可信跨链服务

开发指南

文档版本 01

发布日期 2021-12-30

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

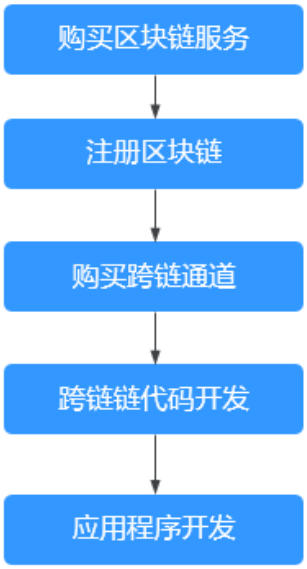
1 简介.....1
2 跨链链代码开发（Hyperledger Fabric）.....3
2.1 开发前准备.....3
2.1.1 开发环境准备.....3
2.1.2 下载源码包.....4
2.2 开发跨链智能合约.....4
2.2.1 跨链资产数据锁定义.....4
2.2.2 跨链智能合约方法定义.....5
2.2.3 跨链智能合约方法示例.....6
2.2.3.1 交易发起方预提交（preCommitSend）.....6
2.2.3.2 交易接收方预提交（preCommitRecv）.....7
2.2.3.3 交易发起方提交（commitSend）.....8
2.2.3.4 交易接收方提交（commitRecv）.....8
2.2.3.5 交易发起方回滚（rollbackSend）.....9
2.2.3.6 交易接收方回滚（rollbackRecv）.....10
2.2.3.7 修改跨链资产数值（putStateWithLock）.....10
2.2.3.8 解锁跨链资产（unlockAccount）.....11
2.2.3.9 回滚跨链资产（rollback）.....12
3 跨链交易.....13
3.1 跨链触发交易.....13
3.2 跨链查询交易.....15
3.3 跨链查账本交易.....18

1 简介

在使用可信跨链服务时，您需要开发自己的链代码和应用。本文档主要介绍跨链业务链代码的开发，专供具备Go/Java开发经验的开发人员使用。

说明

当前仅“华北-北京四”区域支持可信跨链服务。



1. 购买区块链服务

   您可通过华为云区块链服务Hyperledger Fabric增强版实例，具体请参考**基于CCE集群部署**。

2. 注册区块链

   将区块链注册到TCS中，具体请参考**注册区块链**。

3. 购买跨链通道

   TCS将为您提供端到端全流程可信的跨链数据互通体验，具体请参考**购买跨链通道**。

4. 跨链链代码开发

如您的跨链业务不涉及跨链资产交换，则无需定制编写跨链智能合约，否则需按可信跨链服务要求编写跨链智能合约，具体请参考**跨链链代码开发（Hyperledger Fabric）**。

5. 应用程序开发

   可信跨链服务将为您提供一系列跨链互操作涉及的API接口，您需要自行开发应用组织这些接口的调用逻辑。您也可使用接口调试功能调用上述接口，具体请参考**触发跨链交易**。

   说明

   如果您对业务链代码和客户端APP的设计和开发有需求，可以联系华为云区块链合作伙伴提供进一步服务，我们会结合您的业务以及华为云的优势和特点为您提供完善的解决方案，联系邮箱如下：sales@huaweicloud.com

2 跨链链代码开发（Hyperledger Fabric）

2.1 开发前准备

链代码（Chaincode）又称智能合约，在Hyperledger Fabric中是用Go、Java或Node.js语言编写的程序，主要用于操作账本上的数据。链代码是运行在区块链上的、特定条件下自动执行的代码逻辑，是用户利用区块链实现业务逻辑的重要途径。基于区块链特点，智能合约的运行结果是可信的，其结果是无法被伪造和篡改的。

在面向Hyperledger Fabric使用可信数据链接服务时，用户需要结合Hyperledger Fabric链代码开发规范、可信跨链服务的要求与自身业务需求开发自己的跨链链代码。

