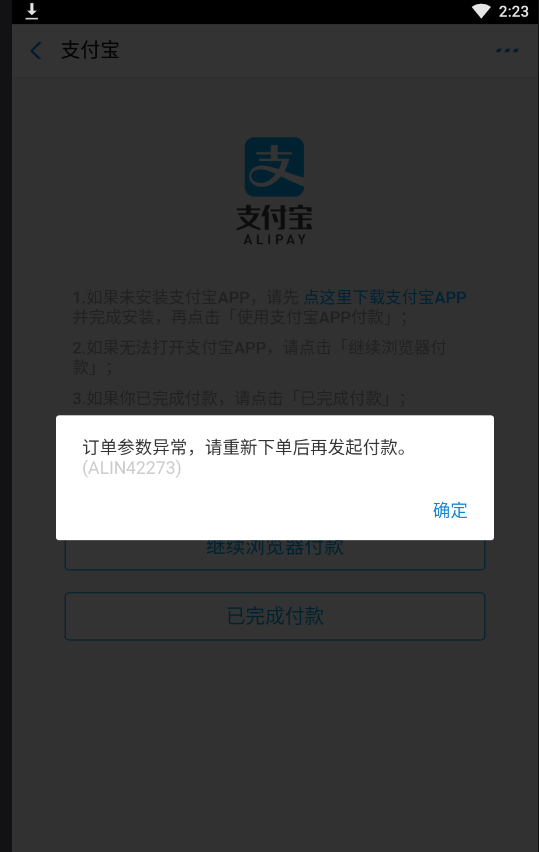
1、进入收费课程详细页面，点击马上学习。

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。



**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**

定义支付宝通知信息DTO

|  |
| --- |
| Java package com.xuecheng.orders.model.dto;  import lombok.Data;  /\*\*  \* @author Mr.M  \* @version 1.0  \* @description 支付结果数据, 用于接收支付结果通知处理逻辑  \* @date 2022/10/4 16:49  \*/ @Data public class PayStatusDto {   //商户订单号  String out\_trade\_no;  //支付宝交易号  String trade\_no;  //交易状态  String trade\_status;  //appid  String app\_id;  //total\_amount  String total\_amount; } |

定义service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); |

service接口实现

|  |
| --- |
| Java @Transactional @Override public void saveAliPayStatus(PayStatusDto payStatusDto) {   //支付结果  String trade\_status = payStatusDto.getTrade\_status();   if (trade\_status.equals("TRADE\_SUCCESS")) {  //支付流水号  String payNo = payStatusDto.getOut\_trade\_no();  //查询支付流水  XcPayRecord payRecord = getPayRecordByPayno(payNo);   //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 (payRecord != null  && payStatusDto.getApp\_id().equals(APP\_ID)  && totalPrice.intValue() == total\_amount.intValue()) {  String status = payRecord.getStatus();  if ("601001".equals(status)) {//未支付时进行处理   *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("收到支付通知，更新支付交易状态成功.付交易流水号:{},支付结果:{}", payNo, trade\_status);  } else {  *log*.error("收到支付通知，更新支付交易状态失败.支付交易流水号:{},支付结果:{}", payNo, trade\_status);  }  //关联的订单号  Long orderId = payRecord.getOrderId();  XcOrders orders = ordersMapper.selectById(orderId);  if(orders!=null){  XcOrders order\_u = new XcOrders();  order\_u.setStatus("600002");  int update = ordersMapper.update(order\_u, new LambdaQueryWrapper<XcOrders>().eq(XcOrders::getId, orderId));  if (update > 0) {  *log*.info("收到支付通知，更新订单状态成功.付交易流水号:{},支付结果:{},订单号:{},状态:{}", payNo, trade\_status, orderId, "600002");  } else {  *log*.error("收到支付通知，更新订单状态失败.支付交易流水号:{},支付结果:{},订单号:{},状态:{}", payNo, trade\_status, orderId, "600002");  }   }else{  *log*.error("收到支付通知，根据交易记录找不到订单,交易记录号:{},订单号:{}",payRecordByPayno.getPayNo(),orderId);  }    }   }   }  } |

