

微服务与区块链

课程大作业技术报告

题目：住房贷款系统

组员：2022104167 徐威

2022104169 方晓坤

2022000911 陈劭

一、系统介绍

1. 系统简介

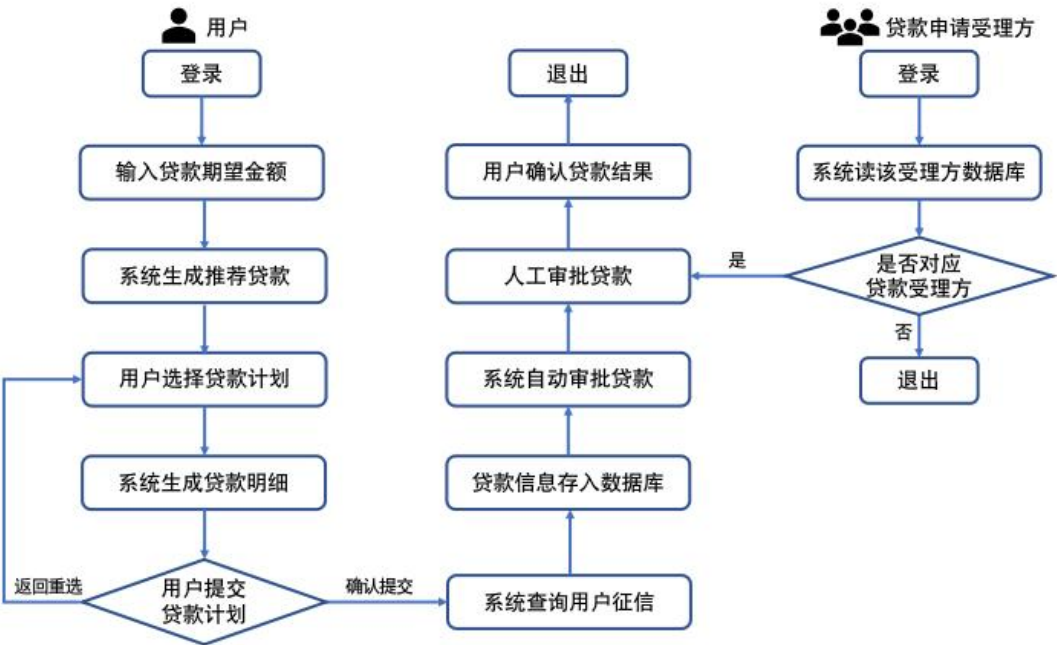
1.1 系统名称

住房贷款系统

1.2 系统总体流程

用户进入系统后，输入账号与密码登录，输入贷款期望，系统在服务端通过查询银行数据库和公积金数据库，给予贷款人相应的贷款推荐。用户结合推荐，在用户端页面选择具体的贷款计划，包括贷款类别、还款方式、贷款期限、贷款金额等。系统利用用户提交的贷款计划计算出贷款明细并返回给用户界面，用户查看贷款明细后可以选择重新选择或者提交贷款计划。用户提交贷款计划后，系统判断贷款人的征信情况，并经过公积金数据库和银行数据库的自动审批后，将贷款明细和自动审批结果存入贷款数据库。最后，由贷款申请受理方（可能是银行或者公积金管理方）给出是否批准贷款的结论，并将最终结果反馈给用户端界面，用户确认最终贷款结果。

1.3 系统流程图



2. 系统功能

2.1 基础贷款功能

用户可以自由选择贷款类型、还款方式、还款计划、挑选商业银行等，选择完毕形成完整的贷款计划并提交给系统后，会由贷款申请受理方（可能是银行或者公积金管理方）给出是否批准贷款的结论。

2.2 贷款推荐功能

用户提交期望的贷款金额，系统会根据用户信息自动推荐最佳贷款计划。具体的贷款推荐逻辑为：

第一，优先为用户推荐公积金贷款，因为公积金贷款的利率低于商业贷款，在公积金贷款额度不满足用户期望的贷款金额时，剩余部分用商业贷款额度补全，即采用混合贷款的方式。若用户的公积金贷款额度与最高的商业贷款额度之和不满足用户期望的贷款金额，则系统会提醒用户降低期望的贷款金额。

第二，如果推荐混合贷款或者商业贷款，在推荐具体的商业银行时，首先看商业银行的

额度是否满足用户期望的贷款金额，在满足用户期望的前提下，为用户计算不同商业银行给该用户的折扣利率，最后为用户推荐利率最低的商业银行。

第三，为用户推荐贷款年限时，优先推荐还款年数较少的贷款计划，这样可以使用户的总还款金额较低。但是考虑到用户每月的还款压力，系统会计算不同年限的月还款金额，并读取用户的月收入信息来进行比较，最后为用户推荐满足月还款金额不超过用户月收入条件下的还款年数最少的贷款年限。若对于 30 年的还款年限，每月还款依然比用户月收入高的话，系统会提醒用户降低期望的贷款金额。

第四，默认为用户推荐等额本息的还款方式，等额本息的还款曲线更平稳，适合大部分用户。

2.3 自动审批功能

用户提交具体的贷款计划，系统会根据查询到的用户信息自动进行审批，审批结果供人工审批参考。

公积金贷款部分的自动审批逻辑为：若用户的征信情况良好，同时用户已经连续缴纳公积金 12 个月以上，并且贷款金额没有超过用户的贷款额度，则自动审批贷款通过，否则拒绝该贷款申请。

商业贷款部分的自动审批逻辑为：若用户的征信情况良好，同时贷款的每月（首月）还款不超过用户月收入的 80%，并且贷款金额没有超过用户的贷款额度，则自动审批贷款通过，否则拒绝该贷款申请。

2.4 人工审批功能

由于现实中用户的贷款申请情况复杂，不能仅凭系统自动审批来最终决定贷款申请结果，因此本系统加入了人工审批功能，在系统自动审批并给出审批建议以后，由贷款申请受理方（可能是银行或者公积金管理方）根据用户的现实情况给出是否批准贷款的结论。