2.1.1 开发环境准备

请根据自身业务选择Go（推荐）或其他语言的开发环境。

Go开发环境准备：

1. 安装Go开发环境。安装包下载地址为：**https://golang.org/dl/** 。（请选择1.9.2之后的版本）

   各个系统对应的包名（以1.14版本为例）

操作系统	包名
Windows	go1.14.windows-amd64.msi
Linux	go1.14.linux-amd64.tar.gz

   - Windows下您可以使用.msi后缀的安装包来安装。默认情况下.msi文件会安装在“C:\Go”目录下。你可以将“C:\Go\bin”目录添加到Path环境变量中。添加后您需要重启命令窗口才能生效。
   - Linux下，您需要将下载的二进制包解压至“/usr/local”目录。将“/usr/local/go/bin”目录添加至Path环境变量：

     ```
     export PATH=$PATH:/usr/local/go/bin
     ```

2. 安装Go编辑器。编辑器可自行选择，推荐使用Goland：**https://www.jetbrains.com/go/download**。

2.1.2 下载源码包

下载Fabric源码包作为三方库。可选择使用1.x或2.x风格开发跨链智能合约：

版本	链接
1.x	https://github.com/hyperledger/fabric/tree/release-1.4
2.x	https://github.com/hyperledger/fabric-contract-api-go/tree/v1.1.1

2.2 开发跨链智能合约

开发跨链智能合约前，需要先了解跨链操作的原理。在跨链资产交换场景中，在减少某个链上资产后，需要相应的在对应链上增加资产，这种转移使各条链的资产发生了变化。因此，跨链操作需要保证整个跨链交易结束后不同链之间的全局事务保持一致性，即同时记账，或同时不记账。

可信跨链服务基于分布式事务两阶段提交的思想设计了一套能确保全局事务保持一致性的跨链资产交换流程。若您的跨链业务不涉及跨链资产交换，则无需定制编写跨链智能合约，否则需设置跨链资产数据锁与跨链智能合约方法。

下面用一个完整的跨链资产交换智能合约为样例说明跨链智能合约开发流程，该样例可完成链A上的A账户与链B上的B账户之间的资产转账。完整智能合约示例获取方法：登录可信跨链服务管理控制台，在“总览”页面的跨链链代码下载用于演示的业务链代码tcsexample.zip。

2.2.1 跨链资产数据锁定义

跨链资产交换基于分布式事务的两阶段提交实现，两阶段提交必须基于每个跨链资产单元携带的数据锁。跨链资产数据锁定义如下：

```
type AccountLock struct {
    PreValue string
    CrossTXID string
}
```

表 2-1 跨链资产数据锁字段说明

字段	说明
PreValue	记录跨链交易开始前被锁定资产的数值，回滚时使用。
CrossTXID	记录跨链资产数据锁所属的交易ID，回滚时使用

另外，一般在智能合约中以资产Key拼接数据锁后缀作为资产数据锁的Key，如Key为A的资产数据锁的Key为A_Lock。因此，可在智能合约中定义该后缀常量，便于后续在其他智能合约中使用：

```
const (
    lockSuffix = "_Lock"
)
```

2.2.2 跨链智能合约方法定义

下表为涉及跨链资产交换的跨链智能合约必须实现的方法，且必须保证方法名相同，否则可能出现跨链资产交换接口调用超时或调用失败的情况：

表 2-2 方法说明

方法名	说明
preCommitSend	在跨链资产交换发起方所属区块链上执行的预提交操作
preCommitRecv	在跨链资产交换接收方所属区块链上执行的预提交操作
commitSend	在跨链资产交换发起方所属区块链上执行的提交操作
commitRecv	在跨链资产交换接收方所属区块链上执行的提交操作
rollbackSend	在跨链资产交换发起方所属区块链上执行的回滚操作
rollbackRecv	在跨链资产交换接收方所属区块链上执行的回滚操作

