jdchain-start快速使用说明

# 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 compile进行编译，在输出路径中找到对应的xxx.contract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：<http://source.jd.com/app/jdchain-starter.git>

## 准备

需要maven配置好setting.xml文件。如果下载有问题，可能需要将原先本地库中的com/jd/blockchain/中相关异常目录删除，再更新。

### artifactory.jd.com

采用artifactory.jd.com私服的配置如下（黄色标记部分需要根据实际位置修改）。

<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository>e:/.m2/repository</localRepository>

<profiles>

<profile>

<repositories>

<repository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>libs-releases</name>

<url>http://artifactory.jd.com/libs-releases</url>

</repository>

<repository>

<snapshots />

<id>snapshots</id>

<name>libs-snapshots</name>

<url>http://artifactory.jd.com/libs-snapshots-local</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>plugins-releases</name>

<url>http://artifactory.jd.com/plugins-releases</url>

</pluginRepository>

<pluginRepository>

<snapshots />

<id>snapshots</id>

<name>plugins-snapshots</name>

<url>http://artifactory.jd.com/plugins-snapshots-local</url>

</pluginRepository>

</pluginRepositories>

<id>artifactory</id>

</profile>

</profiles>

<activeProfiles>

<activeProfile>artifactory</activeProfile>

</activeProfiles>

</settings>

### 使用nexus构建私库

<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository>e:/.m2/repository</localRepository>

<profiles>

<profile>

<id>dev</id>

<repositories>

<repository>

<id>nexus</id>

<url>http://192.168.151.39:7890/content/groups/public/</url>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>nexus</id>

<url>http://192.168.151.39:7890/content/groups/public/</url>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</profile>

</profiles>

<activeProfiles>

<activeProfile>dev</activeProfile>

</activeProfiles>

</settings>

# 步骤

## 开发

入门样例可参照：com.jd.blockchain.contract.AssetContract3，合约类实现EventProcessingAwire接口，同时在合约的入口方法上添加注解：@ContractEvent(name = "xxx")，形参为：ContractEventContext eventContext。

格式如下：

public class AssetContract3 implements EventProcessingAwire{

@ContractEvent(name = "xxx")  
public void test1(ContractEventContext eventContext){

}

}

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

*BlockchainAccount holderAccount = eventContext.getLedger().getAccount(currentLedgerHash(), assetHolderAddress);*

## 编译

样例工程中内嵌了编译插件，需要在pom.xml中预先配置后测试链及其相关参数，这样就可以通过：mvn clean compile来直接编译生成所需的合约压缩包。

Pom.xml文件中相关参数：

<sParam>${project.build.sourceDirectory}</sParam>

<oParam>${project.build.directory}/contract/</oParam>

<cpParam>lib/share</cpParam>

<eParam>utf-8</eParam>

<host>192.168.151.45</host>

<port>8081</port>

<ownerPubPath>${project.basedir}/conf/jd-com.pub</ownerPubPath>

<ownerPrvPath>${project.basedir}/conf/jd-com.priv</ownerPrvPath>

<ownerPassword>${project.basedir}/conf/ownerPassword.txt</ownerPassword>

<ledgerHash>6FEQDTQMnBGANpfX4haXuWHKHw5cZ6P1h2ocqwckYBxp4</ledgerHash>

<!--变化频繁的参数-->

<cParam>com.jd.blockchain.contract.AssetContract3</cParam>

<chainCodePath>${oParam}AssetContract3.contract</chainCodePath>

<event>issue-asset</event>

<contractArgs>10##20##5Sm9VDWAGZ8q3g8Zy2moiT29aufe5xogngPH</contractArgs>

目前测试链自身参数基本不变化，重点关注黄色标记下方区域的参数。

<cParam>：源文件的位置；

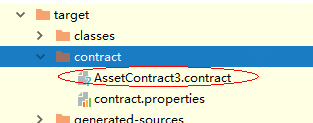
<chainCodePath>：合约压缩文件存放位置；

<event>：调用的合约事件名（@ContractEvent注解对应的名字）；

<contractArgs>：合约方法携带的形参。

## 发布

编译完成之后，根据如上的配置，会在target-》contract目录下生成对应的合约压缩文件：AssetContract3.contract。



运行IntegrationTest.java中的one\_deploy\_exe\_contract\_on\_test\_gateway()方法来发布和执行合约，在执行之前，需要配置好测试资源中的sys-contract.properties文件。

### sys-contract.properties

#常规使用;

ownerPubPath=xxx/conf/jd-com.pub

ownerPrvPath=xxx/conf/jd-com.priv

ownerPassword=xxx/conf/ownerPassword.txt

ledgerHash=6Gw3cK4uazegy4HjoaM81ck9NgYLNoKyBMb7a1TK1jt3d

host=192.168.151.45

port=8081

#合约使用如下;

event = issue-asset

chainCodePath=xxx/AssetContract3.contract

contractArgs=10##4##abc

### 发布和执行方法

/\*\*

\* 在测试链上仅发布和执行合约;

\*/

private void deploy\_exe\_contract\_on\_test\_gateway(){

//then exe the contract;

//由于合约发布之后需要后台进行共识处理，需要一定的时间消耗，先休息5秒钟之后再执行;

try {

Thread.sleep(1000L);

boolean deployResult = ContractDeployExeUtil.instance.deploy(host, port, ledger,ownerPubPath, ownerPrvPath, ownerPassword, chainCodePath,contractPub);

System.out.println("deployResult="+deployResult);

Thread.sleep(2000L);

boolean exeResult = false;

exeResult = ContractDeployExeUtil.instance.exeContract(ledger,ownerPubPath, ownerPrvPath, ownerPassword,eventName,contractArgs);

System.out.println("execute the contract,result= "+exeResult);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

//发布完成之后，会在控制台中生成如下信息：

contract's address=5SmDBPXfXoSsJmfJskkpeVbZkxfkawqWE9CE

deployResult=true

## 执行

执行即如上的方法：

ContractDeployExeUtil.instance.exeContract(ledger,ownerPubPath, ownerPrvPath, ownerPassword,eventName,contractArgs);

在执行完成之后，可以在api接口中查询相关的信息。

### 根据合约地址查询

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

结果：

{

"data": {

"address": "5SmDBPXfXoSsJmfJskkpeVbZkxfkawqWE9CE",

"chainCode": "",

"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

}