**3.7.3 接口完善**

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。

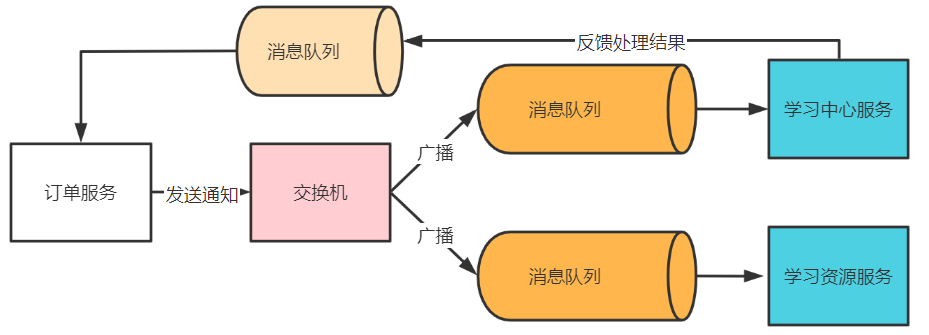
**4 通知支付结果**

**4.1 需求分析**

**4.1.1 消息通知方式**

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通知给与订单服务对接的其它微服务。

为了保证通知过程的简便还要保证消息全部到达消费服务，采用发布订阅的方式通知支付结果。



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，根据支付结果的内容去更新自己的业务数据。

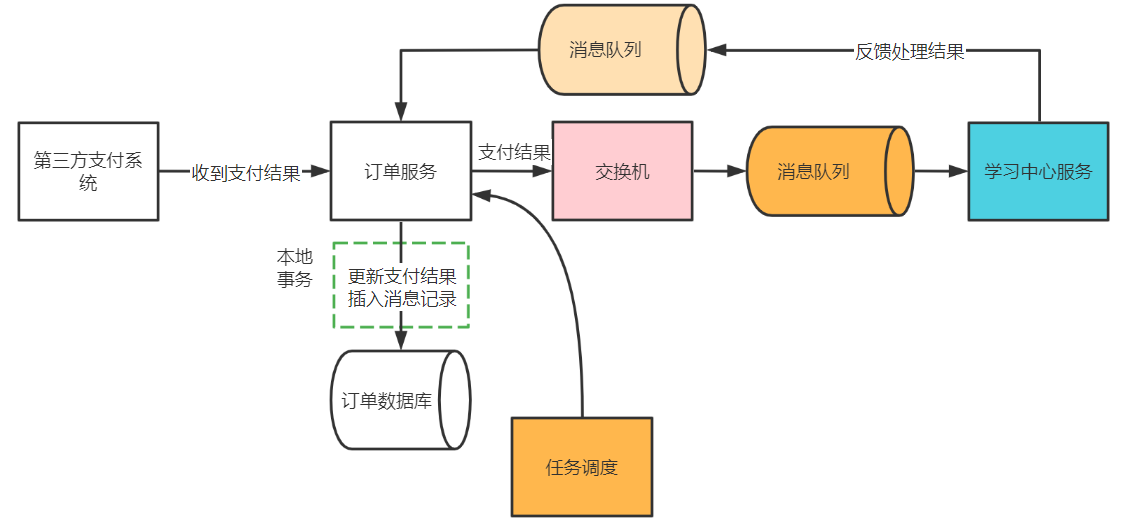
学习中心等微服务收到消息并处理完成通过消息队列回复订单服务。

**4.1.2 分布式事务问题**

订单服务收到第三方支付系统的通知更新支付结果，订单服务将支付结果通知给其它微服务，订单服务需要保证更新支付结果成功并且向其它微服务通知支付结果也成功，两件事跨多个微服务并且需要保证一致性，存在分布式事务控制的需求。

针对该业务场景如何控制分布式事务？

根据需求可知，订单服务先将支付结果更新成功后再将支付结果通知给其它微服务，只要保证最终将支付结果通知到微服务保证最终一致性即可，可以采用课程发布模块的技术方案，先通过本地事务更新支付结果的同时添加一条消息表记录，再由任务调度去定时调度将支付结果通知给其它微服务。



**4.2 集成消息队列**

**4.2.1 支付结果通知队列**

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Fanout广播模式。

首先需要在内容管理服务层工程和消息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、编写MQ配置类，配置交换机

|  |
| --- |
| Java package com.xuecheng.orders.config;  import org.springframework.amqp.core.DirectExchange; import org.springframework.amqp.core.FanoutExchange; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;  /\*\*  \* @description 消息队列配置  \* @author Mr.M  \* @date 2022/10/4 22:25  \* @version 1.0  \*/ @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";  //声明交换机  @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);  } } |

重启订单服务，登录rabbitmq，查看交换机自动创建成功



下边在学习中心服务配置消息队列

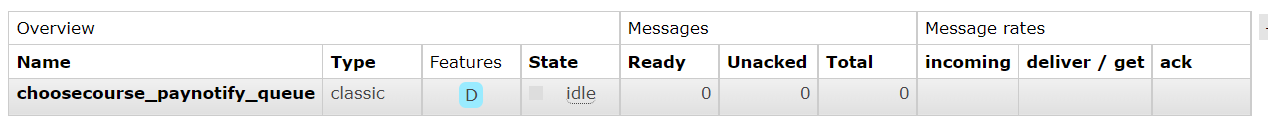
1、引入rabbitmq依赖

2、在学习中心服务引入rabbitmq-dev.yaml配置文件

3、配置队列、绑定交换机

|  |
| --- |
| Java package com.xuecheng.learning.config;  import org.springframework.amqp.core.\*; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;   /\*\*  \* @description 消息队列配置  \* @author Mr.M  \* @date 2022/10/4 22:25  \* @version 1.0  \*/ @Configuration public class PayNotifyConfig {   //交换机 public static final String *PAYNOTIFY\_EXCHANGE\_FANOUT* = "paynotify\_exchange\_fanout"; //支付通知队列 public static final String *PAYNOTIFY\_QUEUE* = "paynotify\_queue";  //支付结果回复 public static final String *PAYNOTIFY\_REPLY\_QUEUE* = "paynotify\_reply\_queue";  //支付结果通知消息类型 public static final String *MESSAGE\_TYPE* = "payresult\_notify";  //声明交换机 @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 @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); } } |

