

万信金融 第3章 讲义-发标

1 需求分析

1.1 什么是发标

P2P行业习惯把平台里某个投资项目称为“标的”，简称“标”。一个标的一般至少包含：描述、借款用途、借款总额、还款方式、借款利率、借款期限等基本信息。通俗来讲：

- “标的”就是：借款人在P2P平台发起的借款项目。
- “发标”就是：借款人在P2P平台申请借款。

<

借钱

凭信用卡/身份证 申请借款

额度高至 3W

申请借款金额(元)

200030,00050000

借款期限

12个月

借款用途

个人生活消费

☐ 我已阅读产品详情页面，知悉并接受产品申请条件及费用说明，并同意《借款相关协议》

立即申请

<

精英标

日常消费(男,59岁,内蒙古自治区乌兰察布市)

12%36月70000元

借款年利率借款期限借款金额

抢

购物消费(男,42岁,河南省郑州市)

11%24月117076.16元

借款年利率借款期限借款金额

抢

购物消费(男,29岁,广东省深圳市)

12%36月76848.35元

借款年利率借款期限借款金额

满

购物消费(男,39岁,上海市)

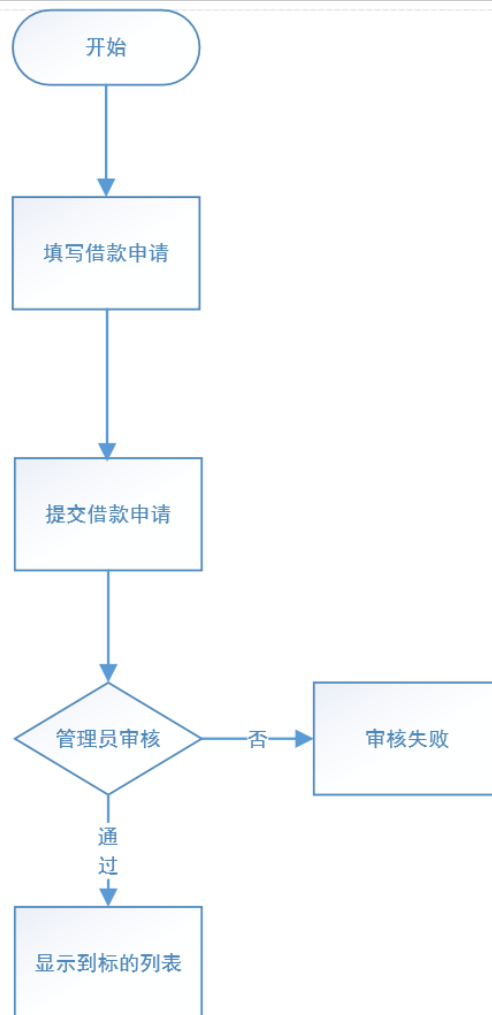
12%36月97333.33元

借款年利率借款期限借款金额

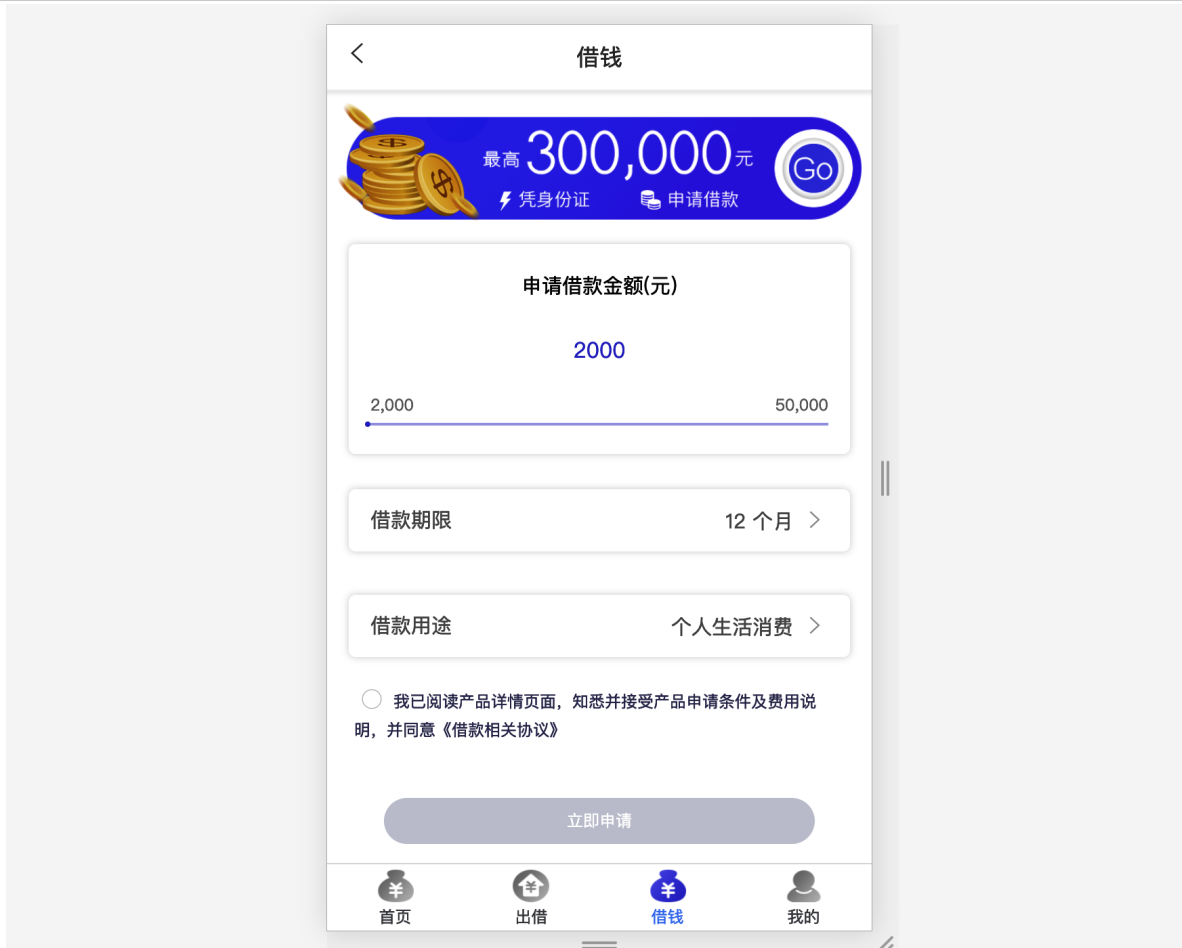
满

1.2 业务流程

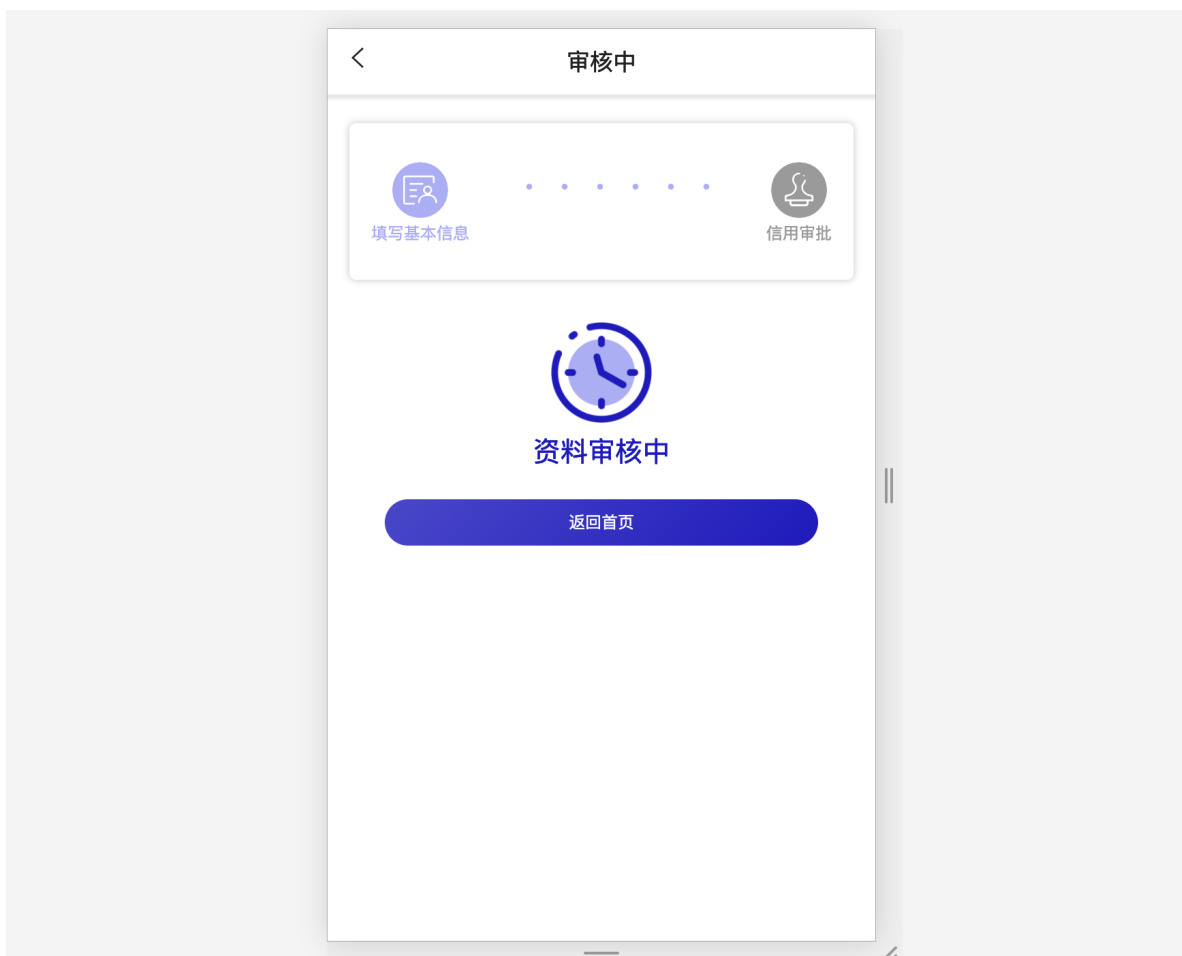
1. 发标流程如下：



2. 填写借款信息



3. 申请成功，等待审核



4. P2P平台管理员审核借款信息

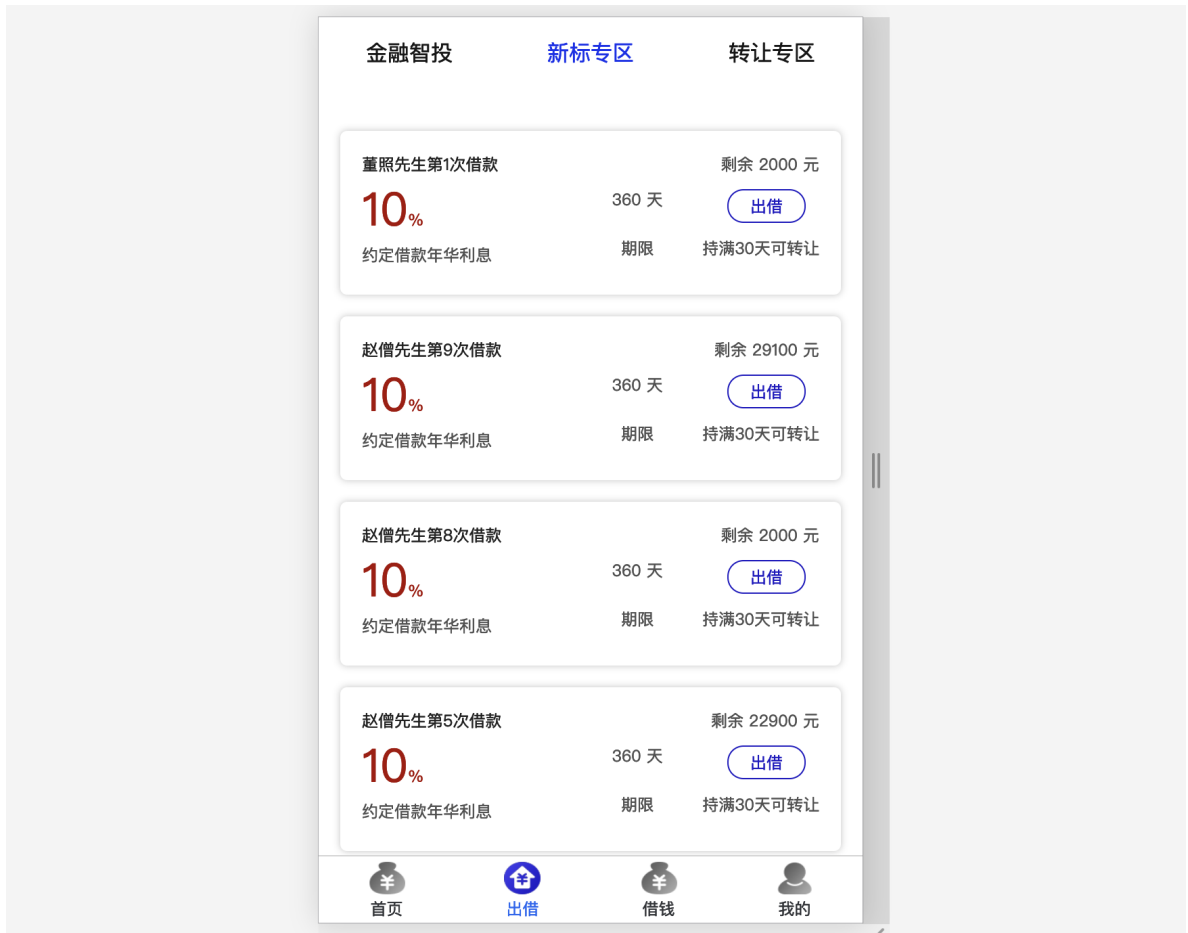
万信金融管理平台

Dashboard / 准备中借款管理 / 准备中待审核

admin | 退出

名称	金额(元)	创建时间	期限(月)	状态	年化利率	操作
董照先生第1次借款	2000	2019-07-31	12	募集中	10%	审核
赵僧先生第9次借款	29100	2019-07-31	12	募集中	10%	审核

