转账问题思考与设计文档

C-2 创建: 党然, 最后修改: 党然 今天 22:28

目录

1、题目描述

代码块

```
题目：请编写【2个用户之间转账功能的接口及其内部实现】的核心代码，尽量是一个可以运行的代码程序，
的测试用例。
要求：完成接口设计、并实现其内部逻辑，以完成A用户转账给B用户的功能。以分布式场景为背景，2个用户
在同一个数据库下。注意尽量不要用伪代码。
提示：请考虑接口规范、安全、幂等、重试、并发、有可能的异常分支、分布式场景下在事务一致性、用户
安全、测试用例设计等的处理。我们将重点评估该部分内容。
        完成后，可书写一份文档，用以描述总结对题目理解、设计思路、场景推演或实现说明等内容。
```

2、题目理解

题意理解

主要功能：两个用户之间的转账功能接口与实现

首先，转账的本身的含义为，一个账户金额减少，另一个账户的金额等值增加，这样的一个交易可以称之为转账。

题目中描述，分布式场景，两个用户的账户不在同一个数据库。

于是有两个方向的理解：

理解1

两个用户的用户类型存在极大的不一致，使得在微服务的架构下，两个账户所对应的领域实体为两个完全不同的领域实体，并且分别由两个微服务进行领域化服务。

在此给出第一个假设：用户A与用户B为两个完全不同的用户类型，假设为商家与用户，则场景转变为 用户A向商家B转账，此时更接近于付款的场景

理解2

两个用户为同一类型的用户，所对应的领域实体为同一实体，但因为表本身体量过大，则进行了分库分表，而两条用户的记录正好分布在两个库的两个表当中。但是从微服务角度来看，用户实体对应了唯一的一个微服务。而转账，则是在同一实体的两条记录之间，进行资金的转移