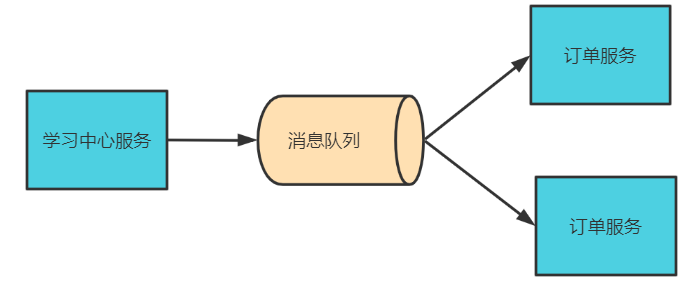
配置完成重启学习中心服务，登录rabbitmq查看队列是否创建成功



**4.2.2 支付通知回复队列**

学习中心服务收到支付结果处理完成向支付通知回复队列发送消息，订单服务收到回复后将不再通知该订单的支付结果。

支付通知回复队列只有订单服务监听，这里使用workQueue模式。如下图：



1、在订单服务、学习中心服务配置支付通知回复队列

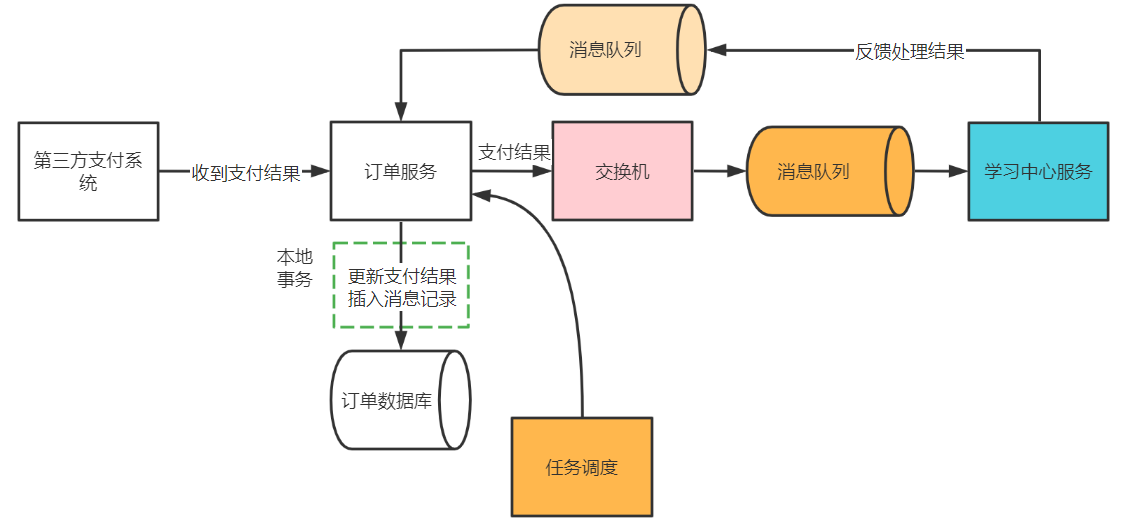
|  |
| --- |
| Java //支付通知回复队列 @Bean(*PAYNOTIFY\_REPLY\_QUEUE*) public Queue msgnotify\_result\_queue(){  return QueueBuilder.*durable*(*PAYNOTIFY\_REPLY\_QUEUE*).build(); } |

**4.3 支付结果通知**

**4.3.1 记录消息表**

订单服务收到第三方支付系统的支付结果，如果支付成功则更新订单的状态，同时向消息表写入记录，这里使用本地事务进行控制，保证订单状态更新成功消息记录写入成功。

如下图：



更改订单服务接收通知的代码，添加写入消息记录的代码，如下：

|  |
| --- |
| Java @Transactional @Override public void saveAliPayStatus(PayStatusDto payStatusDto) {   //支付结果  String trade\_status = payStatusDto.getTrade\_status();   if (trade\_status.equals("TRADE\_SUCCESS")) {  //支付流水号  String payNo = payStatusDto.getOut\_trade\_no();  //查询支付流水  XcPayRecord payRecord = getPayRecordByPayno(payNo);   //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 (payRecord != null  && payStatusDto.getApp\_id().equals(APP\_ID)  && totalPrice.intValue() == total\_amount.intValue()) {  String status = payRecord.getStatus();  if ("601001".equals(status)) {//未支付时进行处理   *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("收到支付通知，更新支付交易状态成功.付交易流水号:{},支付结果:{}", payNo, trade\_status);  } else {  *log*.error("收到支付通知，更新支付交易状态失败.支付交易流水号:{},支付结果:{}", payNo, trade\_status);  }  //关联的订单号  Long orderId = payRecord.getOrderId();  XcOrders orders = ordersMapper.selectById(orderId);  if(orders!=null){  XcOrders order\_u = new XcOrders();  order\_u.setStatus("600002");  int update = ordersMapper.update(order\_u, new LambdaQueryWrapper<XcOrders>().eq(XcOrders::getId, orderId));  if (update > 0) {  *log*.info("收到支付通知，更新订单状态成功.付交易流水号:{},支付结果:{},订单号:{},状态:{}", payNo, trade\_status, orderId, "600002");  //订单类型,购买课程、购买学习资料..  String orderType = orders.getOrderType();  //写消息记录  mqMessageService.addMessage(PayNotifyConfig.*MESSAGE\_TYPE*,orders.getOutBusinessId(),orderType,null);  } else {  *log*.error("收到支付通知，更新订单状态失败.支付交易流水号:{},支付结果:{},订单号:{},状态:{}", payNo, trade\_status, orderId, "600002");  }    }    }   }   }  } |

