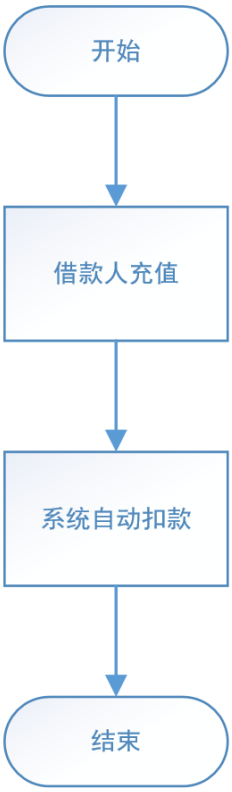
万信金融 第5章-用户还款

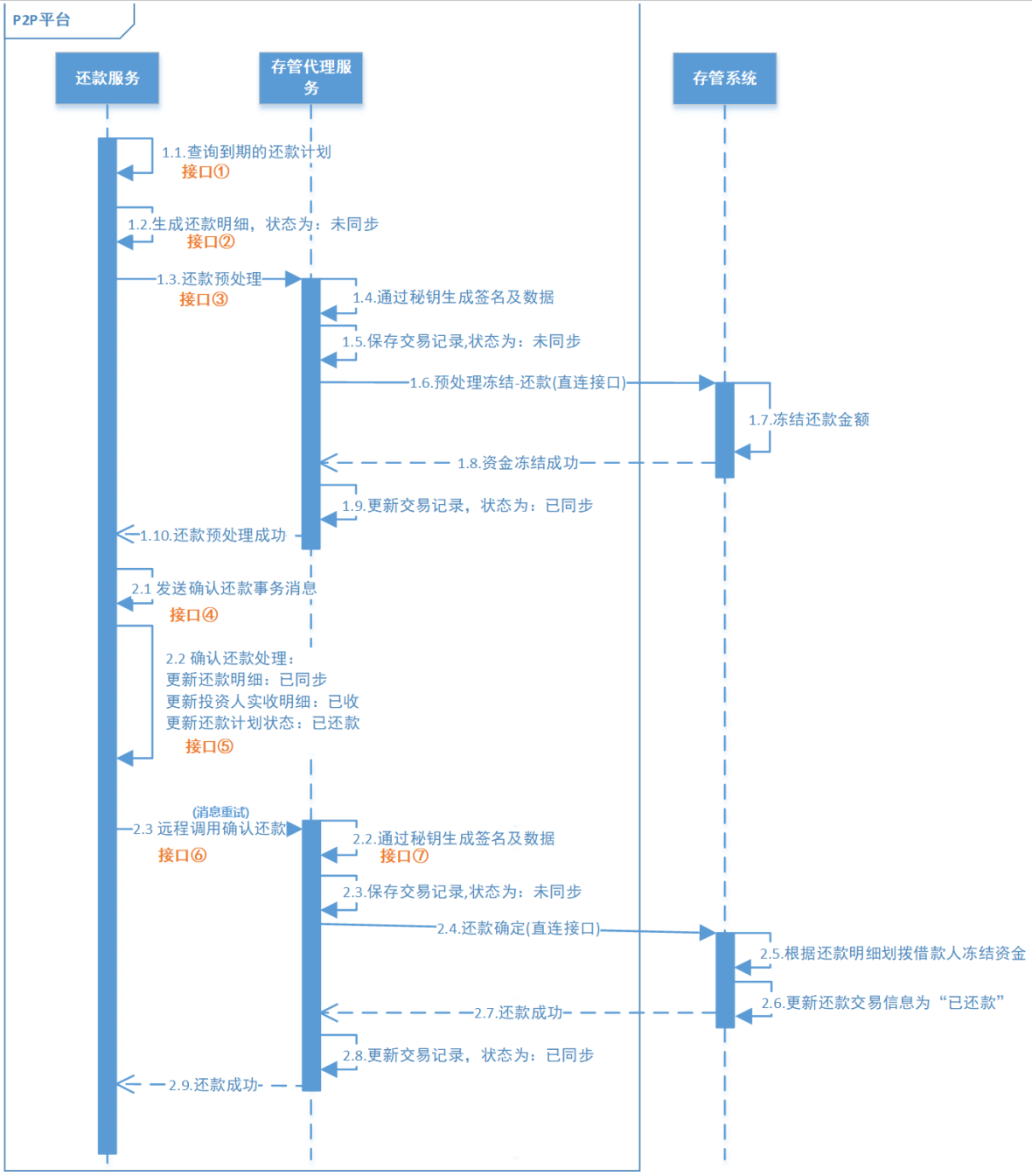
1 需求概述

满标放款审核通过后，就意味着交易已经达成。借款人以后就需要按照借款时约定的还款方式，在还款日当天将应还本息通过平台归还给投资人，这叫用户还款。借款人应该在临近还款日时，把应还的金额充值到平台账户中，平台在还款日当天会自动进行扣款。业务流程如下所示：



2 需求分析

用户还款一共涉及到三个服务：还款服务、存管代理服务和银行存管系统。其中银行存管系统还是像之前一样不用开发，直接使用即可。用户还款业务跟前端没有关系，由定时任务驱动业务执行，到期自动还款。



第一阶段：生成还款明细（图中1.1-1.2）

1. 还款服务每天定时查询到期的还款计划
2. 根据还款计划生成还款明细，状态为：未同步

第二阶段：还款预处理（图中1.3-1.10）

1. 还款服务通过feign请求存管代理服务进行还款预处理
2. 存管代理服务生成签名及数据，并保存交易记录(未同步)
3. 存管代理服务请求银行存管系统进行资金冻结
4. 银行存管系统返回预处理冻结结果给存管代理服务
5. 存管代理服务更新交易记录为：已同步，返回预处理结果给还款服务

第三阶段：确认还款（图中2.1-2.2）

1. 还款服务发送确认还款事务消息(半消息)
2. 还款服务执行处理本地事务：
    - 更新还款明细为：已同步
    - 更新投资人实收明细为：已收

- 更新还款计划状态为：已还款

3. 还款服务根据本地事务执行结果发生commit或rollback

第四阶段：还款成功（图中2.3-2.9）

1. 还款服务消费消息，并通过feign请求存管代理服务进行确认还款
2. 存管代理服务生成签名及数据，并保存交易记录（未同步）
3. 请求银行存管系统进行还款确定
4. 银行存管系统返回还款成功
5. 存管代理服务更新交易记录为：已同步，并返回结果给还款服务
6. 如果这个阶段处理失败，还款服务会重试消费