3. 系统使用方法

3.1 用户使用方法

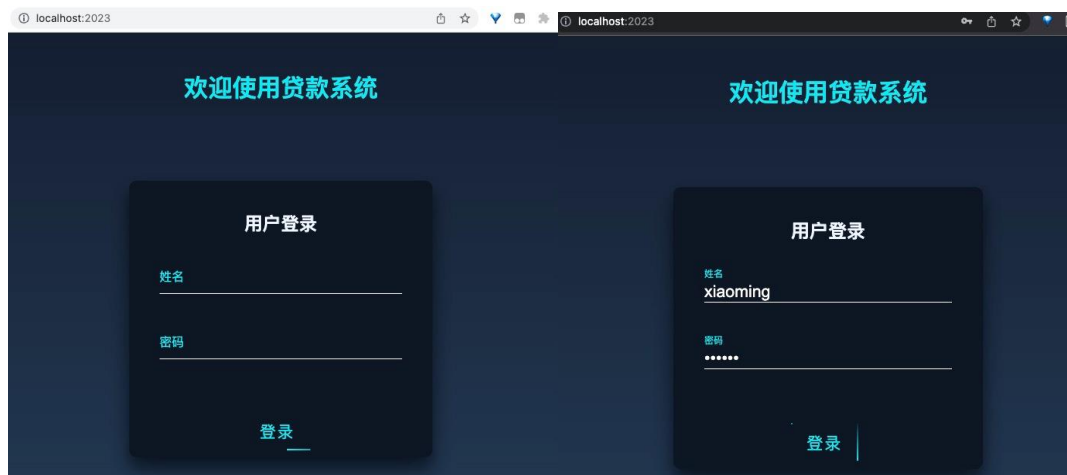
进入系统后，用户需要输入账号密码进行登录，登录后可以提交贷款期望金额，系统显示相应的贷款推荐。用户结合系统推荐贷款选择贷款计划，具体包括贷款的类别、还款方式、贷款期限、贷款金额等，选择完毕后系统给出相应的贷款明细，用户可以选择重新选择或提交贷款计划。提交贷款计划后用户需等待系统自动审批和人工审批完成，可获取最终贷款是否批准的结果。

3.2 用户使用示例：

1) 用户登录。

登录错误情况：





2) 输入期望贷款金额。



3) 系统给予相应推荐, 用户结合推荐贷款选择贷款计划, 具体包括贷款的类别、还款方式、贷款期限、贷款金额等。

贷款计划

欢迎使用贷款系统, 根据您的期望贷款金额100.00万元, 给您如下贷款建议:

推荐您采用混合贷款, 其中公积金贷款20.00万元, 贷款利率3.25%, 商业贷款银行选择 工商银行, 商业贷款80.00万元, 贷款利率4.75%, 贷款15年, 等额本息还款, 月供7628.0元, 总还款137.30万元~

请选择贷款计划

-
-
-

贷款类别

☐ 公积金贷款
☐ 混合贷款
☐ 商业贷款

还款方式

等额本息

贷款期限

5年

公积金贷款金额

0

选择银行

中国银行

银行贷款金额

0

计算房贷

- 4) 系统根据用户选择的贷款计划计算贷款计划明细,用户可以根据明细选择重选贷款计划或提交贷款申请。

您的贷款计划明细

贷款类型	混合贷款
贷款总额	60.00万元
还款类型	等额本息
还款总额	109.56万元
支付利息	49.56万元
还款年数	30年
每月还款	3043.45元
每月递减	0.00元

返回重选

提交申请

- 5) 提交申请后用户等待人工审批结果,可刷新状态。

localhost:2023/wait_judge

请您耐心等待人工审批结果...

刷新状态

- 6) 经过系统自动审批和不同的申请受理方的人工审批后,用户可获取最终贷款申请结果。

localhost:2023/wait_judge

您的贷款申请结果

贷款类型	混合贷款
贷款总额	45.77万元
还款类型	等额本息
还款总额	49.80万元
支付利息	4.03万元
还款年数	5年
每月还款	8299.91元
每月递减	0.00元

很遗憾! 贷款失败!

其中公积金贷款成功, 商业贷款失败。

localhost:2023/wait_judge

您的贷款申请结果

贷款类型	混合贷款
贷款总额	45.77万元
还款类型	等额本息
还款总额	49.80万元
支付利息	4.03万元
还款年数	5年
每月还款	8299.91元
每月递减	0.00元

恭喜您! 贷款成功!

3.3 申请受理方使用方法

进入系统后,申请受理方(公积金管理方或者某个银行的管理方)需要输入账号密码进行登录,登录后系统会自动读取该受理方对应的贷款数据库,若有未审批的贷款申请,系统会直接显示贷款详情让该受理方进行审批,审批完成后可返回再审批其他未审批的贷款申请。若该受理方没有需要审批的贷款申请,则会直接显示暂时没有贷款需要审批。

3.4 申请受理方使用示例

- 1) 申请受理方登录(登录过程同用户登录,截图省略)
- 2) 系统查询对应数据库后,受理方审批贷款或因不需要审批贷款而退出

假设用户选择了混合贷款,且商业银行选择了中国银行(银行1),公积金管理方、中国银行管理方、工商/建设银行(银行2/3)管理方登录后界面如下:

请审批用户xiaoming的贷款申请

用户信息

编号	1
姓名	xiaoming
征信情况	信用良好
月收入	10000.00元
公积金存款	10.00万元
贷款额度	20.00万元
缴纳月数	20

贷款信息

贷款类型	混合贷款
还款类型	等额本息
还款年数	5年
贷款利率	3.25%
贷款总额	42.34万元
支付利息	3.59万元
还款总额	45.93万元
每月还款	7655.65元

系统自动审批结果为 拒绝 该贷款申请，请仔细审批！

拒绝

通过

请审批用户xiaoming的贷款申请

用户信息

编号	1
姓名	xiaoming
征信情况	信用良好
月收入	10000.00元
本行存款	10.00万元
贷款额度	20.00万元

贷款信息

贷款类型	混合贷款
还款类型	等额本息
还款年数	5年
贷款利率	4.90%
贷款总额	3.42万元
支付利息	0.44万元
还款总额	3.87万元
每月还款	644.26元