**4.3.2 发送通知**

由xxl-job进行任务调度读取消息表中未处理的消息，向学习中心服务等其它微服务通知支付结果。

1、在订单服务集成消息SDK与xxl-job任务调度通信。

1) 在订单服务service工程中添加sdk依赖

|  |
| --- |
| XML <dependency>  <groupId>com.xuecheng</groupId>  <artifactId>xuecheng-plus-message-sdk</artifactId>  <version>0.0.1-SNAPSHOT</version> </dependency> |

2、实现sdk中的MessageProcessAbstract类

|  |
| --- |
| Java package com.xuecheng.orders.jobhandler;  import com.xuecheng.messagesdk.model.po.MqMessage; import com.xuecheng.messagesdk.service.MessageProcessAbstract; import com.xuecheng.orders.service.impl.PayNotifyService; import com.xxl.job.core.context.XxlJobHelper; import com.xxl.job.core.handler.annotation.XxlJob; import lombok.extern.slf4j.Slf4j; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component;  /\*\*  \* @author Mr.M  \* @version 1.0  \* @description 支付通知任务  \* @date 2022/9/22 10:16  \*/ @Slf4j @Component public class PayNotifyTask extends MessageProcessAbstract {   @Autowired  private RabbitTemplate rabbitTemplate;    @Autowired  MqMessageService mqMessageService;   @Autowired  PayNotifyService payNotifyService;   //任务调度入口  @XxlJob("NotifyPayResultJobHandler")  public void notifyPayResultJobHandler() throws Exception {  // 分片参数  int shardIndex = XxlJobHelper.*getShardIndex*();  int shardTotal = XxlJobHelper.*getShardTotal*();  *log*.debug("shardIndex="+shardIndex+",shardTotal="+shardTotal);  //只查询支付通知的消息  process(shardIndex,shardTotal, PayNotifyConfig.*MESSAGE\_TYPE*,5,60);  }   //课程发布任务处理  @Override  public boolean execute(MqMessage mqMessage) {   *log*.debug("开始进行支付结果通知:{}",mqMessage.toString());  //发送到消息队列  send(mqMessage);   //由于消息表的记录需要等到订单服务收到回复后才能删除，这里返回false不让消息sdk自动删除  return false;  }   /\*\*  \* @description 发送支付结果通知  \* @param m*essage 消*息内容  \* @return void  \* @author Mr.M  \* @date 2022/9/20 9:43  \*/  private void send(MqMessage message){  //转json  String msg = JSON.t*oJSONString(*message);  // 发送消息  rabbitTemplate.convertAndSend(PayNotifyConfig.P*AYNOTIFY\_EXCHANGE\_FANOUT,* "", msg);    }  } |

3 配置xxl-job

在订单服务的service工程配置执行器

|  |
| --- |
| YAML xxl:  job:  admin:   addresses: http://192.168.101.65:8088/xxl-job-admin  executor:  appname: payresultnotify-job  address:   ip:   port: 8989  logpath: /data/applogs/xxl-job/jobhandler  logretentiondays: 30  accessToken: default\_token |