3 第一阶段：生成还款明细

3.1 接口定义

3.1.1 还款服务查询到期还款计划接口(接口①)

1、接口描述

根据日期查询所有到期的还款计划

2、接口定义

在RepaymentService接口中新增selectDueRepayment方法：

```
/**
 * 查询到期还款计划
 * @param date 格式为：yyyy-MM-dd
 * @return
 */
List<RepaymentPlan> selectDueRepayment(String date);
```

3.1.2 还款服务生成还款明细接口(接口②)

1、接口描述

1）根据还款计划id查询是否已存在记录

2）根据查询结果判断是否生成还款明细

2、接口定义

在RepaymentService接口中新增saveRepaymentDetail方法：

```
/**
 * 根据还款计划生成还款明细并保存
 * @param repaymentPlan
 * @return
 */
RepaymentDetail saveRepaymentDetail(RepaymentPlan repaymentPlan);
```

3.2 功能实现

3.2.1 还款服务查询到期还款计划接口(接口①)

1. 实现数据访问层，在PlanMapper接口中定义selectDueRepayment方法：

```
/**
* 查询所有到期的还款计划
* @return
*/
@Select("SELECT * FROM repayment_plan WHERE DATE_FORMAT(SHOULD_REPAYMENT_DATE,
'%Y-%m-%d') = #{date} " +
" AND REPAYMENT_STATUS = '0'")
List<RepaymentPlan> selectDueRepayment(String date);
```

2. 在业务层RepaymentServiceImpl类中调用mapper实现查询功能

```
@Override
public List<RepaymentPlan> selectDueRepayment(String date) {
    return repaymentPlanMapper.selectDueRepayment(date);
}
```

3.2.2 还款服务生成还款明细接口(接口②)

在业务层RepaymentServiceImpl类中实现生成还款明细功能：

```
@Override
public RepaymentDetail saveRepaymentDetail(RepaymentPlan repaymentPlan) {
    RepaymentDetail repaymentDetail = repaymentDetailMapper.selectOne(
            Wrappers.
<RepaymentDetail>lambdaQuery().eq(RepaymentDetail::getRepaymentPlanId,
                                                    repaymentPlan.getId()));

    if (repaymentDetail == null) {
        repaymentDetail = new RepaymentDetail();
        // 还款计划项标识
        repaymentDetail.setRepaymentPlanId(repaymentPlan.getId());
        // 实还本息
        repaymentDetail.setAmount(repaymentPlan.getAmount());
        // 实际还款时间
        repaymentDetail.setRepaymentDate(LocalDateTime.now());
        // 请求流水号

 repaymentDetail.setRequestNo(CodeNoUtil.getNo(CodePrefixCode.CODE_REQUEST_PREFI
X));
        // 未同步
        repaymentDetail.setStatus(StatusCode.STATUS_OUT.getCode());
        // 保存数据
        repaymentDetailMapper.insert(repaymentDetail);
    }
    return repaymentDetail;
}
```

3.2.3 业务触发入口

将来用户还款功能会由定时任务触发，但是目前定时任务尚未实现，因此为了测试方便，我们在RepaymentController类中定义一个testExecuteRepayment方法，通过往这里发请求去手动触发用户还款功能的执行。

```
@ApiOperation("测试用户还款")
@GetMapping("/execute-repayment/{date}")
public void testExecuteRepayment(@PathVariable String date) {
      repaymentService.executeRepayment(date);
}
```

由于需要调用业务层实现用户还款功能，所以需要在业务层RepaymentService接口中定义入口方法：

```
/**
 * 执行还款
 */
void executeRepayment(String date);
```

在业务层实现类RepaymentServiceImpl中实现该方法：

```
@Override
public void executeRepayment(String date) {
    //查询所有到期的还款计划
    List<RepaymentPlan> repaymentPlanList = selectDueRepayment(date);
    repaymentPlanList.forEach(repaymentPlan -> {
        //生成还款明细（未同步）
        RepaymentDetail repaymentDetail = saveRepaymentDetail(repaymentPlan);
        //未完待续...

    });
}
```

3.3 功能测试

4 第二阶段：还款预处理

4.1 接口定义(接口③)

在RepaymentService接口中新增preRepayment方法：

```
/**
 * 还款预处理：冻结借款人应还金额
 * @param repaymentPlan
 * @param preRequestNo
 * @return
 */
Boolean preRepayment(RepaymentPlan repaymentPlan, String preRequestNo);
```

4.2 功能实现

1. 定义Feign代理

```
@FeignClient(value = "depository-agent-service")
public interface DepositoryAgentApiAgent {

    @PostMapping("/depository-agent/l/user-auto-pre-transaction")
    RestResponse<String> userAutoPreTransaction(
        UserAutoPreTransactionRequest userAutoPreTransactionRequest);
}
```

2. 在RepaymentServiceImpl类中实现preRepayment方法：

```
@Autowired
private DepositoryAgentApiAgent depositoryAgentApiAgent;

@Override
public Boolean preRepayment(RepaymentPlan repaymentPlan, String preRequestNo) {
    // 1.构造请求数据
    final UserAutoPreTransactionRequest userAutoPreTransactionRequest =
generateUserAutoPreTransactionRequest(
            repaymentPlan, preRequestNo);

    // 2.请求存管代理服务
    final RestResponse<String> restResponse = depositoryAgentApiAgent
            .userAutoPreTransaction(userAutoPreTransactionRequest);

    // 3.返回结果
    return
DepositoryReturnCode.RETURN_CODE_00000.getCode().equals(restResponse.getResult()
);
}

/**
 * 构造存管代理服务预处理请求数据
 * @param repaymentPlan
 * @param preRequestNo
 * @return
 */
private UserAutoPreTransactionRequest
generateUserAutoPreTransactionRequest(RepaymentPlan repaymentPlan,
        String preRequestNo) {
    // 构造请求数据
    UserAutoPreTransactionRequest userAutoPreTransactionRequest = new
UserAutoPreTransactionRequest();
    // 冻结金额
    userAutoPreTransactionRequest.setAmount(repaymentPlan.getAmount());
    // 预处理业务类型

 userAutoPreTransactionRequest.setBizType(PreprocessBusinessTypeCode.REPAYMENT.g
etCode());
    // 标的号
    userAutoPreTransactionRequest.setProjectNo(repaymentPlan.getProjectNo());
```

```
    // 请求流水号
    userAutoPreTransactionRequest.setRequestNo(preRequestNo);
    // 标的用户编码
    userAutoPreTransactionRequest.setUserNo(repaymentPlan.getUserNo());
    // 关联业务实体标识
    userAutoPreTransactionRequest.setId(repaymentPlan.getId());
    // 返回结果
    return userAutoPreTransactionRequest;
}
```