5. 审核通过后，就可在前端出借列表中看到标的信息



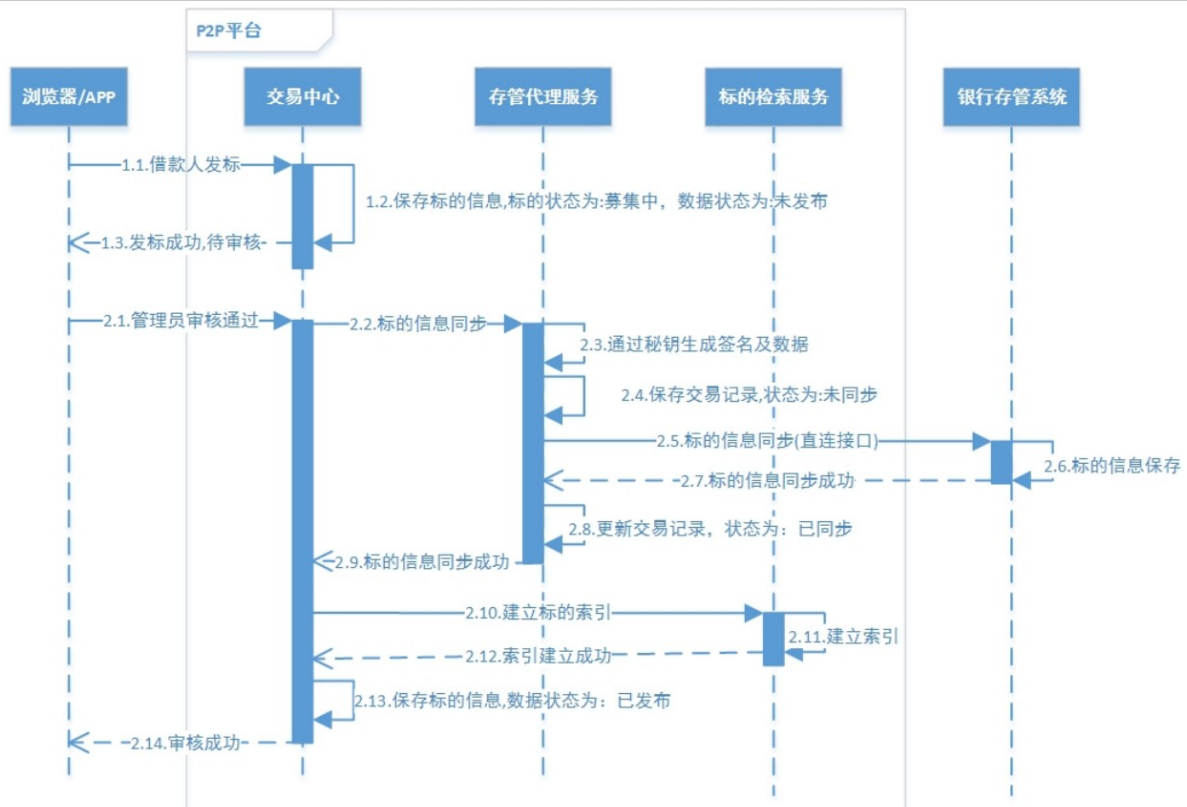
2 分库分表

参考资料文件夹中的“分库分表专题”

3 用户发标

3.1 需求分析

“发标”就是：借款人在P2P平台申请借款，具体交互流程如下：



1. 用户在前端填写借款信息
2. 前端请求交易中心保存标的信息
3. 管理员审核标的信息
4. 交易中心请求存管代理服务生成交易记录（未同步），并对标的数据进行签名
5. 存管代理服务携带标的的签名数据请求银行存管系统
6. 银行存管系统保存标的的信息，返回同步成功给存管代理服务
7. 存管代理服务更新交易记录，返回同步成功给交易中心
8. 交易中心确认同步成功，更新标的的信息

3.2 交易中心保存标的的信息

3.2.1 定义接口

1. 在wanxinp2p-api工程中新建一个transaction包，在该包中定义一个TransactionApi接口，在该接口中定义一个发标的方法：

```

/**
 * <P>
 * 交易中心服务API
 * </p>
 */
public interface TransactionApi {
    /**
     * 借款人发标
     * @param projectDTO
     * @return
     */
    RestResponse<ProjectDTO> createProject(ProjectDTO projectDTO);
}
    
```

2. 在wanxinp2p-transaction-service工程中，定义TransactionController类实现createProject方法：

```
@Api(value = "交易中心服务", tags = "transaction")
@RestController
public class TransactionController implements TransactionApi {
    @Override
    @ApiOperation("借款人发标")
    @ApiImplicitParam(name = "project", value = "标的信息", required = true,
        dataType = "Project", paramType = "body")
    @PostMapping("/my/projects")
    public RestResponse<ProjectDTO> createProject(@RequestBody ProjectDTO
projectDTO) {
        return null;
    }
}
```

3.2.2 功能实现

1. 在Mapper包中定义ProjectMapper接口，并继承MP的BaseMapper接口：

```
/**
 * <p>
 * 标的信息Mapper接口
 * </p>
 */
public interface ProjectMapper extends BaseMapper<Project> {
}
```

2. 在Mapper包中新建ProjectMapper.xml文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.itcast.wanxinp2p.transaction.mapper.ProjectMapper">

</mapper>
```

3. 在Service包中新建ProjectService接口，并定义createProject方法：

```
/**
 * <P>
 * 交易中心service接口
 * </p>
 */
public interface ProjectService {
    /**
     * 创建标的
     *
     * @param project
     * @return ProjectDTO
     */
    ProjectDTO createProject(ProjectDTO project);
}
```

4. 由于保存标的时，需要用到当前登录用户的一些信息，因此交易中心需要远程访问用户中心，在ConsumerAPI接口中新增一个获得当前登录用户的方法：

```
/**
 * 获得当前登录用户
 * @return
 */
RestResponse<ConsumerDTO> getCurrConsumer();
```

在ConsumerController类中实现该方法：

```
@Override
@ApiOperation("获取登录用户信息")
@GetMapping("/1/currConsumer")
public RestResponse<ConsumerDTO> getCurrConsumer() {
    ConsumerDTO consumerDTO = consumerService
        .getByMobile(SecurityUtil.getUser().getMobile());
    return RestResponse.success(consumerDTO);
}
```

5. 在交易中心微服务工程中创建Feign代理，远程访问上面的Controller

```
/**
 * C端服务代理
 */
@FeignClient(value = "consumer-service")
public interface ConsumerApiAgent {

    @GetMapping("/consumer/1/currConsumer")
    public RestResponse<ConsumerDTO> getCurrConsumer();

}
```

6. 在service包中新建ProjectServiceImpl类实现createProject方法：

```
@Service
@Slf4j
public class ProjectServiceImpl extends ServiceImpl<ProjectMapper, Project>
implements ProjectService {

    @Autowired
    private ConsumerApiAgent consumerApiAgent;

    @Override
    public ProjectDTO createProject(ProjectDTO projectDTO) {
        final RestResponse<ConsumerDTO> consumer =
            consumerApiAgent.getCurrConsumer();

        // 设置用户编码
        projectDTO.setUserNo(consumer.getResult().getUserNo());
        // 设置用户id
        projectDTO.setConsumerId(consumer.getResult().getId());
        // 生成标的编码
```

```

projectDTO.setProjectNo(CodeNoUtil.getNo(CodePrefixCode.CODE_PROJECT_PREFIX));
// 标的状态修改
projectDTO.setProjectStatus(ProjectCode.COLLECTING.getCode());
// 标的可用状态修改，未同步
projectDTO.setStatus(StatusCode.STATUS_OUT.getCode());
// 设置标的创建时间
projectDTO.setCreateDate(LocalDateTime.now());
// 设置还款方式
projectDTO.setRepaymentWay(RepaymentWayCode.FIXED_REPAYMENT.getCode());
// 设置标的类型
projectDTO.setType("NEW");
// 把dto转换为entity
final Project project = convertProjectDTOToEntity(projectDTO);
// 设置利率(需要在Apollo上进行配置)
// 年化利率(借款人视图)
project.setBorrowerAnnualRate(configService.getBorrowerAnnualRate());
// 年化利率(投资人视图)
project.setAnnualRate(configService.getAnnualRate());
// 年化利率(平台佣金，利差)

project.setCommissionAnnualRate(configService.getCommissionAnnualRate());
// 债权转让
project.setIsAssignment(0);
// 设置标的名字，姓名+性别+第N次借款
// 判断男女
String sex = Integer.parseInt(consumer.getResult().getIdNumber()
    .substring(16, 17)) % 2 == 0 ? "女士" : "先生";
// 构造借款次数查询条件
QueryWrapper<Project> queryWrapper = new QueryWrapper<>();
queryWrapper.lambda().eq(Project::getConsumerId,
consumer.getResult().getId());
project.setName(
    consumer.getResult().getFullname() + sex
    + "第" + (count(queryWrapper) + 1) + "次借款");
// 保存到数据库
save(project);
// 设置主键
projectDTO.setId(project.getId());
projectDTO.setName(project.getName());
return projectDTO;
}

private Project convertProjectDTOToEntity(ProjectDTO projectDTO) {
    if (projectDTO == null) {
        return null;
    }
    Project project = new Project();
    BeanUtils.copyProperties(projectDTO, project);
    return project;
}
}
    
```

7. 完善TransactionController的代码，调用ProjectService完成保存标的信息的功能：

```
@Api(value = "交易中心服务", tags = "transaction")
```



```
@RestController
public class TransactionController implements TransactionApi {

    @Autowired
    private ProjectService projectService;

    @Override
    @ApiOperation("借款人发标")
    @ApiImplicitParam(name = "project", value = "标的信息", required = true,
        dataType = "Project", paramType = "body")
    @PostMapping("/my/projects")
    public RestResponse<ProjectDTO> createProject(@RequestBody ProjectDTO
projectDTO) {
        ProjectDTO dto = projectService.createProject(projectDTO);
        return RestResponse.success(dto);
    }
}
```

3.2.3 发标业务的先决条件

用户登录成功后，可以直接进行发标，但是根据业务需求，只有在已开户的情况下才可以发标。因此在发标时，前端需要判断当前登录用户是否已经开户(绑定银行卡)，如果未开，就跳转到开户界面，如果已开就继续发标。同时，我们还要判断用户申请的借款金额是否超标，如果超标就不允许发标。要想知道当前登录用户是否开户以及可贷额度，就必须获取当前登录用户的相关信息，并返回给前端使用。

1. 在ConsumerAPI接口中新增获取当前登录用户的方法：

```
/**
 * 获取当前登录用户
 * @return
 */
RestResponse<ConsumerDTO> getMyConsumer();
```

2. 在ConsumerController类中实现该方法：