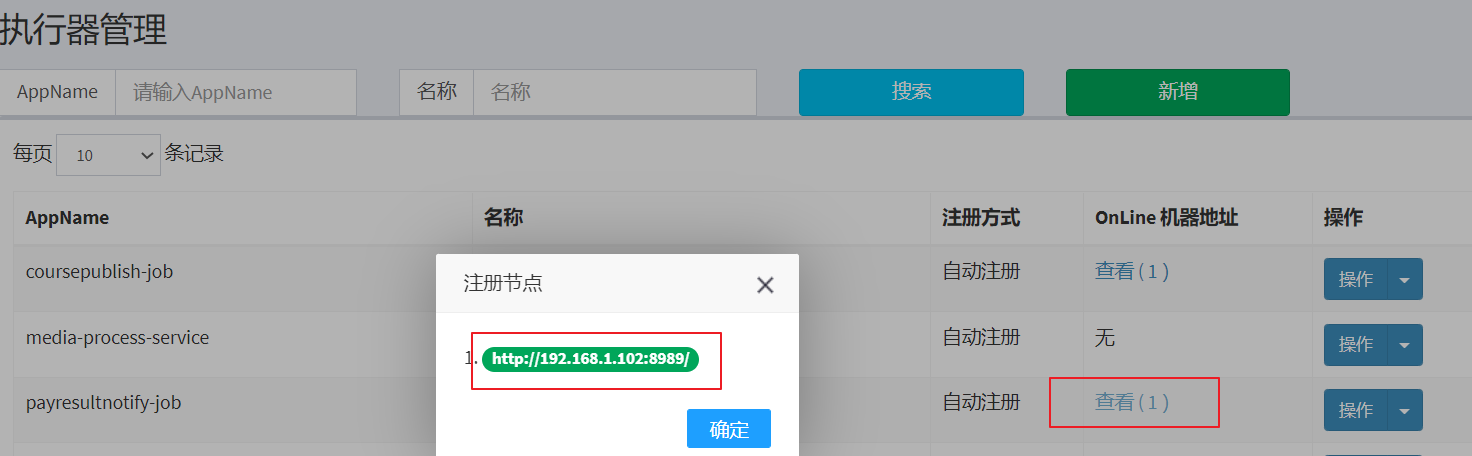
在订单服务添加xxl-job配置类

从内容管理模块的service拷贝一个即可。

在xxl-job添加执行器



重启订单服务，登录xxl-job，查看执行器是否正常



**4.3.3 接收通知**

在学习中心服务接收支付结果通知，收到通知保存选课状态。

1) 学习中心服service工程中添加sdk依赖

|  |
| --- |
| XML <dependency>  <groupId>com.xuecheng</groupId>  <artifactId>xuecheng-plus-message-sdk</artifactId>  <version>0.0.1-SNAPSHOT</version> </dependency> |

2）监听消息队列，接收通知

|  |
| --- |
| Java package com.xuecheng.learning.service.impl;  import com.alibaba.fastjson.JSON; 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.ExchangeTypes; import org.springframework.amqp.rabbit.annotation.Exchange; import org.springframework.amqp.rabbit.annotation.Queue; import org.springframework.amqp.rabbit.annotation.QueueBinding; 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;  /\*\*  \* @author Mr.M  \* @version 1.0  \* @description 接收支付结果通知service  \* @date 2022/10/5 5:06  \*/ @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(String message) {  //获取消息  MqMessage mqMessage = JSON.*parseObject*(message, 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){  //向订单服务回复    }   }   }    } |

3）编写保存选课状态方法

|  |
| --- |
| Java @Slf4j @Service public class MyCourseTablesServiceImpl implements MyCourseTablesService { ...... @Transactional @Override public boolean saveChooseCourseStauts(String choosecourseId) {   //根据choosecourseId查询选课记录  XcChooseCourse xcChooseCourse = chooseCourseMapper.selectById(choosecourseId);  if(xcChooseCourse == null){  *log*.debug("收到支付结果通知没有查询到关联的选课记录,choosecourseId:{}",choosecourseId);  return false;  }  String status = xcChooseCourse.getStatus();  if("701001".equals(status)){  //添加到课程表  addCourseTabls(xcChooseCourse);  return true;  }  //待支付状态才处理  if ("701002".equals(status)) {  //更新为选课成功  xcChooseCourse.setStatus("701001");  int update = chooseCourseMapper.updateById(xcChooseCourse);  if(update>0){  *log*.debug("收到支付结果通知处理成功,选课记录:{}",xcChooseCourse);  //添加到课程表  addCourseTabls(xcChooseCourse);  return true;  }else{  *log*.debug("收到支付结果通知处理失败,选课记录:{}",xcChooseCourse);  return false;  }  }   return false; } |

**4.4 支付结果通知回复**

**4.4.1 发送回复**

学习中心服务收到支付结果并处理完成后发送回复。

|  |
| --- |
| Java package com.xuecheng.learning.service.impl;  import com.alibaba.fastjson.JSON; 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.ExchangeTypes; import org.springframework.amqp.rabbit.annotation.Exchange; import org.springframework.amqp.rabbit.annotation.Queue; import org.springframework.amqp.rabbit.annotation.QueueBinding; 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;  /\*\*  \* @author Mr.M  \* @version 1.0  \* @description 接收支付结果通知service  \* @date 2022/10/5 5:06  \*/ @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(String message) {  //获取消息  MqMessage mqMessage = JSON.*parseObject*(message, 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){  //向订单服务回复  send(mqMessage);  }   }   }  /\*\*  \* @description 回复消息  \* @param *message* 回复消息  \* @return void  \* @author Mr.M  \* @date 2022/9/20 9:43  \*/ public void send(MqMessage message){  //转json  String msg = JSON.*toJSONString*(message);  // 发送消息  rabbitTemplate.convertAndSend(PayNotifyConfig.*PAYNOTIFY\_REPLY\_QUEUE*, msg);  *log*.debug("学习中心服务向订单服务回复消息:{}",message); }    } |

**4.4.2 接收回复**

订单服务监听队列，接收回复，删除消息表的记录。

|  |
| --- |
| Java package com.xuecheng.orders.jobhandler;  import com.alibaba.fastjson.JSON; import com.xuecheng.messagesdk.model.po.MqMessage; import com.xuecheng.messagesdk.service.MessageProcessAbstract; import com.xuecheng.messagesdk.service.MqMessageService; import com.xuecheng.orders.config.PayNotifyConfig; import com.xxl.job.core.context.XxlJobHelper; import com.xxl.job.core.handler.annotation.XxlJob; import lombok.extern.slf4j.Slf4j; 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.Component;  /\*\*  \* @author Mr.M  \* @version 1.0  \* @description *TODO* \* @date 2022/10/24 23:27  \*/ @Slf4j @Component public class PayNotifyTask extends MessageProcessAbstract {   @Autowired  private RabbitTemplate rabbitTemplate;   @Autowired  MqMessageService mqMessageService;   .....   //接收回复  @RabbitListener(queues = PayNotifyConfig.*PAYNOTIFY\_REPLY\_QUEUE*)  public void receive(String message) {  //获取消息  MqMessage mqMessage = JSON.*parseObject*(message, MqMessage.class);  *log*.debug("接收支付结果回复:{}", mqMessage);   //完成支付通知  mqMessageService.completed(mqMessage.getId());   } } |