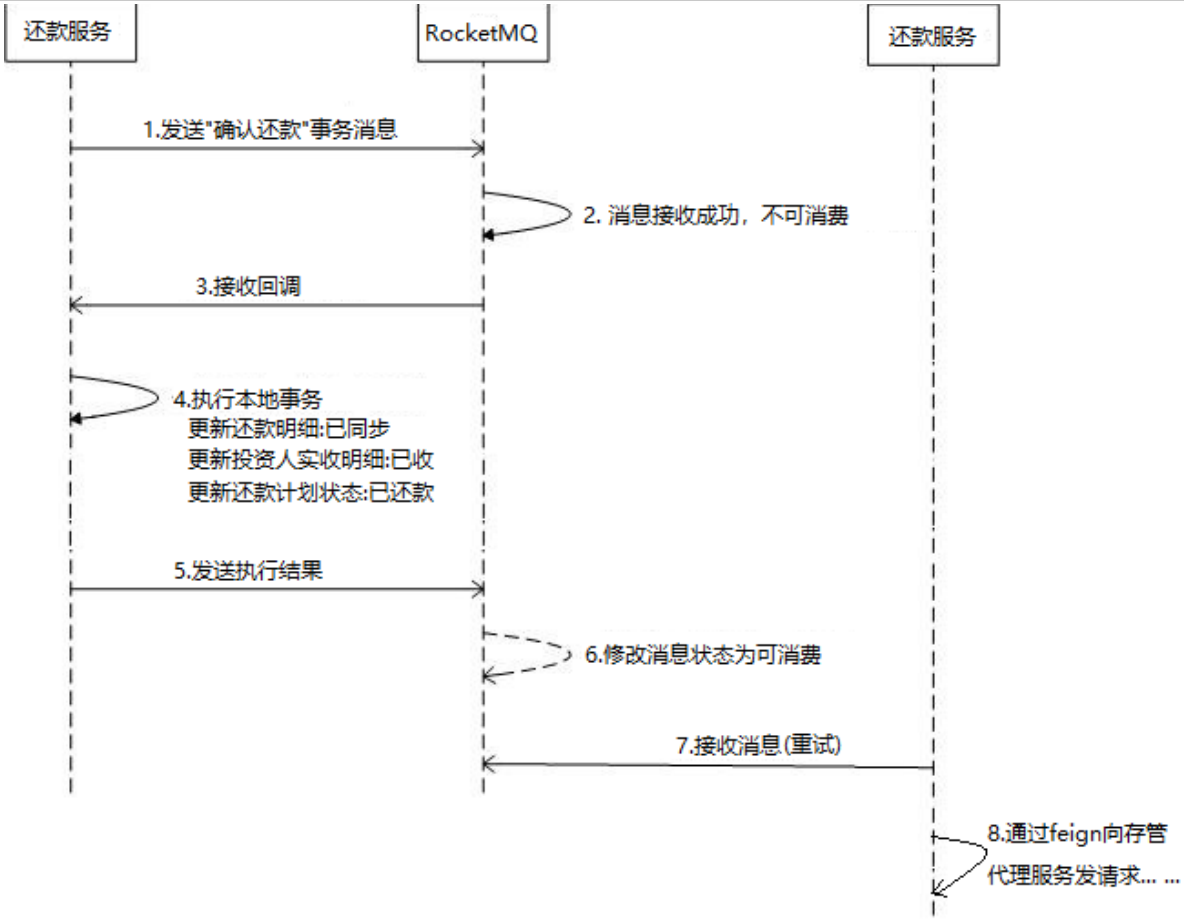
3. 完善业务触发入口代码，调用preRepayment方法执行还款预处理

```
@Override
public void executeRepayment(String date) {
    //查询所有到期的还款计划
    List<RepaymentPlan> repaymentPlanList = selectDueRepayment(date);
    repaymentPlanList.forEach(repaymentPlan -> {
        //生成还款明细（未同步）
        RepaymentDetail repaymentDetail = saveRepaymentDetail(repaymentPlan);

        //还款预处理
        String preRequestNo = repaymentDetail.getRequestNo();
        Boolean preRepaymentResult = preRepayment(repaymentPlan, preRequestNo);
        if (preRepaymentResult) {
            // 未完待续... ...

        }
    });
}
```

4.3 功能测试

5 第三阶段：确认还款

第三阶段和第四阶段的业务存在分布式事务问题，即：第三阶段业务执行成功，那么第四阶段业务也必须成功，这里通过RocketMQ可靠消息来解决该问题，具体如下图所示：



5.1 确认还款事务消息生产接口(接口④)

在message包中新建RepaymentProducer类，实现发送事务消息：

```
@Component
public class RepaymentProducer {

    @Resource
    private RocketMQTemplate rocketMQTemplate;

    public void confirmRepayment(RepaymentPlan repaymentPlan, RepaymentRequest
repaymentRequest) {
        //1.构造消息
        JSONObject object = new JSONObject();
        object.put("repaymentPlan", repaymentPlan);
        object.put("repaymentRequest", repaymentRequest);

        Message<String> msg =
MessageBuilder.withPayload(object.toJSONString()).build();
        //2.发送消息
        rocketMQTemplate
                .sendMessageInTransaction("PID_CONFIRM_REPAYMENT",
"TP_CONFIRM_REPAYMENT", msg, null);
    }
}
```

5.2 确认还款处理接口(接口⑤)