```
@Override
@ApiOperation("获取登录用户信息")
@GetMapping("/my/consumers")
public RestResponse<ConsumerDTO> getMyConsumer() {
    ConsumerDTO consumerDTO = consumerService
        .getByMobile(SecurityUtil.getUser().getMobile());
    return RestResponse.success(consumerDTO);
}
```

consumerDTO中的loanAmount属性表示可贷额度，isBindCard表示是否开户，前端发标页面会自动访问该方法，获得这些数据后会自动进行业务处理。

注意：每个借款人的可贷额度都不相同，这个跟借款人自身的综合情况(信用、月收入、工作性质等)有关，可贷额度由P2P平台根据借款人自身的综合情况进行评估并设置，这里暂时简化处理，由我们自己直接去表里设置值即可(不能低于2000)。

表: consumer

Columns (13)

Field	Type	Comment
ID	bigint(20) NOT NULL	主键
USERNAME	varchar(50) NOT NULL	用户名
FULLNAME	varchar(50) NULL	真实姓名
ID_NUMBER	varchar(50) NULL	身份证号
USER_NO	varchar(50) NULL	用户编码,生成唯一,用户在存管系统标识
MOBILE	varchar(50) NULL	平台预留手机号
USER_TYPE	varchar(50) NULL	用户类型,个人or企业, 预留
ROLE	varchar(50) NULL	用户角色:B借款人or I投资人
AUTH_LIST	varchar(50) NULL	存管授权列表
IS_BIND_CARD	tinyint(1) NULL	是否开户(已绑定银行卡)
LOAN_AMOUNT	decimal(10,0) NULL	可贷额度
STATUS	tinyint(1) NULL	可用状态
REQUEST_NO	varchar(50) NULL	请求流水号

3.2.4 前后端集成测试

由于要进行前后端集成测试，所以一定要记得在Apollo的网关项目中新增路由配置：

application

表格	文本	更改历史	实例列表 1
1	zuul.routes.consumer-service.stripPrefix = false		
2	zuul.routes.consumer-service.path = /consumer/**		
3	zuul.routes.account-service.stripPrefix = false		
4	zuul.routes.account-service.path = /account/**		
5	zuul.routes.uaa-service.stripPrefix = false		
6	zuul.routes.uaa-service.path = /uaa/**		
7	zuul.routes.transaction-service.stripPrefix = false		
8	zuul.routes.transaction-service.path = /transaction/**		
9			
10	zuul.retryable = true		
11	zuul.add-host-header = true		
12	zuul.ignoredServices = *		

细节问题：SecurityUtil.getUser()不能随意使用，必须是收到了网关转发的请求的微服务才可以使用。请大家参考视频或课件源码修复前面的代码。

3.3 部署P2P管理平台前端工程

交易中心保存完标的信息后，P2P公司管理员就需要登录万信金融管理平台对该标的进行审核，该功能同样放在wanxinp2p-transaction-service工程中实现，但是该功能需要一个独立的前端(管理平台)作为审核的入口。

万信金融管理平台

首页

准备中待审核

满标放款审核

Dashboard / 准备中借款管理 / 准备中待审核

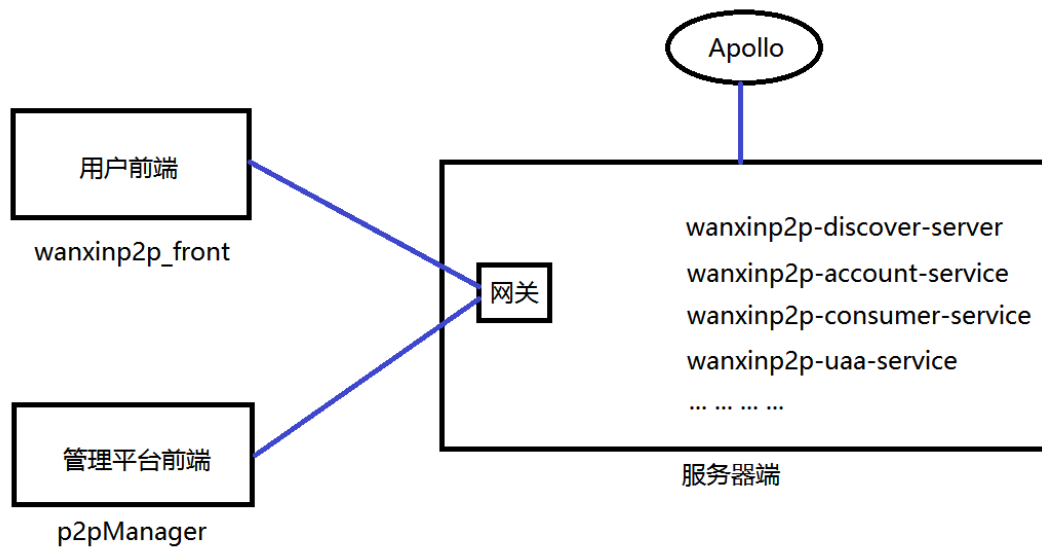
admin

退出

名称	金额(元)	创建时间	期限(月)	状态	年化利率	操作
董照先生第1次借款	2000	2019-07-31	12	募集中	10%	审核
赵僧先生第9次借款	29100	2019-07-31	12	募集中	10%	审核

3.3.1 打开前端工程

在资料文件夹中找到p2pManager，这是万信金融管理平台的前端工程，在HBuilder X中打开。该前端工程使用 vue-element-admin 集成方案，具体采用了vue [2.6.10] + element-ui[2.7.0] + axios技术栈。该前端工程直接提供给大家使用，无需我们动手编码。



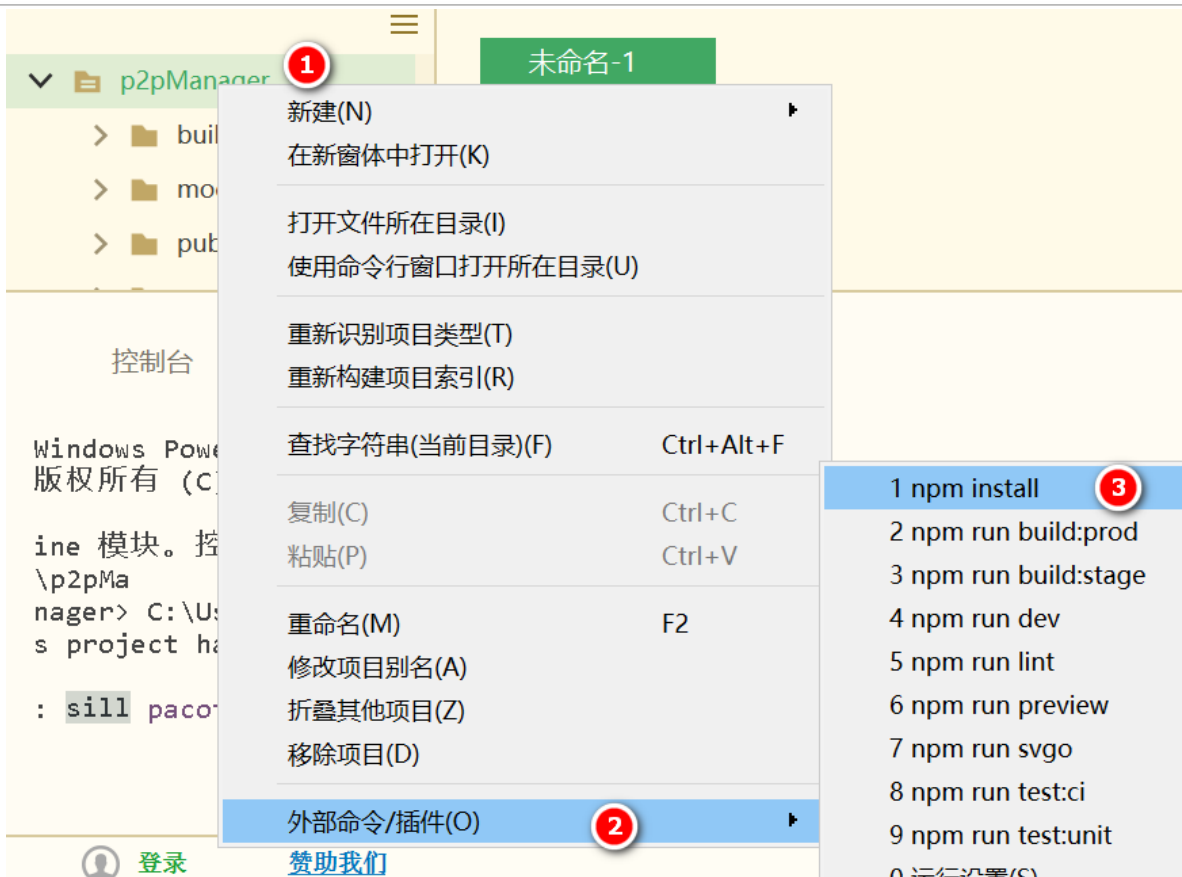
3.3.2 搭建运行环境

1. 在HBuilder中安装一个插件：内置终端

H 插件安装

emmet JAVA插件	通过缩写快速编写HTML、CSS代码
NPM Node.js插件	nodejs包管理，众多插件的安装基础
内置浏览器 C++插件	内置浏览器，支持边改边预览
内置终端 C++插件	内置的命令行终端，可在菜单视图中呼出

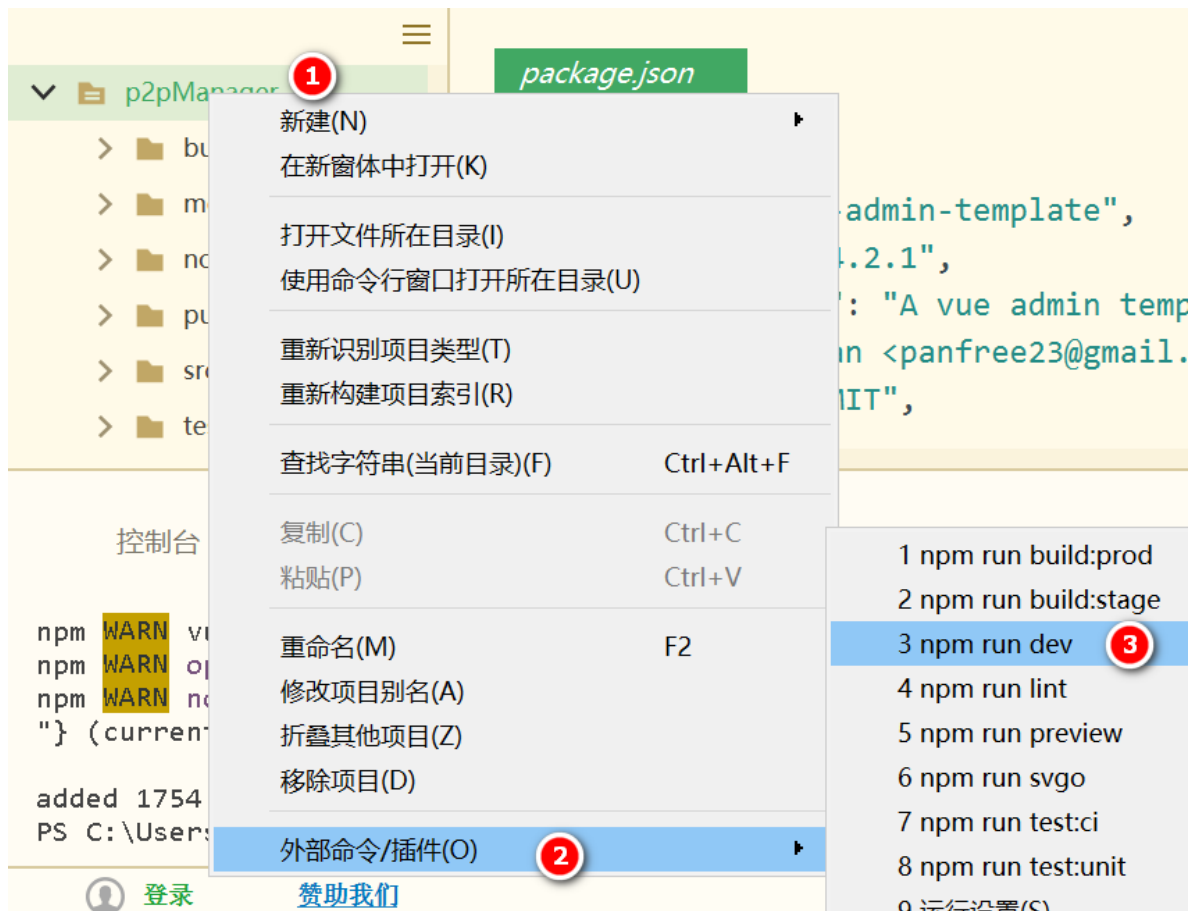
2. 在工程名上点击右键，执行npm install菜单，安装依赖库，如果没有npm install菜单，直接跳过该步骤



3. 执行npm install后，会在下方看到日志输出，等待联网下载依赖库



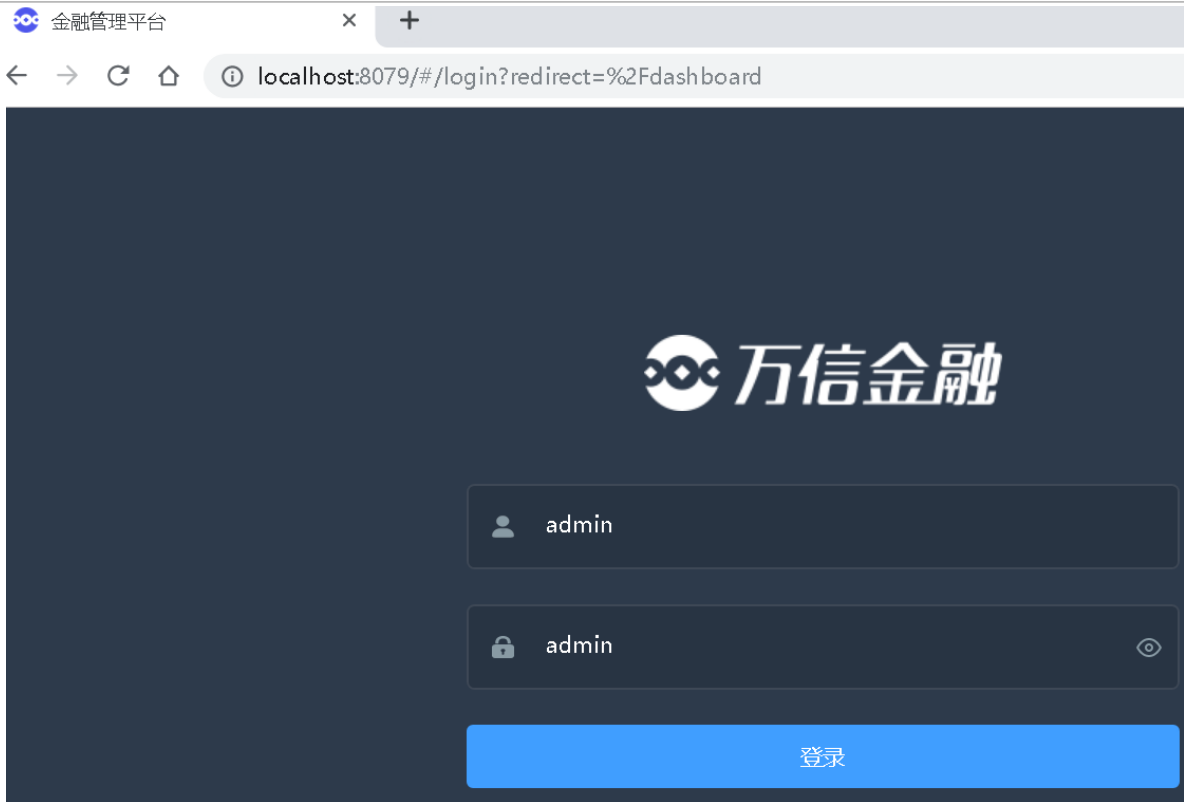
4. 在工程名上点击右键，执行npm run dev菜单，启动该前端工程



5. 执行npm run dev后，会在下方看到日志输出，前端工程占用8079端口



6. 在浏览器中访问登录页面，账号和密码都是admin。要想登录成功，需要启动Apollo以及wanxinp2p-discover-server、wanxinp2p-account-service、wanxinp2p-consumer-service、wanxinp2p-gateway-server、wanxinp2p-uaa-service等微服务。



3.4 查询标的信息

审核标的业务需要首先实现标的信息的分页查询，查询出来的标的信息会在管理平台中展示出来，只有P2P公司的管理员看到这些标的信息了，才能进行审核操作。

 万信金融管理平台

首页

准备中待审核

满标放款审核

Dashboard / 准备中借款管理 / 准备中待审核

admin

退出

名称	金额(元)	创建时间	期限(月)	状态	年化利率	操作
董照先生第1次借款	2000	2019-07-31	12	募集中	10%	审核
赵借先生第9次借款	29100	2019-07-31	12	募集中	10%	审核

3.4.1 定义接口

1. 在TransactionApi接口中新增查询方法queryProjects，实现带条件的分页查询：