**4.5 通知支付结果测试**

测试准备：

1、制造一门待支付的选课记录。

2、提前在订单服务的发送通知、接收回复方法内打上断点。

3、提前在学习中心的接收通知、发送回复方法内打上断点。

4、启动订单服务、学习中心服务、xxl-job。

测试流程：

1、对待支付的选课记录进行支付

2、观察四个断点的执行情况。

3、观察学习中心服务是否添加课程到课程表成功、是否更新选课状态为成功。

4、观察消息在处理完后是否删除消息表的记录移动到消息历史表。

**4.6 面试**

**未支付订单如何处理？**

基于RabbitMQ延迟队列处理未支付订单

1）创建订单的同时向MQ发送一条消息给一个队列并设置消息的TTL（过期时间），比如：设置30分钟，由于该队列设置了死信交换机，30分钟后消息将投递到死信交换机。

2）当消息过期，消息发到了死信交换机，同时发给了死信队列（死信队列绑定了死信交换机）

3）程序监听了死信队列，收到了过期的订单。

4）程序收到消息判断如果订单未支付则取消订单。如果说已支付不用处理。

**如何保证RabbitMQ的消息可靠性?**

1、设置消息持久化

首先设置交换机支持持久化（定义交换机时设置持久化为true）

其次设置队列支持持久化（定义队列时设置持久化为true）

发送消息时设置消息要持久化

2、消费者收到消息处理完成要确认

设置 消费者确认模式为自动确认 acknowledge-mode=auto ，当程序处理正常没有异常会发送ack，抛出异常则发送nack

也可以设置为手动确认，在程序处理完成的代码处手动发送ack。

3、消费失败重试

消费失败后在消费者本地进行重试，达到最大重试次数会将失败消息投递到指定交换机，交换机绑定一个异常消息队列，程序监听这个队列收到异常消息后放在数据库中单独处理，或由人工处理。

**如何避免消息重复消费？**

重复投递的原因：等待超时后需要重试。

避免重复投递：消息生产时，生产者发送的消息携带一个`Message ID`（全局唯一ID），

作为去重和幂等的依据，避免重复的消息进入队列

重复消费的原因：消费者接收消息后，在确认之前断开了连接或者取消订阅，

消息会被重新分发给下一个订阅的消费者。

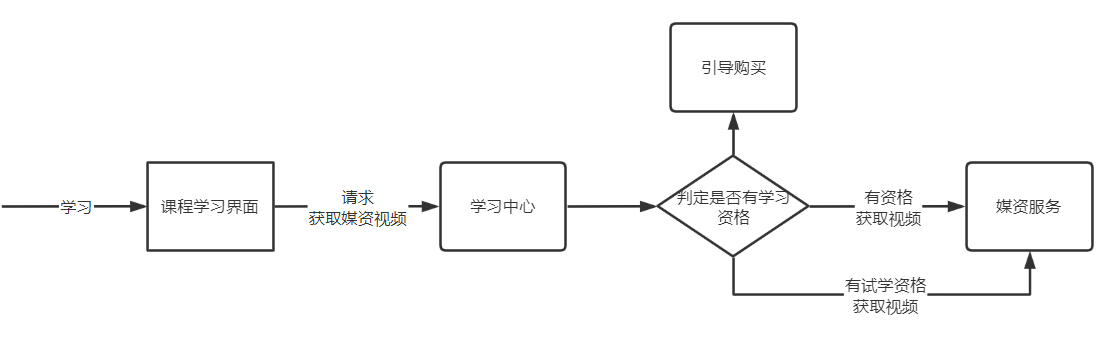
避免重复消费：消息消费时，要求消息体中必须要有一个全局唯一ID，作为去重和幂等的依据，避免同一条消息被重复消费。