可在跨链智能合约按以下示例定义方法名常量与Invoke方法，确保已实现了必选方法：

```
/*
 * Function name list of the TCS example chaincode
 * Note: the chaincode for TCS should have all of the functions below with the same function names
 */
const (
    fromPreCommitFuncName = "preCommitSend"
    toPreCommitFuncName   = "preCommitRecv"
    fromCommitFuncName    = "commitSend"
    toCommitFuncName      = "commitRecv"
    fromRollbackFuncName  = "rollbackSend"
    toRollbackFuncName    = "rollbackRecv"
)

/*
 * Invoke is the entrance of the chaincode invoking
 */
func (t *TCSExampleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    fmt.Println("TCS Example Function Invoke")
    function, args := stub.GetFunctionAndParameters()
    if function == fromPreCommitFuncName {
        return t.preCommitSend(stub, args)
    } else if function == toPreCommitFuncName {
        return t.preCommitRecv(stub, args)
    } else if function == fromCommitFuncName {
        return t.commitSend(stub, args)
    } else if function == toCommitFuncName {
        return t.commitRecv(stub, args)
```



```
    } else if function == fromRollbackFuncName {
        return t.rollbackSend(stub, args)
    } else if function == toRollbackFuncName {
        return t.rollbackRecv(stub, args)
    } else if function == "query" {
        return t.query(stub, args)
    } else if function == "init" {
        return t.Init(stub)
    }
    return shim.Error("Invalid invoke function name. Expecting \"preCommitSend\" \"preCommitRev\" " +
        "\"commitSend\" \"commitRecv\" \"rollbackSend\" \"rollbackRecv\" \"query\", but got: " + function)
}
```

2.2.3 跨链智能合约方法示例

介绍跨链智能合约方法示例。

2.2.3.1 交易发起方预提交（preCommitSend）

该方法用于在跨链资产交换发起方所属区块链上执行预提交操作，即修改发起方所属区块链上对应资产的值，并对该资产上锁。

本例中，该方法将修改发起方所属区块链上args[0]对应账户的余额为跨链资产交换完成后的数值，同时对该账户上锁，并在数据锁中保存跨链资产交换发生前该账户的余额。putStateWithLock方法的实现请参考**修改跨链资产数值（putStateWithLock）**。

说明

该方法为必选方法，需在智能合约中以相同命名定义该方法，否则将导致跨链资产交换失败。

```
/*
 * preCommitSend is the first step of two-phase commit cross-chain transaction process, it will be executed on the client side and lock the related assets/data
 * In this example, this function transfer X units from client's local
 * account(first arg) to remote account(second arg) and lock the local account
 * The example args[] is {"a","b","1","txid123"}
 * @Param args[0]: The name of the account on blockchain A that will transfer the amount of units to the account on blockchain B
 * @Param args[1]: The name of the account on blockchain B that will receive the amount of units from the account on blockchainA
 * @Param args[2]: The amount of unit that will be transferred from args[0](account on blockchain A) to args[1](account on blockchain B)
 * @Param args[3]: The id of this transaction that transfer units of one account to another account
 */
func (t *TCSExampleChaincode) preCommitSend(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments. Expecting 4")
    }
        txID := args[3]
    // Get account balance from blockchain
    account := args[0]
    balanceBytes, err := stub.GetState(account)
    if err != nil {
        return shim.Error(err.Error())
    }
    if balanceBytes == nil {
        return shim.Error("account not found: " + account)
    }
    balance, err := strconv.Atoi(string(balanceBytes))
    if err != nil {
        return shim.Error(err.Error())
    }
    // Get transfer amount from args
    amount, err := strconv.Atoi(args[2])
```

```
    if err != nil {
        return shim.Error(err.Error())
    }
    // Execute the business process
    balance = balance - amount
    // Use specific function to put state for cross-chain transaction
    err = putStateWithLock(stub, txID,account, []byte(strconv.Itoa(balance)))
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(nil)
}
```

2.2.3.2 交易接收方预提交（preCommitRecv）

该方法用于在跨链资产交换接收方所属区块链上执行预提交操作，即修改接收方所属区块链上对应资产的值，并对该资产上锁。

本例中，该方法将修改接收方所属区块链上args[1]对应账户的余额为跨链资产交换完成后的数值，同时对该账户上锁，并在数据锁中保存跨链资产交换发生前该账户的余额。putStateWithLock方法的实现请参考**修改跨链资产数值（putStateWithLock）**。

