Java SDK 1.4.0 使用说明

1. 开发简介

蚂蚁区块链合约平台 Java SDK 是通过Service的形式对外提供了功能，BaseService包含服务接口。SDK提供了同步或异步方式发送交易、查询交易、订阅事件等接口。无论以同步或异步的方式发送交易，SDK封装了发送交易后查询收据的逻辑，这样方便了业务开发者查看交易的执行结果。

2. 应用示例

2.1 版本说明

1. Java SDK版本说明：

2. netty依赖包说明（SDK压缩包中包含）：

文件	用途	说明
netty-tcnative-openssl-static-2.0.17-Final-mychain-all.jar	centos/mac/windows x64 操作系统下sdk所依赖的运行库	支持k1曲线/RSA
netty-tcnative-boringssl-static-2.0.17-Final.jar	centos/mac/windows x64 操作系统下sdk所依赖的运行库	支持r1曲线/RSA

3. 运行环境说明：

- JDK 7 及以上版本
  在终端输入 `java -version` 查看当前java版本。
- maven 3.5.4及以上版本
  在终端输入 `mvn -v` 查看当前maven版本。
- Linux下使用sdk，要求GLIBC version > 2.14

4. maven引入SDK包：

- 安装下载的jar到本地仓库
  从命令终端进入到下载的文件根目录执行以下命令：

```

//如需使用K1，则需安装netty依赖到本地仓库，注意请选择对应平台netty-tcnative-openssl-static版本，注意修改classifier，macOS :osx-x86_64 ,linux:linux-x86_64 ,windows:windows-x86_64
mvn install:install-file -Dfile=netty-tcnative-openssl-static-2.0.
```

```
17-Final-mychain-all.jar -DgroupId=io.netty -DartifactId=netty-tcn
ative-openssl-static -Dversion=2.0.17-Final-mychain-all -Dpackagin
g=jar
```

注意：netty的版本一定要保证是下面的版本

```
<dependencies>
    <dependency>
      <artifactId>mychain-api</artifactId>
      <groupId>com.alipay.intelligent</groupId>
      <version>1.1.0-SNAPSHOT</version>
    </dependency>
<dependency>
    <groupId>io.netty</groupId>
    <artifactId>netty-all</artifactId>
    <version>4.1.29.Final</version>
</dependency>
<dependency>
    <groupId>io.netty</groupId>
    <artifactId>netty-tcnative-boringssl-static</artifactId>
    <version>2.0.17.Final</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
</dependency>
</dependencies>
```

2.2 运行指南

1. 环境准备

- 准备SSL连接文件和账户私钥文件
  与平台建立ssl连接，需准备三个证书文件：ca机构的根证书（trustCa），客户端的证书文件（client.crt），客户端的私钥文件（client.key）。此外提交交易还需要账户的私钥文件（user.key），这几个文件的详细说明如下：

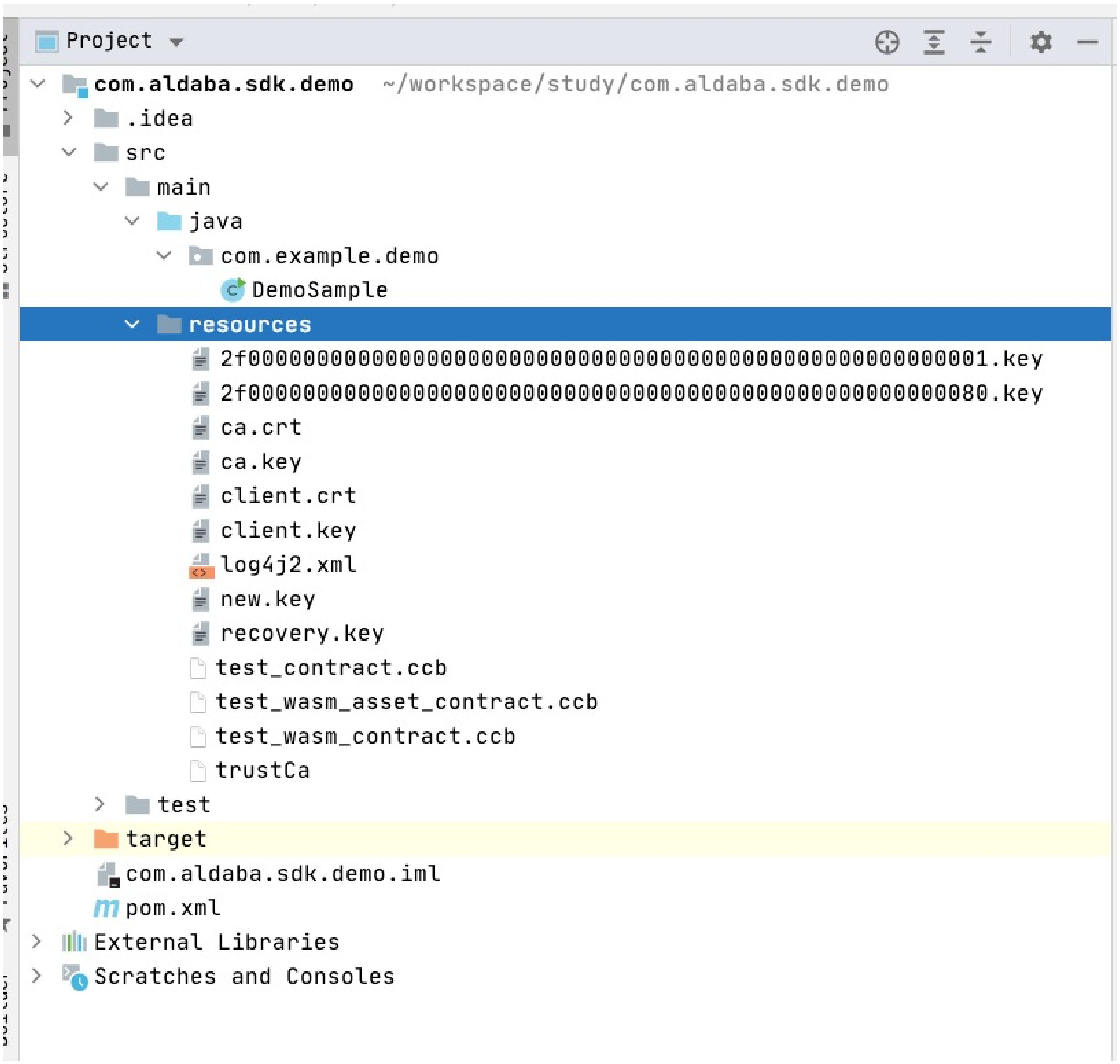
文件名称	文件描述	文件来源

client.crt	客户端的证书文件	链部署者分发
client.key	客户端的私钥文件	
trustCa	存储CA证书的TrustStore	
user.key	账户私钥文件	