系统自动审批结果为 通过 该贷款申请，请仔细审批！

拒绝

通过

暂时没有贷款需要审批

3) 申请受理方审批完成（均审批完成后用户才能获取最终贷款结果）

分别为公积金管理方和中国银行管理方审批贷款申请为通过：

localhost:2023/store_fund_judge/1

审批完成

localhost:2023/store_bank_judge/1

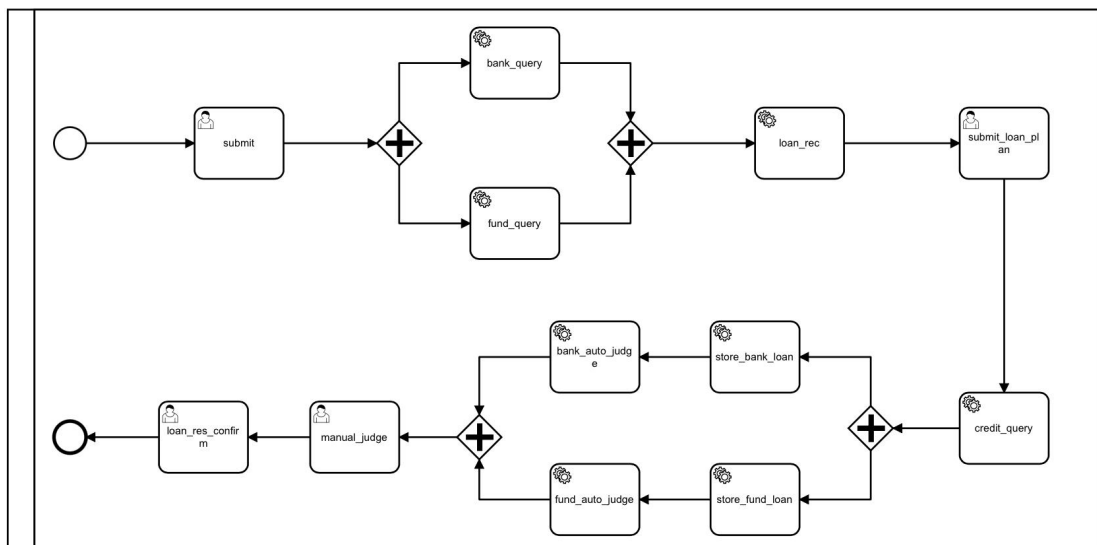
审批完成

二、方案

我们组的住房贷款管理系统采用微服务来实现，具体实现方案是由 BPMN 引擎驱动、MySQL 数据库作为后台资源仓库和 Flask 框架所组成的，下面将对这三个组成部分分别作出介绍，最后再介绍系统的具体部署流程。

1. BPMN 数据流设计

首先我们介绍在 BPMN 引擎下，我们是如何设计数据流并执行的。



我们所设计的 BPMN 工作流程图如上图所示。该流程为了简易起见，使用了单泳道，主体上呈现串行执行，并且只使用了并行网关。该系统允许用户，也即贷款申请人先行登录系统并提交自己预期的贷款数额，然后系统根据用户 ID 从后端的 MySQL 数据库查询出他在各个银行以及公积金的贷款额度、贷款利息比率以及利息优惠率，然后系统会综合这些因素给出一个较优的贷款方案推荐给用户。用户在前端根据推荐的贷款方案和自己的实际情况提交自己最终的贷款方案，其中包括最终的贷款额度、贷款银行和还款计划等。系统会在前端统计用户的输入，反馈给用户一张贷款方案表让用户确认。用户可以返回修改，也可以在确认无误后提交系统后台。在用户提交之后，后端会查询出用户的征信情况加入数据流，作为后面审批的依据。系统收到用户提交的贷款方案后，会根据方案的实际情况分别存入商业银行和公积金管理方的数据库中并进行自动地审批，审批角度主要有征信是否良好以及额度是否超出。自动审批的结果会合并交付对应的人员作为参考进行二次审批，商业银行专员和公积金管理人员在给出人工审批的结果后，用户会最终得到确认信息和贷款是否通过的情况。

下面简要介绍下该 BPMN 流程图中各个任务的功能，详细介绍我们放在后面的实现细节中。我们的 BPMN 流程图由 4 个用户任务和 8 个服务任务组成。众所周知，用户任务往往与前端结合，负责接收用户在前端的输入并交付后端。我们第一个用户任务 `submit` 是在用户登录后，负责接收用户输入的贷款数额，比如 10 万。随即经过一个并行网关，用户信息和贷款数额被传给服务任务 `bank_query` 和 `fund_query`。这两个任务分别从数据库中查询出该用户在各个商业银行和公积金方的贷款额度、利息比率和优惠比率，然后这些信息会合并交付下一个服务任务 `loan_rec`。该服务任务的功能是做出一个较优的贷款方案推荐给用户以供参考，也是我们流程中最具有创新的一点。在接下来的用户任务 `submit_loan_plan` 中，用户在前端依据推荐的贷款方案自行填写好最终的方案，并确认提交。用户提交方案之后，服务任务 `credit_query` 会从后台数据库中查询出该用户的征信情况。值得一提的是，我们的 BPMN 流程图经过了多次改版。在最初的一版中，流程在查询征信后会经过一个排他网关，如果征信差则直接拒绝贷款方案。后来我们经过和助教的沟通，发现该 BPMN 引擎并没有实现排他网关功能，便将流程改版，将个人征信情况对贷款方案通过与否的影响放到了后面的自动审批中。回到现在的流程，在得到用户征信情况后经过一个并行网关，`store_bank_loan` 和 `store_fund_loan` 两个服务任务会将收到的贷款方案分别存入商业银行或公积金管理方的数据库中。存入数据库后，会由后面两个服务任务 `bank_auto_judge` 和 `fund_auto_judge` 自动对贷款方案进行审批。用户任务 `manual_judge` 即允许人员登录系统对合并过后的审批结果进行人工审批，审批完成后用户可以在前端通过用户任务 `loan_res_confirm` 确认最终得到的审批结果。用户确认后，该流程结束。