1、接口描述

1）更新还款明细为：已同步

2）更新应收明细状态为: 已收

3）更新还款计划状态：已还款

2、接口定义

在RepaymentService接口中，新增confirmRepayment方法：

```
/**
 * 确认还款处理
 * @param repaymentPlan
 * @param repaymentRequest
 * @return
 */
Boolean confirmRepayment(RepaymentPlan repaymentPlan, RepaymentRequest
repaymentRequest);
```

3、功能实现

由于在该功能中我们需要操作receivable_detail表，所以需要先实现操作该表的数据访问层代码：

ReceivableDetailMapper接口：

```
/**
操作receivable_detail的Mapper接口
*/
public interface ReceivableDetailMapper extends BaseMapper<ReceivableDetail> {
}
```

ReceivableDetailMapper.xml:

```
<?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.repayment.mapper.ReceivableDetailMapper">
</mapper>
```

在RepaymentServiceImpl类中实现confirmRepayment方法：

```
@Override
@Transactional
public Boolean confirmRepayment(RepaymentPlan repaymentPlan, RepaymentRequest
repaymentRequest) {
     //1. 更新还款明细：已同步
     String preRequestNo=repaymentRequest.getPreRequestNo();
        repaymentDetailMapper.update(null,Wrappers.
<RepaymentDetail>lambdaUpdate().set(RepaymentDetail::getStatus,StatusCode.STATUS
_IN.getCode()).eq(RepaymentDetail::getRequestNo,preRequestNo));

        //2.1 更新receivable_plan表为：已收
        //根据还款计划id，查询应收计划
```

```
        List<ReceivablePlan> rereceivablePlanList =
receivablePlanMapper.selectList(Wrappers.
<ReceivablePlan>lambdaQuery().eq(ReceivablePlan::getRepaymentId,repaymentPlan.ge
tId()));
        rereceivablePlanList.forEach(receivablePlan -> {
            receivablePlan.setReceivableStatus(1);
            receivablePlanMapper.updateById(receivablePlan);

            //2.2 保存数据到receivable_detail
            // 构造应收明细
            ReceivableDetail receivableDetail = new ReceivableDetail();
            // 应收项标识
            receivableDetail.setReceivableId(receivablePlan.getId());
            // 实收本息
            receivableDetail.setAmount(receivablePlan.getAmount());
            // 实收时间
            receivableDetail.setReceivableDate(DateUtil.now());
            // 保存投资人应收明细
            receivableDetailMapper.insert(receivableDetail);
        });

        //3. 更新还款计划：已还款
        repaymentPlan.setRepaymentStatus("1");
        int rows = planMapper.updateById(repaymentPlan);
        return rows>0;
}
```

5.3 确认还款事务消息监听类

在message包中新建ConfirmRepaymentTransactionListener类，接收消息，并实现调用本地事务和进行事务回查。

```
@Component
@RocketMQTransactionListener(txProducerGroup = "PID_CONFIRM_REPAYMENT")
public class ConfirmRepaymentTransactionListener implements
RocketMQLocalTransactionListener {

    @Autowired
    private RepaymentService repaymentService;

    @Override
    public RocketMQLocalTransactionState executeLocalTransaction(Message msg,
Object arg) {
        //1.解析消息
        final JSONObject jsonObject = JSON.parseObject(new String((byte[])
msg.getPayload()));
        RepaymentPlan repaymentPlan = JSONObject
                .parseObject(jsonObject.getString("repaymentPlan"),
RepaymentPlan.class);
        RepaymentRequest repaymentRequest = JSONObject
                .parseObject(jsonObject.getString("repaymentRequest"),
RepaymentRequest.class);

        //2.执行本地事务
        final Boolean isCommit =
repaymentService.confirmRepayment(repaymentPlan, repaymentRequest);
```

```
        //3.返回结果
        if (isCommit) {
            return RocketMQLocalTransactionState.COMMIT;
        } else {
            return RocketMQLocalTransactionState.ROLLBACK;
        }
    }

    @Override
    public RocketMQLocalTransactionState checkLocalTransaction(Message msg) {
        //1.解析消息
        final JSONObject jsonObject = JSON.parseObject(new String((byte[])
msg.getPayload()));
        RepaymentPlan repaymentPlan = JSONObject
                .parseObject(jsonObject.getString("repaymentPlan"),
RepaymentPlan.class);
        //2.事务状态回查
        RepaymentPlan newRepaymentPlan =
repaymentPlanMapper.selectById(repaymentPlan.getId());
        //3.返回结果
        if (newRepaymentPlan != null &&
newRepaymentPlan.getRepaymentStatus().equals("1")) {
            return RocketMQLocalTransactionState.COMMIT;
        }else {
            return RocketMQLocalTransactionState.ROLLBACK;
        }
    }
}
```

5.4 完善业务触发入口代码

在executeRepayment方法中调用RepaymentProducer发送消息：

```
@Autowired
private RepaymentProducer repaymentProducer;

@Override
public void executeRepayment(String date) {
    //查询所有到期的还款计划
    List<RepaymentPlan> repaymentPlanList = selectDueRepayment(date);
    repaymentPlanList.forEach(repaymentPlan -> {
        //生成还款明细（未同步）
        RepaymentDetail repaymentDetail = saveRepaymentDetail(repaymentPlan);

        //还款预处理
        String preRequestNo = repaymentDetail.getRequestNo();
        Boolean preRepaymentResult = preRepayment(repaymentPlan, preRequestNo);
        if (preRepaymentResult) {
            //构造还款信息请求数据
            RepaymentRequest repaymentRequest =
generateRepaymentRequest(repaymentPlan, preRequestNo);
            //发送确认还款事务消息
            repaymentProducer.confirmRepayment(repaymentPlan, repaymentRequest);
        }
    });
```

```
}

/**
 * 构造还款信息请求数据
 */
private RepaymentRequest generateRepaymentRequest(RepaymentPlan repaymentPlan,
String preRequestNo) {
    //根据还款计划id, 获取应收计划
    final List<ReceivablePlan> receivablePlanList =
receivablePlanMapper.selectList(
            Wrappers.
<ReceivablePlan>lambdaQuery().eq(ReceivablePlan::getRepaymentId,
repaymentPlan.getId()));

    //封装请求数据
    RepaymentRequest repaymentRequest = new RepaymentRequest();
    // 还款总额
    repaymentRequest.setAmount(repaymentPlan.getAmount());
    // 业务实体id
    repaymentRequest.setId(repaymentPlan.getId());
    // 向借款人收取的佣金
    repaymentRequest.setCommission(repaymentPlan.getCommission());
    // 标的编码
    repaymentRequest.setProjectNo(repaymentPlan.getProjectNo());
    // 请求流水号

 repaymentRequest.setRequestNo(CodeNoUtil.getNo(CodePrefixCode.CODE_REQUEST_PREF
IX));
    // 预处理业务流水号
    repaymentRequest.setPreRequestNo(preRequestNo);
    // 放款明细
    List<RepaymentDetailRequest> detailRequests = new ArrayList<>();
    receivablePlanList.forEach(receivablePlan -> {
        RepaymentDetailRequest detailRequest = new RepaymentDetailRequest();
        // 投资人用户编码
        detailRequest.setUserNo(receivablePlan.getUserNo());
        // 向投资人收取的佣金
        detailRequest.setCommission(receivablePlan.getCommission());
        // 派息 - 无
        // 投资人应得本金
        detailRequest.setAmount(receivablePlan.getPrincipal());
        // 投资人应得利息
        detailRequest.setInterest(receivablePlan.getInterest());
        // 添加到集合
        detailRequests.add(detailRequest);
    });
    // 还款明细请求信息
    repaymentRequest.setDetails(detailRequests);
    return repaymentRequest;
}
```