2. 应用编写

- 创建完成后，项目目录结构应如下：

- 使用 Intellij IDEA 创建一个基于maven构建的空项目，在下图中java 目录创建自定义包名，例如：com.example.demo ,并将以下 DemoSample.java 完整拷贝创建的package中，并将sdk必须使用的client.crt、client.key、trustCA，user.key 放入到resources目录中，如下图：



```
Project
com.aldaba.sdk.demo  ~/workspace/study/com.aldaba.sdk.demo
  .idea
  src
    main
      java
        com.example.demo
          DemoSample
      resources
        2f00000000000000000000000000000000000000000000000000000000000001.key
        2f00000000000000000000000000000000000000000000000000000000000080.key
        ca.crt
        ca.key
        client.crt
        client.key
        log4j2.xml
        new.key
        recovery.key
        test_contract.ccb
        test_wasm_asset_contract.ccb
        test_wasm_contract.ccb
        trustCa
    test
  target
  com.aldaba.sdk.demo.iml
  pom.xml
External Libraries
Scratches and Consoles
```

```
package com.example.demo;

package com.example.demo;

import com.alipay.intelligent.mychain.sdk.api.BaseService;
import com.alipay.intelligent.mychain.sdk.api.request.AccountCre
ateRequest;
import com.alipay.intelligent.mychain.sdk.api.request.ContractCr
```

```java
eateRequest;
import com.alipay.intelligent.mychain.sdk.api.request.WasmContractCallRequest;
import com.alipay.intelligent.mychain.sdk.crypto.MyCrypto;
import com.alipay.intelligent.mychain.sdk.crypto.PublicKey;
import com.alipay.intelligent.mychain.sdk.crypto.keyoperator.Pkcs8KeyOperator;
import com.alipay.intelligent.mychain.sdk.crypto.keypair.Keypair;
import com.alipay.intelligent.mychain.sdk.crypto.signer.SignerBase;
import com.alipay.intelligent.mychain.sdk.env.*;
import com.alipay.intelligent.mychain.sdk.message.TransactionReceipt;
import com.alipay.intelligent.mychain.sdk.message.TxReceiptEvent;
import com.alipay.intelligent.mychain.sdk.message.api.TransactionPackResponse;
import com.alipay.intelligent.mychain.sdk.network.IAsynCallBack;
import com.alipay.intelligent.mychain.sdk.utils.ByteUtils;
import com.alipay.intelligent.mychain.sdk.utils.vm.Type;
import com.alipay.intelligent.mychain.sdk.utils.vm.TypeEnum;
import com.alipay.intelligent.mychain.sdk.utils.vm.VMOutput;
import com.alipay.intelligent.mychain.sdk.utils.wasm.WASMParameter;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.InetSocketAddress;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

/**
 * Hello world!
 */
public class DemoSample {
    protected BaseService baseService; //用于建立连接
    protected String sender = "2f000000000000000000000000000000000000000000000000000080"; //发起请求的账户
    protected String senderKey = "2f000000000000000000000000000000000000000000000000000080.key"; //发起请求的账户密钥文件
    protected String recoveryKey = "recovery.key";
    protected String newSenderKey = "new.key";
    protected String keyPwd = "123abc"; //发起请求的账户密钥文件的密码
    protected Keypair mainKeypair;
    protected Keypair recoveryKeypair;
    final protected int asyncWaitTime = 10;
    // keypair used to replace main keypair that
    protected Keypair newKeypair;

    public void init() throws Exception {
        //------------
        List<SignerBase> signerBases = new ArrayList<>();
        //加载密钥文件
        mainKeypair = new Pkcs8KeyOperator().loadKey(DemoSample.class.getClassLoader().getResourceAsStream(
                senderKey), keyPwd);
        //存储签名者信息的类
        SignerBase signerBase = new MyCrypto().createSigner(mainKeypair);
        recoveryKeypair = new Pkcs8KeyOperator().loadKey(DemoSample.class.getClassLoader().getResourceAsStream(
                recoveryKey), keyPwd);
        SignerBase recoverySignerBase = new MyCrypto().createSigner(recoveryKeypair);
        newKeypair = new Pkcs8KeyOperator().loadKey(DemoSample.class.getClassLoader().getResourceAsStream(
                newSenderKey), keyPwd);
        SignerBase newSignerBase = new MyCrypto().createSigner(newKeypair);
        signerBases.add(signerBase);
        signerBases.add(recoverySignerBase);
        signerBases.add(newSignerBase);

        SignerOption signerOption = new SignerOption(); //签名配置项
        signerOption.setSigners(signerBases); //设置签名

        //------------
        List<InetSocketAddress> inetSocketAddresses = new ArrayList<>();
        //mychain master节点的 “client_endpoints”中指定的用于客户端进行TCP连接的IP和端口。
        InetSocketAddress inetSocketAddress1 = new InetSocketAddress("100.83.1.225", 18000);
        inetSocketAddresses.add(inetSocketAddress1);
        NetworkOption networkOption = new NetworkOption(); //网络配置项
        networkOption.setConnectTimeoutMs(10000); //设置连接超时时间
        networkOption.setSocketAddressList(inetSocketAddresses);
        //-----------
        //配置SSL连接，链式调用
        ISslOption sslOption = new SslOption.Builder().keyFilePath("client.key").keyPassword("123abc").
                certFilePath("client.crt").trustStoreFilePath("trustCa").trustStorePassword("123abc").build();
        //-----------
        //请求配置
        RequestOption requestOption = new RequestOption();
        //-----------
        //设置签名，网络，SSL连接，请求配置
        baseService = new BaseService();
        baseService.setSignerOption(signerOption);
        baseService.setNetworkOption(networkOption);
        baseService.setiSslOption(sslOption);
        baseService.setRequestOption(requestOption);

        baseService.init();

    }

    public void stop() throws Exception {
        baseService.shutdown();
    }

```

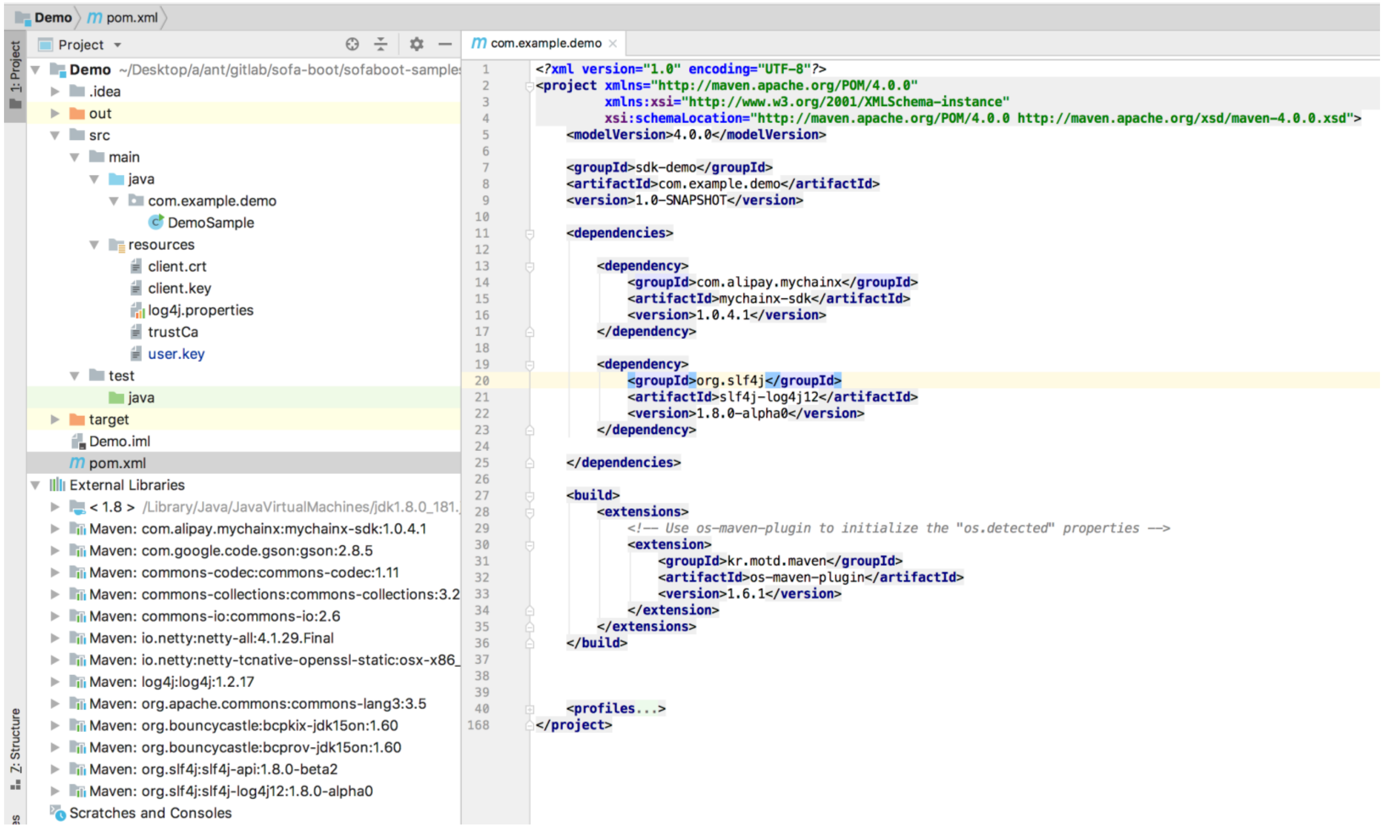
```java
  public String deployContract() throws Exception {
    ContractCreateRequest contractCreateRequest = new ContractCreateRequest();
    contractCreateRequest.setSender(sender);
    byte[] content = readFilebyByte("/test_wasm_contract.ccb");
    contractCreateRequest.setContractContent(content);

    final CountDownLatch deployContractCountDown = new CountDownLatch(1);
    final List<String> contractAddress = new ArrayList<>();
    TransactionPackResponse transactionResponse = baseService.createContract(contractCreateRequest,
        new IAsynCallBack() {
          @Override
          public void callBack(Object event) {
            assert (event instanceof TxReceiptEvent);
            TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
            byte[] txHash = new byte[txReceiptEvent.txHashLength()];
            txReceiptEvent.txHashAsByteBuffer().get(txHash);
            TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
            if (!(transactionReceipt.result() == 0)) {
              return;
            }
            byte[] output = new byte[transactionReceipt.outputLength()];
            transactionReceipt.outputAsByteBuffer().get(output);
            VMOutput vmOutput = new VMOutput(output);
            try {
              Type address = vmOutput.getOutput().get(0);
```

```java
                            if (address.getTypeEnum() == TypeEnum.contract) {
                                contractAddress.add(ByteUtils.toHexString((byte[]) address.getValue()));
                            }
                        } catch (Exception e) {
                        }
                        deployContractCountDown.countDown();
                    }
                });
        if (deployContractCountDown.await(asyncWaitTime, TimeUnit.SECONDS)) {
            return contractAddress.get(0);
        } else {
            throw new RuntimeException("Deploy contract failed");
        }
    }

    public void callContract(String account, String contractAddress) throws Exception {
        WasmContractCallRequest contractCallRequest = new WasmContractCallRequest();
        WASMParameter wasmParameter = new WASMParameter();
        wasmParameter.addString("XX");
        contractCallRequest.setWasmParameter(wasmParameter);
        contractCallRequest.setContractAddr(contractAddress);
        contractCallRequest.setMethod("set_value");
        contractCallRequest.setSender(account);
        final CountDownLatch callContractLatch = new CountDownLatch(1);
        TransactionPackResponse transactionResponse = baseService.callContract(contractCallRequest,
                new IAsynCallBack() {
                    @Override
                    public void callBack(Object event) {
                        if (event instanceof TxReceiptEvent) {
                            TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                            if ((txReceiptEvent.blockNum() > 0))
```

```java
{
                                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
                                assert (transactionReceipt.result() == 0);
                                byte[] output = new byte[transactionReceipt.outputLength()];
                                transactionReceipt.outputAsByteBuffer().get(output);
                                callContractLatch.countDown();
                            }
                        } else {
                            System.out.println("Call contract failed");
                        }
                    }
                });
        if (!callContractLatch.await(asyncWaitTime, TimeUnit.SECONDS)) {
            System.out.println("Call contract failed");
        }
    }

    public String createAccount() throws Exception {
        AccountCreateRequest createAccountRequest = new AccountCreateRequest(); //新建创建账户请求实例
        createAccountRequest.setSender(sender); //设置发送者
        byte[] publicKeyBytes = mainKeypair.getPubkeyEncoded(); //获得创建账户的公钥
        createAccountRequest.setPublicKey(new PublicKey(publicKeyBytes)); //设置创建账户的公钥

        //创建响应实例，并设置回调函数
        final CountDownLatch transactionCountDownLatch = new CountDownLatch(1);
        final List<String> accounts = new ArrayList<>();
        TransactionPackResponse transactionResponse = baseService.createAccount(createAccountRequest,
                //回调接口
        new IAsynCallBack() {
          //回调函数
          @Override
          public void callBack(Object event) throws IOException {

            //检查响应的合法性
            assert (event instanceof TxReceiptEvent);
            TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
            assert (txReceiptEvent.blockNum() > 0);

            TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer()); //获取交易收据

            if (!(transactionReceipt.result() == 0)) {
              ; //交易执行结果, 0 代表成功, 其他值代表失败
              return;
            }
            //从交易收据中获得返回的创建账户的地址信息
            byte[] output = new byte[transactionReceipt.outputLength()];
            transactionReceipt.outputAsByteBuffer().get(output);
            VMOutput vmOutput = new VMOutput(output);
            byte[] accountAddressBytes = (byte[]) vmOutput.getOutput().get(0).getValue(); //创建的账户的地址信息

            accounts.add(ByteUtils.toHexString(accountAddressBytes));
            transactionCountDownLatch.countDown();
          }
        });

    if (transactionCountDownLatch.await(asyncWaitTime, TimeU
nit.SECONDS)) {
            return accounts.get(0);
        } else {
            throw new RuntimeException("Create account failed");
        }
    }

    public static byte[] readFilebyByte(String file) throws IOException {
        InputStream in = DemoSample.class.getResourceAsStream(file);
        byte[] result = inputStreamToByte(in);
        in.close();
        return result;
    }

    public static byte[] inputStreamToByte(InputStream inStream) throws IOException {
        int bufferSize = 1024;
        ByteArrayOutputStream swapStream = new ByteArrayOutputStream();
        byte[] buff = new byte[bufferSize];
        int rc = 0;
        while ((rc = inStream.read(buff, 0, bufferSize)) > 0) {
            swapStream.write(buff, 0, rc);
        }
        return swapStream.toByteArray();
    }


    public static void main(String[] args) throws Exception {
        DemoSample demoSample = new DemoSample();
        try {
            demoSample.init();
            System.out.println("Step 1: Initialize client success");
            String account = demoSample.createAccount();
            System.out.println("Step 2: Create account success");
            String contractAddress = demoSample.deployContract()
```

```
;
      System.out.println("Step 3: Deploy contract success"
);
      demoSample.callContract(account, contractAddress);
      System.out.println("Step 4: Call contract success");
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
      demoSample.stop();
      System.out.println("Step 5: Stop client success");
    }
  }
}