2. MySQL 数据库设计

下面介绍我们的 MySQL 数据库设计。这部分会详细解释 MySQL 作为后端数据仓库，是如何支撑上游这些服务的。

写在前面。微服务之所以作为微服务，它应该由各个商业银行和公积金管理方独立编写，然后使用 BPMN 数据流引擎将这些个微服务组合起来。也就是说，每个服务，比如查询用户在银行的贷款额度，需要由多个银行的微服务组合生成。而每个银行实际上也不会将业务的数据存储在一个数据库中，各个银行的微服务理应是访问各自不同的数据库，共同组合成这样的一个住房贷款管理系统。但是为了开发的方便起见，我们组在实际设计中将所有银行和公积金管理方的数据库合并设计成了一个。这可能成为本系统最大的黑点，需要之后进行完善。

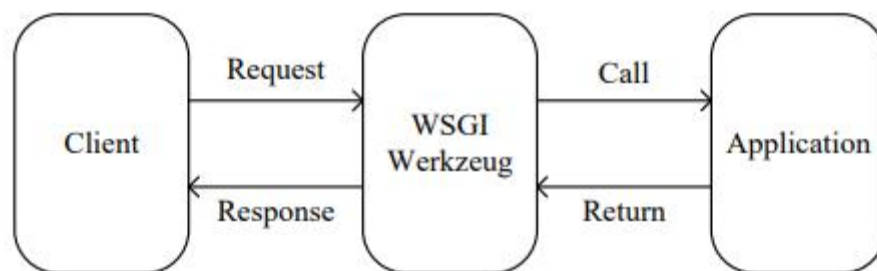
下面介绍 MySQL 的数据表设计。首先我们在 MySQL 中建立了数据库 `loan`，统一存放所有的数据表。数据表一共有 6 个，分别是 `User` 表、`Bank` 表、`BankUser` 表、`FundUser` 表、`BankLoan` 表和 `FundLoan` 表。`User` 表储存用户的编号、用户名、密码、征信水平和工资水

平。该表支持了用户前端登录、征信查询等服务。**Bank** 表存储各个商业银行的编号、名字和基础利息率。**BankUser** 表储存了不同用户在各个银行的存款数额、贷款额度和利息折扣率。**FundUser** 表储存了各个用户公积金的存款数额、贷款额度、贷款利息率和公积金的缴纳年限。**BankUser** 表和 **FundUser** 表联合起来，共同支持了贷款方案推荐等服务。至于 **BankLoan** 表和 **FundLoan** 表，它们存储了用户提交上来的贷款方案，同时也包含了自动审批和人工审批结果等字段，主要支持了贷款方案存储、自动审批和人工审批等业务。

3. Flask 框架介绍

Flask 是一个轻量级的可定制框架，使用 Python 语言编写，较其他同类型框架更为灵活、轻便、安全且容易上手。它可以很好地结合 MVC 模式进行开发，开发人员分工合作，小型团队在短时间内就可以完成功能丰富的中小型网站或 Web 服务的实现。另外，Flask 还有很强的定制性，用户可以根据自己的需求来添加相应的功能，在保持核心功能简单的同时实现功能的丰富与扩展，其强大的插件库可以让用户实现个性化的网站定制，开发出功能强大的网站。

Flask 的基本模式为在程序里将一个视图函数分配给一个 URL，每当用户访问这个 URL 时，系统就会执行给该 URL 分配好的视图函数，获取函数的返回值并将其显示到浏览器上，其工作过程见图。



Flask 有以下四大特点：

- 1) Flask 主要包括 Werkzeug 和 Jinja2 两个核心函数库，他们分别负责批处理和安全方面的工程，这些基础函数为 Web 项目开发过程提供了丰富的基础组件。
- 2) Flask 中的 Jinja2 模板引擎，提高了前端代码的复用率。可以大大提高开发效率并且有利于后期的开发与维护。
- 3) Flask 不会指定数据库和模板引擎等对象，用户可以根据需要自己选择各种数据库。
- 4) Flask 不提供表单验证功能，在项目实施过程中可以自由配置，从而为应用程序开发提供数据库抽象层基础组件，支持进行表单数据合法性验证、文件上传处理、用户身份认证和数据库集成等功能。

由于 Flask 框架有以上基本模式和特点，因此非常适合作为本系统的开发框架。首先在服务管理端，采用 Flask 框架来进行微服务的开发，可以只运行一个 python Flask 程序，为每个微服务分配一个视图函数即 URL，大大提高了小组合作的开发效率。另外，Flask 编写的微服务在进行调试时较 Java 而言更加方便快捷，修改代码后便可以直接运行调试，不需要打包重新部署等流程。在系统迭代开发的过程中，所需要实现的功能以及扩展会逐渐增多，针对这一特点更是需要使用易扩展的 Flask 框架。

另外，本系统的用户端，即前端 web 页面也采用 Flask 框架来编写，这样组内三个成员写的代码更容易对接，在合并调试时也能提高效率。

4. 系统部署流程

4.1 创建微服务

在 nacos 服务器上创建一个服务。

脚本: `curl -X POST '10.77.70.184:8848/nacos/v1/ns/service?serviceName=2022104167loan&groupName=WORKFLOW&protectThreshold=0'`

服务名	分组名称	集群数目	实例数	健康实例数	触发保护阈值	操作
2022104167	DEFAULT_GROUP	1	2	0	true	详情 示例代码 删除
2022104167test	WORKFLOW	1	1	0	true	详情 示例代码 删除
2022104167loan	WORKFLOW	1	1	0	true	详情 示例代码 删除

4.2 实例化微服务

把 `loan_service.py` 程序运行得到的 ip 与端口号实例化到上面创建的服务。

脚本: `curl -X POST 'http://10.77.70.184:8848/nacos/v1/ns/instance?port=2024&healthy=true&ip=10.9.32.221&weight=1.0&serviceName=2022104167loan&groupName=WORKFLOW&encoding=GBK'`

```
$ python loan_service/loan_service.py
* Serving Flask app 'loan_service'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment
. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:2024
* Running on http://10.9.32.221:2024
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 120-506-687
```

服务名	分组名称	集群数目	实例数	健康实例数	触发保护阈值	操作
2022104167	DEFAULT_GROUP	1	2	0	true	详情 示例代码 删除
2022104167test	WORKFLOW	1	1	0	true	详情 示例代码 删除
2022104167loan	WORKFLOW	1	1	1	false	详情 示例代码 删除