说明

该方法为必选方法，需在智能合约中以相同命名定义该方法，否则将导致跨链资产交换失败。

```
/*
 * preCommitRecv is the second step of two-phase cross-chain transaction process, it will be executed on
the server side and lock the related assets/data
 * In this example, this function will make local account(second arg) received
 * X units transferred by remote account(first arg) and lock the local account
 * The example args[] is {"a","b","1","txid123"}
 * @Param args[0]: The name of the account on blockchain A that will transfer the amount of units to the
account on blockchain B
 * @Param args[1]: The name of the account on blockchain B that will receive the amount of units from the
account on blockchainA
 * @Param args[2]: The amount of unit that will be transferred from args[0](account on blockchain A) to
args[1](account on blockchain B)
 * @Param args[3]: The id of this transaction that transfer units of one account to another account
 */
func (t *TCSExampleChaincode) preCommitRecv(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments. Expecting 4")
    }
        txID := args[3]
    // Get account balance from blockchain
    account := args[1]
    balanceBytes, err := stub.GetState(account)
    if err != nil {
        return shim.Error(err.Error())
    }
    if balanceBytes == nil {
        return shim.Error("account not found: " + account)
    }
    balance, err := strconv.Atoi(string(balanceBytes))
    if err != nil {
        return shim.Error(err.Error())
    }
    // Get transfer amount from args
    amount, err := strconv.Atoi(args[2])
    if err != nil {
        return shim.Error(err.Error())
    }
    // Execute the business process
    balance = balance + amount
```

```
    // Use specific function to put state for cross-chain transaction
    err = putStateWithLock(stub, txID, account, []byte(strconv.Itoa(balance)))
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(nil)
}
```

2.2.3.3 交易发起方提交（commitSend）

该方法用于在跨链资产交换发起方所属区块链上执行提交操作，即解锁发起方所属区块链上对应资产，使得其可以继续处理下一笔跨链操作。

本例中，该方法将删除发起方所属区块链上args[0]对应的数据锁，代表该笔跨链资产交换操作在发起方已端到端完成。unlockAccount方法的实现请参考**解锁跨链资产（unlockAccount）**。

说明

该方法为必选方法，需在智能合约中以相同命名定义该方法，否则将导致跨链资产交换失败。

```
/*
 * commitSend is the third step of two-phase cross-chain transaction process, it will be executed on the client side and unlock the related assets/data
 * In this example, this function will unlock server side's local account(second arg),
 * so that it can be used by next cross-chain transaction
 * The example args[] is {"a","b","1","txid123"}
 * @Param args[0]: The name of the account on blockchain A that will transfer the amount of units to the account on blockchain B
 * @Param args[1]: The name of the account on blockchain B that will receive the amount of units from the account on blockchainA
 * @Param args[2]: The amount of unit that will be transferred from args[0](account on blockchain A) to args[1](account on blockchain B)
 * @Param args[3]: The id of this transaction that transfer units of one account to another account
 */
func (t *TCSExampleChaincode) commitSend(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments. Expecting 4")
    }
        txID := args[3]
    // unlock client side's local account
    account := args[0]
    err := unlockAccount(stub, txID, account)
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(nil)
}
```

2.2.3.4 交易接收方提交（commitRecv）

该方法用于在跨链资产交换接收方所属区块链上执行提交操作，即解锁接收方所属区块链上对应资产，使得其可以继续处理下一笔跨链操作。

本例中，该方法将删除接收方所属区块链上args[0]对应的数据锁，代表该笔跨链资产交换操作在接收方已端到端完成。unlockAccount方法的实现请参考**解锁跨链资产（unlockAccount）**。