```

在 pom.xml 中添加依赖，将 sdk 与要演示使用的 slf4j-log4j12 引入到 pom.xml 中，并在 resource 中添加log4j的配置文件，如下图：



```
Demo ~/Desktop/a/ant/gitlab/sofa-boot/sofaboot-sample
  .idea
  out
  src
    main
      java
        com.example.demo
          DemoSample
      resources
        client.crt
        client.key
        log4j.properties
        trustCa
        user.key
    test
      java
  target
  Demo.iml
  pom.xml
External Libraries
  < 1.8 > /Library/Java/JavaVirtualMachines/jdk1.8.0_181.
  Maven: com.alipay.mychainx:mychainx-sdk:1.0.4.1
  Maven: com.google.code.gson:gson:2.8.5
  Maven: commons-codec:commons-codec:1.11
  Maven: commons-collections:commons-collections:3.2
  Maven: commons-io:commons-io:2.6
  Maven: io.netty:netty-all:4.1.29.Final
  Maven: io.netty:netty-tcnative-openssl-static:osx-x86_
  Maven: log4j:log4j:1.2.17
  Maven: org.apache.commons:commons-lang3:3.5
  Maven: org.bouncycastle:bcpkix-jdk15on:1.60
  Maven: org.bouncycastle:bcprov-jdk15on:1.60
  Maven: org.slf4j:slf4j-api:1.8.0-beta2
  Maven: org.slf4j:slf4j-log4j12:1.8.0-alpha0
Scratches and Consoles
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>sdk-demo</groupId>
    <artifactId>com.example.demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>

        <dependency>
            <groupId>com.alipay.mychainx</groupId>
            <artifactId>mychainx-sdk</artifactId>
            <version>1.0.4.1</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.8.0-alpha0</version>
        </dependency>

    </dependencies>

    <build>
        <extensions>
            <!-- Use os-maven-plugin to initialize the "os.detected" properties -->
            <extension>
                <groupId>kr.motd.maven</groupId>
                <artifactId>os-maven-plugin</artifactId>
                <version>1.6.1</version>
            </extension>
        </extensions>
    </build>


    <profiles...>
</project>
```

注意：上图中SDK依赖的版本号请使用最新版本。

> 带有 slf4j-log4j12 的完整pom依赖参考：

```
```

```xml
    <dependencies>
        <dependency>
          <artifactId>mychain-api</artifactId>
          <groupId>com.alipay.intelligent</groupId>
          <version>1.1.0-SNAPSHOT</version>
      </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.8.0-alpha0</version>
        </dependency>
    </dependencies>

    <build>
        <extensions>
            <extension>
                <groupId>kr.motd.maven</groupId>
                <artifactId>os-maven-plugin</artifactId>
                <version>1.6.1</version>
            </extension>
        </extensions>
    </build>
```

`log4j.properties`

```
log4j.rootLogger=INFO, R

# 日志输出位置为控制台
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[QC] %p [%t] %C.%M
(%L) | %m%n

# 日志输出位置为文件
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File=./sdk.log
log4j.appender.R.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.R.layout.ConversionPattern=%d-[TS] %p %t %c - %m%n
```

3. 应用编译

- 项目根路径运行 `mvn clean compile` 执行项目编译。

4. 应用执行

- 在 DemoSample.java 中，运行该项目。生成的log文件位于：项目根路径 ./sdk.log，从log中搜索到 Hand shake success， 则代表与区块链平台链接成功。
- 预期输出

```
Step 1: Initialize client success
Step 2: Create account success
Step 3: Deploy contract success
Step 4: Call contract success
Step 5: Stop client success
```

2.3 样例执行流程

1. 初始化环境

```
//step 1: 初始化客户端
init();

//step 2:创建账号交易.
createAccount();

//step 3: 部署智能合约
deployContract();

//step 4: 合约调用
callContract();

//step 5: 关闭环境
stop();
```

2.4 指定密码学套件

合约链的链环境当前仅支持classic：

- classic：使用国际商用密码算法，包括 SHA256 摘要、ECC 公钥算法、AES 对称加密等，`标准合约链` 默认为此套件配置；
- ~~china-sm：使用中国国家商用密码算法，包括 SM3 摘要、SM2 公钥算法、SM4 对称加密等，`国密算法合约链` 默认为此套件配置。~~

如果不清楚 SDK 连接的目标合约链使用的是哪一种密码套件，请咨询该链的管理员。构建 `ClientEnv` 时，必须显式的指定SignerBase，示例如下：

```
Pkcs8KeyOperator pkcs8KeyOperator = new Pkcs8KeyOperator();
Keypair keyPair = pkcs8KeyOperator.load(privateKeyPath, keyPassword);
SignerBase signerBase = MyCrypto.getInstance().createSigner(keyPair);
```