```
/**
 * 检索标的信息
 * @param projectQueryDTO 封装查询条件
 * @param order
 * @param pageNo
 * @param pageSize
 * @param sortBy
 * @return
 */
RestResponse<PageVO<ProjectDTO>> queryProjects(ProjectQueryDTO projectQueryDTO,
String order, Integer pageNo,
Integer pageSize, String sortBy);
```

2. 在TransactionController类中实现该方法：

```
@Override
```

```
@ApiOperation("检索标的信息")
@ApiImplicitParams({
    @ApiImplicitParam(name = "projectQueryDTO", value = "标的信息查询对象",
        required = true, dataType = "ProjectQueryDTO", paramType =
        "body"),
    @ApiImplicitParam(name = "order", value = "顺序", required = false,
        dataType = "string", paramType =
        "query"),
    @ApiImplicitParam(name = "pageNo", value = "页码", required = true,
        dataType = "int", paramType =
        "query"),
    @ApiImplicitParam(name = "pageSize", value = "每页记录数", required =
        true,
        dataType = "int", paramType =
        "query"),
    @ApiImplicitParam(name = "sortBy", value = "排序字段", required = true,
        dataType = "string", paramType =
        "query")})
@PostMapping("/projects/q")
public RestResponse<PageVO<ProjectDTO>> queryProjects(@RequestBody
    ProjectQueryDTO projectQueryDTO, String order, Integer pageNo, Integer pageSize,
    String sortBy) {
    return null;
}
```

3.4.2 功能实现

1. 数据访问层在前面保存标的信息时已经搞定，这里不用再管了
2. 在ProjectService接口中新增一个查询方法：

```
/**
 * 根据分页条件检索标的信息
 *
 * @param projectQueryDTO
 * @param order
 * @param pageNo
 * @param pageSize
 * @param sortBy
 * @return
 */
PageVO<ProjectDTO> queryProjectsByQueryDTO(ProjectQueryDTO projectQueryDTO,
    String order, Integer pageNo, Integer pageSize, String
    sortBy);
```

3. 在ProjectServiceImpl类中实现该方法：

```
@Override
public PageVO<ProjectDTO> queryProjectsByQueryDTO(ProjectQueryDTO
    projectQueryDTO,
    String order, Integer pageNo, Integer pageSize, String
    sortBy){

    // 条件构造器
    QueryWrapper<Project> queryWrapper = new QueryWrapper();
    // 标的类型
    if (StringUtils.isNotBlank(projectQueryDTO.getType())) {
```



```

        queryWrapper.lambda().eq(Project::getType, projectQueryDTO.getType());
    }
    // 起止年化利率(投资人) -- 区间
    if (null != projectQueryDTO.getStartAnnualRate()) {
        queryWrapper.lambda().ge(Project::getAnnualRate,
            projectQueryDTO.getStartAnnualRate());
    }
    if (null != projectQueryDTO.getEndAnnualRate()) {
        queryWrapper.lambda().le(Project::getAnnualRate,
            projectQueryDTO.getStartAnnualRate());
    }
    // 借款期限 -- 区间
    if (null != projectQueryDTO.getStartPeriod()) {
        queryWrapper.lambda().ge(Project::getPeriod,
            projectQueryDTO.getStartPeriod());
    }
    if (null != projectQueryDTO.getEndPeriod()) {
        queryWrapper.lambda().le(Project::getPeriod,
            projectQueryDTO.getEndPeriod());
    }
    // 标的状态
    if (StringUtils.isNotBlank(projectQueryDTO.getProjectStatus())) {
        queryWrapper.lambda().eq(Project::getProjectStatus,
            projectQueryDTO.getProjectStatus());
    }

    // 构造分页对象
    Page<Project> page = new Page<>(pageNo, pageSize);

    // 处理排序 order值: desc 或者 asc
    if (StringUtils.isNotBlank(order) && StringUtils.isNotBlank(sortBy)) {
        if (order.toLowerCase().equals("asc")) {
            queryWrapper.orderByAsc(sortBy);
        } else if (order.toLowerCase().equals("desc")) {
            queryWrapper.orderByDesc(sortBy);
        }
    } else {
        //默认按发标时间倒序排序
        queryWrapper.lambda().orderByDesc(Project::getCreateDate);
    }
    // 执行查询
    IPage<Project> projectIPage = page(page, queryWrapper);

    // ENTITY转换为DTO, 不向外部暴露ENTITY
    List<ProjectDTO> dtoList =
        convertProjectEntityListToDTOList(projectIPage.getRecords());
    // 封装结果集
    return new PageVO<>(dtoList, projectIPage.getTotal(), pageNo, pageSize);
}

private List<ProjectDTO> convertProjectEntityListToDTOList(List<Project>
    projectList) {
    if (projectList == null) {
        return null;
    }
    List<ProjectDTO> dtoList = new ArrayList<>();
    projectList.forEach(project -> {
        ProjectDTO projectDTO = new ProjectDTO();
    });
}

```



```
        BeanUtils.copyProperties(project, projectDTO);
        dtoList.add(projectDTO);
    });
    return dtoList;
}
```

4. 完善TransactionController类中的代码：

```
@PostMapping("/projects/q")
public RestResponse<PageVO<ProjectDTO>> queryProjects(@RequestBody
ProjectQueryDTO projectQueryDTO, String order, Integer pageNo, Integer pageSize,
String sortBy) {
    PageVO<ProjectDTO> projects =
projectService.queryProjectsByQueryDTO(projectQueryDTO, order, pageNo, pageSize,
sortBy);
    return RestResponse.success(projects);
}
```

3.4.3 前后端集成测试

3.5 审核标的

用户发标后，P2P平台的管理员会对标的信息进行审核，审核通过后会通过存管代理服务将标的信息同步到银行存管系统，请通过前面的时序图回顾具体的业务流程。

3.5.1 定义接口

3.5.1.1 交易中心审核标的接口

1. 在TransactionApi接口中定义一个审核标的信息的方法：

```
/**
 * 管理员审核标的信息
 *
 * @param id
 * @param approveStatus
 * @return
 */
RestResponse<String> projectsApprovalStatus(Long id, String approveStatus);
```

2. 在TransactionController类中实现projectsApprovalStatus方法：

```
@Override
@ApiOperation("管理员审核标的信息")
@ApiImplicitParams({
    @ApiImplicitParam(name = "id", value = "标的id", required = true,
        dataType = "long", paramType = "path"),
    @ApiImplicitParam(name = "approveStatus", value = "审批状态",
        required = true, dataType = "ref", paramType = "path")
})
@PutMapping("/m/projects/{id}/projectStatus/{approveStatus}")
```

```
public RestResponse<String> projectsApprovalStatus(  
    @PathVariable("id") Long id,  
    @PathVariable("approveStatus") String approveStatus){  
    return null;  
}
```

3.5.1.2 存管代理服务标的同步接口

1. 在DepositoryAgentApi接口中定义一个向银行存管系统发送标的信息的方法：

```
/**  
 * 向银行存管系统发送标的信息  
 *  
 * @param projectDTO  
 * @return  
 */  
RestResponse<String> createProject(ProjectDTO projectDTO);
```

2. 在DepositoryAgentController类中实现该方法：

```
@Override  
@ApiOperation(value = "向存管系统发送标的信息")  
@ApiImplicitParam(name = "projectDTO", value = "向存管系统发送标的信息",  
    required = true, dataType = "ProjectDTO", paramType = "body")  
@PostMapping("/l/createProject")  
public RestResponse<String> createProject(@RequestBody ProjectDTO projectDTO) {  
    return null;  
}
```

3.5.2 功能实现

3.5.2.1 交易中心审核标的信息

1. 定义一个Feign代理接口，用来向存管代理微服务发送标的信息：

```
@FeignClient(value = "depository-agent-service")  
public interface DepositoryAgentApiAgent {  
    @PostMapping(value = "/depository-agent/l/createProject")  
    public RestResponse<String> createProject(ProjectDTO projectDTO);  
}
```

2. 在ProjectService接口中新增projectsApprovalStatus方法，用来审核标的信息：

```
/**  
 * 管理员审核标的信息  
 *  
 * @param id  
 * @param approveStatus  
 * @return String  
 */  
String projectsApprovalStatus(Long id, String approveStatus);
```

3. 在ProjectServiceImpl类中实现该方法：

```

@Override
public String projectsApprovalStatus(Long id, String approveStatus) {
    // 1. 根据id查询出标的信息并转换为DTO对象
    final Project project = getById(id);
    final ProjectDTO projectDTO = convertProjectEntityToDTO(project);
    // 2. 生成请求流水号

    projectDTO.setRequestNo(CodeNoUtil.getNo(CodePrefixCode.CODE_REQUEST_PREFIX));
    // 3. 调用存管代理服务同步标的信息
    final RestResponse<String> restResponse = depositoryAgentApiAgent

.createProject(projectDTO);
    if (DepositoryReturnCode.RETURN_CODE_00000.getCode()
        .equals(restResponse.getResult())) {
        // 4. 修改状态为：已发布
        update(Wrappers.<Project>lambdaUpdate().set(Project::getStatus,
            Integer.parseInt(approveStatus)).eq(Project::getId,
id));
        return "success";
    }
    // 5. 失败抛出一个业务异常
    throw new BusinessException(TransactionErrorCode.E_150113);
}

private ProjectDTO convertProjectEntityToDTO(Project project) {
    if (project == null) {
        return null;
    }
    ProjectDTO projectDTO = new ProjectDTO();
    BeanUtils.copyProperties(project, projectDTO);
    return projectDTO;
}
    
