1. 服务连接

```
//创建服务代理
public static BlockchainKeyPair CLIENT_CERT = BlockchainKeyGenerator.getInstance().generate();
final String GATEWAY_IP = "127.0.0.1";
final int GATEWAY_PORT = 80;
final boolean SECURE = false;
GatewayServiceFactory serviceFactory = GatewayServiceFactory.connect(GATEWAY_IP, GATEWAY_PORT, SECURE,
        CLIENT_CERT);
// 创建服务代理;
BlockchainService service = serviceFactory.getBlockchainService();
```

2. 用户注册

```
// 创建服务代理;
BlockchainService service = serviceFactory.getBlockchainService();
// 在本地定义注册账号的 TX;
TransactionTemplate txTemp = service.newTransaction(ledgerHash);
SignatureFunction signatureFunction = asymmetricCryptography.getSignatureFunction(CryptoAlgorithm.ED25519);
CryptoKeyPair cryptoKeyPair = signatureFunction.generateKeyPair();
BlockchainKeyPair user = new BlockchainKeyPair(cryptoKeyPair.getPubKey(), cryptoKeyPair.getPrivKey());

txTemp.users().register(user.getIdentity());

// TX 准备就绪;
PreparedTransaction prepTx = txTemp.prepare();
// 使用私钥进行签名;
CryptoKeyPair keyPair = getSponsorKey();
prepTx.sign(keyPair);

// 提交交易;
prepTx.commit();
```

3. 数据账户注册

```
// 创建服务代理;
BlockchainService service = serviceFactory.getBlockchainService();
// 在本地定义注册账号的 TX;
TransactionTemplate txTemp = service.newTransaction(ledgerHash);
SignatureFunction signatureFunction = asymmetricCryptography.getSignatureFunction(CryptoAlgorithm.ED25519);
CryptoKeyPair cryptoKeyPair = signatureFunction.generateKeyPair();
BlockchainKeyPair dataAccount = new BlockchainKeyPair(cryptoKeyPair.getPubKey(), cryptoKeyPair.getPrivKey());

txTemp.dataAccounts().register(dataAccount.getIdentity());

// TX 准备就绪;
PreparedTransaction prepTx = txTemp.prepare();
// 使用私钥进行签名;
CryptoKeyPair keyPair = getSponsorKey();
prepTx.sign(keyPair);

// 提交交易;
prepTx.commit();
```

4. 写入数据

```
// 创建服务代理；
BlockchainService service = serviceFactory.getBlockchainService();

HashDigest ledgerHash = getLedgerHash();
// 在本地定义注册账号的 TX；
TransactionTemplate txTemp = service.newTransaction(ledgerHash);

// ------------------------------------
// 将商品信息写入到指定的账户中；
// 对象将被序列化为 JSON 形式存储，并基于 JSON 结构建立查询索引；
String commodityDataAccount = "GGhhreGeasdfasfUUfehf9932lkae99ds66jf==";
Commodity commodity1 = new Commodity();
txTemp.dataAccount(commodityDataAccount).set("ASSET_CODE", commodity1.getCode().getBytes(), -1);

// TX 准备就绪；
PreparedTransaction prepTx = txTemp.prepare();

String txHash = ByteArray.toBase64(prepTx.getHash().toBytes());
// 使用私钥进行签名；
CryptoKeyPair keyPair = getSponsorKey();
prepTx.sign(keyPair);

// 提交交易；
prepTx.commit();
```