SDK 与合约平台之间的 SSL 通信不受密码学套件影响，由颁发证书的 PKI 机构决定。

3. 数据模型

3.1 账户模型

账户模型是aldaba中的重要概念，账户模型主要分为两部分，跟系统合约签约产生的主账户，主要是系统合约定义的相关数据模型。跟其他服务合约签约的分账户，主要是服务合约自己定义的相关数据模型。
一下是主账户和分账户的基本参数及说明

3.1.1 主账户(系统合约)

- Account

参数	类型	说明

header	Header	主账户头部
service_state	SystemServiceState	系统合约状态
assets_state	[AssetEntry]	未启用

- Header

参数	类型	说明
type	uint8_t	对象类型，主账户为4
version	uint8_t	版本号
status	uint8_t	状态
extra	uint8_t	保留字段，无用

- SystemServiceState

参数	类型	说明
sys_contract_addr	[byte]	系统合约地址
sys_contract_roles	uint64	系统合约权限
sys_access_type	uint8_t	系统通道类型
sys_access_pk	[byte]	系统通道公钥
sys_access_contract	[byte]	系统通道可访问合约，通常是通配符(所有合约)
sys_access_roles	uint64	系统通道权限
contracts_nonce	uint32	保留
contracts	[AccountCSB]	内联账户，**暂未启用**
accesses	[GrantedAccessEntry]	通道列表

- GrantedAccessEntry

参数	类型	说明
id	uint16	通道序号
type	uint64	通道类型，目前只有tx_access一种
pk	[byte]	通道公钥
contract	[byte]	通道允许访问的合约（所有或者某个特定合约）

roles	uint64	通道访问的权限

3.1.2 分账户(服务合约)

- AccountCSB

参数	类型	说明
header	Header	主账户头部，同系统合约header
service_state	SignedContractEntry	分账户的服务合约相关数据模型

- SignedContractEntry

参数	类型	说明
csb	ContractStateBlock	主账户头部，同主账户header
contract_addr	[byte]	分账户的服务合约相关数据模型
roles	uint64	保留字段

3.2 合约模型

下面是合约的基本参数及说明：

- Contract

参数	类型	说明
header	Header	合约头部，同主账户header
ccb	ContractCCB	合约代码块

- ContractCCB

参数	类型	说明
compiler_id	string	编译器型号
compiler_version	string	编译器commit id
language	string	编程语言
source_hash	string	code sha256 hash
role_set	uint64	保留字段
asset_tab	[string]	合约读写的资产类型里列表

contract_tab	[string]	合约读写的合约类型里列表
account_tab	[string]	合约读写的账户类型里列表
shared_csb_cnt	uint8	cso数量
private_csb_cnt	uint8	csb数量 只能是1
function_tab	[ContractFunction]	所有合约函数定义
code	string	合约代码
control_block	ContractControlBlock	合约控制模块，关于自动签约的配置信息，在部署的时候通过参数指定

ContractFunction的定义一般用户不需要感知，感兴趣的可以参考这个文档
https://yuque.antfin-inc.com/antchain/syqo3d/gg2gpx#7QpTz

- ContractControlBlock

参数	类型	说明
enable_default_signup	uint8	是否启用默认签约
default_signup_args	string	自动签约参数列表
default_signup_roles	uint64	无需感知，填0

3.3 交易模型

Transaction包含了一次交易所需要的完整的信息，但针对于各种请求类型所需的填充的参数是可能不完全一样的。使用sdk不需要构造Transaction，只需要使用对应的Service。

- Transaction

参数	类型	说明
sender	String	交易的发送者
contract	String	合约地址
method	String	合约方法
args	String	调用合约方法参数，datastream编码
access_id	int	访问表中的访问ID,需要保护

timestamp	long	时间戳
nonce	String	交易的接受者
gas	Fixed64BitUnsignedInteger	交易执行的消耗费用
memo	String	扩展数据

3.4 收据模型

只有查到一个交易的Receipt才能证明出块成功，交易被确认。

- TransactionReceipt

参数	类型	说明
result	long	交易结果
gasUsed	BigInteger	交易执行的消耗费用
newAddr	String	执行时创建的地址
logs	List	交易执行的日志集合
output	byte[]	合约的ouptut

3.5 日志模型

LogEntry区块链输出日志的数据存储结构。

- LogEntry

参数	类型	说明
sender	String	交易的发送者
contract	String	交易的接受者
topic	List	订阅的主题，topic字段是通过16进制编码
desc	byte[]	交易产生的日志

3.6 区块模型

区块链是由一个个区块组成的，区块由区块头和区块体构成。

- Block

参数	类型	说明
blockHeader	BlockHeader	区块头
blockBody	blockBody	区块体
proof	*BlockProof*	区块证明

- BlockHeader

参数	类型	说明
version	long	版本
extra	long	用于填充或者扩展
number	BigInteger	区块号
timestamp	long	时间戳
parentHash	String	上一区块哈希
txRoot	String	区块体中的交易构成的默克尔哈希根
receiptRoot	String	区块体中的收据构成的默克尔哈希根
stateRoot	String	世界状态的默克尔哈希根
validatorRoot	String	有效公钥默克尔哈希根
gasUsed	BigInteger	交易执行的总消耗量
logBloom	String	日志布隆过滤器

- BlockBody

参数	类型	说明
transactionList	List	交易列表
receiptList	List	收据列表

- BlockProof

参数	类型	说明
number	long	区块高度

hash	String	哈希
version	long	版本
type	int	类型
epoch	long	世纪
validatorSet	String	校验集
proof	String	证明
antiReplayProof	String	反重放证明

3.7 环境相关模型

- ClientEnv

参数	类型	说明
signerOption	SignerOption	签名配置选项
sslOption	ISslOption	tls接口，实现类分别为SslBytesOption、SslOption
networkOption	NetworkOption	网络配置选项
requestOption	RequestOption	消息请求配置选项
logger	ILogger	日志接口

- SignerOption

参数	类型	说明
signers	List	签名接口

- SslOption

参数	类型	说明
keyFilePath	String	客户端的私钥
certFilePath	String	客户端的证书
keyPassword	String	客户端的私钥密码
trustStoreFilePath	String	ca机构的根证书路径
trustStorePassword	String	ca机构的根证书的密码

- NetworkOption

参数	类型	说明
socketAddressList	List	节点的IP和端口信息
enableCompress	Boolean	是否压缩消息
compressSizeLimit	Integer	压缩消息最大值，暂未使用
maxMessageSize	Integer	https接受消息最大值，tls通道发送和接收缓存最大值
connectTimeoutMs	Integer	连接超时时间，单位毫秒
heartbeatIntervalMs	Integer	心跳间隔时间，单位毫秒
retryHeartbeatTimes	Integer	心跳重试次数
retryConnectTimes	Integer	连接单个节点重连次数
networkThreadPoolSize	Integer	netty处理网络事件线程数量，消息解码器的线程数量
msgProcessThreadPoolSize	Integer	消息处理线程池线程数量
msgPoolQueueSize	Integer	网络层发送消息的任务队列大小，netty处理网络事件的任务队列大小

- RequestOption

参数	类型	说明
queryReceiptTimeoutMs	Integer	查询收据的超时时间，单位毫秒
sendRequestTimeoutMs	Integer	发送消息的超时时间，单位毫秒
queryReceiptIntervalMs	Integer	查询收据的间隔时间，单位毫秒
enableQueryTxReceipt	Boolean	是否自动查询交易回执，默认true

- ILogger

参数	类型	说明
logger	ILogger	日志接口，默认可不修改

3.8 权限

- SystemRole

参数	类型	说明
setAccountCreate	boolean	account_create, sender，默认false
setAccountDestroy	boolean	account_destroy, sender ，默认false
setAccountDeactivate	boolean	account_deactive，默认false
setAccountActivate	boolean	account_activate，默认false
setContractCreate	boolean	contract_create, sender && operator，默认false
setContractDestroy	boolean	contract_destory, sender，默认false
setContractDeactivate	boolean	contract_deactive，默认false
setContractActivate	boolean	contract_activate，默认false
setDomainManager	boolean	domain 类的写接口，默认false
setConfigManager	boolean	config_set，默认false