4.3 部署 bpmn 图

采用课堂上给的部署 python 代码, 把 bpmn 图部署到对应的服务器。

脚本: `python3 deploy.py --BpmnPath=./bpmn/2022104167loan.bpmn --deploymentName=2022104167loan.bpmn`

4.4 实例化 bpmn 图

把部署好的 bpmn 图实例化 (每次系统执行前都需要重新实例化获取新的 oid), 实例化时 `processData` 和 `businessData` 提交空的, `.table` 文件填写 `anyone`。

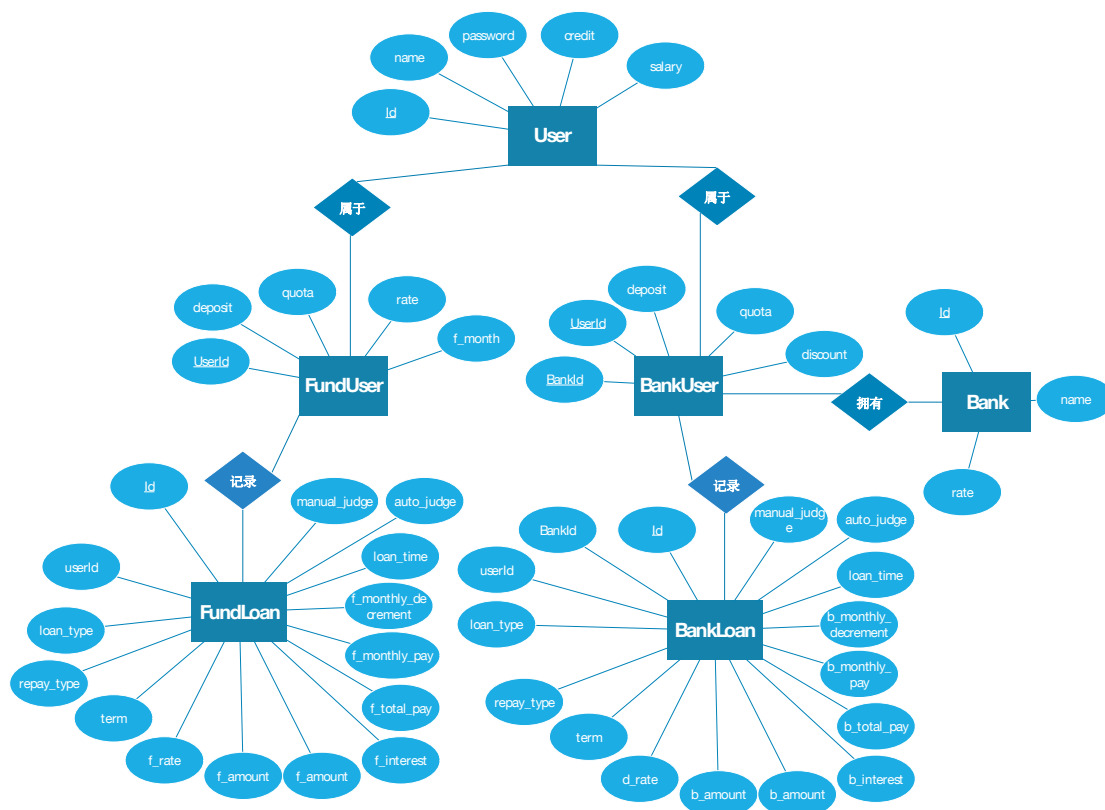
脚本: `python3 instance.py --allocationTablePath=./table/2022104167loan.bpmn.table --deploymentName=2022104167loan.bpmn --processDataPath=./data/processData.txt --businessDataPath=./data/instanceData.txt`

4.5 运行本地的用户端前端程序

```
$ python loan_local/loan_local.py
* Serving Flask app 'loan_local'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://127.0.0.1:2023
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 120-506-687
```

三、技术细节

1. 数据库 ER 图



对于现实中的贷款流程的一些细节问题，在数据库设计的过程中都有考虑到，包括贷款基准利率、商业银行给用户的个人贷款折扣、公积金缴纳月数等。

2. 微服务输入输出数据流

输入：input，输出：output，用户端（前端）获取数据：get，注释：#。

2.1 login&submit

get {userId, loanEXP}

output {"userId":userId, "loanEXP":loanEXP} #loanEXP:期望贷款金额

2.2 bank_query

input {"userId":"init.userId"}

output {

```
"bank":[
  {"bankId":0, "d_rate":d_rate0, "b_quota":b_quota0},
  {"bankId":1, "d_rate":d_rate1, "b_quota":b_quota1},
  {"bankId":2, "d_rate":d_rate2, "b_quota":b_quota2}
], # d_rate:对应银行的打折后利率, b_quota:对应银行的贷款额度
"salary":salary
}
```

2.3 fund_query

input {"userId":"init.userId"}

output {"f_rate":f_rate, "f_quota":f_quota}

2.4 loan_rec

input {

```
"userId":"init.userId",
"loanEXP":"init.loanEXP",
"f_rate":"fund_query.f_rate",
"f_quota":"fund_query.f_quota",
"bankId":"bank_query.bank.bankId",
"d_rate":"bank_query.bank.d_rate",
"b_quota":"bank_query.bank.b_quota",
```

```

        "salary":"bank_query.salary"
    }
output {
    "userId":userId,
    "loan_type":loan_type,      # 推荐时默认计算等额本息
    "bank":{
        "bankId":bankId,
        "b_amount":b_amount,    # b_amount:在银行贷款的金额
        "d_rate":d_rate
    },
    "fund":{
        "f_amount":f_amount,
        "f_rate":f_rate
    },
    "term":term,
    "monthly_pay":monthly_pay,
    "total_pay":total_pay,
    "f_quota":f_quota,
    "b_quota":b_quota
    }
}