**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 视频插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：



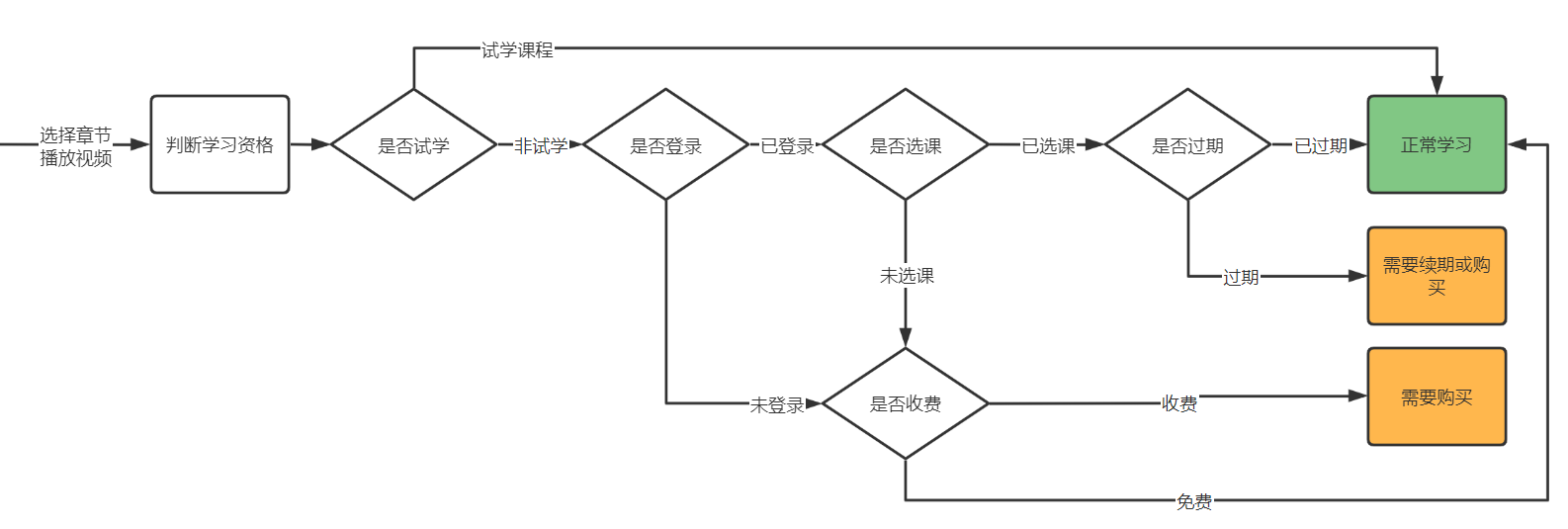
拥有学习资格则继续播放视频，不具有学习资格则引导去购买、续期等操作。

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。

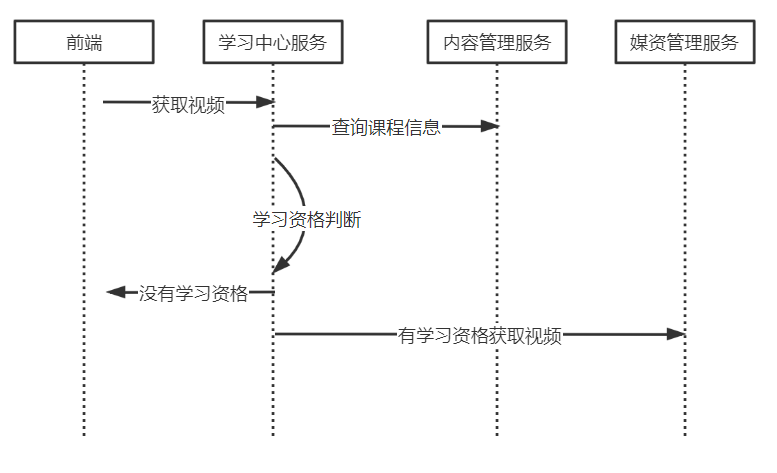
进入内容管理服务api工程CoursePublishController 类，定义查询课程预览信息接口如下：

|  |
| --- |
| 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}/{mediaId}")  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**

|  |
| --- |
| 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、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。

2、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

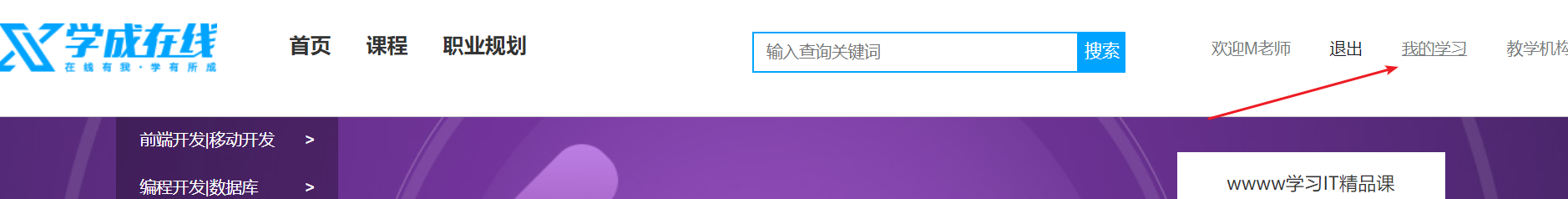
可修改选课记录表中的课程id为不存在进行测试，测试完再恢复原样。

**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.xuecheng-plus.com;  #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**

|  |
| --- |
| 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**

|  |
| --- |
| 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;  } |

**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。



**4.5 面试**

**项目使用redis缓存了哪些数据?**

redis缓存的是白名单接口（无需认证即可访问）所需要的数据，缓存了普通用户所要查询的数据（我的订单、我的选课），缓存热点数据（最新发布的课程信息、推荐课程信息等）。