4. 接口说明

4.1 环境接口

4.1.1 服务初始化

函数原型

```
public boolean init()
```

返回字段

返回字段	字段类型	说明

response	Boolean	sdk 是否初始化成功

使用样例

```java
protected BaseService baseService; //用于建立连接
protected String sender = "2f000000000000000000000000000000000000000000000000000080"; //发起请求的账户
protected String senderKey = "2f000000000000000000000000000000000000000000000000000080.key"; //发起请求的账户密钥文件
protected String recoveryKey = "recovery.key";
protected String newSenderKey = "new.key";
protected String keyPwd = "abc123"; //发起请求的账户密钥文件的密码
protected Keypair mainKeypair;
protected Keypair recoveryKeypair;
final protected int asyncWaitTime = 10;
// keypair used to replace main keypair that
protected Keypair newKeypair;

//------------
List<SignerBase> signerBases = new ArrayList<>();
//加载密钥文件
mainKeypair =new Pkcs8KeyOperator().loadKey(SDKBaseTest.class.getClassLoader().getResourceAsStream(
            senderKey),keyPwd);
//存储签名者信息的类
SignerBase signerBase=new MyCrypto().createSigner(mainKeypair);
recoveryKeypair = new Pkcs8KeyOperator().loadKey(SDKBaseTest.class.getClassLoader().getResourceAsStream(
                recoveryKey), keyPwd);
SignerBase recoverySignerBase = new MyCrypto().createSigner(recoveryKeypair);
newKeypair = new Pkcs8KeyOperator().loadKey(SDKBaseTest.class.getClassLoader().getResourceAsStream(
                newSenderKey), keyPwd);
SignerBase newSignerBase = new MyCrypto().createSigner(newKeypair);
signerBases.add(signerBase);
signerBases.add(recoverySignerBase);
signerBases.add(newSignerBase);

SignerOption signerOption = new SignerOption(); //签名配置项
signerOption.setSigners(signerBases); //设置签名

//------------
List<InetSocketAddress> inetSocketAddresses = new ArrayList<>();
//mychain master节点的 “client_endpoints”中指定的用于客户端进行TCP连接的IP和端口。
InetSocketAddress inetSocketAddress1 = new InetSocketAddress("127.0.0.1", 18000);
inetSocketAddresses.add(inetSocketAddress1);
NetworkOption networkOption = new NetworkOption(); //网络配置项
networkOption.setConnectTimeoutMs(10000); //设置连接超时时间
networkOption.setSocketAddressList(inetSocketAddresses);
//-----------
//配置SSL连接，链式调用
ISslOption sslOption = new SslOption.Builder().keyFilePath("client.key").keyPassword("123abc").
            certFilePath("client.crt").trustStoreFilePath("trustCa").trustStorePassword("123abc").build();
//-----------
//请求配置
RequestOption requestOption = new RequestOption();


BaseService baseService = new BaseService();
baseService.setSignerOption(signerOption);
baseService.setNetworkOption(networkOption);
baseService.setiSslOption(sslOption);
baseService.setRequestOption(requestOption);

boolean successful = baseService.init();
```

4.2 账户接口

4.2.1 创建账户

- createAccount

> 创建账户

函数原型

```
public TransactionPackResponse createAccount(AccountCreateRequest createAccountRequest, IAsynCallBack callBack) throws Exceptionount Request) throws Exception
```

请求参数

参数	必选	类型	说明
createAccountRequest	true	AccountCreateRequest	创建账户的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	创建账号的响应

使用样例

```
AccountCreateRequest createAccountRequest = new AccountCreateRequest(); //新建创建账户请求实例
createAccountRequest.setSender(sender); //设置发送者
byte[] publicKeyBytes = mainKeypair.getPubkeyEncoded(); //获得创建账户的公钥
createAccountRequest.setPublicKey(new PublicKey(publicKeyBytes)); //设置创建账户的公钥
System.out.println("pbk " + Hex.toHexString(publicKeyBytes));
```

```java
final List<byte[]> accountAddress = new ArrayList<>(); //用于存储返回的创建账户的地址

//创建响应实例，并设置回调函数
TransactionPackResponse transactionResponse = baseService.createAccount(createAccountRequest,
        //回调接口
        new IAsynCallBack() {
          //回调函数
          @Override
          public void callBack(Object event) throws IOException {

            //检查响应的合法性
            assert (event instanceof TxReceiptEvent);
            TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
            assert (txReceiptEvent.blockNum() > 0);

            TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer()); //获取交易收据

            System.out.println("res: " + transactionReceipt.result());
            assert (transactionReceipt.result() == 0); //交易执行结果，0 代表成功，其他值代表失败

            //从交易收据中获得返回的创建账户的地址信息
            byte[] output = new byte[transactionReceipt.outputLength()];
            transactionReceipt.outputAsByteBuffer().get(output);
            VMOutput vmOutput = new VMOutput(output);
            byte[] accountAddressBytes = (byte[]) vmOutput.getOutput().get(0).getValue(); //创建的账户的地址信息
            System.out.println("account created:" + Hex.toHexString(accountAddressBytes));

            accountAddress.add(accountAddressBytes);
```

```
                    }
          });
```

4.2.2 冻结账户

- deactivateAccount

> 冻结账户

函数原型

```
public TransactionPackResponse deactivateAccount(AccountDeactivateRequest deactivateAccountRequest, IAsynCallBack callBack) throws Exception
```

请求参数

参数	必选	类型	说明
deactivateAccountRequest	true	AccountDeactivateRequest	冻结账户的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	冻结账号的响应

使用样例

```
AccountDeactivateRequest deactiveAccountRequest = new AccountDeactivateRequest(); //新建冻结账户请求实例
deactiveAccountRequest.setSender(sender); //设置发送者
deactiveAccountRequest.setAccountAddress(account); //设置要冻结的账户

//创建响应实例，并设置回调函数
transactionResponse = baseService.deactivateAccount(deactiveAccountRequest,
        //回调接口
        new IAsynCallBack() {
            //回调函数
            @Override
            public void callBack(Object event) {

                //检查响应的合法性
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);

                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer()); //获取交易收据

                System.out.println("res: " + transactionReceipt.result());
                assert (transactionReceipt.result() == 0); //交易执行结果，0 代表成功，其他值代表失败

                // TODO check accout is really frozen
                System.out.println("account frozen: " + Hex.toHexString(accountAddress.get(0)));
            }
```

4.2.3 解冻账户

- activateAccount

> 创建账户，同步方式调用

函数原型

```java
public TransactionPackResponse activateAccount(AccountActivateRequest activateAccountRequest, IAsynCallBack callBack) throws Exception
```

请求参数

参数	必选	类型	说明
activateAccountRequest	true	AccountActivateRequest	解冻账户的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	解冻账号的响应

使用样例

```java
AccountActivateRequest activateAccountRequest = new AccountActivateRequest();
activateAccountRequest.setSender(sender);
activateAccountRequest.setAccountAddress(account);

transactionResponse = baseService.activateAccount(activateAccountRequest, new IAsynCallBack() {
    @Override
    public void callBack(Object event) {
        assert (event instanceof TxReceiptEvent);
        TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
        assert (txReceiptEvent.blockNum() > 0);
        TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());

        System.out.println("res: " + transactionReceipt.result());
        assert (transactionReceipt.result() == 0);
        // TODO check accout is really unfrozen
        System.out.println("account unfrozen: " + Hex.toHexString(accountAddress.get(0)));
    }
});
```

4.2.4 销毁账户

- destroyAccount

> 销毁账户

函数原型

```java
public TransactionPackResponse destroyAccount(AccountDestroyRequest destroyAccountRequest, IAsynCallBack callBack) throws Exception
```

请求参数

参数	必选	类型	说明
destroyAccountRequest	true	AccountDestroyRequest	销毁账户的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	销毁账号的响应

使用样例

```java
AccountDestroyRequest accountDestroyRequest = new AccountDestroyRequest();
accountDestroyRequest.setSender(sender);
accountDestroyRequest.setAccount(account);

transactionResponse = baseService.destroyAccount(accountDestroyRequest, new IAsynCallBack() {
    @Override
    public void callBack(Object event) {
        assert (event instanceof TxReceiptEvent);
        TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
        assert (txReceiptEvent.blockNum() > 0);
        TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());

        System.out.println("res: " + transactionReceipt.result());
        assert (transactionReceipt.result() == 0);
        // TODO check accout is really unfrozen
        System.out.println("account unfrozen: " + Hex.toHexString(accountAddress.get(0)));
    }
});
```

4.3 合约接口

4.3.1 创建合约

- createContract

> 创建合约

函数原型

```java
public TransactionPackResponse createContract(ContractCreateRequest contractCreateRequest, IAsynCallBack callBack) throws Exception
```

请求参数

参数	必选	类型	说明
contractCreateRequest	true	ContractCreateRequest	创建合约的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	创建合约的响应

使用样例

```java
ContractCreateRequest contractCreateRequest = new ContractCreateRequest();
contractCreateRequest.setSender(sender);
byte[] content = readFilebyByte("/test_wasm_contract.ccb");
contractCreateRequest.setContractContent(content);

TransactionPackResponse transactionResponse = baseService.createContract(contractCreateRequest,
        new IAsynCallBack() {
            @Override
            public void callBack(Object event) {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
                System.out.printf("transaction receipt %d\n", transactionReceipt.result());
                assert (transactionReceipt.result() == 0);
                byte[] output = new byte[transactionReceipt.outputLength()];
                transactionReceipt.outputAsByteBuffer().get(output);
                System.out.println(Hex.toHexString(output));
                VMOutput vmOutput = new VMOutput(output);
                try {
                    Type address = vmOutput.getOutput().get(0);
                    if (address.getTypeEnum() == TypeEnum.contract) {
                        setContractAddress(ByteUtils.toHexString((byte[]) address.getValue()));
                    }
                    System.out.println(vmOutput.getOutput().get(0));
                } catch (Exception e) {
                    fail();
                }
            }
        });
```