```

2.5 submit_loan_plan

```

get {loan_rec 的全部 output}
output {
    "userId":userId,
    "name":name,
    "loan_type":loan_type,
    "repay_type":repay_type,    # repay_type:等额本息/等额本金
    "term":term,
    "f_rate":f_rate,
    "f_amount":f_amount,
    "f_interest":f_interest,
    "f_total_pay":f_total_pay,
    "f_monthly_pay":f_monthly_pay,
    # monthly_pay:等额本息则每期还款，等额本金则首期还款
    "f_monthly_decrement":f_monthly_decrement,
    "bankId":bankId,
    "d_rate":d_rate,
    "b_amount":b_amount,
    "b_interest":b_interest,
    "b_total_pay":b_total_pay,
    "b_monthly_pay":b_monthly_pay,
    "b_monthly_decrement":b_monthly_decrement
    }
}

```

2.6 credit_query

```

input {"userId":"init.userId"}
output {"credit":credit, "loan_time": loan_time}

```

2.7 store_bank_loan

```

input {
    "credit":"credit_query.credit",
    "userId":"init.userId",
    "bankId":"init.bankId",
    "loan_type":"init.loan_type",
    "repay_type":"init.repay_type",
    "term":"init.term",
    "d_rate":"init.d_rate",

```

```

        "b_amount": "init.b_amount",
        "b_interest": "init.b_interest",
        "b_total_pay": "init.b_total_pay",
        "b_monthly_pay": "init.b_monthly_pay",
        "b_monthly_decrement": "init.b_monthly_decrement",
        "loan_time": "credit_query.loan_time"
    }
    output {"credit": credit, "loan_time": loan_time}

```

2.8 store_fund_loan

```

    input {
        "credit": "credit_query.credit",
        "userId": "init.userId",
        "loan_type": "init.loan_type",
        "repay_type": "init.repay_type",
        "term": "init.term",
        "f_rate": "init.f_rate",
        "f_amount": "init.f_amount",
        "f_interest": "init.f_interest",
        "f_total_pay": "init.f_total_pay",
        "f_monthly_pay": "init.f_monthly_pay",
        "f_monthly_decrement": "init.f_monthly_decrement",
        "loan_time": "credit_query.loan_time"
    }
    output {}      #output 为空，但需把 input 存数据库

```

2.9 bank_auto_judge

```

    input {
        "userId": "init.userId",
        "bankId": "init.bankId",
        "b_amount": "init.b_amount",
        "b_monthly_pay": "init.b_monthly_pay",
        "credit": "store_bank_loan.credit",
        "loan_time": "store_bank_loan.loan_time"
    }
    output {"result": "ok", "loan_time": loan_time}

```

2.10 fund_auto_judge

```

    input {
        "userId": "init.userId",
        "f_amount": "init.f_amount",
        "credit": "store_fund_loan.credit",
        "loan_time": "store_fund_loan.loan_time"
    }
    output {"result": "ok", "loan_time": loan_time}

```

2.11 manual_judge
get 为读数据库的贷款表

```

    output {}      #output 为空，但需把人工审批结果存数据库

```

2.12 loan_res_confirm
get {b_judge_res, f_judge_res} #读数据库中的人工审批结果

3. 部分代码技术细节介绍

3.1 用户端表单验证

用户端的表单验证采用了 Flask-WTF 扩展。Web 应用程序的一个重要方面是为用户提供用户界面。HTML 提供了一个<form>标签，用于设计界面。可以适当地使用 Form（表单）元素，例如文本输入，单选按钮，选择等。

用户输入的数据以 Http 请求消息的形式通过 GET 或 POST 方法提交给服务器端脚本。服务器端脚本必须从 http 请求数据重新创建表单元素。因此，实际上，表单元素必须定义两

次 - 一次在 HTML 中,另一次在服务器端脚本中。使用 HTML 表单的另一个缺点是很难(如果不是不可能的话)动态呈现表单元素。HTML 本身无法验证用户的输入。

这就是 WTForms 的作用,一个灵活的表单,渲染和验证库,能够方便使用。Flask-WTF 扩展为这个 WTForms 库提供了一个简单的接口。使用 Flask-WTF,我们可以在 Python 脚本中定义表单字段,并使用 HTML 模板进行渲染。还可以将验证应用于 WTF 字段。

以 choose_loan_plan 用户选择贷款计划的代码为例,具体实现如下:

```
loan_local > forms.py > LoanForm
1 from flask_wtf import FlaskForm
2 from wtforms import StringField, IntegerField, TextAreaField, SubmitField, RadioField, SelectField
3 from wtforms import validators, ValidationError
4 from wtforms.validators import DataRequired, NumberRange
5
6 class LoanForm(FlaskForm):
7     loan_type = RadioField('贷款类别', choices = [(1, '公积金贷款'), (2, '混合贷款'), (3, '商业贷款')], validators=[DataRequired('')], coerce=int)
8     repay_type = SelectField('还款方式', choices = [(1, '等额本息'), (2, '等额本金')], coerce=int)
9     term = SelectField('贷款期限', choices = [(5, '5年'), (10, '10年'), (15, '15年'), (25, '25年'), (30, '30年')], coerce=int)
10    f_amount = IntegerField('公积金贷款金额', validators=[NumberRange(0, 99999999)])
11    bankId = SelectField('选择银行', choices = [(1, '中国银行'), (2, '工商银行'), (3, '建设银行')], coerce=int)
12    b_amount = IntegerField('银行贷款金额', validators=[NumberRange(0, 99999999)])
13    submit = SubmitField('计算房贷')
```

```
loan_local > loan_local.py > ...
1 from flask import Flask, redirect, url_for, request, render_template, session
2 import pymysql, time, json, subprocess
3 from urllib import request as rq
4 from forms import LoanForm
5 from loan_calculator import loan_calculator
6
7 app = Flask(__name__)
8 app.secret_key = 'development key'
```

```
@app.route('/choose_loan_plan', methods = ['GET', 'POST'])
def choose_loan_plan():
    loanEXP = session["loanEXP"]
    loan_rec = session["loan_rec"]
    # print(loan_rec)
    loan_rec_dict = json.loads(loan_rec)

    form = LoanForm()
    if form.validate_on_submit():
        loan_type = form.loan_type.data
        repay_type = form.repay_type.data
        term = form.term.data
        f_amount = form.f_amount.data
        bankId = form.bankId.data
        b_amount = form.b_amount.data
        session['loan_type'] = loan_type
        session['repay_type'] = repay_type
        session['term'] = term
        session['f_amount'] = f_amount
        session['bankId'] = bankId
        session['b_amount'] = b_amount
        return redirect(url_for('confirm_loan_plan'))
    else:
        return render_template('choose_loan_plan.html', loan_rec_dict = loan_rec_dict, loanEXP = loanEXP, form = form)
```