说明

该方法为必选方法，需在智能合约中以相同命名定义该方法，否则将导致跨链资产交换失败。

```
/*
 * commitRecv is the forth step of two-phase cross-chain transaction process, it will be executed on the server side and unlock the related assets/data
```

```
 * In this example, this function will unlock server side's local account(second arg),
 * so that it can be used by next cross-chain transaction
 * The example args[] is {"a","b","1","txid123"}
 * @Param args[0]: The name of the account on blockchain A that will transfer the amount of units to the
account on blockchain B
 * @Param args[1]: The name of the account on blockchain B that will receive the amount of units from the
account on blockchainA
 * @Param args[2]: The amount of unit that will be transferred from args[0](account on blockchain A) to
args[1](account on blockchain B)
 * @Param args[3]: The id of this transaction that transfer units of one account to another account
 */
func (t *TCSExampleChaincode) commitRecv(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 3 {
        return shim.Error("Incorrect number of arguments. Expecting 3")
    }
        txID := args[3]
    // unlock server side's local account
    account := args[1]
    err := unlockAccount(stub, txID, account)
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(nil)
}
```

2.2.3.5 交易发起方回滚（rollbackSend）

该方法用于在跨链资产交换发起方所属区块链上执行回滚操作，即还原发起方所属区块链上对应资产至跨链资产交换开始前的状态，并解锁该资产，使得其可以继续处理下一笔跨链操作。

本例中，该方法将根据发起方所属区块链上args[0]对应数据锁中的PreValue进行资产回滚，并删除该数据锁，使发起方所属区块链上args[0]对应的资产回滚至跨链资产交换开始前的状态。

说明

该方法为必选方法，需在智能合约中以相同命名定义该方法，否则将导致跨链资产交换失败。

```
/*
 * rollbackSend is the rollback function for client side, it will be executed when error happened during the
cross-chain tx process
 * In this example, this function will recover the balance of client side's local account and unlock it,
 * so that it can be used by next cross-chain transaction
 * The example args[] is {"a","b","1","txid123"}
 * @Param args[0]: The name of the account on blockchain A that will transfer the amount of units to the
account on blockchain B
 * @Param args[1]: The name of the account on blockchain B that will receive the amount of units from the
account on blockchainA
 * @Param args[2]: The amount of unit that will be transferred from args[0](account on blockchain A) to
args[1](account on blockchain B)
 * @Param args[3]: The id of this transaction that transfer units of one account to another account
 */
func (t *TCSExampleChaincode) rollbackSend(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 3 {
        return shim.Error("Incorrect number of arguments. Expecting 3")
    }
        txID := args[3]
    account := args[0]
    err := rollback(stub, txID, account)
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(nil)
}
```

2.2.3.6 交易接收方回滚（rollbackRecv）

该方法用于在跨链资产交换接收方所属区块链上执行回滚操作，即还原接收方所属区块链上对应资产至跨链资产交换开始前的状态，并解锁该资产，使得其可以继续处理下一笔跨链操作。

本例中，该方法将根据接收方所属区块链上args[1]对应数据锁中的PreValue进行资产回滚，并删除该数据锁，使接收方所属区块链上args[1]对应的资产回滚至跨链资产交换开始前的状态。

说明

该方法为必选方法，需在智能合约中以相同命名定义该方法，否则将导致跨链资产交换失败。

```
/*
 * rollbackRecv is the rollback function for server side, it will be executed when error happened during the cross-chain tx process
 * In this example, this function will recover the balance of server side's local account and unlock it,
 * so that it can be used by next cross-chain tx
 * The example args[] is {"a","b","1","txid123"}
 * @Param args[0]: The name of the account on blockchain A that will transfer the amount of units to the account on blockchain B
 * @Param args[1]: The name of the account on blockchain B that will receive the amount of units from the account on blockchainA
 * @Param args[2]: The amount of unit that will be transferred from args[0](account on blockchain A) to args[1](account on blockchain B)
 * @Param args[3]: The id of this transaction that transfer units of one account to another account
 */
func (t *TCSExampleChaincode) rollbackRecv(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 3 {
        return shim.Error("Incorrect number of arguments. Expecting 3")
    }
    account := args[1]
    err := rollback(stub, txID, account)
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(nil)
}
```