6 第四阶段：还款成功

6.1 定义接口(接口⑦)

1、接口描述

1）请求存管系统进行确认还款

2）返回结果给还款服务

2、接口定义

在DepositoryAgentApi接口中新增confirmRepayment方法：

```
/**
 * 还款确认
 * @param repaymentRequest 还款信息
 * @return
 */
RestResponse<String> confirmRepayment(RepaymentRequest repaymentRequest);
```

在DepositoryAgentController类中实现该方法：

```
@Override
@ApiOperation(value = "确认还款")
@ApiImplicitParam(name = "repaymentRequest", value = "还款信息",
                  required = true, dataType = "RepaymentRequest", paramType =
"body")
@PostMapping("1/confirm-repayment")
public RestResponse<String> confirmRepayment(@RequestBody RepaymentRequest
repaymentRequest) {
    return null;
}
```

6.2 业务层

在DepositoryRecordService接口中新增confirmRepayment方法：

```
/**
 * 还款确认
 * @param repaymentRequest
 * @return
 */
DepositoryResponseDTO<DepositoryBaseResponse> confirmRepayment(RepaymentRequest
repaymentRequest);
```

在DepositoryRecordServiceImpl类中实现该方法:

```
@Override
public DepositoryResponseDTO<DepositoryBaseResponse>
confirmRepayment(RepaymentRequest repaymentRequest) {
    try {
        //构造交易记录
        DepositoryRecord depositoryRecord = new
DepositoryRecord(repaymentRequest.getRequestNo(),
                PreprocessBusinessTypeCode.REPAYMENT.getCode(), "Repayment",
repaymentRequest.getId());
```

```
        // 分布式事务幂等性实现
        DepositoryResponseDTO<DepositoryBaseResponse> responseDTO =
handleIdempotent(depositoryRecord);
        if (responseDTO != null) {
            return responseDTO;
        }

        // 获取最新交易记录
        depositoryRecord =
getEntityByRequestNo(repaymentRequest.getRequestNo());

        /**
         * 确认还款(调用银行存管系统)
         */
        final String jsonString = JSON.toJSONString(repaymentRequest);
        // 业务数据报文，base64处理，方便传输
        String reqData = EncryptUtil.encodeUTF8StringBase64(jsonString);
        // 拼接银行存管系统请求地址
        String url = configService.getDepositoryUrl() + "/service";
        // 封装通用方法，请求银行存管系统
        return sendHttpGet("CONFIRM_REPAYMENT", url, reqData, depositoryRecord);
    } catch (Exception e) {
        throw new BusinessException(DepositoryErrorCode.E_160101);
    }
}
```

6.3 完善Controller代码

```
@Override
@ApiOperation(value = "确认还款")
@ApiImplicitParam(name = "repaymentRequest", value = "还款信息",
                  required = true, dataType = "RepaymentRequest", paramType =
"body")
@PostMapping("l/confirm-repayment")
public RestResponse<String> confirmRepayment(@RequestBody RepaymentRequest
repaymentRequest) {
    DepositoryResponseDTO<DepositoryBaseResponse> depositoryResponse =
depositoryRecordService
            .confirmRepayment(repaymentRequest);
    return getRestResponse(depositoryResponse);
}
```

6.4 确认还款接口(接口⑥)

1、接口描述

1）请求存管代理服务进行确认还款

2）根据返回结果处理后续流程

2、接口定义

在RepaymentService接口中，新增invokeConfirmRepayment方法，接收到消息后需要调用该方法：

```
/**
 * 远程调用确认还款接口
 * @param repaymentPlan
 * @param repaymentRequest
 */
void invokeConfirmRepayment(RepaymentPlan repaymentPlan, RepaymentRequest
repaymentRequest);
```

3、定义Feign代理

在DepositoryAgentApiAgent接口中新增confirmRepayment方法，用于向存管代理服务发请求

```
@PostMapping("/depository-agent/1/confirm-repayment")
RestResponse<String> confirmRepayment(RepaymentRequest repaymentRequest);
```

4、在RepaymentServiceImpl类中实现invokeConfirmRepayment方法：

```
@Override
public void invokeConfirmRepayment(RepaymentPlan repaymentPlan, RepaymentRequest
repaymentRequest) {
    RestResponse<String> repaymentResponse =
depositoryAgentApiAgent.confirmRepayment(repaymentRequest);
    if
(!DepositoryReturnCode.RETURN_CODE_00000.getCode().equals(repaymentResponse.getR
esult())) {
        throw new RuntimeException("还款失败");
    }
}
```