3.2 贷款计算器

贷款计算器采用了最基础的数学公式来计算,具体代码如下:

```

loan_local > loan_calculator.py > loan_calculator
1  import math
2
3  def loan_calculator(amount, rate, term, repay_type):    # 传入的rate要乘100 (比如5%传入5)
4      if repay_type == 1:                                # 等额本息
5          up = amount * math.pow((1 + rate / 1200), term * 12)
6          down = 1
7          for i in range(1, term * 12):
8              down = down + math.pow((1 + rate / 1200), i)
9          A = up / down
10         total_pay = round(A * term * 12, 2)
11         monthly_pay = round(A, 2)
12         monthly_decrement = 0
13         loan_cal_dict = {}
14         loan_cal_dict["total_pay"] = total_pay
15         loan_cal_dict["monthly_pay"] = monthly_pay
16         loan_cal_dict["monthly_decrement"] = monthly_decrement
17         return loan_cal_dict
18     elif repay_type == 2:                                # 等额本金
19         A = float(amount) / (term * 12)
20         B = amount * (rate / 1200)
21         C = A * (rate / 1200)
22         D = (B + C) * term * 6
23         total_pay = round(D + amount, 2)
24         monthly_pay = round(A + B, 2)                    # 首月还款
25         monthly_decrement = round(C, 2)                 # 每月递减
26         loan_cal_dict = {}
27         loan_cal_dict["total_pay"] = total_pay
28         loan_cal_dict["monthly_pay"] = monthly_pay
29         loan_cal_dict["monthly_decrement"] = monthly_decrement
30     return loan_cal_dict

```

3.3 数据库连接

MySQL 数据库连接采用 python 的 pymysql 库来实现,采用 pymysql 库可以直接编写 SQL 语句来实现 python 代码对数据库的各种操作,便于小组成员的微服务开发,不需要额外的学习成本。

Flask 自带了 SQLAlchemy 工具包可以对数据库进行操作,该工具包的 ORM API 提供了执行 CRUD 操作的方法,而不必编写原始 SQL 语句。但是在实际的开发过程中发现,SQLAlchemy 工具包需要一定的学习成本,并且并不稳定,偶尔会出现 bug,因此我们小组最终采用的 pymysql 库来连接数据库。

具体的代码示例如下:

```

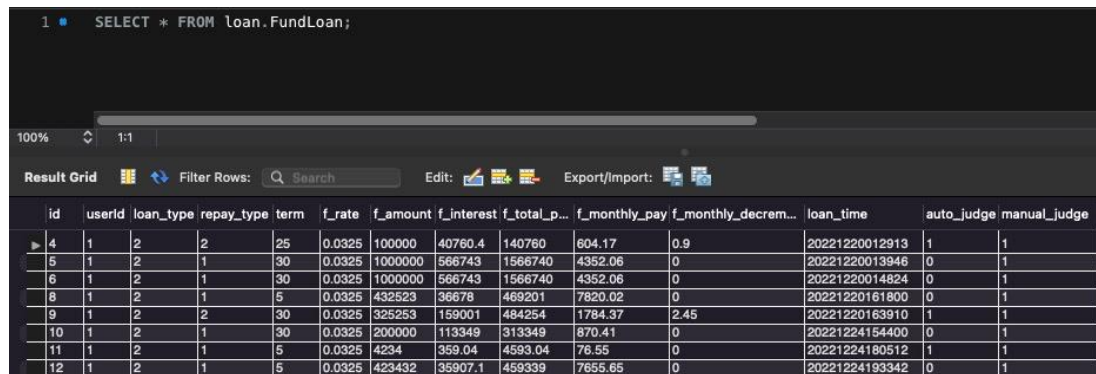
loan_service > loan_service.py > bank_query
10 @app.route('/bank_query', methods = ['POST', 'GET'])
11 def bank_query():
12
13     '''
14     每个servicetask用request.getdata接收到一个json格式的数据
15     '''
16     datajsonstr = request.get_data()
17     print(datajsonstr)
18     input = json.loads(datajsonstr)
19     userId = input["userId"]
20
21     mysql_conn = pymysql.connect(host= '127.0.0.1', port= 3306, user= 'root', password= 'rootroot', db= 'loan')
22
23     sql1 = "SELECT id, rate FROM Bank"
24     try:
25         with mysql_conn.cursor() as cursor:
26             cursor.execute(sql1)
27             Bank_data = cursor.fetchall()
28             # print(Bank_data)
29     except Exception as e:
30         print(e)

```


3.4 贷款申请受理方身份区分

由于用户的贷款申请可能会给公积金管理方和不同商业银行的管理方进行审批,因此有必要对贷款申请受理方的身份进行区分,确保对应的受理方审批到对应的贷款申请。我们小组采用了将贷款申请存入不同受理方数据库表的方式,并给每个贷款申请加上了唯一属性“贷款时间”来区分不同的贷款申请。在某个申请受理方登录系统时读对应的数据库表,可以确保审批到对应的贷款申请。

公积金贷款申请表的部分数据如下图:



	id	userid	loan_type	repay_type	term	f_rate	f_amount	f_interest	f_total_p...	f_monthly_pay	f_monthly_decre...	loan_time	auto_judge	manual_judge
▶	4	1	2	2	25	0.0325	100000	40760.4	140760	604.17	0.9	20221220012913	1	1
	5	1	2	1	30	0.0325	1000000	566743	1566740	4352.06	0	20221220013946	0	1
	6	1	2	1	30	0.0325	1000000	566743	1566740	4352.06	0	20221220014824	0	1
	8	1	2	1	5	0.0325	432523	36678	469201	7820.02	0	20221220161800	0	1
	9	1	2	2	30	0.0325	325253	159001	484254	1784.37	2.45	20221220163910	1	1
	10	1	2	1	30	0.0325	200000	113349	313349	870.41	0	20221224154400	0	1
	11	1	2	1	5	0.0325	4234	359.04	4593.04	76.55	0	20221224180512	1	1
	12	1	2	1	5	0.0325	423432	35907.1	459339	7656.65	0	20221224193342	0	1