```

4. 完善TransactionController类中的代码：

```

@PutMapping("/m/projects/{id}/projectStatus/{approveStatus}")
public RestResponse<String> projectsApprovalStatus(
    @PathVariable("id") Long id,
    @PathVariable("approveStatus") String approveStatus) {
    String result = projectService.projectsApprovalStatus(id, approveStatus);
    return RestResponse.success(result);
}
    
```

3.5.2.2 存管代理服务标的同步

1. 在DepositoryRecordService接口中新增createProject方法：

```

/**
 * 保存标的
 * @param projectDTO
 * @return
 */
DepositoryResponseDTO<DepositoryBaseResponse> createProject(ProjectDTO
projectDTO);
    
```

2. 在DepositoryRecordServiceImpl类中实现该方法:

```
@Override
public DepositoryResponseDTO<DepositoryBaseResponse> createProject(ProjectDTO
projectDTO) {
    //1. 保存交易记录
    DepositoryRecord depositoryRecord =
        saveDepositoryRecord(projectDTO.getRequestNo(),
            DepositoryRequestTypeCode.CREATE.getCode(), "Project",
            projectDTO.getId());

    //2. 签名数据
    // ProjectDTO 转换为 ProjectRequestDataDTO
    ProjectRequestDataDTO projectRequestDataDTO =
        convertProjectDTOToProjectRequestDataDTO(projectDTO,
            depositoryRecord.getRequestNo());
    //转换为JSON
    String jsonString=JSON.toJSONString(projectRequestDataDTO);
    //base64编码
    String reqData=EncryptUtil.encodeUTF8StringBase64(jsonString);

    //3. 往银行存管系统发送数据(标的信息),根据结果修改状态并返回结果
    // url地址    发送哪些数据
    String url=configService.getDepositoryUrl()+"/service";
    // 怎么发 OKHttpClient    发送HttpRequest
    return sendHttpGet("CREATE_PROJECT",url,reqData,depositoryRecord);
}
/**
 * 保存交易记录
 */
private DepositoryRecord saveDepositoryRecord(String requestNo,
                                                String requestType,
                                                String objectType,
                                                Long objectId) {

    DepositoryRecord depositoryRecord = new DepositoryRecord();
    // 设置请求流水号
    depositoryRecord.setRequestNo(requestNo);
    // 设置请求类型
    depositoryRecord.setRequestType(requestType);
    // 设置关联业务实体类型
    depositoryRecord.setObjectType(objectType);
    // 设置关联业务实体标识
    depositoryRecord.setObjectId(objectId);
    // 设置请求时间
    depositoryRecord.setCreateDate(LocalDateTime.now());
    // 设置数据同步状态
    depositoryRecord.setRequestStatus(StatusCode.STATUS_OUT.getCode());
    // 保存数据
    save(depositoryRecord);
    return depositoryRecord;
}

private ProjectRequestDataDTO convertProjectDTOToProjectRequestDataDTO(
    ProjectDTO projectDTO, String requestNo) {
    if(projectDTO==null){
        return null;
    }
}
```

```
ProjectRequestDataDTO requestDataDTO = new ProjectRequestDataDTO();
BeanUtils.copyProperties(projectDTO, requestDataDTO);
requestDataDTO.setRequestNo(requestNo);
return requestDataDTO;
}
```

3. 复制OkHttpService.java和SignatureInterceptor.java到存管代理工程中

OkHttpService类封装了一个方法，该方法通过OkHttpClient发送HttpRequest，我们通过该方法往银行存管系统发起请求。SignatureInterceptor类是一个通用请求拦截器，用来拦截请求并进行签名和验签。

4. 向银行存管系统发送数据

```
private DepositoryResponseDTO<DepositoryBaseResponse> sendHttpGet(
    String serviceName, String url, String reqData,
    DepositoryRecord depositoryRecord){
    // 银行存管系统接收的4大参数: serviceName, platformNo, reqData, signature
    // signature会在okHttp拦截器(SignatureInterceptor)中处理
    // 平台编号
    String platformNo = configService.getP2pCode();
    // redData签名
    // 发送请求，获取结果，如果检验签名失败，拦截器会在结果中放入: "signature", "false"
    String responseBody = okHttpService
        .doSyncGet(url + "?serviceName=" + serviceName + "&platformNo=" +
            platformNo + "&reqData=" + reqData);
    DepositoryResponseDTO<DepositoryBaseResponse> depositoryResponse = JSON
        .parseObject(responseBody,
            new TypeReference<DepositoryResponseDTO<DepositoryBaseResponse>>()
        {
            });

    // 响应后，根据结果更新数据库(进行签名判断)
    // 判断签名(signature)是为 false，如果是说明验签失败！
    if (!"false".equals(depositoryResponse.getSignature())) {
        // 成功 - 设置数据同步状态
        depositoryRecord.setRequestStatus(StatusCode.STATUS_IN.getCode());
        // 设置消息确认时间
        depositoryRecord.setConfirmDate(LocalDateDateTime.now());
        // 更新数据库
        updateById(depositoryRecord);
    } else {
        // 失败 - 设置数据同步状态
        depositoryRecord.setRequestStatus(StatusCode.STATUS_FAIL.getCode());
        // 设置消息确认时间
        depositoryRecord.setConfirmDate(LocalDateDateTime.now());
        // 更新数据库
        updateById(depositoryRecord);
        // 抛业务异常
        throw new BusinessException(DepositoryErrorCode.E_160101);
    }
    return depositoryResponse;
}
```

5. 完善DepositoryAgentController类的代码

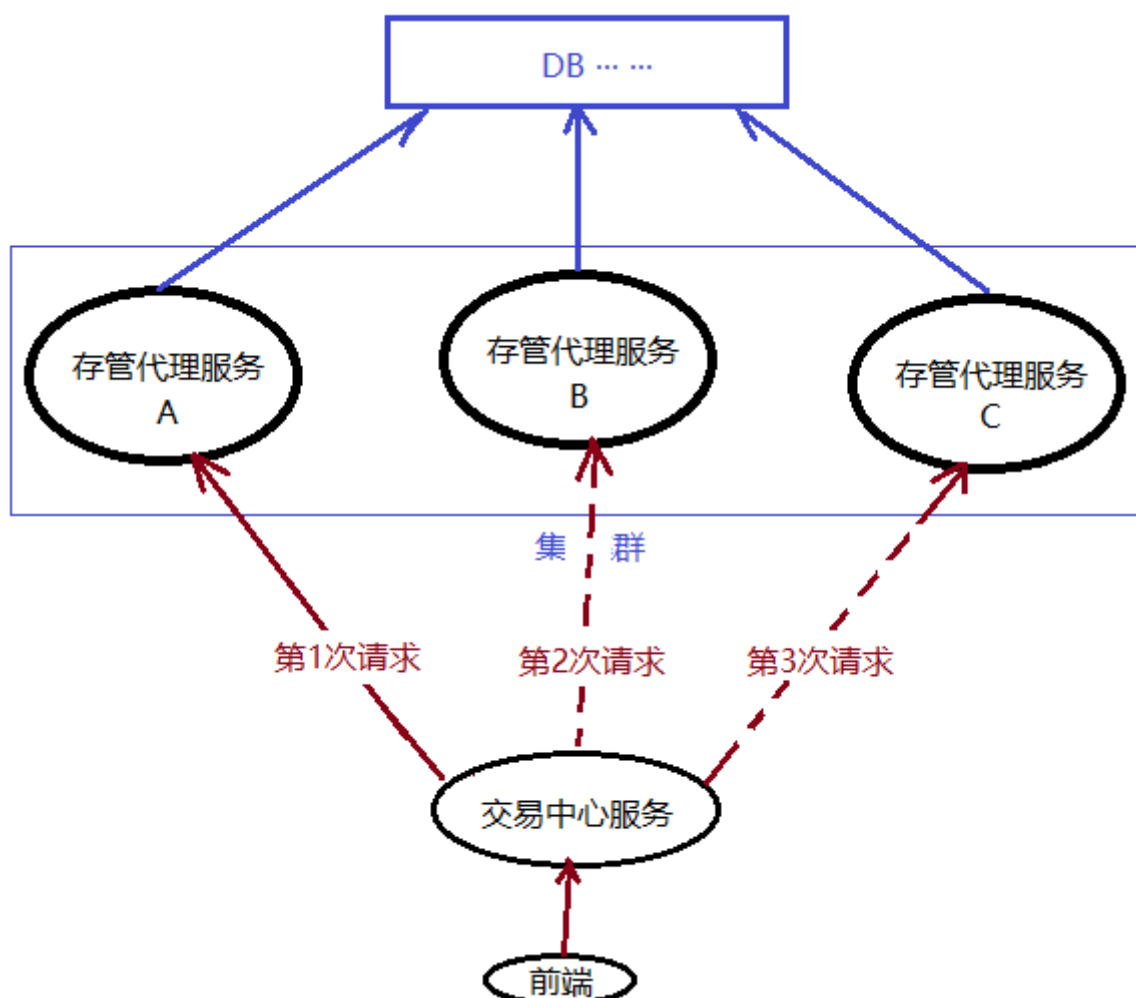
```
@Override
@ApiOperation(value = "向银行存管系统发送标的信息")
@ApiImplicitParam(name = "projectDTO", value = "向银行存管系统发送标的信息",
    required = true, dataType = "ProjectDTO", paramType = "body")
@PostMapping("/1/create-project")
public RestResponse<String> createProject(@RequestBody ProjectDTO projectDTO) {
    DepositoryResponseDTO<DepositoryBaseResponse> depositoryResponse =
        depositoryRecordService.createProject(projectDTO);
    RestResponse<String> restResponse=new RestResponse<String>();
    restResponse.setResult(depositoryResponse.getRespData().getRespCode());
    restResponse.setMsg(depositoryResponse.getRespData().getRespMsg());
    return restResponse;
}
```

3.5.2.3 前后端集成测试

3.6 分布式事务幂等性

3.6.1 问题分析

在审核标的业务中，交易中心服务会向存管代理服务发起远程请求，存管代理服务会向银行存管系统发起远程请求，在整个过程中，业务处理可能会比较耗时。另外，在生产环境里，我们的微服务很有可能会以集群形式进行部署，此时会出现一个问题需要解决，我们结合下图一起分析。



假设：前端请求交易中心服务审核标的，交易中心服务向存管代理服务A发起远程请求，存管代理服务A在向数据库保存完交易记录后出现网络问题，此时，交易中心服务一直等待，前端也收不到响应信息，那么客户在等待一段时间后，很有可能会在前端再次点击按钮请求交易中心服务审核标的，交易中心服务向存管代理服务B发起远程请求，存管代理服务B向数据库保存交易记录... ..，最终导致数据库中存在重复数据。

还有一个问题也是需要解决的，就是客户通过前端在短时间(例如5秒)内多次点击按钮重复发送请求，此时，我们应该只允许其中一个请求拥有执行权。

怎么解决上述问题呢？需要实现幂等性。幂等性指的是：一个业务操作，不管重复执行多少次，最终产生的效果或返回的结果都和执行一次是一样的。我们接下来就需要编码实现审核标的业务中的幂等性。

3.6.2 功能实现

当存管代理服务被交易中心远程请求时，首先根据该请求的requestNo获取交易记录，然后分三种情况进行处理：

1. 若交易记录不存在则新增交易记录，requestNo为唯一索引，新保存的数据的状态为“未同步”。
2. 若交易记录存在并且数据状态为“未同步”，此阶段存在并发可能，利用redis原子性自增，来争夺请求执行权，若count大于1，说明已有线程在执行该操作，直接返回“正在处理”。
3. 若交易记录存在并且数据状态为已同步，直接返回处理结果。

在DepositRecordServiceImpl类中单独定义一个方法实现幂等性：

```
/**
 * 实现幂等性
 * @param depositRecord
 * @return
 */
private DepositRecordResponseDTO<DepositRecordBaseResponse>
handleIdempotent(DepositRecord depositRecord) {
    //根据requestNo查询交易记录
    String requestNo = depositRecord.getRequestNo();
    DepositRecordDTO depositRecordDTO = getByRequestNo(requestNo);

    // 1. 交易记录不存在则新增交易记录
    if (depositRecordDTO == null) {
        //保存交易记录
        saveDepositRecord(depositRecord);
        return null;
    }

    // 2. 交易记录存在并且数据状态为未同步，则利用redis原子性自增，来争夺请求执行权
    if (StatusCode.STATUS_OUT.getCode() ==
        depositRecordDTO.getRequestStatus()) {
        //如果requestNo不存在则返回1,如果已经存在,则会返回 (requestNo已存在个数+1)
        Long count = cache.incrBy(requestNo, 1L);

        if (count == 1) {
            cache.expire(requestNo, 5); //设置requestNo有效期5秒
            return null;
        }