6.5 确认还款消息监听

在wanxinp2p-repayment-service工程的message包中定义ConfirmRepaymentConsumer类，用于接收消息，并向存管代理服务发起请求:

```
@Component
@RocketMQMessageListener(topic = "TP_CONFIRM_REPAYMENT", consumerGroup =
"CID_CONFIRM_REPAYMENT")
public class ConfirmRepaymentConsumer implements RocketMQListener<String> {

    @Autowired
    private RepaymentService repaymentService;

    @Override
    public void onMessage(String msg) {
        //1.解析消息
        JSONObject jsonObject = JSON.parseObject(msg);
        RepaymentPlan repaymentPlan = JSONObject
                .parseObject(jsonObject.getString("repaymentPlan"),
RepaymentPlan.class);
        RepaymentRequest repaymentRequest = JSONObject
                .parseObject(jsonObject.getString("repaymentRequest"),
RepaymentRequest.class);
```

```
        //2.执行业务
        repaymentService.invokeConfirmRepayment(repaymentPlan,
repaymentRequest);
    }
}
```

6.6 功能测试

7 定时还款任务

7.1 分布式任务调度Elastic-job

详见“Elastic-Job分布式任务调度”专题

7.2 功能实现

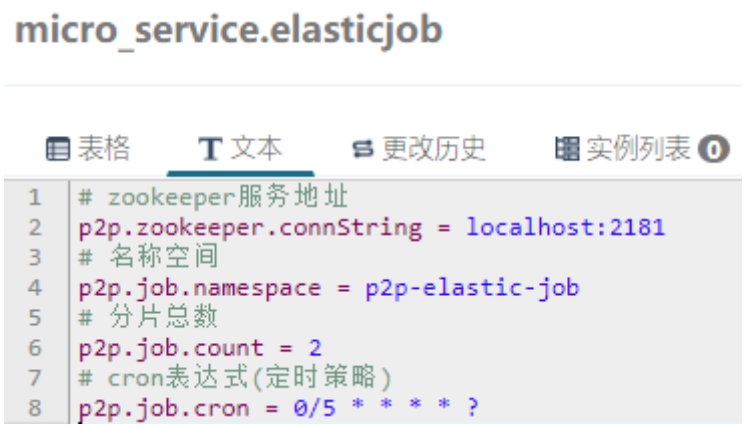
前面我们已经把用户还款功能实现了，但是需要通过定时任务去自动触发功能的执行。这里可以采用Elastic-Job实现分布式定时还款任务，在查询到期的还款计划时根据NUMBER_OF_PERIODS(期数)进行分片。

1. 依赖检查

```
<dependency>
    <groupId>com.dangdang</groupId>
    <artifactId>elastic-job-lite-spring</artifactId>
    <version>2.1.5</version>
</dependency>
```

2. springBoot配置

在Apollo中找到repayment-service项目，然后新建一个名字为“**micro_service.elasticjob**”的名称空间，并增加如下配置：



cron表达式的值为每5秒，这是为了测试方便，在实际运行中，不可能是这个值，例如可以是每天早上5点等，要结合具体业务去设置。

application.yml

```yaml
app:
  id: repayment-service
apollo:
  bootstrap:
    enabled: true
    namespaces: micro_service.elasticjob,micro_service.spr
```

3. 在PlanMapper接口中再增加一个selectDueRepayment方法，用来进行分片查询

```
/**
查询所有到期的还款计划
@return
*/
@Select("SELECT * FROM repayment_plan WHERE DATE_FORMAT(SHOULD_REPAYMENT_DATE,
'%Y-%m-%d') = #{date} AND REPAYMENT_STATUS = '0' AND MOD(number_of_periods,#
{shardingCount}) = #{shardingItem}")
List<RepaymentPlan> selectDueRepayment(@Param("date") String date,
                                       @Param("shardingCount") int
shardingCount,
                                       @Param("shardingItem") int shardingItem);
```

4. 修改业务层代码

RepaymentService.java

```
    /**
     * 查询所有到期的还款计划
     * @param  date 格式：yyyy-MM-dd
     * @param shardingCount 分片数量
     * @param shardingItem 分片值
     * @return
     */
    //List<RepaymentPlan> selectDueRepayment(String date);
    List<RepaymentPlan> selectDueRepayment(String date, int shardingCount, int shardingItem);

    /**
     * 执行用户还款
     * @param  date 格式：yyyy-MM-dd
     * @param shardingCount 分片数量
     * @param shardingItem 分片值
     */
    // void executeRepayment(String date);
    void executeRepayment(String date, int shardingCount, int shardingItem);
```

RepaymentServiceImpl.java

```
@Override
public List<RepaymentPlan> selectDueRepayment(String date, int shardingCount, int shardingItem) {
    return planMapper.selectDueRepayment(date, shardingCount, shardingItem);
}

@Override
public void executeRepayment(String date, int shardingCount, int shardingItem) {
    //查询到期的还款计划
    List<RepaymentPlan> repaymentPlanList=selectDueRepayment(date, shardingCount, shardingItem);

    //生成还款明细
    repaymentPlanList.forEach(repaymentPlan -> {
        System.out.println("当前分片:"+shardingItem+"\n"+repaymentPlan);
        RepaymentDetail repaymentDetail=saveRepaymentDetail(repaymentPlan);
```

5. 新建job包，在该包中新建RepaymentJob类执行定时任务

```
@Component
public class RepaymentJob implements SimpleJob {

    @Autowired
    private RepaymentService repaymentService;

    @Override
    public void execute(ShardingContext shardingContext) {
        //调用业务层执行任务

repaymentService.executeRepayment(LocalDate.now().format(DateTimeFormatter.ISO_LOCAL_DATE),
                shardingContext.getShardingTotalCount(),
                shardingContext.getShardingItem());
    }
}
```

6. zookeeper配置

```
@Configuration
public class ZKRegistryCenterConfig {

    //zookeeper服务地址
    @Value("${p2p.zookeeper.connString}")
    private  String ZOOKEEPER_CONNECTION_STRING ;

    //定时任务命名空间
    @Value("${p2p.job.namespace}")
    private  String JOB_NAMESPACE;

    //创建注册中心
    @Bean(initMethod = "init")
    public  ZookeeperRegistryCenter setUpRegistryCenter(){
        //zk的配置
        ZookeeperConfiguration zookeeperConfiguration = new
            ZookeeperConfiguration(ZOOKEEPER_CONNECTION_STRING, JOB_NAMESPACE);
        //创建注册中心
        ZookeeperRegistryCenter zookeeperRegistryCenter = new
```

```
            ZookeeperRegistryCenter(zookeeperConfiguration);
        return zookeeperRegistryCenter;
    }
}
```