2.2.3.7 修改跨链资产数值（putStateWithLock）

在跨链资产交换涉及的智能合约方法中，所有对跨链资产的修改都必须与资产上锁同时进行。可将上述逻辑封装至一个方法中，便于后续在其他智能合约方法（主要是preCommitSend与preCommitRecv）中调用：

```
/*
 * putStateWithLock will update the account balance and lock the account
 * @Param account: The name of the account that whose balance is going to be updated with lock
 * @Param balance: The amount of units that will be put to the account
 * @Param txID: The id of this transaction that transfer units of one account to another account
 */
func putStateWithLock(stub shim.ChaincodeStubInterface, txID string, account string, balance []byte) error {
    accountBytes, err := stub.GetState(account)
    if err != nil {
        return fmt.Errorf("failed to get account state: %v", err)
    }
    if accountBytes == nil {
        accountBytes = []byte("")
    }
    accountLockKey := account + lockSuffix
    accountLockBytes, err := stub.GetState(accountLockKey)
    if err != nil {
        return fmt.Errorf("failed to get accountLock state: %v", err)
    }
    accountLock := AccountLock{}
```

```
/*
 * Detect the account lock's status
 * if account is not locked:
 * 1. put lock to the blockchain ledger with current account balance
 * 2. update the account balance
 * If account is locked:
 * 1. is locked by the same txID, then return success
 * 2. is not locked by this txID,then not allowed to update account balance
 */
if accountLockBytes == nil {
    accountLock = AccountLock{
        PreValue: string(accountBytes),
        CrossTXID: txID,
    }
    accountLockBytes, err = json.Marshal(accountLock)
    if err != nil {
        return fmt.Errorf("failed to marshal accountLockBytes: %v", err)
    }
    err = stub.PutState(accountLockKey, accountLockBytes)
    if err != nil {
        return fmt.Errorf("failed to put accountLock state: %v", err)
    }
    err = stub.PutState(account, balance)
    if err != nil {
        return fmt.Errorf("failed to put account state: %v", err)
    }
    return nil
} else {
    err = json.Unmarshal(accountLockBytes, &accountLock)
    if err != nil {
        return fmt.Errorf("failed to unmarshal accountLockBytes: %v", err)
    }
    if accountLock.CrossTXID == txID {
        return nil
    }
    return fmt.Errorf("account %s locked", account)
  }
}
```

2.2.3.8 解锁跨链资产（unlockAccount）

在跨链资产交换即将完成时，需要解锁跨链资产交换中涉及的资产。可将上述逻辑封装至一个方法中，便于后续在其他智能合约方法（主要是commitSend、commitRecv、rollbackSend与rollbackRecv）中调用：

```
/*
 * unlockAccount will delete the account's lock from the blockchain
 * @Param account: The name of the account whose lock will be unlocked
 * @Param txID: The id of this transaction that transfer units of one account to another account
 */
func unlockAccount(stub shim.ChaincodeStubInterface, txID string, account string) error {
    // get account lock state with account lock's key
    accountLockKey := account + lockSuffix
    accountLockBytes, err := stub.GetState(accountLockKey)
    if err != nil {
        return fmt.Errorf("failed to get accountLock state: %v", err)
    }
    if accountLockBytes == nil {
        // if the account lock state does not exist, no need to execute unlock
        return nil
    } else {
        accountLock := &AccountLock{}
        err = json.Unmarshal(accountLockBytes, &accountLock)
        if err != nil {
            return fmt.Errorf("failed to unmarshal accountLockBytes: %v", err)
        }
        if accountLock.CrossTXID != txID {
```

```
            return fmt.Errorf("not the owner of the lock")
        }
        err = stub.DelState(accountLockKey)
        if err != nil {
            return fmt.Errorf("failed to delete state: %v", err)
        }
    }
    return nil
}
```

