jdchain-start快速使用说明

1. 简介

本工程已经构建了基本的开发环境，意在快速帮助用户开发、编译、发布和执行合约。用户在contract-compile模块下，首先根据com.jd.blockchain.contract包下的合约样例，编写适合自己的新合约。然后执行mvn clean package进行编译，在输出路径中找到对应的xxx.jar合约文件。然后将此合约压缩包在测试链上部署并执行。

快速使用样例下载地址：https://github.com/blockchain-jd-com/jdchain-starter.git ，在此工程的doc/JDChain-Start快速使用指南.md也包含此文件。

1.1. 准备

用户可直接通过maven中央库来获取jar。

1.1.1 基于SDK“数据快速上链”方式的maven坐标

```
<dependency>
    <groupId>com.jd.blockchain</groupId>
    <artifactId>sdk-client</artifactId>
    <version>1.0.1.RELEASE</version>
</dependency>
```

#####1.1.2 基于“快速开发合约代码”方式的maven坐标

```
<dependency>
    <groupId>com.jd.blockchain</groupId>
    <artifactId>sdk-pack</artifactId>
    <version>1.0.1.RELEASE</version>
</dependency>
```

2. 数据快速上链

2.1. 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();
```

2.2. 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();
```

2.3. 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();
```

2.4. 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();
```

2.5. 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());
    }
}
```

2.6. 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());
```

2.7. 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()));
    }
```

3. 快速开发合约代码

3.1. 开发

入门样例可参照：com.jd.blockchain.contract.TransferContractImpl，合约类实现EventProcessingAw ire接口和其声明的接口，同时在合约接口的执行方法上添加注解：@ContractEvent(name = "xxx")，形参自行定义，目前支持String、int、long和实现了@DataContract注解的类型。

格式如下：

```
//接口
@Contract
public interface TransferContract {

    @ContractEvent(name = "create")
    String create(String address, String account, long money);
}

//实现;
public class TransferContractImpl implements EventProcessingAware, TransferContract {
    private ContractEventContext eventContext;
    private HashDigest ledgerHash;

    @Override
    public String create(String address, String account, long money) {
        //......
    }
}
```

合约中可以通过ContractEventContext对象来调用账本中的相关方法，比如：

```
KVDataEntry[] kvDataEntries = eventContext.getLedger().getDataEntries(ledgerHash, address, account);
// 肯定有返回值，但若不存在则返回version=-1
if (kvDataEntries != null && kvDataEntries.length > 0) {
    long currVersion = kvDataEntries[0].getVersion();
    if (currVersion > -1) {
        throw new IllegalStateException(String.format("%s -> %s already have created !!!", address, account));
    }
    eventContext.getLedger().dataAccount(address).setInt64(account, money, -1L);
} else {
    throw new IllegalStateException(String.format("Ledger[%s] inner Error !!!", ledgerHash.toBase58()));
}
```

3.2. 编译

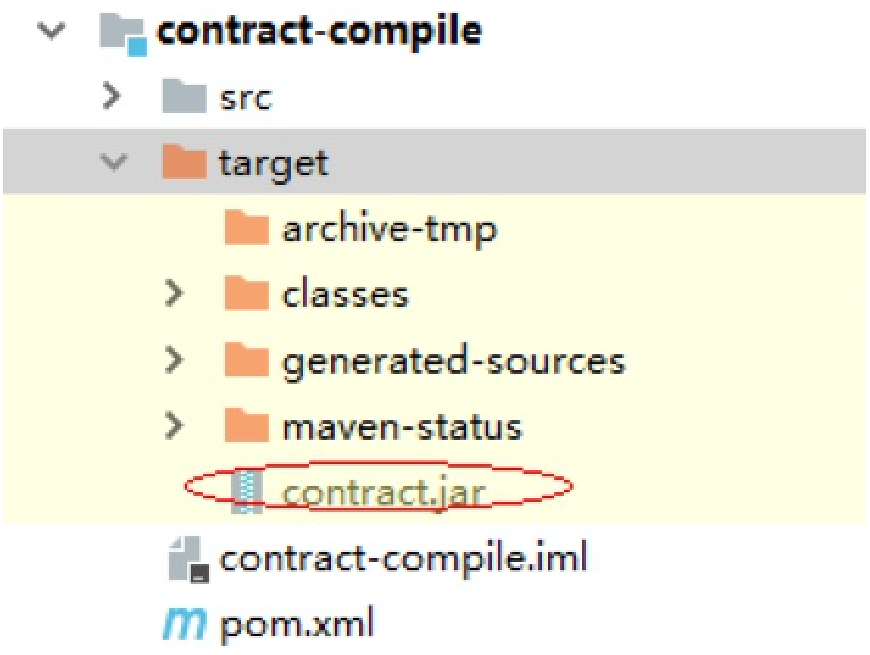
编译工作在contract-compile工程中进行。在控制台中执行：mvn clean package来直接编译生成所需的合约压缩包。

注意修改pom.xml文件中和这两个属性：