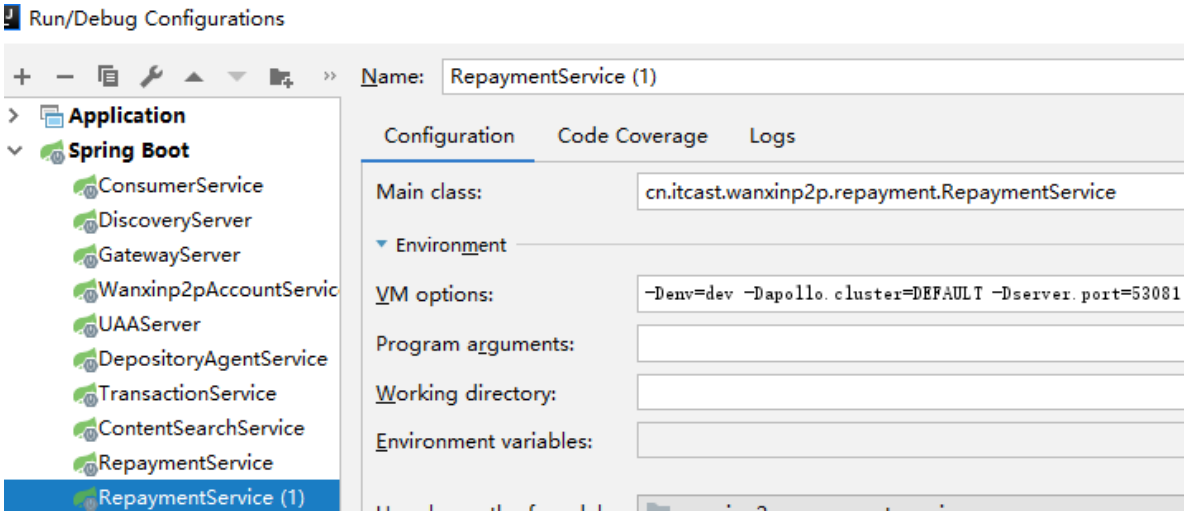
7. elastic-job配置

```
@Configuration
public class ElasticJobConfig {

    @Autowired
    RepaymentJob repaymentJob;

    @Autowired
    ZookeeperRegistryCenter registryCenter;

    @Value("${p2p.job.count}")
    private int shardingCount;

    @Value("${p2p.job.cron}")
    private String cron;

    /**
     * 配置任务详细信息
     * @param jobClass 任务执行类
     * @param cron  执行策略
     * @param shardingTotalCount 分片数量
     * @return
     */
    private LiteJobConfiguration createJobConfiguration(final Class<? extends
SimpleJob> jobClass,
                                                        final String cron,
                                                        final int
shardingTotalCount){
        //创建JobCoreConfigurationBuilder
        JobCoreConfiguration.Builder JobCoreConfigurationBuilder =
JobCoreConfiguration.newBuilder(jobClass.getName(), cron, shardingTotalCount);

        JobCoreConfiguration jobCoreConfiguration =
JobCoreConfigurationBuilder.build();
        //创建SimpleJobConfiguration
        SimpleJobConfiguration simpleJobConfiguration = new
SimpleJobConfiguration(jobCoreConfiguration, jobClass.getCanonicalName());
        //创建LiteJobConfiguration
        LiteJobConfiguration liteJobConfiguration = LiteJobConfiguration.
            newBuilder(simpleJobConfiguration).overwrite(true).build();
        return liteJobConfiguration;
    }

    @Bean(initMethod = "init")
    public SpringJobScheduler initSimpleElasticJob() {
        //创建SpringJobScheduler
        SpringJobScheduler springJobScheduler = new
SpringJobScheduler(repaymentJob, registryCenter,
                createJobConfiguration(repaymentJob.getClass(), cron,
shardingCount));
```

```
        return springJobScheduler;
    }
}
```

7.3 功能测试

由于分片总数设置为2，所以这里需要再配置一个还款服务，端口号是53081，最终一共要启动两个还款服务。



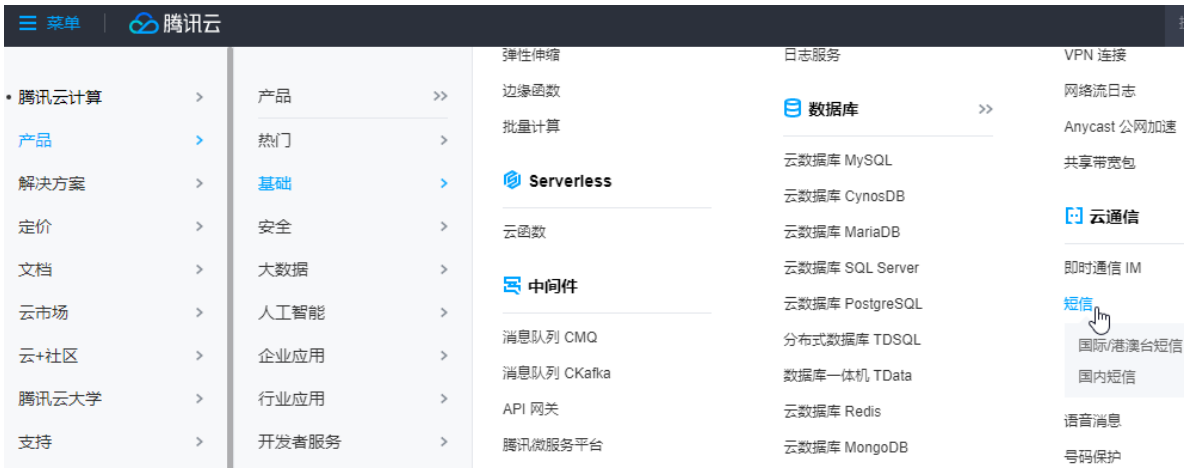
为了测试方便，需要从repayment_plan表中挑选一些数据，把SHOULD_REPAYMENT_DATE字段的值改为当前时间。

8 还款短信提醒

前面我们已经实现由系统自动执行定时还款任务，但是如果用户账户余额没钱，或余额不足，那么还款就会失败，所以用户必须保证在到期还款日之前往账户中充值。为了避免用户忘记此事，很有必要给用户发送短信进行还款提醒。