2.2.3.9 回滚跨链资产（rollback）

在跨链资产交换的过程中如遇到异常情况，需要读取跨链资产对应数据锁中的PreValue，并根据该值回滚跨链资产交换中涉及资产已发生的变化。可将上述逻辑封装至一个方法中，便于后续在其他智能合约方法（主要是rollbackSend与rollbackRecv）中调用：

```
/*
 * rollback function will recover the account balance to the previous status
 * @Param account: The name of the account whose balance will be rollback
 * @Param txID: The id of this transaction that transfer units of one account to another account
 */
func rollback(stub shim.ChaincodeStubInterface, txID string, account string) error {
    // get server side's local account lock(with preValue inside)
    accountLockKey := account + lockSuffix
    accountLockBytes, err := stub.GetState(accountLockKey)
    if err != nil {
        return fmt.Errorf("failed to get accountLock state: %v", err)
    }
    if accountLockBytes == nil {
        return nil
    }
    accountLock := AccountLock{}
    err = json.Unmarshal(accountLockBytes, &accountLock)
    if err != nil {
        return fmt.Errorf("failed to unmarshal accountLockBytes: %v", err)
    }
    // recover the balance of server side's local account to the previous balance
    err = stub.PutState(account, []byte(accountLock.PreValue))
    if err != nil {
        return fmt.Errorf("failed to put state of preValue account: %v", err)
    }
    // unlock server side's local account
    err = unlockAccount(stub, txID, account)
    if err != nil {
        return fmt.Errorf("failed to unlock in rollbackRev: %v", err)
    }
    return nil
}
```

3 跨链交易

3.1 跨链触发交易

功能介绍

发起由源端区块链到目标端区块链的跨链交易

URI

POST /v1/cross/transaction/invoke

请求参数

表 3-1 请求 Body 参数

参数	是否必选	参数类型	描述
from_chaincode_id	是	String	源端链码名称 最小长度：**1** 最大长度：**64**
args	是	Array of strings	链码交易函数所需的参数列表
to_chain	是	String	目标区块链ID 最小长度：**1** 最大长度：**64**
to_chaincode_id	是	String	目标端链码名称 最小长度：**1** 最大长度：**64**

响应参数

状态码： 200

表 3-2 响应 Body 参数

参数	参数类型	描述
code	String	成功响应码
data	**Data** object	响应数据

表 3-3 Data

参数	参数类型	描述
message	String	响应信息

状态码： 400

表 3-4 响应 Body 参数

参数	参数类型	描述
error_code	String	错误码
error_msg	String	错误信息

状态码： 500

表 3-5 响应 Body 参数

参数	参数类型	描述
error_code	String	错误码
error_msg	String	错误信息

请求示例

```
{
  "from_chaincode_id" : "tcsexample",
  "args" : [ "a", "b", "1" ],
  "to_chain" : "cexxaxef-1475-11xx-b225-0255xx10043x",
  "to_chaincode_id" : "tcsexample"
}
```

响应示例

状态码： 200

请求成功

```
{
  "code" : "TCS.2000000",
  "data" : {
    "message" : "SUCCESS!"
  }
}
```

状态码： 400

请求不合法

```
{
  "error_code" : "TCS.4000001",
  "error_message" : "Invalid request args"
}
```

状态码： 500

请求失败

```
{
  "error_code" : "TCS.5000002",
  "error_message" : "Failed to send cross tx : Failed to sendPreCrossTransaction: Failed to preCrossRequest: Failed to Generate PreCrossRequest: Failed to get VerificationResponse from relay: rpc error: code = Unknown desc = Permission not granted: %!s(<nil>) "
}
```

状态码

状态码	描述
200	请求成功
400	请求不合法
500	请求失败

3.2 跨链查询交易

功能介绍

发起由源端区块链到目标端区块链的跨链查询交易

URI

POST /v1/cross/transaction/query

请求参数

表 3-6 请求 Body 参数

参数	是否必选	参数类型	描述
to_chain	是	String	目标区块链ID 最小长度：**1** 最大长度：**64**
from_chaincode_id	否	String	源端链码名称 最小长度：**1** 最大长度：**64**
to_chaincode_id	是	String	目标端链码名称 最小长度：**1** 最大长度：**64**
to_query_func_name	是	String	链码中的查询函数 最小长度：**1** 最大长度：**64**
args	是	Array of strings	链码查询函数所需的参数列表