每类信息有不同的缓存过期时间，为了避免缓存雪崩缓存时间加了随机数。

验证码

30秒 字符串

课程发布信息

7天 hash

课程视频信息

7天 hash

我的课程

3分钟 hash

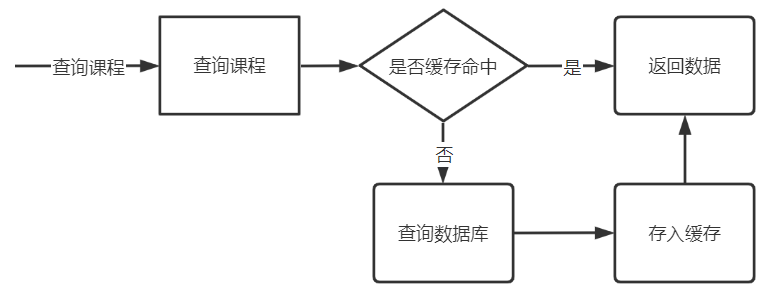
我的订单

3分钟 hash

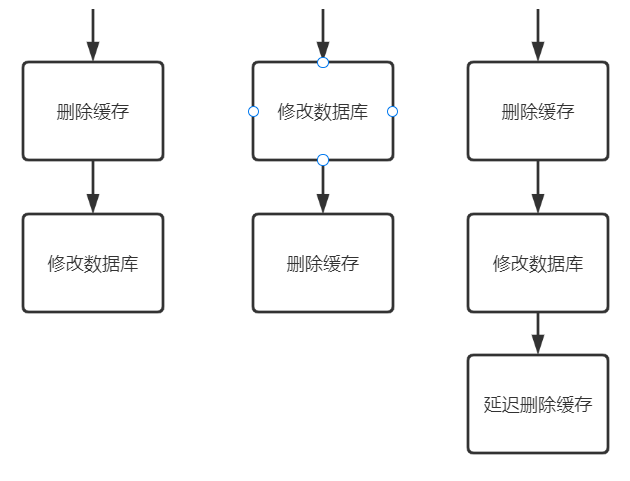
**如何保证Redis缓存一致性？**

缓存一致性是数据库和缓存保持一致，当修改了数据库的信息缓存的数据也要同时更新和数据库保持一致。

去查询数据时先查询缓存，如果缓存有就返回，如果没有就查询数据库，如果查不到则缓存一个null字符串（过期时间设置的小一些），如果查询到了，缓存到redis具体的信息。



当修改数据时常用的方案：



1、先删除缓存，再更新数据库

如果删除了缓存更新数据库的操作没有成功，此时查询数据的请求会把旧数据存储到缓存中。

2、先更新数据库，再删除缓存。

如果更新了数据库，删除缓存的操作失败了，此时查询数据的请求查到的数据仍然是旧数据。

3、延迟双删，先删除缓存、再更新数据库，再延迟一定的时间去删除缓存。

为什么要两次删除缓存，因为有可能第一次删除缓存后其它查询请求将旧数据存储到了缓存。

为什么要延迟一定的时间去删除缓存，为了给mysql主向从同步的时间，如果立即删除缓存很可能其它请求读到的数据还是旧数据。

延迟的时间不好确定，延迟双删仍然可能导致脏数据。

所以结论：以上方案当存在高并发时都无法解决数据库和缓存强一致性的问题。

如何做缓存一致性？

需要根据需求来定：

1、实现强一致性 需要使用分布式锁控制，修改数据和向缓存存储数据使用同一个分布式锁。

2、实现最终一致性，缓存数据要加过期时间，即使出现数据不致性当过期时间一到缓存失效又会从数据库查询最新的数据存入缓存。

3、对于实时性要求强的，要实现数据强一致性要尽量避免使用缓存，可以直接操作数据库。

使用工具对数据进行同步方案如下：

1、使用任务表加任务调度的方案进行同步。

2、使用Canal基于MySQL的binlog进行同步。

**redis内存回收机制是什么？**

1、过期删除策略

当redis中的key的生命时间到了，不会立即删除，使用下边的过期删除策略进行删除

定时删除：一个key设置一个定时器去，当key过期就删除，如果key比较多则设置的定时器也比较多，耗费cpu，或者一次随机扫描多个过期的key进行删除。

惰性删除：当去get查询key时如果过期再删除，优点是节省cpu但是浪费内存，因为如果一直不get则过期key将一直存在。

2、内存淘汰策略

两种算法：

LRU 全称为：Least Recently Used。即：最近最长时间未被使用。这个主要针对的是使用时间。

LFU 全称为：Least Frequently Used。即：最近最少频率使用，这个主要针对的是使用频率。



设置方法

|  |
| --- |
| Plain Text 获取当前内存淘汰策略： 127.0.0.1:6379> config get maxmemory-policy 通过配置文件设置淘汰策略（修改redis.conf文件） maxmemory-policy allkeys-lru  通过命令修改淘汰策略： 127.0.0.1:6379> config set maxmemory-policy allkeys-lru |

配置maxmemory\_samples选取固定数目的key，然后比较它们的lru访问时间，然后淘汰最近最久没有访问的key

样本的范围大小决定了性能，

* 范围越大：越精确，性能越差；
* 范围越小：越不精确，性能越好。

maxmemory-samples 5 # 保持默认

**面对高并发是如何对缓存进行处理的**

首先使用redis集群。

其次使用分布式锁进行控制避免缓存击穿，通过分布式锁控制只有一个线程去查询数据库，查完数据库后存入缓存。

我们使用redisson实现分布锁。

redisson实现Lock接口，基于此接口使用，具体方法是获取锁调用lock()方法，用完释放锁调用unlock()。

当线程还没有执行完时会有看门狗对锁进行续期，保证线程在执行过程中不会让锁过期。

缓存雪崩问题。

缓存穿透问题。

缓存一致性问题。