8.1 腾讯云概述和环境准备

此次我们采用腾讯云来实现发送短信的功能，下面是腾讯云官方截图。



个人或企业都可以注册并登录腾讯云，然后在后台需要进行一些必要的设置，如下图所示：

其中需要设置的是短信签名和短信正文模板，设置后需要通过腾讯审核。另外还需要创建使用短信服务的应用，得到AppID和APP Key，这些都需要在编码中用到。接下来就可以参考官方开发者指南(https://cloud.tencent.com/document/product/382)进行环境准备：

1. 导入依赖

```
<dependency>
    <groupId>com.github.qcloudsms</groupId>
    <artifactId>qcloudsms</artifactId>
    <version>1.0.6</version>
</dependency>
```

2. 在Apollo的repayment-service项目中新建sms名称空间，并配置短信发送参数

micro_service.sms

表格 文本 更改历史 实例列表 0

Key	Value
sms.qcloud.appId	14[redacted]69
sms.qcloud.appKey	36ff7b[redacted]25052
sms.qcloud.templateId	362314
sms.qcloud.sign	[redacted]

记得在application.yml中引入该名称空间

8.2 接口定义

8.2.1 还款服务发送短信接口

1、接口描述

调用腾讯云发送短信

2、接口定义

新建sms包，在该包中定义如下接口：

```
public interface SmsService {

    /**
     * 发送还款短信通知
     * @param mobile 还款人手机号
     * @param date 日期
     * @param amount 应还金额
     */
    void sendRepaymentNotify(String mobile, String date, BigDecimal amount);

}
```

8.2.2 还款服务还款提醒接口

1、接口描述

1）查询所有到期的还款计划

2）根据还款计划查询用户手机号

3）调用发送短信接口进行还款提醒

2、接口定义

在RepaymentService接口中，定义如下方法：

```
/**
 * 查询还款人相关信息，并调用发送短信接口进行还款提醒
 */
void sendRepaymentNotify(String date);
```

8.3 功能实现

8.3.1 还款服务还款提醒接口

1. 在ConsumerAPI中新增一个方法，用来根据用户id获取用户信息

```
 /**
  * 获取借款人用户信息-供微服务访问
  * @param id 用户标识
  * @return
  */
 RestResponse<BorrowerDTO> getBorrowerMobile(Long id);
```

2. 在ConsumerController类中实现该方法

```
@Override
@ApiOperation("获取借款人用户信息-供微服务访问")
@ApiImplicitParam(name = "id", value = "用户标识", required = true,
        dataType = "Long", paramType = "path")
@GetMapping("/l/borrowers/{id}")
public RestResponse<BorrowerDTO> getBorrowerMobile(@PathVariable Long id) {
    return RestResponse.success(consumerService.getBorrower(id));
}
```

3. 在还款微服务中创建Feign代理

```
@FeignClient(value = "consumer-service")
public interface ConsumerApiAgent {
    @GetMapping(value = "/consumer/l/borrowers/{id}")
    RestResponse<BorrowerDTO> getBorrowerMobile(@PathVariable("id") Long id);
}
```

4. 在RepaymentServiceImpl类中实现sendRepaymentNotify方法

```
@Autowired
private ConsumerApiAgent consumerApiAgent;

@Autowired
private SmsService smsService;

@Override
public void sendRepaymentNotify(String date) {
    //1.查询到期的还款计划
    List<RepaymentPlan> repaymentPlanList = selectDueRepayment(date);
```

```
    //2.遍历还款计划
    repaymentPlanList.forEach(repaymentPlan -> {
        //3.得到还款人的信息
        RestResponse<BorrowerDTO> consumerReponse = consumerApiAgent
                            .getBorrowerMobile(repaymentPlan.getConsumerId());

        //4.得到还款人的手机号
        String mobile = consumerReponse.getResult().getMobile();

        //5.发送还款短信
        smsService.sendRepaymentNotify(mobile,date,repaymentPlan.getAmount());
    });
}
```

8.3.2 还款服务发送短信接口

创建QCloudSmsServiceImpl实现发送短信功能

```
@Slf4j
@Service
public class QCloudSmsServiceImpl implements SmsService {

    @Value("${sms.qcloud.appId}")
    private int appId;

    @Value("${sms.qcloud.appKey}")
    private String appKey;

    @Value("${sms.qcloud.templateId}")
    private int templateId;

    @Value("${sms.qcloud.sign}")
    private String sign;

    @Override
    public void sendRepaymentNotify(String mobile, String date, BigDecimal
amount) {
        log.info("给手机号{}，发送还款提醒：{},金额:{}",mobile,date,amount);
        SmsSingleSender ssender = new SmsSingleSender(appId, appKey);
        try {
            ssender.sendWithParam("86", mobile,
                    templateId, new String[]{date, amount.toString()}, sign, "",
"");
        }catch (Exception ex){
            log.error("发送失败：{}",ex .getMessage());
        }
    }
}
```

8.4 功能测试

1. 在RepaymentController中添加测试方法：

```
@ApiOperation("测试还款短信提醒")
@GetMapping("/repayment-notify/{date}")
public void testRepaymentNotify(@PathVariable String date) {
    repaymentService.sendRepaymentNotify(date);
}
```

2. 启动Apollo、Zookeeper、还款微服务和用户中心微服务，通过浏览器访问 http://127.0.0.1:53080/repayment/repayment-notify/2019-08-30

> 注意：Mapper接口中的方法不能重载，因为Mybatis内部默认把方法名作为ID使用

8.5 定时还款短信提醒

还款短信提醒功能靠人工触发运行，明显是不现实的。因此，我们需要把该功能设计成定时任务由系统自动执行。

在RepaymentJob任务类的execute方法中，加入调用业务层执行还款短信提醒任务的代码：

```
@Component
public class RepaymentJob implements SimpleJob {

    @Autowired
    private RepaymentService repaymentService;

    @Override
    public void execute(ShardingContext shardingContext) {

    //调用业务层执行还款任务
    repaymentService.executeRepayment(LocalDate.now().format(
                                    DateTimeFormatter.ISO_LOCAL_DATE),
                                    shardingContext.getShardingTotalCount(),
                                    shardingContext.getShardingItem());

    //调用业务层执行还款短信提醒任务(提前一天)
    repaymentService.sendRepaymentNotify(LocalDate.now().plusDays(1)

.format(DateTimeFormatter.ISO_LOCAL_DATE));
    }
}
```