与面试官确认后，第一种理解为正确理解

3、场景推演

1、转账，本身是一个**高流量场景**，会存在对同一个账户多笔资金汇入的情况，则需要考虑并发问题。

- 微服务下，一个服务多台部署，需要采用**分布式加锁**的形式
- 资金更新时，可以采用**CAS思想**来进一步保证

2、转账为逻辑不同的两个实体账户间转账，在微服务架构下，则为跨服务转账，而涉及资金，需要强一致性，则考虑引入**分布式事务两阶段提交**

3、转账本身为一种交易，交易幂等性可以以**token令牌**的形式来保障，一阶段申请令牌，二阶段根据令牌进行交易

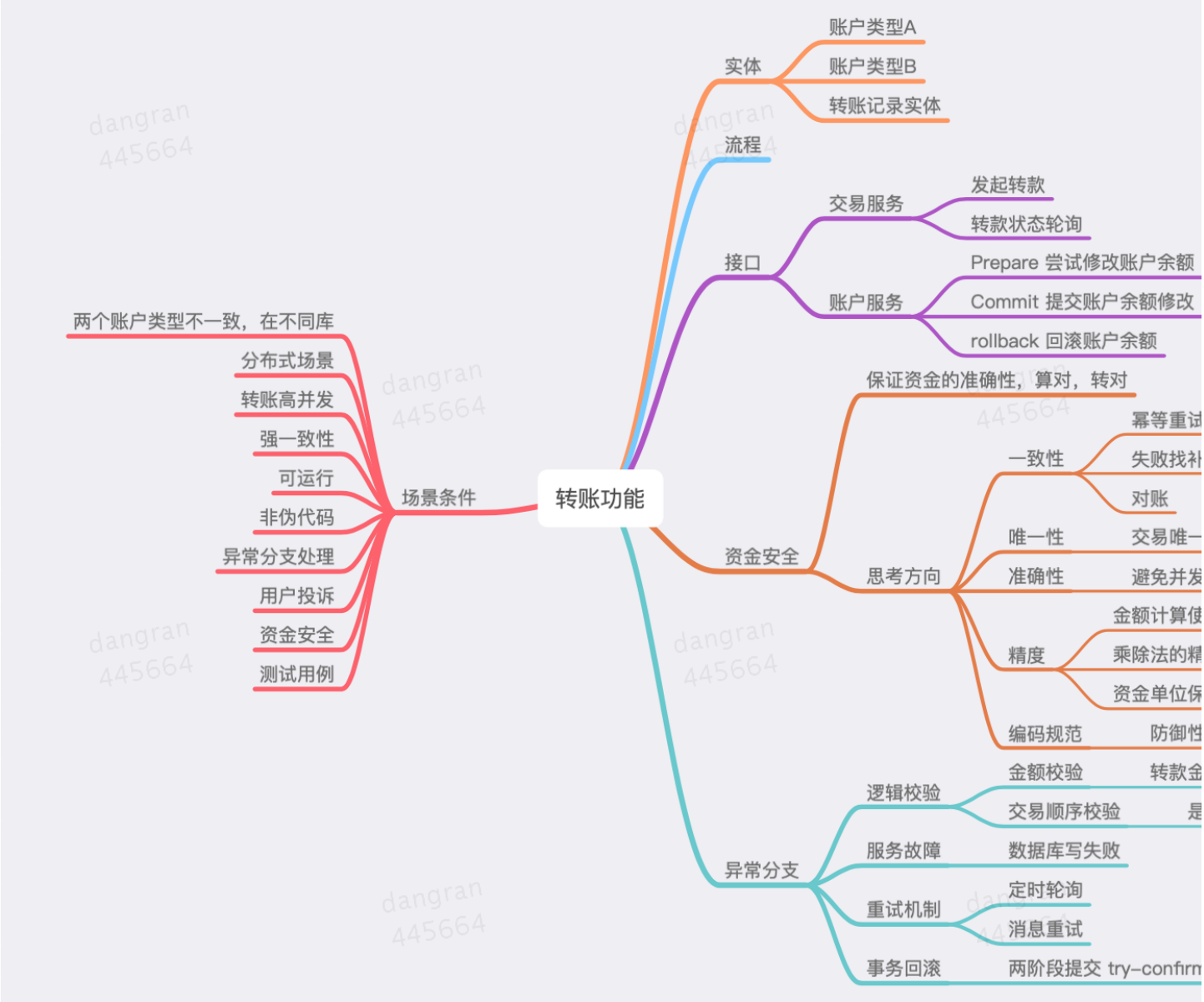
token的生成要全局唯一化，生成后可以考虑存在分布式缓存当中。

4、整个交易过程中，可能出现失败，失败原因多种多样。对于一笔失败中的转账，有多种方式进行处理

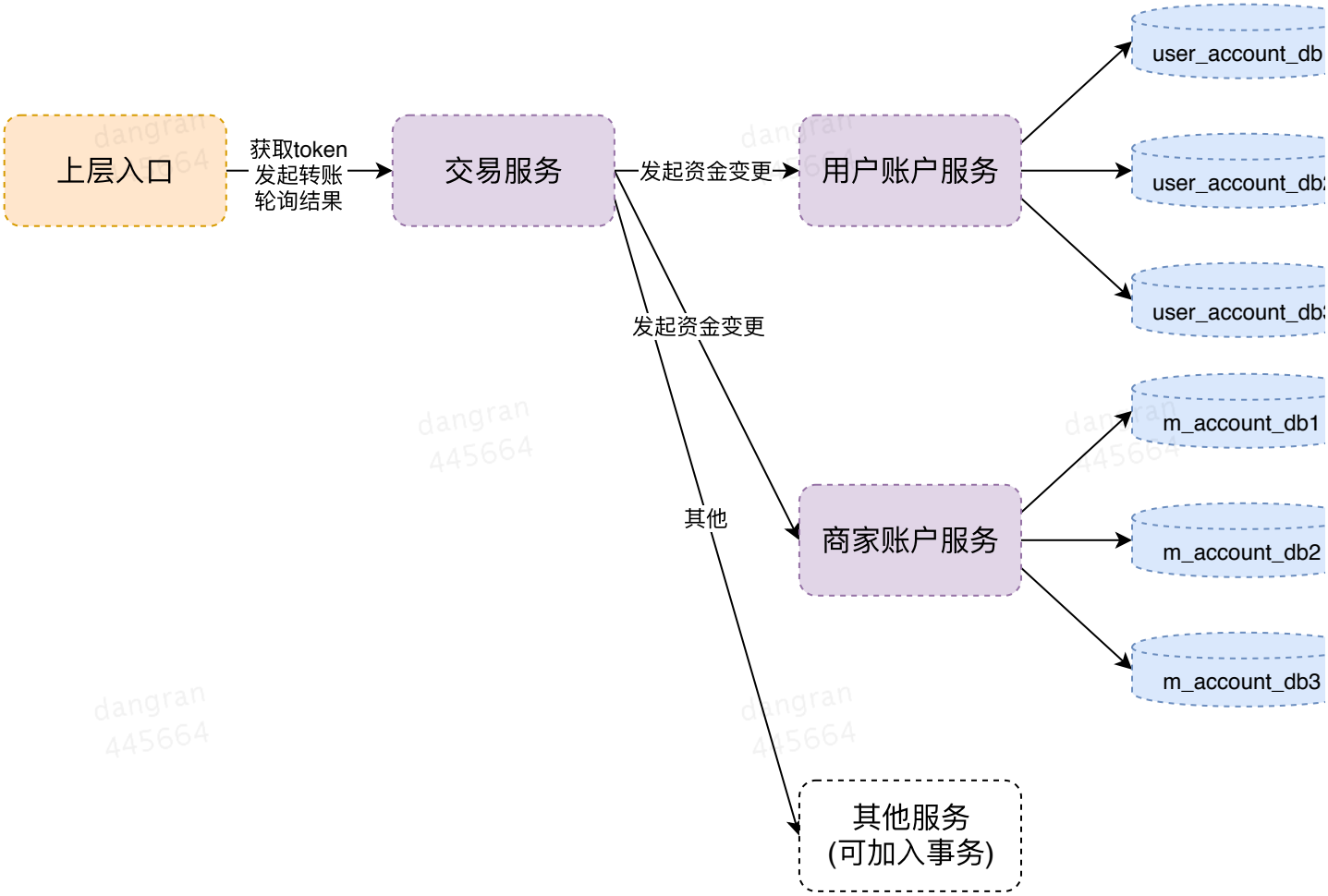
- 不进行重试，将转账失败化，数据回滚，反映给上层，由上层用户决定如何处理
- 保存失败原因，有重试可能的转账记录，定时轮询进行重试，重试次数设定阈值，达到阈值后，向上抛出

故关于重试这个场景，其实我是抱有一定的怀疑，我个人没做过金融领域下的重试策略，但是一笔转账如果出现失败，我认为回滚的优先级是否应当大于重试？因为转账失败的原因各种各样，每次转账失败都进行重试的话，即便有重试阈值，是否会对整个服务的吞吐量造成很大的影响？

4、问题拆解


转账功能
场景条件
两个账户类型不一致，在不同库
分布式场景
转账高并发
强一致性
可运行
非伪代码
异常分支处理
用户投诉
资金安全
测试用例
实体
账户类型A
账户类型B
转账记录实体
流程
接口
交易服务
发起转款
转款状态轮询
账户服务
Prepare 尝试修改账户余额
Commit 提交账户余额修改
rollback 回滚账户余额
资金安全
保证资金的准确性，算对，转对
思考方向
一致性
幂等重试
失败找补
对账
唯一性
交易唯一
准确性
避免并发
精度
金额计算使
乘除法的精
资金单位保
编码规范
防御性
异常分支
逻辑校验
金额校验
转款金
交易顺序校验
是
服务故障
数据库写失败
重试机制
定时轮询
消息重试
事务回滚
两阶段提交 try-confirm