4.3.2 冻结合约

- deactivateContract

> 冻结合约

函数原型

```java
public TransactionPackResponse deactivateContract(ContractDeactivateRequest deactivateContractRequest, IAsynCallBack callBack) throws Exception
```

请求参数

参数	必选	类型	说明
deactivateContractRequest	true	ContractDeactivateRequest	创建合约的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	创建合约的响应

使用样例

```
ContractDeactivateRequest contractDeactivateRequest = new ContractDeactivateRequest();
contractDeactivateRequest.setSender(sender);
contractDeactivateRequest.setContract(contractAddress);

transactionResponse = baseService.deactivateContract(contractDeactivateRequest,
        new IAsynCallBack() {
            @Override
            public void callBack(Object event) {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
                System.out.printf("transaction receipt %d\n", transactionReceipt.result());
                assert (transactionReceipt.result() == 0);
                byte[] output = new byte[transactionReceipt.outputLength()];
                transactionReceipt.outputAsByteBuffer().get(outpu
t);
                System.out.println(Hex.toHexString(output));
            }
        });
```

4.3.3 解冻合约

- activateContract

> 解冻合约

函数原型

```
public TransactionPackResponse activateContract(ContractActivateRequest activateContractRequest, IAsynCallBack callBack) throws Exception
```

请求参数

参数	必选	类型	说明
activateContractRequest	true	ContractActivateRequest	解冻合约的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	解冻合约的响应

使用样例

```
ContractActivateRequest contractactiveCallRequest = new ContractA
```

```
ctivateRequest();
contractactiveCallRequest.setSender(sender);
contractactiveCallRequest.setContract(contractAddress);

transactionResponse = baseService.activateContract(contractactive
CallRequest,
        new IAsynCallBack() {
            @Override
            public void callBack(Object event) {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent)
event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = Transacti
onReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsB
yteBuffer());
                System.out.printf("transaction receipt %d\n", tra
nsactionReceipt.result());
                assert (transactionReceipt.result() == 0);
                byte[] output = new byte[transactionReceipt.outpu
tLength()];
                transactionReceipt.outputAsByteBuffer().get(outpu
t);
                System.out.println(Hex.toHexString(output));
            }
        });
```

4.3.4 销毁合约

- destroyAccount

> 销毁合约

函数原型

```

```

```java
public TransactionPackResponse destroyContract(ContractDestroyRequest destroyContractRequest, IAsynCallBack callBack) throws Exception
```

请求参数

参数	必选	类型	说明
destroyContractRequest	true	ContractDestroyRequest	销毁合约的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	销毁合约的响应

使用样例

```java
ContractDestroyRequest contractDestroyRequest = new ContractDestroyRequest();
contractDestroyRequest.setContract(contractAddress);
contractDestroyRequest.setSender(sender);

TransactionPackResponse transactionResponse = baseService.destroyContract(contractDestroyRequest,
        new IAsynCallBack() {
            @Override
            public void callBack(Object event) {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
                System.out.printf("transaction receipt %d\n", tra
nsactionReceipt.result());
                assert (transactionReceipt.result() == 0);
                byte[] output = new byte[transactionReceipt.outputLength()];
                transactionReceipt.outputAsByteBuffer().get(output);
                System.out.println(Hex.toHexString(output));
            }
        });
```

4.4 授权接口

4.4.1 交易访问授权

- grantTransactionAccess

> 交易访问授权

函数原型

```
public TransactionPackResponse grantTransactionAccess(TxAccessGrantRequest createAccessRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
createAccessRequest	true	TxAccessGrantRequest	交易访问授权的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	交易访问授权的响应

使用样例

```java
ContractAccessGrantRequest contractAccessGrantRequest = new ContractAccessGrantRequest(); // 创建交易请求
contractAccessGrantRequest.setSender(sender); // 设置发送者
contractAccessGrantRequest.setCaller(contractAddress); // 固定为数字 0
contractAccessGrantRequest.setRoles(new SystemRole());
contractAccessGrantRequest.setContract(ContractConstants.SYSTEM_CONTRACT_ADDR);

transactionResponse = baseService.grantContractAccessId(contractAccessGrantRequest,
        new IAsynCallBack() {
            // 回调函数
            @Override
            public void callBack(Object event) throws IOException {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());

                System.out.println("res: " + transactionReceipt.result());
                assert (transactionReceipt.result() == 0); //检查是否成功

                VMOutput vmOutput = new VMOutput(GetHelper.get(transactionReceipt, "output"));

                int accessId = ((Short) (vmOutput.getOutput().get(0).getValue())).intValue(); //获取通道ID
                System.out.println("accessid: " + accessId);
                access.add(accessId);
            }
        });
```

4.4.2 交易访问撤回

- revokeTransactionAccess

> 交易访问撤回

函数原型

```
public TransactionPackResponse revokeTransactionAccess(TxAccessRevokeRequest revokeTransactionAccessRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
revokeTransactionAccessRequest	true	TxAccessRevokeRequest	交易访问撤回的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	交易访问撤回的响应

使用样例

```
ContractAccessRevokeRequest contractAccessRevokeRequest = new ContractAccessRevokeRequest(); // 创建交易请求
contractAccessRevokeRequest.setSender(sender); // 设置发送者
contractAccessRevokeRequest.setAccessId(BigInteger.valueOf(access
.get(0))); // 固定为数字 0

transactionResponse = baseService.revokeContractAccess(contractAccessRevokeRequest,
        new IAsynCallBack() {
            // 回调函数
            @Override
            public void callBack(Object event) throws IOException {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());

                System.out.println("res: " + transactionReceipt.result());
                assert (transactionReceipt.result() == 0); //检查是否成功
            }
        });
```

4.4.3 合约访问授权

- grantContractAccess

> 合约访问授权

函数原型

```
public TransactionPackResponse grantContractAccess(GrantContractAccessCallRequest grantContractAccessCallRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
grantContractAccessCallRequest	true	GrantContractAccessCallRequest	合约访问授权的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	交易访问授权的响应

使用样例

```
ContractAccessGrantRequest contractAccessGrantRequest = new ContractAccessGrantRequest(); // 创建交易请求
contractAccessGrantRequest.setSender(sender); // 设置发送者
contractAccessGrantRequest.setCaller(contractAddress); // 固定为数字 0
contractAccessGrantRequest.setRoles(new SystemRole());
contractAccessGrantRequest.setContract(ContractConstants.SYSTEM_CONTRACT_ADDR);

transactionResponse = baseService.grantContractAccess(contractAccessGrantRequest,
        new IAsynCallBack() {
            // 回调函数
            @Override
            public void callBack(Object event) throws IOException {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());

                System.out.println("res: " + transactionReceipt.result());
                assert (transactionReceipt.result() == 0); //检查是否成功

                VMOutput vmOutput = new VMOutput(GetHelper.get(transactionReceipt, "output"));

                int accessId = ((Short) (vmOutput.getOutput().get(0).getValue())).intValue(); //获取通道ID
                System.out.println("accessid: " + accessId);
                access.add(accessId);
            }
        });
```

4.4.4 合约访问撤回

- revokeContractAccess

> 合约访问撤回

函数原型

```
public TransactionPackResponse revokeContractAccess(ContractAccessRevokeRequest contractAccessRevokeRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
contractAccessRevokeRequest	true	ContractAccessRevokeRequest	合约访问撤回的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	合约访问撤回的响应

使用样例

```java
ContractAccessRevokeRequest contractAccessRevokeRequest = new ContractAccessRevokeRequest(); // 创建交易请求
contractAccessRevokeRequest.setSender(sender); // 设置发送者
contractAccessRevokeRequest.setAccessId(BigInteger.valueOf(access.get(0))); // 固定为数字 0

transactionResponse = baseService.revokeContractAccess(contractAccessRevokeRequest,
        new IAsynCallBack() {
            // 回调函数
            @Override
            public void callBack(Object event) throws IOException {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());

                System.out.println("res: " + transactionReceipt.result());
                assert (transactionReceipt.result() == 0); //检查是否成功
            }
        });
```

4.5 管理接口

4.5.1 设置配置

- setConfig

> 设置配置

函数原型

```java
public TransactionPackResponse setConfig(ConfigSetRequest setConfigRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
setConfigRequest	true	ConfigSetRequest	设置配置的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	设置配置的响应

使用样例

```java
ConfigSetRequest setConfigRequest = new ConfigSetRequest();
setConfigRequest.setSender(sender);
setConfigRequest.setKey("admin.account".getBytes());