响应参数

状态码： 200

表 3-7 响应 Body 参数

参数	参数类型	描述
code	String	成功响应码
data	**DataPlayload** object	响应链上数据

表 3-8 DataPlayload

参数	参数类型	描述
message	String	响应信息
payload	String	链上信息

状态码： 400

表 3-9 响应 Body 参数

参数	参数类型	描述
error_code	String	错误码
error_msg	String	错误信息

状态码： 500

表 3-10 响应 Body 参数

参数	参数类型	描述
error_code	String	错误码
error_msg	String	错误信息

请求示例

```
{
  "to_chain" : "cexxaxef-1475-11xx-b225-0255xx10043x",
  "from_chaincode_id" : "tcsexample",
  "to_chaincode_id" : "tcsexample",
  "to_query_func_name" : "query",
  "args" : [ "a" ]
}
```

响应示例

状态码： 200

请求成功

```
{
  "code" : "TCS.2000000",
  "data" : {
    "message" : "SUCCESS!",
    "payload" : "937"
  }
}
```

状态码： 400

请求不合法

```
{
  "error_code" : "TCS.4000001",
  "error_message" : "Invalid request args"
}
```

状态码： 500

请求失败

```
{
  "error_code" : "TCS.5000002",
  "error_message" : "Failed to send cross tx : Failed to sendPreCrossTransaction: Failed to preCrossRequest: Failed to Generate PreCrossRequest: Failed to get VerificationResponse from relay: rpc error: code =
```



```
Unknown desc = Permission not granted: %!s(<nil>) "
}
```

状态码

状态码	描述
200	请求成功
400	请求不合法
500	请求失败

3.3 跨链查账本交易

功能介绍

跨链查询目标区块链的账本信息

URI

POST /v1/cross/ledger/query

请求参数

表 3-11 请求 Body 参数

参数	是否必选	参数类型	描述
from_chaincode_id	是	String	源端链码名称 最小长度：**1** 最大长度：**64**
to_chain	是	String	目标端区块链ID 最小长度：**1** 最大长度：**64**
to_query_func_name	是	String	目标端查询函数，如:QueryBlock 最小长度：**1** 最大长度：**64**
args	是	Array of strings	查询账本所需要的参数列表。为数字字符串形式，如：[“1”]。

响应参数

状态码： 200

表 3-12 响应 Body 参数

参数	参数类型	描述
code	String	成功响应码
data	**DataPlayload** object	响应链上数据

表 3-13 DataPlayload

参数	参数类型	描述
message	String	响应信息
payload	String	链上信息

状态码： 400

表 3-14 响应 Body 参数

参数	参数类型	描述
error_code	String	错误码
error_msg	String	错误信息

状态码： 500

表 3-15 响应 Body 参数

参数	参数类型	描述
error_code	String	错误码
error_msg	String	错误信息

请求示例

```
{
  "from_chaincode_id" : "tcsexample",
  "to_chain" : "cexxaxef-1475-11xx-b225-0255xx10043x",
  "to_query_func_name" : "QueryBlock",
  "args" : [ "1" ]
}
```

响应示例

状态码： 200

请求成功

```
{
  "code" : "TCS.2000000",
  "data" : {
    "message" : "SUCCESS!",
    "payload" : "... -----BEGIN CERTIFICATE-----\n ... \n-----END CERTIFICATE-----\n ... "
  }
}
```

状态码： 400

请求不合法

```
{
  "error_code" : "TCS.4000001",
  "error_message" : "Invalid request args"
}
```

状态码： 500

请求失败

```
{
  "error_code" : "TCS.5000002",
  "error_message" : "Failed to send cross tx : Failed to sendPreCrossTransaction: Failed to preCrossRequest: Failed to Generate PreCrossRequest: Failed to get VerificationResponse from relay: rpc error: code = Unknown desc = Permission not granted: %!s(<nil>) "
}
```

状态码

状态码	描述
200	请求成功
400	请求不合法
500	请求失败