        // 若count大于1,说明已有线程在执行该操作,直接返回“正在处理”
        if (count > 1) {
            throw new BusinessException(DepositRecordErrorCode.E_160103);
        }
    }
}
```



```

    }

    // 3. 交易记录存在并且数据状态为已同步，直接返回处理结果
    return JSON.parseObject(depositaryRecordDTO.getResponseData(),
        new TypeReference<DepositaryResponseDTO<DepositaryBaseResponse>>() {
        });

}

private DepositaryRecordDTO getByRequestNo(String requestNo) {
    DepositaryRecord depositaryRecord = getEntityByRequestNo(requestNo);
    if (depositaryRecord == null) {
        return null;
    }
    DepositaryRecordDTO depositaryRecordDTO = new DepositaryRecordDTO();
    BeanUtils.copyProperties(depositaryRecord, depositaryRecordDTO);
    return depositaryRecordDTO;
}

private DepositaryRecord getEntityByRequestNo(String requestNo) {
    return getOne(new QueryWrapper<DepositaryRecord>().lambda()
        .eq(DepositaryRecord::getRequestNo, requestNo));
}

```

修改DepositaryRecordServiceImpl类中createProject方法的代码：

```

DepositaryRecord depositaryRecord = new DepositaryRecord(
    projectDTO.getRequestNo(),
    DepositaryRequestTypeCode.CREATE.getCode(),
    "Project",
    projectDTO.getId());

// 1. 幂等性实现
DepositaryResponseDTO<DepositaryBaseResponse> responseDTO =
    handleIdempotent(depositaryRecord);
if (responseDTO != null) {
    return responseDTO;
}

// 根据requestNo获取交易记录
depositaryRecord = getEntityByRequestNo(projectDTO.getRequestNo());

//2. 签名数据
// ProjectDTO 转换为 ProjectRequestDataDTO
... ..

```

3.6.3 功能测试

在之前的代码中，requestNo每次在请求时都会新生成一个值，因此无法满足实现幂等性的需求。在进行功能测试前，需要做出一些修改，保证只有第一次请求时才为requestNo生成新的值。

1. 给标的信息表新增一个字段：REQUEST_NO


```
ALTER TABLE `p2p_transaction_0`.`project_0`  
ADD COLUMN `REQUEST_NO` varchar(50) NULL COMMENT '发标请求流水号' AFTER  
`IS_ASSIGNMENT`;  
  
ALTER TABLE `p2p_transaction_0`.`project_1`  
ADD COLUMN `REQUEST_NO` varchar(50) NULL COMMENT '发标请求流水号' AFTER  
`IS_ASSIGNMENT`;  
  
ALTER TABLE `p2p_transaction_1`.`project_0`  
ADD COLUMN `REQUEST_NO` varchar(50) NULL COMMENT '发标请求流水号' AFTER  
`IS_ASSIGNMENT`;  
  
ALTER TABLE `p2p_transaction_1`.`project_1`  
ADD COLUMN `REQUEST_NO` varchar(50) NULL COMMENT '发标请求流水号' AFTER  
`IS_ASSIGNMENT`;
```

2. 在Project类中新增对应的属性

```
/**  
 * 发标请求流水号  
 */  
@TableField("REQUEST_NO")  
private String requestNo;
```

3. 在ProjectDTO类中新增对应的属性

```
//发标请求流水号  
private String requestNo;
```

4. 在ProjectServiceImpl类的projectsApprovalStatus方法中修改部分逻辑

```
@Override  
public String projectsApprovalStatus(Long id, String approveStatus) {  
    //1.根据id查询标的信息并转换为DTO对象  
    Project project= getById(id);  
    ProjectDTO projectDTO=convertProjectEntityToDTO(project);  
    //2.生成流水号(不存在才生成)  
    if(StringUtils.isBlank(project.getRequestNo())){  
        projectDTO.setRequestNo(CodeNoUtil.getNo(CodePrefixCode  
            .CODE_REQUEST_PREFIX));  
        update(Wrappers.<Project>lambdaUpdate().set(Project::getRequestNo,  
            projectDTO.getRequestNo()).eq(Project::getId,id));  
    }  
    ... ..  
    ... ..  
}
```

另外，针对幂等性问题中的第3种情况，我们也需要做一些修改，在银行存管系统返回处理结果时，需要把该结果保存到数据库中，这样才能保证第3种情况能拿到并返回处理结果。

1. 在depository_record表中新增一个字段RESPONSE_DATA，用来保存银行存管系统返回的处理结果

```
ALTER TABLE `p2p_depository_agent`.`depository_record`  
ADD COLUMN `RESPONSE_DATA` text NULL COMMENT '处理结果' AFTER `CONFIRM_DATE`;
```

2. 在DepositoryRecord类中新增对应的属性

```
@TableField("RESPONSE_DATA")  
private String responseData;
```

3. 在DepositoryRecordDTO类中新增对应的属性

```
private String responseData;
```

4. 在DepositoryRecordServiceImpl类的sendHttpGet方法中新增一行代码

```
DepositoryRecordServiceImpl.java ×  
  
// 发送请求，获取结果，如果检验签名失败，拦截器会在结果中放入："signature", "false"  
String responseBody = okHttpClient.  
    .doSyncGet(url + "?serviceName=" + serviceName + "&platformNo=" +  
        platformNo + "&reqData=" + reqData);  
DepositoryResponseDTO<DepositoryBaseResponse> depositoryResponse = JSON  
    .parseObject(responseBody,  
        new TypeReference<DepositoryResponseDTO<DepositoryBaseResponse>>() {});  
  
depositoryRecord.setResponseData(responseBody);  
  
// 响应后，根据结果更新数据库（进行签名判断）  
// 判断签名(signature)是为 false，如果是说明验签失败！  
if (!"false".equals(depositoryResponse.getSignature())) {  
    // 成功 - 设置数据同步状态  
    depositoryRecord.setRequestStatus(StatusCode.STATUS_IN.getCode());  
}
```

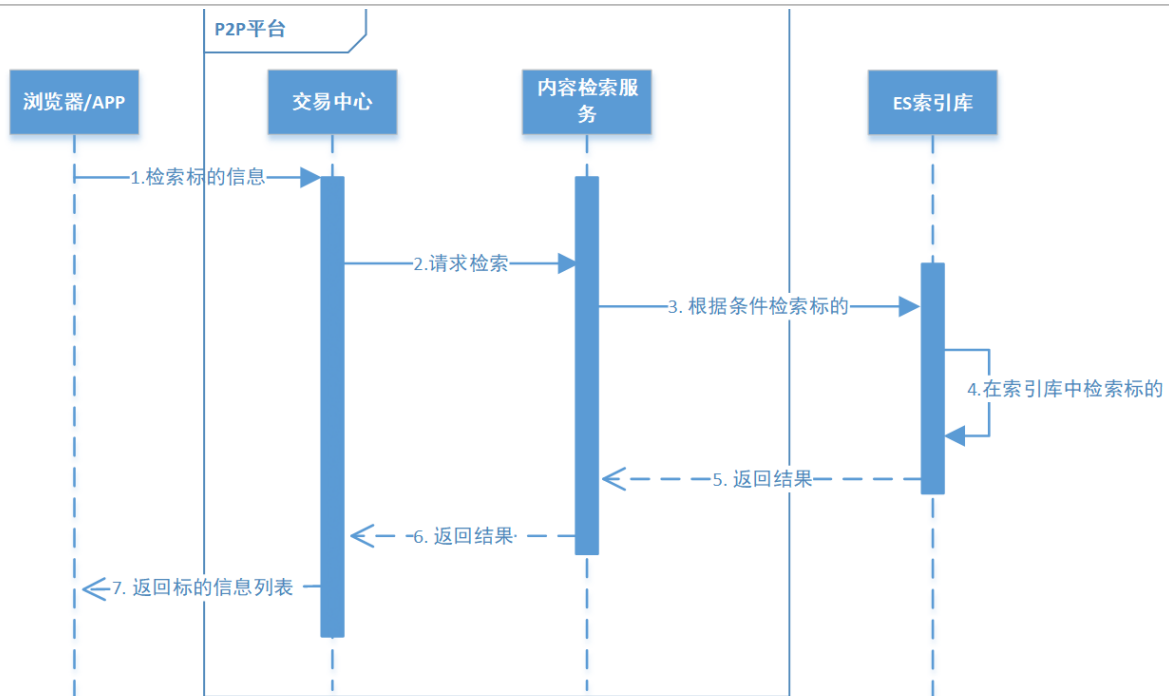
注意：为了看到测试效果，需要至少在本机启动三个存管代理服务(端口号为53070、53071、53072)用来模拟集群环境，另外还需要启动redis（密码默认foobared123）。

4 标的快速检索

4.1 需求分析

为了提高数据库性能，我们前面对交易中心数据库p2p_transaction进行了分库分表、读写分离，但是即便如此，当我们进行某些查询时，仍然面临性能瓶颈。例如：我要进行模糊查询，我要进行多字段过滤，在数据量很大的情况下，类似这种查询仍然会比较耗时。在这种情况下，我们考虑使用全文检索技术来进一步优化性能。

由于P2P项目中会有多处业务数据有这种需求，所以我们单独创建一个微服务工程(下图中的内容检索服务)为整个项目提供全文检索服务，该工程主要采用ElasticSearch+Logstash技术实现。具体交互流程如下所示：

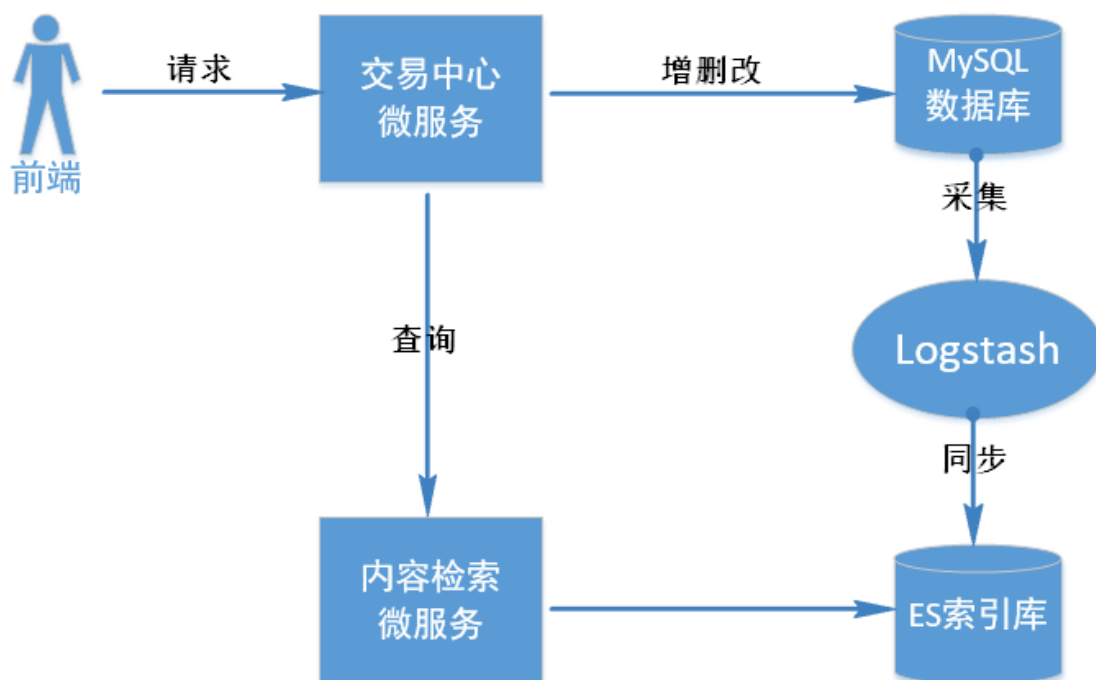


1. 前端请求交易中心检索标的
2. 交易中心接到请求后，请求内容检索服务进行查询
3. 内容检索服务请求ES索引库查询标的信息
4. 返回查询结果

4.2 技术方案

由于标的信息是存储在MySQL数据库中的，而全文检索是从ES索引库获取的数据。因此，我们需要解决MySQL数据库和ES索引库之间的数据同步问题，即：MySQL数据库中的标的信息发生变化时，ES索引库中的数据也随之变化。

Logstash作为Elasticsearch常用的实时数据采集引擎，可以采集来自不同数据源的数据。我们通过Logstash就可以解决MySQL数据库和ES索引库之间的数据同步问题，具体解决方案如下图所示：



4.3 搭建ES环境

4.3.1 安装和配置

ElasticSearch、IK分词器以及head插件的安装和配置由大家自行完成，这里暂时采用单机环境，在企业实际部署时，视情况可以采用集群。elasticsearch.yml配置如下：