TransactionPackResponse transactionResponse = baseService.setConfig(setConfigRequest,
        new IAsynCallBack() {
            @Override
            public void callBack(Object event) {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = Transacti
onReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
                assert (transactionReceipt.result() == 0);
                System.out.println("mychain returns " + transactionReceipt.result());
            }
        });
```

4.5.2 域添加

- addDomain

> 域添加

函数原型

```
public TransactionPackResponse addDomain(DomainAddRequest addDomainRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
addDomainRequest	true	DomainAddRequest	域添加的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	域添加的响应

使用样例

```
ContractAccessRevokeRequest contractAccessRevokeRequest = new ContractAccessRevokeRequest(); // 创建交易请求
contractAccessRevokeRequest.setSender(sender); // 设置发送者
contractAccessRevokeRequest.setAccessId(BigInteger.valueOf(access.get(0))); // 固定为数字 0

transactionResponse = baseService.revokeContractAccess(contractAccessRevokeRequest,
        new IAsynCallBack() {
            // 回调函数
            @Override
            public void callBack(Object event) throws IOException {
                assert (event instanceof TxReceiptEvent);
                TxReceiptEvent txReceiptEvent = (TxReceiptEvent) event;
                assert (txReceiptEvent.blockNum() > 0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());

                System.out.println("res: " + transactionReceipt.result());
                assert (transactionReceipt.result() == 0); //检查是否成功
            }
        });
```

4.5.3 域更新

- updateDomain

> 域更新

函数原型

```
public TransactionPackResponse updateDomain(DomainUpdateRequest updateDomainRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
updateDomainRequest	true	DomainUpdateRequest	域更新的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	域更新的响应

使用样例

```java
DomainUpdateRequest updateDomainRequest = new DomainUpdateRequest();
updateDomainRequest.setDomainId(HashFactory.getHash().hash(mainKeypair.getPubkeyEncoded()));
updateDomainRequest.setDomainRole(BigInteger.valueOf(DomainRole.DOMAIN_ROLE_CONSENSUS));
updateDomainRequest.setDomainState(BigInteger.valueOf(DomainState.DOMAIN_STATE_NORMAL));
List<Type>  endpoints = new ArrayList<>();
Type endpoint = new Type(TypeEnum.string,"tcp://123.0.0.1:8080".getBytes());
endpoints.add(endpoint);
updateDomainRequest.setEndpoints(endpoints);
updateDomainRequest.setPublicKey(new PublicKey(mainKeypair.getPubkeyEncoded()));
TransactionPackResponse transactionResponse = baseService.updateDomain(updateDomainRequest,
    new IAsynCallBack() {
        @Override
        public void callBack(Object event) {
            assert(event instanceof TxReceiptEvent);
            TxReceiptEvent txReceiptEvent = (TxReceiptEvent)event;
```

```
                assert(txReceiptEvent.blockNum()>0);
                TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
                assert (transactionReceipt.result() == 100005);
                System.out.println("mychain returns " + transactionReceipt.result());
            }
    });
```

4.5.4 域移除

- deleteDomain

> 域移除

函数原型

```
public TransactionPackResponse deleteDomain(DomainRemoveRequest deleteDomainRequest, IAsynCallBack asynCallBack) throws Exception
```

请求参数

参数	必选	类型	说明
deleteDomainRequest	true	DomainRemoveRequest	域移除的请求
callBack	true	IAsynCallBack	交易回调

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	合约访问撤回的响应

使用样例

```java
DomainRemoveRequest deleteDomainRequest = new DomainRemoveRequest();
deleteDomainRequest.setSender(sender);
// any fake domain works
deleteDomainRequest.setDomainId(Hex.decode(sender));
TransactionPackResponse transactionResponse = baseService.deleteDomain(deleteDomainRequest,
    new IAsynCallBack() {
        @Override
        public void callBack(Object event) {
            assert(event instanceof TxReceiptEvent);
            TxReceiptEvent txReceiptEvent = (TxReceiptEvent)event;
            assert(txReceiptEvent.blockNum()>0);
            TransactionReceipt transactionReceipt = TransactionReceipt.getRootAsTransactionReceipt(txReceiptEvent.txReceiptAsByteBuffer());
            assert (transactionReceipt.result() == 100005);
            System.out.println("mychain returns " + transactionReceipt.result());
        }
    });
```

4.6 查询接口

4.6.1 交易查询

- queryTransaction

> 交易查询

函数原型

```java
public TransactionPackResponse queryTransaction(QueryTransactionRequest queryTransactionRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryTransactionRequest	true	QueryTransactionRequest	交易查询的请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	交易查询的响应

使用样例

```
QueryTransactionRequest queryTransactionRequest = new QueryTransactionRequest();
queryTransactionRequest.setSender(sender);
queryTransactionRequest.setHash(ByteUtils.toHexString(txHash));
queryTransactionRequest.setRequireProof(false);
TransactionPackResponse transactionPackResponse = baseService.queryTransaction(queryTransactionRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
```

4.6.2 交易收据查询

- queryTransactionReceipt

> 交易收据查询

函数原型

```
public TransactionPackResponse queryTransactionReceipt(QueryTransactionReceiptRequest queryTransactionReceiptRequest) throws IOExcep
```

```
tion
```

请求参数

参数	必选	类型	说明
queryTransactionReceiptRequest	true	QueryTransactionReceiptRequest	交易收据查询的请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	交易收据查询的响应

使用样例

```
QueryTransactionReceiptRequest queryTransactionReceiptRequest = new QueryTransactionReceiptRequest();
queryTransactionReceiptRequest.setSender(sender);
queryTransactionReceiptRequest.setHash(ByteUtils.toHexString(txHash));
queryTransactionReceiptRequest.setRequireProof(false);
TransactionPackResponse transactionReceiptPackResponse = baseService.queryTransactionReceipt(queryTransactionReceiptRequest);
assertEquals(transactionReceiptPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionReceiptPackResponse.getTransactionReceipt().result(),0);
```

4.6.3 区块查询

- queryBlock

> 区块查询

函数原型

```java
public TransactionPackResponse queryBlock(QueryBlockRequest queryBlockRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryBlockRequest	true	QueryBlockRequest	区块查询的请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	区块查询的响应

使用样例

```java
long queryBlockNumber = 1L;
QueryBlockRequest queryBlockRequest = new QueryBlockRequest();
queryBlockRequest.setSender(sender);
queryBlockRequest.setBlockNum(BigInteger.valueOf(queryBlockNumber));
queryBlockRequest.setRequireBody(true);
queryBlockRequest.setRequireProof(true);
TransactionPackResponse transactionPackResponse = baseService.queryBlock(queryBlockRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffer().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] blockInfoBytes = (byte[])vmOutput.getOutput().get(0).getValue();
```

```
BlockInfo blockInfo = BlockInfo.getRootAsBlockInfo(ByteBuffer.wrap(blockInfoBytes));
BlockHeader blockHeader = BlockHeader.getRootAsBlockHeader(blockInfo.blockHeaderAsByteBuffer());
long blockNumber = blockHeader.number();
assert(blockNumber == queryBlockNumber);

BlockBody blockBody = BlockBody.getRootAsBlockBody(blockInfo.blockBodyAsByteBuffer());
byte[] receiptListBytes = new byte[blockBody.receiptListLength()];
blockBody.receiptListAsByteBuffer().get(receiptListBytes);
List<byte[]> receiptList = DecodeBytesVector.decodeBytesArray(receiptListBytes);
byte[] transactionListBytes = new byte[blockBody.txListLength()];
blockBody.txListAsByteBuffer().get(transactionListBytes);
List<byte[]> transactionList = DecodeBytesVector.decodeBytesArray(transactionListBytes);
System.out.println(receiptList.size());
List<TransactionReceipt> transactionReceipts = new ArrayList<>();
List<TransactionRequest> transactionRequests = new ArrayList<>();
for(int i =0;i<receiptList.size();i++){
    transactionReceipts.add(TransactionReceipt.getRootAsTransactionReceipt(ByteBuffer.wrap(receiptList.get(i))));
    TransactionRequest transactionRequest = new TransactionRequest(transactionList.get(i));
    transactionRequests.add(transactionRequest);
}
assert(true);
```

4.6.4 最新区块头查询

- queryLastBlockHeader

> 最新区块头查询

函数原型

```java
public TransactionPackResponse queryLastBlockHeader(QueryLastBlockHeaderRequest queryLastBlockHeaderRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryLastBlockHeaderRequest	true	QueryLastBlockHeaderRequest	最新区块头查询的请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	最新区块头查询的响应