5、系统设计



6、实体表结构设计

代码块 SQL

```sql
-- User账户服务
CREATE TABLE `user_account` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `account_no` int(11) NOT NULL COMMENT '账户唯一逻辑标识',
  `balance` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '账户余额',
  `frozen_amount` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '账户冻结金额(两步交)',
  `version` int(11) NOT NULL DEFAULT '0' COMMENT '账户版本(每次更改++)',
  `create_time` int(11) NOT NULL,
  `update_time` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `account_no` (`account_no`)
) ENGINE=InnoDB;
CREATE TABLE `user_account_transaction` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `trade_no` varchar(32) NOT NULL DEFAULT '' COMMENT '唯一交易识别号',
  `account_no` int(11) NOT NULL COMMENT '账户id',
```

```
  `origin_balance` decimal(10,2) NOT NULL COMMENT '变动前余额',
  `amount` decimal(10,2) NOT NULL COMMENT '转账金额',
  `type` tinyint(3) NOT NULL COMMENT '操作类型(0加钱,1减钱)',
  `status` tinyint(3) NOT NULL COMMENT '是否提交 0未提交 1已提交 2已回滚',
  `create_time` int(11) NOT NULL COMMENT '操作时间',
  PRIMARY KEY (`id`),
  KEY `transaction_no` (`trade_no`),
  KEY `account_no` (`account_no`)
) ENGINE=InnoDB;

-- Merchant账户服务
CREATE TABLE `merchant_account` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `account_no` int(11) NOT NULL COMMENT '账户唯一逻辑标识',
  `balance` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '账户余额',
  `frozen_amount` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '账户冻结金额(两步交)',
  `version` int(11) NOT NULL DEFAULT '0' COMMENT '账户版本(每次更改++)',
  `create_time` int(11) NOT NULL,
  `update_time` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `account_no` (`account_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE TABLE `merchant_account_transaction` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `trade_no` varchar(32) NOT NULL DEFAULT '' COMMENT '唯一交易识别号',
  `account_no` int(11) NOT NULL COMMENT '账户id',
  `origin_balance` decimal(10,2) NOT NULL COMMENT '变动前余额',
  `amount` decimal(10,2) NOT NULL COMMENT '转账金额',
  `type` tinyint(3) NOT NULL COMMENT '操作类型(0加钱,1减钱)',
  `status` tinyint(3) NOT NULL COMMENT '是否提交 0未提交 1已提交 2已回滚',
  `create_time` int(11) NOT NULL COMMENT '操作时间',
  PRIMARY KEY (`id`),
  KEY `transaction_no` (`trade_no`),
  KEY `account_no` (`account_no`)
) ENGINE=InnoDB;


-- trade 系统
CREATE TABLE `transfer_record` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `transaction_no` varchar(32) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT '' C
'转账操作唯一凭证(交易识别号)',
  `from_account_no` int(11) NOT NULL COMMENT '出账账户id',
```

```
  `from_account_type` tinyint(3) NOT NULL COMMENT '0用户 1商家',
  `to_account_no` int(11) NOT NULL COMMENT '入账账户id',
  `to_account_type` tinyint(11) DEFAULT NULL COMMENT '0用户 1商家',
  `amount` decimal(10,2) NOT NULL COMMENT '转账金额',
  `status` tinyint(3) NOT NULL COMMENT '转账状态 0初始 1成功 2失败',
  `fail_code` tinyint(3) DEFAULT NULL COMMENT '错误码',
  `credit_time` int(11) DEFAULT NULL COMMENT '转账完成时间',
  `comments` int(11) DEFAULT NULL COMMENT '备注',
  `create_time` int(11) NOT NULL,
  `update_time` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `transaction_no` (`transaction_no`)
) ENGINE=InnoDB;
```