```
cluster.name: wanxinp2p
node.name: wanxinp2p_node
network.host: 0.0.0.0
http.port: 9200
transport.tcp.port: 9300
discovery.zen.minimum_master_nodes: 1
bootstrap.memory_lock: true
discovery.type: single-node
http.cors.enabled: true
http.cors.allow-origin: "*"
path.data: C:\Users\apple\Desktop\elasticsearch-7.3.1\data
path.logs: C:\Users\apple\Desktop\elasticsearch-7.3.1\logs
```

4.3.2 创建索引库和映射

1. 启动本机ES和head插件，通过head插件快速创建wanxinp2p_project索引库（3个分片，0个副本）。
2. 通过Postman创建映射：http://localhost:9200/wanxinp2p_project/_mapping（Post）

```
{
  "properties":{
    "isassignment":{
      "type":"keyword"
    },
    "amount":{
      "type":"double"
    },
    "period":{
      "type":"integer"
    },
    "repaymentway":{
      "type":"keyword"
    },
    "consumerid":{
      "type":"long"
    },
    "userno":{
      "type":"keyword"
    },
    "description":{
      "analyzer":"ik_max_word",
      "type":"text"
    },
    "annualrate":{
      "type":"double"
    },
  },
}
```

```
"type":{
  "type":"keyword"
},
"borrowerannualrate":{
  "type":"double"
},
"projectstatus":{
  "type":"keyword"
},
"projectno":{
  "type":"keyword"
},
"commissionannualrate":{
  "type":"keyword"
},
"name":{
  "analyzer":"ik_max_word",
  "type":"text"
},
"id":{
  "type":"long"
},
"createdate":{
  "type":"date"
},
"modifydate":{
  "type":"date"
},
"status":{
  "type":"keyword"
}
}
```

4.4 Logstash数据采集和同步

Logstash作为Elasticsearch常用的实时数据采集引擎，可以采集来自不同数据源的数据，并输出到ES索引库中创建索引。在这里，我们使用Logstash来保证MySQL数据库和ES索引库的标的数据同步。

4.4.1 下载和安装Logstash

1. 从官网<https://www.elastic.co/cn/downloads/logstash>下载Logstash 7.3.1版本（ZIP），和本项目使用的Elasticsearch 7.3.1版本一致。

Download Logstash

Want to upgrade? We'll give you a hand. [Migration Guide »](#)

Version: 7.3.1

Release date: August 23, 2019

License: Elastic License

Downloads: [TAR.GZ](#) [sha asc](#)

[DEB](#) [sha asc](#)

[ZIP](#) [sha asc](#)

[RPM](#) [sha asc](#)

2. 解压到任意路径

3. 安装logstash-input-jdbc插件，Logstash通过该插件实现从数据库采集数据。由于该插件是由ruby脚本语言开发的，所以需要先大家自行下载ruby并安装，下载地址: <https://rubyinstaller.org/downloads/>，下载2.6版本即可。安装完成后在命令行窗口执行ruby -v命令检查是否安装成

```
命令提示符
Microsoft Windows [版本 10.0.17134.885]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\apple>ruby -v
ruby 2.6.3p62 (2019-04-16 revision 67580) [x64-mingw32]
```

安装完ruby后，就可以安装logstash-input-jdbc插件。在命令行窗口中切换到Logstash的bin目录下，执行如下命令即可：

```
.\logstash-plugin.bat install logstash-input-jdbc
```

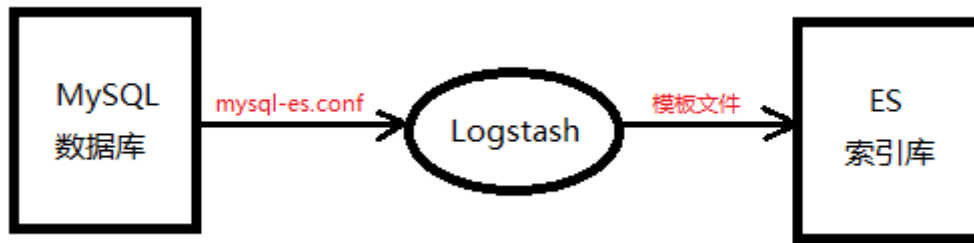
```
PS D:\Elastic\logstash-6.2.1\bin> .\logstash-plugin.bat install logstash-input-jdbc
Validating logstash-input-jdbc
Installing logstash-input-jdbc
Installation successful
```

大家也可以直接使用课件中提供的logstash-7.3.1，此logstash中已安装了logstash-input-jdbc插件。

4.4.2 使用Logstash

Logstash的工作是从MySQL中读取数据，然后向ES中写入数据从而创建索引。因此它需要知道数据库相关的信息和ES相关的信息，我们需要提供两个配置文件：

1. mysql-es.conf: 配置从哪个数据库和表读取数据，配置把读取到的数据写入到哪个ES索引库中
2. 模板文件: 索引库的映射配置



4.4.2.1 创建模板文件

在logstash的config目录中创建wanxinp2p_project_template.json作为模板文件，配置索引库映射，内容如下：

```
{
  "mappings": {
    "properties": {
      "isassignment": {
        "type": "keyword"
      },
      "amount": {
        "type": "double"
      },
      "period": {
        "type": "integer"
      },
      "repaymentway": {
        "type": "keyword"
      },
      "consumerid": {
        "type": "long"
      },
      "userno": {
        "type": "keyword"
      },
      "description": {
        "analyzer": "ik_max_word",
        "type": "text"
      },
      "annualrate": {
        "type": "double"
      },
      "type": {
        "type": "keyword"
      },
      "borrowerannualrate": {
        "type": "double"
      },
      "projectstatus": {
        "type": "keyword"
      },
      "projectno": {
        "type": "keyword"
      },
      "commissionannualrate": {
        "type": "keyword"
      }
    }
  }
}
```

```
    },
    "name": {
      "analyzer": "ik_max_word",
      "type": "text"
    },
    "id": {
      "type": "long"
    },
    "createdate": {
      "type": "date"
    },
    "modifydate": {
      "type": "date"
    },
    "status": {
      "type": "keyword"
    }
  }
},
"template": "wanxinp2p_project"
}
```

4.4.2.2 配置mysql-es.conf

1. 我们需要执行如下sql语句，在标的信息表中增加一个字段：

```
ALTER TABLE `p2p_transaction_0`.`project_0` ADD COLUMN `MODIFY_DATE` datetime(0)
NULL DEFAULT NULL COMMENT '修改时间' AFTER `CREATE_DATE`;

ALTER TABLE `p2p_transaction_0`.`project_1` ADD COLUMN `MODIFY_DATE` datetime(0)
NULL DEFAULT NULL COMMENT '修改时间' AFTER `CREATE_DATE`;

ALTER TABLE `p2p_transaction_1`.`project_0` ADD COLUMN `MODIFY_DATE` datetime(0)
NULL DEFAULT NULL COMMENT '修改时间' AFTER `CREATE_DATE`;

ALTER TABLE `p2p_transaction_1`.`project_1` ADD COLUMN `MODIFY_DATE` datetime(0)
NULL DEFAULT NULL COMMENT '修改时间' AFTER `CREATE_DATE`;
```

2. 在Logstash的config目录下创建mysql-es.conf配置文件，Logstash会根据该配置文件从MySQL中读取数据并同步到ES库中。

```
input {
  jdbc {
    jdbc_connection_string => "jdbc:mysql://localhost:3306/p2p_transaction_0?
useUnicode=true&characterEncoding=utf-8&useSSL=false"
    jdbc_user => "root"
    jdbc_password => "123"
    jdbc_driver_library => "e:/mysql-connector-java-8.0.11.jar"
    jdbc_driver_class => "com.mysql.cj.jdbc.Driver"
    jdbc_paging_enabled => true
    jdbc_page_size => "50000"
    #时区设置
    jdbc_default_timezone => "Asia/Shanghai"
    #要执行的sql
    statement_filepath => "e:/wanxinp2p-project_0.sql"
    #每隔10秒执行一次
```



```
schedule => "*/10 * * * * *"  
#是否记录最后一次的运行时间  
record_last_run => true  
#记录最后一次运行时间的位置  
last_run_metadata_path => "./logstash_metadata"  
}  
  
jdbc {  
  jdbc_connection_string => "jdbc:mysql://localhost:3306/p2p_transaction_0?  
useUnicode=true&characterEncoding=utf-8&useSSL=false"  
  jdbc_user => "root"  
  jdbc_password => "123"  
  jdbc_driver_library => "e:/mysql-connector-java-8.0.11.jar"  
  jdbc_driver_class => "com.mysql.cj.jdbc.Driver"  
  jdbc_paging_enabled => true  
  jdbc_page_size => "50000"  
  jdbc_default_timezone => "Asia/Shanghai"  
  
  statement_filepath => "e:/wanxinp2p-project_1.sql"  
  schedule => "*/10 * * * * *"  
  record_last_run => true  
  last_run_metadata_path => "./logstash_metadata"  
}  
  
jdbc {  
  jdbc_connection_string => "jdbc:mysql://localhost:3306/p2p_transaction_1?  
useUnicode=true&characterEncoding=utf-8&useSSL=false"  
  jdbc_user => "root"  
  jdbc_password => "123"  
  jdbc_driver_library => "e:/mysql-connector-java-8.0.11.jar"  
  jdbc_driver_class => "com.mysql.cj.jdbc.Driver"  
  jdbc_paging_enabled => true  
  jdbc_page_size => "50000"  
  jdbc_default_timezone => "Asia/Shanghai"  
  
  statement_filepath => "e:/wanxinp2p-project_0.sql"  
  schedule => "*/10 * * * * *"  
  record_last_run => true  
  last_run_metadata_path => "./logstash_metadata"  
}  
  
jdbc {  
  jdbc_connection_string => "jdbc:mysql://localhost:3306/p2p_transaction_1?  
useUnicode=true&characterEncoding=utf-8&useSSL=false"  
  jdbc_user => "root"  
  jdbc_password => "123"  
  jdbc_driver_library => "e:/mysql-connector-java-8.0.11.jar"  
  jdbc_driver_class => "com.mysql.cj.jdbc.Driver"  
  jdbc_paging_enabled => true  
  jdbc_page_size => "50000"  
  jdbc_default_timezone => "Asia/Shanghai"  
  
  statement_filepath => "e:/wanxinp2p-project_1.sql"  
  schedule => "*/10 * * * * *"  
  record_last_run => true  
  last_run_metadata_path => "./logstash_metadata"  
}  
}
```

```
output {
  elasticsearch {
    #ES服务器地址
    hosts => "localhost:9200"
    #ES索引库名字
    index => "wanxinp2p_project"
    #取表中主键值作为文档ID
    document_id => "%{id}"
    #模板文件地址
    template => "C:/Users/apple/Desktop/logstash-
7.3.1/config/wanxinp2p_project_template.json"
    #模板文件名字
    template_name => "wanxinp2p_project"
    #覆盖默认模板
    template_overwrite => true
  }
  #日志输出
  stdout {
    codec => json_lines
  }
}
```

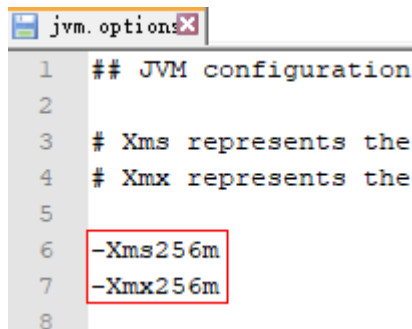
上述配置文件中涉及到的jar文件和sql文件都在课件中。Logstash会根据上述配置自动执行数据采集和同步工作，每次执行完成会在config的logstash_metadata文件中记录执行时间，下次以此时间为基准进行增量同步，该时间在sql语句中可以通过:sql_last_value获取。

4.4.3 测试

1. 找到课件中的"标的信息表_测试数据.sql"并执行
2. 启动ES和head插件，然后在Logstash的bin目录中执行如下命令启动Logstash