5. 查询数据

> 注：详细的查询可参考模块sdk-samples中SDK_GateWay_Query_Test_相关测试用例

```
// 创建服务代理；
BlockchainService service = serviceFactory.getBlockchainService();

// 查询区块信息；
// 区块高度；
long ledgerNumber = service.getLedger(LEDGER_HASH).getLatestBlockHeight();
// 最新区块；
LedgerBlock latestBlock = service.getBlock(LEDGER_HASH, ledgerNumber);
// 区块中的交易的数量；
long txCount = service.getTransactionCount(LEDGER_HASH, latestBlock.getHash());
// 获取交易列表；
LedgerTransaction[] txList = service.getTransactions(LEDGER_HASH, ledgerNumber, 0, 100);
// 遍历交易列表
for (LedgerTransaction ledgerTransaction : txList) {
    TransactionContent txContent = ledgerTransaction.getTransactionContent();
    Operation[] operations = txContent.getOperations();
    if (operations != null && operations.length > 0) {
        for (Operation operation : operations) {
            operation = ClientOperationUtil.read(operation);
            // 操作类型：数据账户注册操作
            if (operation instanceof  DataAccountRegisterOperation) {
                DataAccountRegisterOperation daro = (DataAccountRegisterOperation) operation;
                BlockchainIdentity blockchainIdentity = daro.getAccountID();
            }
            // 操作类型：用户注册操作
            else if (operation instanceof UserRegisterOperation) {
                UserRegisterOperation uro = (UserRegisterOperation) operation;
                BlockchainIdentity blockchainIdentity = uro.getUserID();
            }
            // 操作类型：账本注册操作
            else if (operation instanceof LedgerInitOperation) {

                LedgerInitOperation ledgerInitOperation = (LedgerInitOperation)operation;
                LedgerInitSetting ledgerInitSetting = ledgerInitOperation.getInitSetting();

                ParticipantNode[] participantNodes = ledgerInitSetting.getConsensusParticipants();
            }
            // 操作类型：合约发布操作
```

```
            else if (operation instanceof ContractCodeDeployOperation) {
                ContractCodeDeployOperation ccdo = (ContractCodeDeployOperation) operation;
                BlockchainIdentity blockchainIdentity = ccdo.getContractID();
            }
            // 操作类型：合约执行操作
            else if (operation instanceof ContractEventSendOperation) {
                ContractEventSendOperation ceso = (ContractEventSendOperation) operation;
            }
            // 操作类型：KV存储操作
            else if (operation instanceof DataAccountKVSetOperation) {
                DataAccountKVSetOperation.KVWriteEntry[] kvWriteEntries =
                        ((DataAccountKVSetOperation) operation).getWriteSet();
                if (kvWriteEntries != null && kvWriteEntries.length > 0) {
                    for (DataAccountKVSetOperation.KVWriteEntry kvWriteEntry : kvWriteEntries) {
                        BytesValue bytesValue = kvWriteEntry.getValue();
                        DataType dataType = bytesValue.getType();
                        Object showVal = ClientOperationUtil.readValueByBytesValue(bytesValue);
                        System.out.println("writeSet.key=" + kvWriteEntry.getKey());
                        System.out.println("writeSet.value=" + showVal);
                        System.out.println("writeSet.type=" + dataType);
                        System.out.println("writeSet.version=" + kvWriteEntry.getExpectedVersion());
                    }
                }
            }
        }
    }
}

// 根据交易的 hash 获得交易；注：客户端生成 PrepareTransaction 时得到交易hash；
HashDigest txHash = txList[0].getTransactionContent().getHash();
Transaction tx = service.getTransactionByContentHash(LEDGER_HASH, txHash);
// 获取数据；
String commerceAccount = "GGhhreGeasdfasfUUfehf9932lkae99ds66jf==";
String[] objKeys = new String[] { "x001", "x002" };
KVDataEntry[] kvData = service.getDataEntries(LEDGER_HASH, commerceAccount, objKeys);

long payloadVersion = kvData[0].getVersion();

// 获取数据账户下所有的KV列表
KVDataEntry[] kvData = service.getDataEntries(ledgerHash, commerceAccount, 0, 100);
if (kvData != null && kvData.length > 0) {
    for (KVDataEntry kvDatum : kvData) {
        System.out.println("kvData.key=" + kvDatum.getKey());
        System.out.println("kvData.version=" + kvDatum.getVersion());
        System.out.println("kvData.type=" + kvDatum.getType());
        System.out.println("kvData.value=" + kvDatum.getValue());
    }
}
```

6. 合约发布

```
// 创建服务代理；
BlockchainService service = serviceFactory.getBlockchainService();

// 在本地定义TX模板
TransactionTemplate txTemp = service.newTransaction(ledgerHash);

// 合约内容读取
byte[] contractBytes = FileUtils.readBytes(new File(CONTRACT_FILE));

// 生成用户
BlockchainIdentityData blockchainIdentity = new BlockchainIdentityData(getSponsorKey().getPubKey());

// 发布合约
txTemp.contracts().deploy(blockchainIdentity, contractBytes);

// TX 准备就绪；
PreparedTransaction prepTx = txTemp.prepare();
```

```
// 使用私钥进行签名；
CryptoKeyPair keyPair = getSponsorKey();

prepTx.sign(keyPair);

// 提交交易；
TransactionResponse transactionResponse = prepTx.commit();

assertTrue(transactionResponse.isSuccess());

// 打印合约地址
System.out.println(blockchainIdentity.getAddress().toBase58());
```

7. 合约执行

```
// 创建服务代理；
BlockchainService service = serviceFactory.getBlockchainService();

// 在本地定义TX模板
TransactionTemplate txTemp = service.newTransaction(ledgerHash);

// 合约地址
String contractAddressBase58 = "";

// 使用接口方式调用合约
TransferContract transferContract = txTpl.contract(contractAddress, TransferContract.class);

// 使用decode方式调用合约内部方法（create方法）
// 返回GenericValueHolder可通过get方法获取结果，但get方法需要在commit调用后执行
GenericValueHolder<String> result = ContractReturnValue.decode(transferContract.create(address, account, money));

PreparedTransaction ptx = txTpl.prepare();

ptx.sign(adminKey);

TransactionResponse transactionResponse = ptx.commit();

String cotractExecResult = result.get();

// TransactionResponse也提供了可供查询结果的接口
OperationResult[] operationResults = transactionResponse.getOperationResults();

// 通过OperationResult获取结果
for (int i = 0; i < operationResults.length; i++) {
    OperationResult opResult = operationResults[i];
    System.out.printf("Operation[%s].result = %s \r\n",
            opResult.getIndex(), BytesValueEncoding.decode(opResult.getResult()));
}
```