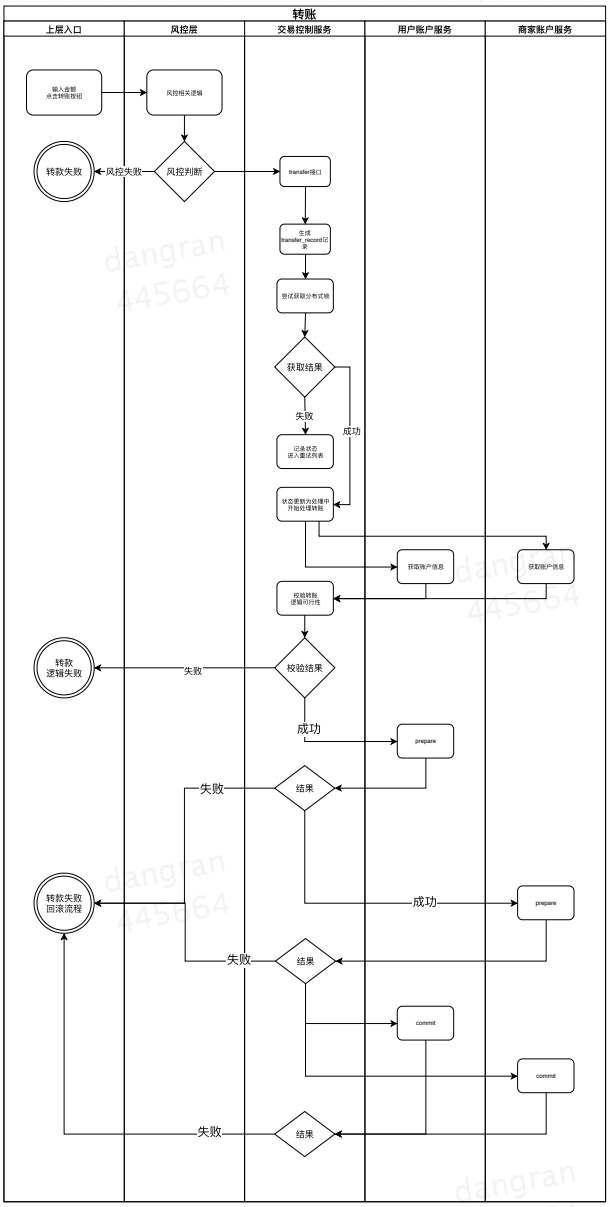
7、接口设计

代码块 Java

```java
//trade系统
public interface TransferService {

    //获取交易token
    public String requestToken(TransferRequestParamDTO transferRequestParamDTO)

    //触发转款
    public TransferResultDTO transfer(TransferRequestParamDTO
transferRequestParamDTO);

    //轮询转款结果
    public TransferResultDTO pollTransferResult(String transactionNo);
}


//account系统(化简实现，两个account系统逻辑相似)
public interface MerchantAccountService {

    //第一阶段提交，准备数据，资金冻结，资金变更记录
    public boolean prepareMerchantAccountOperate(TransactionContext
transactionContext, MerchantAccountOperateParamDTO accountOperateParamDTO);

    //第二阶段提交，数据提交，余额清算完成，资金记录状态流转
    public boolean commit(TransactionContext transactionContext,
MerchantAccountOperateParamDTO accountOperateParamDTO);
```

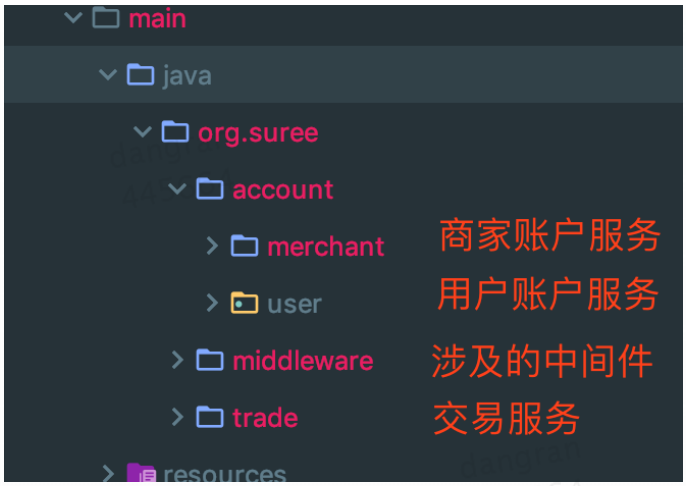
```
23
24      //异常处理，回滚资金状态到最初，资金记录状态异常
25      public boolean rollback(TransactionContext transactionContext,
    MerchantAccountOperateParamDTO accountOperateParamDTO);
26
27      //账户查询
28      public MerchantAccount getMerchantAccountByAccountNo(Integer accountNo);
29  }
```

8、交互流程



9、工程结构说明

因为时间关系，模拟实现的三个服务，放在了一个工程里



因为时间关系，部分细节没有全部完成，故工程本身没法像线上服务一样可部署起来

其中，user与merchant只是题目条件中设定为两个领域服务，但是主要考察点不在领域服务本身，而在于跨服务之间的数据一致性问题，故接口定义、表结构设计与实现大致逻辑基本一致

user服务的完成度更高

各个服务中的package命名规则目前我尽量以我在美团的代码规范形式来进行编码(阿里的编码规范可能还需要额外的学习)

整个实现当中，因为时间原因，省略了关于日志、数据库、单测、安全加密、风控等场景的编码

其中关于中间件当中对分布式事务中间件的一些考虑与简易实现，可以在TransactionContext的注释中见到