```
.\logstash.bat -f ..\config\mysql-es.conf
```

3. 如果遇到内存溢出的报错，请修改config目录下的jvm.options文件，把1g的默认内存改小：

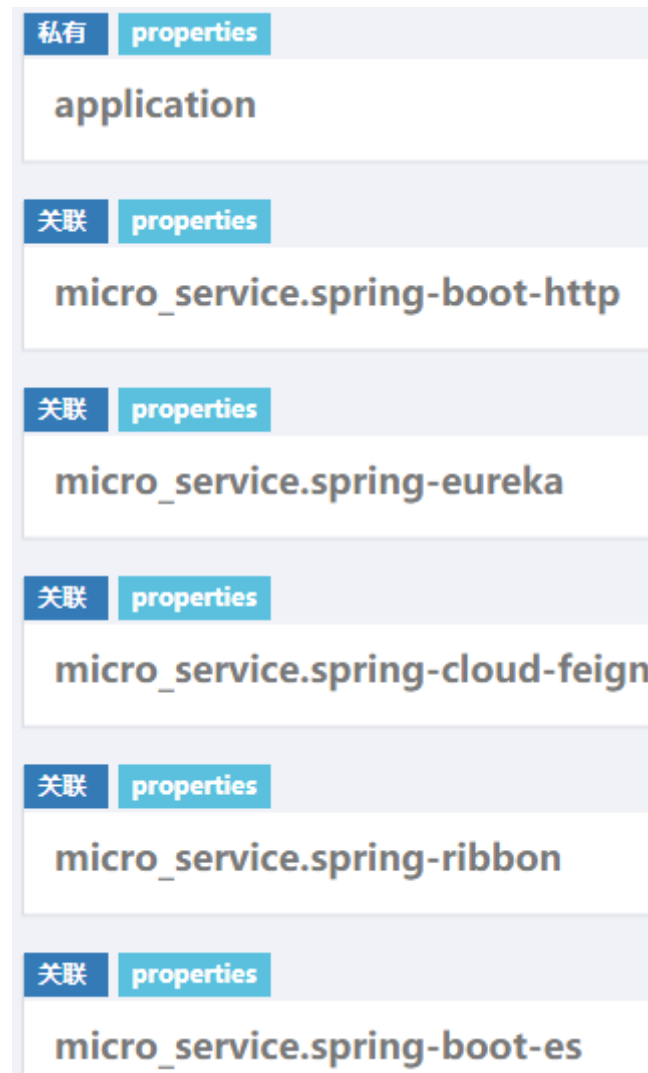


```
jvm.options
1  ## JVM configuration
2
3  # Xms represents the
4  # Xmx represents the
5
6  -Xms256m
7  -Xmx256m
8
```

4. 修改标的信息表中的任意数据，并且修改MODIFY_DATE为当前时间
 - A. 观察ES索引库是否发生数据同步
 - B. 观察Logstash控制台

4.5 搭建开发环境

1. 在IDEA中导入内容检索微服务工程（wanxinp2p-content-search-service），并熟悉里面的初始代码
2. 在Apollo上新建content-search-service项目用来存储内容检索微服务工程的相关配置



application

表格	文本	更改历史	实例列表 0
1	swagger.enable = true		
2	spring.mvc.throw-exception-if-no-handler-found = true		

micro_service.spring-boot-http

表格	文本	更改历史	实例列表 0
1	server.servlet.context-path = /content-search		

micro_service.spring-boot-es

表格 文本 更改历史 实例列表 ①

```
1 wanxinp2p.es.host = 127.0.0.1:9200
2 wanxinp2p.es.index = wanxinp2p_project
```

3. 启动微服务工程进行环境测试

(启动参数: -Denv=dev -Dapollo.cluster=DEFAULT -Dserver.port=53090)

4.6 标的快速检索

4.6.1 接口定义

4.6.1.1 交易中心标的查询接口

1. 在TransactionApi接口中新增查询方法:

```
/**
 * 标的信息快速检索
 * @param projectQueryDTO
 * @param pageNo
 * @param pageSize
 * @param sortBy
 * @param order
 * @return
 */
RestResponse<PageVO<ProjectDTO>> queryProjects(ProjectQueryDTO projectQueryDTO,
Integer pageNo, Integer pageSize, String sortBy, String order);
```

2. 在TransactionController类中实现该方法:

```
@Override
@ApiOperation("从ES检索标的信息")
@ApiImplicitParams({
    @ApiImplicitParam(name = "projectQueryDTO", value = "标的信息条件对象",
        required = true, dataType = "ProjectQueryDTO", paramType = "body"),
    @ApiImplicitParam(name = "order", value = "顺序", required = false,
        dataType = "string", paramType = "query"),
    @ApiImplicitParam(name = "pageNo", value = "页码", required = true,
        dataType = "int", paramType = "query"),
    @ApiImplicitParam(name = "pageSize", value = "每页记录数", required =
true,
        dataType = "int", paramType = "query"),
    @ApiImplicitParam(name = "sortBy", value = "排序字段", required = false,
        dataType = "string", paramType = "query")})
@PostMapping("/projects/indexes/q")
public RestResponse<PageVO<ProjectDTO>> queryProjects(@RequestBody
ProjectQueryDTO projectQueryDTO, Integer pageNo, Integer pageSize, String
sortBy, String order){
    return null;
}
```

4.6.1.2 检索服务标的查询接口

1. 在wanxinp2p-api工程中新建search包，在该包中新建一个ContentSearchApi接口，并定义一个查询方法：

```
/**
 * <P>
 * 内容检索服务API
 * </p>
 */
public interface ContentSearchApi {
    /**
     * 检索标的
     * @param projectQueryParamsDTO
     * @return
     */
    RestResponse<PageVO<ProjectDTO>> queryProjectIndex(
        ProjectQueryParamsDTO projectQueryParamsDTO,
        Integer pageNo, Integer pageSize, String sortBy, String order);
}
```

2. 在wanxinp2p-content-search-service工程中新建ContentSearchController类：

```
@Slf4j
@RestController
@Api(value = "检索服务", tags = "ContentSearch", description = "检索服务API")
public class ContentSearchController {
    @ApiOperation("检索标的")
    @ApiImplicitParams({
        @ApiImplicitParam(name = "projectQueryParamsDTO",
            value = "标的检索参数", required = true,
            dataType = "ProjectQueryParamsDTO", paramType = "body"),
        @ApiImplicitParam(name = "pageNo", value = "页码", required = true,
            dataType = "int", paramType = "query"),
        @ApiImplicitParam(name = "pageSize", value = "每页记录数",
            required = true, dataType = "int", paramType = "query"),
        @ApiImplicitParam(name = "sortBy", value = "排序字段",
            dataType = "String", paramType = "query"),
        @ApiImplicitParam(name = "order", value = "顺序", dataType =
            "String",
            paramType = "query")})
    @PostMapping(value = "/1/projects/indexes/q")
    public RestResponse<PageVO<ProjectDTO>> queryProjectIndex(
        @RequestBody ProjectQueryParamsDTO projectQueryParamsDTO,
        @RequestParam Integer pageNo,
        @RequestParam Integer pageSize,
        @RequestParam(required = false) String sortBy,
        @RequestParam(required = false) String order){
        return null;
    }
}
```

4.6.2 功能实现

4.6.2.1 交易中心标的查询功能

1. 定义Feign代理

```
@FeignClient(value = "content-search-service")
public interface ContentSearchApiAgent {
    @PostMapping(value = "/content-search/1/projects/indexes/q")
    RestResponse<PageVO<ProjectDTO>> queryProjectIndex(
        @RequestBody ProjectQueryDTO projectQueryParamsDTO,
        @RequestParam Integer pageNo,
        @RequestParam Integer pageSize,
        @RequestParam(required = false) String sortBy,
        @RequestParam(required = false) String order);
}
```

2. 在ProjectService接口中新增查询方法:

```
PageVO<ProjectDTO> queryProjects(ProjectQueryDTO projectQueryDTO, String order,
Integer pageNo, Integer pageSize, String sortBy);
```

3. 在ProjectServiceImpl类中实现该方法:

```
@Autowired
private ContentSearchApiAgent contentSearchApiAgent;

@Override
public PageVO<ProjectDTO> queryProjects(ProjectQueryDTO projectQueryDTO,
String order, Integer pageNo, Integer pageSize, String sortBy) {
    RestResponse<PageVO<ProjectDTO>> esResponse =
contentSearchApiAgent.queryProjectIndex(projectQueryDTO,
pageNo, pageSize, sortBy,
order);
    if (!esResponse.isSuccessful()) {
        throw new BusinessException(CommonErrorCode.UNKOWN);
    }
    return esResponse.getResult();
}
```

4. 完善TransactionController中的代码:

```
@PostMapping("/projects/indexes/q")
public RestResponse<PageVO<ProjectDTO>> queryProjects(@RequestBody
ProjectQueryDTO projectQueryDTO, String order, Integer pageNo, Integer pageSize,
String sortBy) {
    PageVO<ProjectDTO> projects = projectService.queryProjects(projectQueryDTO,
order, pageNo, pageSize, sortBy);
    return RestResponse.success(projects);
}
```

4.6.2.2 检索服务标的查询功能

1. 新建一个业务层接口ProjectIndexService:

```
/**
 * 标的检索业务层接口
 */
public interface ProjectIndexService {
    PageVO<ProjectDTO> queryProjectIndex(ProjectQueryParamsDTO
projectQueryParamsDTO,
                                         Integer pageNo,Integer pageSize,
                                         String sortBy, String order);
}
```

2. 创建ProjectIndexServiceImpl实现类，通过Elasticsearch的API实现标的检索(条件+分页+排序):

```
@Service
public class ProjectIndexServiceImpl implements ProjectIndexService {

    @Autowired
    private RestHighLevelClient restHighLevelClient;

    @Value("${wanxinp2p.es.index}")
    private String projectIndex;

    public PageVO<ProjectDTO> queryProjectIndex(ProjectQueryParamsDTO
projectQueryParamsDTO, Integer pageNo,Integer pageSize,
                                         String sortBy, String order) {

        //1.创建搜索请求对象
        SearchRequest searchRequest = new SearchRequest(projectIndex);

        //2.搜索条件
        //2.1.创建条件查询对象
        BoolQueryBuilder queryBuilder = QueryBuilders.boolQuery();
        //2.2.非空判断并封装条件
        if (StringUtils.isNotBlank(projectQueryParamsDTO.getName())) {
            queryBuilder.must(QueryBuilders.termQuery("name",
projectQueryParamsDTO.getName()));
        }
        if(projectQueryParamsDTO.getStartPeriod() != null){
            queryBuilder.must(QueryBuilders.rangeQuery("period")
                .gte(projectQueryParamsDTO.getStartPeriod()));
        }
        if(projectQueryParamsDTO.getEndPeriod() != null){
            queryBuilder.must(QueryBuilders.rangeQuery("period")
                .lte(projectQueryParamsDTO.getEndPeriod()));
        }
        //3.创建searchSourceBuilder对象
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //3.1.封装条件查询对象
        searchSourceBuilder.query(queryBuilder);
        //3.2.设置排序信息
        if(StringUtils.isNotBlank(sortBy)&&StringUtils.isNotBlank(order)){
            if(order.toLowerCase().equals("asc")){
                searchSourceBuilder.sort(sortBy, SortOrder.ASC);
            }
            if(order.toLowerCase().equals("desc")){
                searchSourceBuilder.sort(sortBy, SortOrder.DESC);
            }
        }
    }
}
```

```
        else{
            searchSourceBuilder.sort("createdate",SortOrder.DESC);
        }
        //3.3.设置分页信息
        searchSourceBuilder.from((pageNo-1)*pageSize);
        searchSourceBuilder.size(pageSize);

        //4.完成封装
        searchRequest.source(searchSourceBuilder);

        List<ProjectDTO> list = new ArrayList<>();
        PageVO<ProjectDTO> pageVO=new PageVO<>();
        try {
            //5.执行搜索
            SearchResponse searchResponse = restHighLevelClient.search(
                searchRequest, RequestOptions.DEFAULT);
            //6.获取响应结果
            SearchHits hits = searchResponse.getHits();
            long totalHits = hits.getTotalHits().value;//匹配的总记录数
            pageVO.setTotal(totalHits);
            SearchHit[] searchHits = hits.getHits();//获取匹配数据
            //7.循环封装DTO
            for(SearchHit hit:searchHits){
                ProjectDTO projectDTO = new ProjectDTO();
                Map<String, Object> sourceAsMap = hit.getSourceAsMap();
                Double amount = (Double) sourceAsMap.get("amount");
                String projectstatus = (String)
sourceAsMap.get("projectstatus");
                Integer period =
Integer.parseInt(sourceAsMap.get("period").toString());
                String name = (String) sourceAsMap.get("name");
                String description = (String) sourceAsMap.get("description");

                projectDTO.setAmount(new BigDecimal(amount));
                projectDTO.setProjectStatus(projectstatus);
                projectDTO.setPeriod(period);
                projectDTO.setName(name);
                projectDTO.setDescription(description);

                list.add(projectDTO);
            }
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        //8.封装为PageVO对象并返回
        pageVO.setContent(list);
        pageVO.setPageNo(pageNo);
        pageVO.setPageSize(pageSize);
        return pageVO;
    }
}
```

2. 完善ContentSearchController中的代码:

```
@Autowired
private ProjectIndexService projectIndexService;
```



```
@PostMapping(value = "/1/projects/indexes/q")
public RestResponse<PageVO<ProjectDTO>> queryProjectIndex(
    @RequestBody ProjectQueryParamsDTO projectQueryParamsDTO,
    @RequestParam Integer pageNo,
    @RequestParam Integer pageSize,
    @RequestParam(required = false) String sortBy,
    @RequestParam(required = false) String order) {
    PageVO<ProjectDTO> projects = projectIndexService
        .queryProjectIndex(projectQueryParamsDTO, pageNo, pageSize, sortBy,
            order);
    return RestResponse.success(projects);
}
```

4.6.3 功能测试