```
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
        <finalName>contract</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <archive>
            <manifest>
                <mainClass>com.jd.blockchain.contract.TransferContractImpl</mainClass>
            </manifest>
        </archive>
......
```

3.3. 发布与执行

编译完成之后，根据如上的配置，会在target目录下生成对应的合约压缩文件：contract.jar。
将此contract.jar复制至contract-starter模块中src/resources目录下。



```
contract-compile
  src
  target
    archive-tmp
    classes
    generated-sources
    maven-status
    contract.jar
  contract-compile.iml
  pom.xml
```

3.3.1. 配置服务器信息

修改SDKDemo_Constant.java文件中对应的GW_IPADDR/GW_PORT/PUB_KEYS/PRIV_KEYS/PASSWORD为实际值；

```
public class SDKDemo_Constant {
    public static final String GW_IPADDR = "127.0.0.1";
    public static final int GW_PORT = 11000;
    public static final String[] PUB_KEYS = {
            "3snPdw7i7PjVKiTH2VnXZu5H8QmNaSXpnk4ei533jFpuifyjS5zzH9"};

    public static final String[] PRIV_KEYS = {
            "177gjzHTznYdPgWqZrH43W3yp37onm74wYXT4v9FukpCHBrhRysBBZh7Pzdo5AMRyQGJD7x"};
    public static final String PASSWORD = "abc";
```

3.3.2. 发布并执行合约

执行contract-starter/src/.../SDK_Contract_Demo.java类中的main()方法：

```
public class SDK_Contract_Demo extends SDK_Base_Demo {
    public static void main(String[] args) {
        new SDK_Contract_Demo().executeContract();
    }
}
```

在执行完成之后，可以在api接口中查询相关的信息。

3.3.3. 根据合约地址查询

http://192.168.151.45:7080/ledgers/6Gw3cK4uazegy4HjoaM81ck9NgYLNoKyBMb7a1TK1jt3d/contracts/5SmDBPXfXoSsJmfJskkpeVbZkxfkawqWE9CE

结果：

```
{
"data": {
"address": "5SmDBPXfXoSsJmfJskkpeVbZkxfkawqWE9CE",
"chainCode": "UEsDBBQACAgIAACAlE0AAAAAAAAAAAAAAAAJAAQATUVUQS1JTkYv/soAAAMAUEsHCAAAAAACAAAAAAAAAFBLAwQUAAgICAAAgJRNAAAAAAAA
"chaincodeProof": {
"dataHash": {
"value": "6LwUAHD8zZbANkdJL8bBHpYM2BYduZpQXzaV5T92XxyLL"
},
"dataNode": {
"key": "CHAIN-CODE",
"level": 0,
"nodeHash": {
"value": "6LwUAHD8zZbANkdJL8bBHpYM2BYduZpQXzaV5T92XxyLL"
},
"sN": 0,
"version": 0
},
"levels": 1,
"rootHash": {
"value": "65WEJwZWpdpv5A8BWDqsJQxW33qBR16q4Bx8tzUTbSUWv"
},
"sN": 0
},
"chaincodeVersion": 0,
"pubKey": {
"value": "mayaPm1C7opEhSyRtmH7rkbmxAX2MvrzUguFkgcJ7mpfQY"
},
"rootHash": {
"value": "65WEJwZWpdpv5A8BWDqsJQxW33qBR16q4Bx8tzUTbSUWv"
}
},
"success": true
}
```