3.5 用户前端页面与 bpmn 微服务间的数据传输

若用户端采用 curl 的方式来向 bpmn 微服务传输数据,对用户来说缺乏便捷性。因此我们小组的系统采用 python 的 urllib 库,让用户端的页面向 bpmn 服务器发出 http 请求并获取反馈数据。每次请求首先获取查询数据的 oid,存储该 oid 后再进行一次 http 请求来获取微服务反馈的业务数据。具体代码实现如下:

```
'''
从complete借来的代码,用于usertask向bpmn中连接的servicetask传数据, 传的业务数据放在businessData
'''
headers = {'Content-Type': 'application/json'}
# 如果程序运行过了,需要重新实例化一次bpmn, 每次实例化返回新的oid
Oid=open("/Users/xuwei/Documents/课程/微服务与区块链/大作业/workflow/data/oid.txt",'r').read()
# Oid = resultMap["Oid"]
# print(Oid)
user = "anyone"
taskName = "submit"
processData= "{}"
businessData=json.dumps({"userId":userId, "loanEXP":loanEXP})
map={}
map["taskName"]=taskName
map["processData"]=processData
map["businessData"]=businessData
map["user"]=user
map["fcn"]="complete"
map["Oid"]=Oid
datajsonstr=json.dumps(map)
r=rq.Request(url="http://10.77.70.173:8999/grafana/wfRequest",data=bytes(datajsonstr,"utf-8"),headers=headers)
result = rq.urlopen(r).read().decode('utf-8')
print(result.encode('utf-8'))
resultMap=json.loads(result)
# return resultMap
```

```

'''
获取执行的结果需要执行query.sh,下面的代码相当于实现了query.sh, 获取到servicetask自动流转的最终结果
'''

responseMap = {}
if resultMap["code"]==500:
    responseMap["code"]=500
    responseMap["body"]=resultMap["body"]
    responseMap["state"]="Failed"
    return responseMap
oid1=resultMap["Oid"]
# print(oid1)
getResponseUrl="http://10.77.70.173:8999/grafana/getResByOid/"+oid1
for i in range(50):
    time.sleep(0.2)
    r=rq.Request(url=getResponseUrl)
    res = rq.urlopen(r).read().decode('utf-8')
    print(res)
    if res!="none":
        resMap=json.loads(res)
        if "isEnd" in resMap:
            responseMap["code"]=200
            responseMap["body"]=resMap["businessData"]
            responseMap["state"]="Success"
            # return responseMap
            res_body = responseMap["body"]
            session["loan_rec"] = res_body
            return redirect(url_for('choose_loan_plan'))
responseMap["code"]=500
responseMap["body"]="time out,It may be that the write to blockchain failed"
responseMap["state"]="Failed"
return responseMap

```

服务端的微服务接收数据采用了 Flask 框架自带的 request.getdata() 函数，接收到业务数据后再对数据进行 json 格式的解码，即可使用业务数据。在完成该微服务的业务流程后，再将业务数据编码为 json 格式返回给 bpmn 服务器，以供下一个微服务或用户端使用。具体代码实现如下：

```

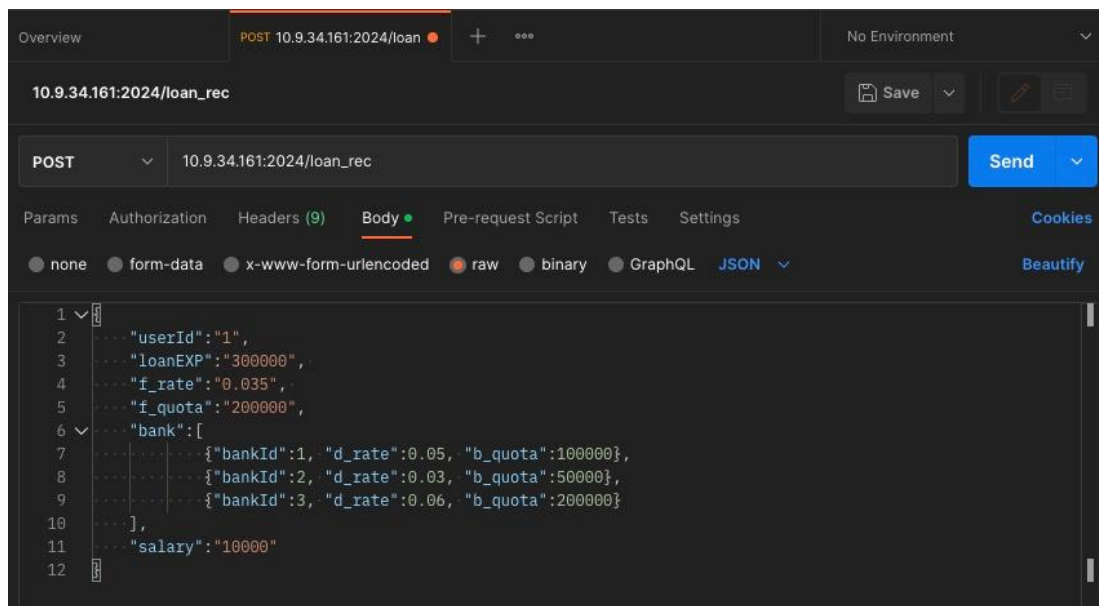
'''
每个servicetask用request.getdata接收到一个json格式的数据
'''

datajsonstr = request.get_data()
print(datajsonstr)
input = json.loads(datajsonstr)
userId = input["userId"]

```

4. 系统测试

我们采用了 Postman 来对微服务进行接口测试，这种测试方式方便快捷，比普通的黑盒测试效率更高。对贷款推荐 loan_rec 微服务的测试过程如下图：



四、成员分工

徐威： bpmn 图制作、微服务数据流编写、本地前端页面编写、系统整合及测试、报告撰写、PPT 制作

方晓坤： bpmn 图制作、数据库设计、征信服务与公积金及商业银行贷款微服务编写、前端页面优化、报告撰写

陈劲： 贷款背景知识调研、数据库设计、个人信息查询与贷款推荐微服务编写、报告撰写