使用样例

```java
QueryLastBlockHeaderRequest queryLastBlockHeaderRequest = new QueryLastBlockHeaderRequest();
queryLastBlockHeaderRequest.setSender(sender);
TransactionPackResponse transactionPackResponse = baseService.queryLastBlockHeader(queryLastBlockHeaderRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffer().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] blockInfoBytes = (byte[])vmOutput.getOutput().get(0).getValue();
BlockInfo blockInfo = BlockInfo.getRootAsBlockInfo(ByteBuffer.wrap(blockInfoBytes));
BlockHeader blockHeader = BlockHeader.getRootAsBlockHeader(blockInfo.blockHeaderAsByteBuffer());
long blockNumber = blockHeader.number();
```

```
ClearHistoryStatusRequest clearHistoryStatusRequest = new ClearHi
storyStatusRequest();
clearHistoryStatusRequest.setSender(sender);
clearHistoryStatusRequest.setType(1);
clearHistoryStatusRequest.setBlockNum(BigInteger.valueOf(200L));
clearHistoryStatusRequest.setBlockTimestamp(BigInteger.valueOf(Sy
stem.currentTimeMillis()));
ClearHistoryStatusResponse clearHistoryStatusResponse = baseServi
ce.clearHistoryStatus(clearHistoryStatusRequest);
assert(clearHistoryStatusResponse.getErrorCode().equals(ErrorCode
.SUCCESS));
```

4.6.5 健康状态查询

- queryHealthStatus

> 健康状态查询

函数原型

```
public TransactionPackResponse queryHealthStatus(QueryHealthStatus
Request queryHealthStatusRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryHealthStatusRequest	true	QueryHealthStatusRequest	健康状态查询请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	健康状态查询的响应

使用样例

```
QueryHealthStatusRequest queryHealthStatusRequest = new QueryHeal
```

```
thStatusRequest();
queryHealthStatusRequest.setSender(sender);
TransactionPackResponse transactionPackResponse = baseService.queryHealthStatus(queryHealthStatusRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffer().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] healthStatusBytes = (byte[])vmOutput.getOutput().get(0).getValue();
HealthStatusInfo healthStatus = HealthStatusInfo.getRootAsHealthStatusInfo(ByteBuffer.wrap(healthStatusBytes));

System.out.println("block: " + healthStatus.lastBlockNum() + " timestamp: " + healthStatus.lastTimestamp());
```

4.6.6 账号查询

- queryAccount

> 账号查询

函数原型

```
public TransactionPackResponse queryAccount(QueryAccountRequest queryAccountRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryAccountReques	true	QueryAccountRequest	账号查询的请求

t			

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	账号查询的响应

使用样例

```java
QueryAccountRequest queryAccountRequest = new QueryAccountRequest();
queryAccountRequest.setSender(sender);
queryAccountRequest.setAccountAddress(sender);
queryAccountRequest.setRequireProof(false);
TransactionPackResponse transactionPackResponse = baseService.queryAccount(queryAccountRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffer().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] accountInfoBytes = (byte[])vmOutput.getOutput().get(0).getValue();
AccountInfo accountInfo = AccountInfo.getRootAsAccountInfo(ByteBuffer.wrap(accountInfoBytes));
int accountLength = accountInfo.accountLength();
byte[] account = new byte[accountLength];;
accountInfo.accountAsByteBuffer().get(account);
```

4.6.7 合约查询

- queryContract

合约查询

函数原型

```java
public TransactionPackResponse queryContract(QueryContractRequest queryContractRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryContractRequest	true	QueryContractRequest	合约查询的请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	合约查询的响应

使用样例

```java
QueryContractRequest queryContractRequest = new QueryContractRequest();
queryContractRequest.setSender(sender);
queryContractRequest.setContractAddress(ContractConstants.SYSTEM_CONTRACT_ADDR);
queryContractRequest.setRequireProof(false);
TransactionPackResponse transactionPackResponse = baseService.queryContract(queryContractRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffe
```

```
r().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] contractInfoBytes = (byte[])vmOutput.getOutput().get(0).getValue();
ContractInfo contractInfo = ContractInfo.getRootAsContractInfo(ByteBuffer.wrap(contractInfoBytes));
int contractLength = contractInfo.contractLength();
byte[] account = new byte[contractLength];;
contractInfo.contractAsByteBuffer().get(account);
System.out.println("contract: length " + String.valueOf(contractLength));
```

4.6.8 共识状态查询

- queryConsensusStatus

> 共识状态查询

函数原型

```
public TransactionPackResponse queryConsensusStatus(QueryConsensusStatusRequest queryConsensusStatusRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryConsensusStatusRequest	true	QueryConsensusStatusRequest	共识状态查询的请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	共识状态查询的响应

使用样例

```
QueryConsensusStatusRequest queryConsensusStatusRequest = new QueryConsensusStatusRequest();
queryConsensusStatusRequest.setSender(sender);
TransactionPackResponse transactionPackResponse = baseService.queryConsensusStatus(queryConsensusStatusRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffer().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] consensusStatusBytes = (byte[])vmOutput.getOutput().get(0).getValue();
ConsensusStatus consensusStatus = ConsensusStatus.getRootAsConsensusStatus(ByteBuffer.wrap(consensusStatusBytes));

byte statusType = consensusStatus.statusType();
assertEquals(statusType, ConsensusStatusType.PbftStatus);

PbftStatus status = new PbftStatus();
consensusStatus.status(status);
```

4.6.9 合约节点状态查询

- queryContractNodesStatus

> 合约节点状态查询

函数原型

```
public QueryContractNodesStatusResponse queryContractNodesStatus(QueryContractNodesStatusRequest queryContractNodesStatusRequest) th
```

```
  rows IOException
```

请求参数

参数	必选	类型	说明
queryContractNodesStatusRequest	true	QueryContractNodesStatusRequest	合约节点状态查询的请求

返回字段

返回字段	字段类型	说明
response	QueryContractNodesStatusRequest	合约节点状态查询的响应

使用样例

```
QueryContractNodesStatusRequest queryContractNodesStatusRequest = new QueryContractNodesStatusRequest();
queryContractNodesStatusRequest.setSender(sender);
queryContractNodesStatusRequest.setBlockNum(BigInteger.valueOf(-1));
TransactionPackResponse transactionPackResponse = baseService.queryContractNodesStatus(queryContractNodesStatusRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffer().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] contractConfigStatusBytes = (byte[])vmOutput.getOutput().get(0).getValue();
ContractNodesStatus contractConfigStatus = ContractNodesStatus.getRootAsContractNodesStatus(ByteBuffer.wrap(contractConfigStatusBytes));
```

```

DomainMeta domainMeta = contractConfigStatus.domains(0);

byte[] domainIdBytes = new byte[domainMeta.domainIdLength()];
domainMeta.domainIdAsByteBuffer().get(domainIdBytes);
```

4.6.10 合约配置状态查询

- queryContractConfigStatus

> 合约配置状态查询

函数原型

```
public TransactionPackResponse queryContractConfigStatus(QueryContractConfigStatusRequest queryContractConfigStatusRequest) throws IOException
```

请求参数

参数	必选	类型	说明
queryContractConfigStatusRequest	true	QueryContractConfigStatusRequest	合约配置状态查询的请求

返回字段

返回字段	字段类型	说明
response	TransactionPackResponse	合约配置状态查询的响应

使用样例

```
QueryContractConfigStatusRequest queryContractConfigStatusRequest = new QueryContractConfigStatusRequest();
queryContractConfigStatusRequest.setSender(sender);
queryContractConfigStatusRequest.setBlockNum(BigInteger.valueOf(-
1));
TransactionPackResponse transactionPackResponse = baseService.queryContractConfigStatus(queryContractConfigStatusRequest);
assertEquals(transactionPackResponse.getErrorCode(), ErrorCode.SUCCESS);
assertEquals(transactionPackResponse.getTransactionReceipt().result(),0);
byte[] output = new byte[transactionPackResponse.getTransactionReceipt().outputLength()];
transactionPackResponse.getTransactionReceipt().outputAsByteBuffer().get(output);
VMOutput vmOutput = new VMOutput(output);
byte[] contractConfigStatusBytes = (byte[])vmOutput.getOutput().get(0).getValue();
ContractConfigStatus contractConfigStatus = ContractConfigStatus.getRootAsContractConfigStatus(ByteBuffer.wrap(contractConfigStatusBytes));

byte[] kv = new byte[contractConfigStatus.kvsLength()];
contractConfigStatus.kvsAsByteBuffer().get(kv);
String kvStr = new String(kv);
System.out.println("contract config kv: " + kvStr);
```

5. 错误码和错误信息

5.1 错误码

```
SUCCESS                 = 0,     // 成功
EXECUTOR_RUNTIME_ERROR = 3001,   // 虚拟机执行合约时候，虚拟机自身发现错误
EXECUTOR_PROGRAM_ERROR = 3002,   // 合约内部出现异常，调用运行时Abort接口，抛出该异常
EXECUTOR_UNKNOWN_ERROR = 3003,   // 未知异常，例如合约代码里丢出异常，并未被合约内部捕获
```

5.2 OUTPUT

- 当transaction_receipt.result == SUCCESS时，output为入口合约函数的返回值的datastream编码，

```
例如返回值是void：0100
说明：01(1byte version)00 (vector size)


例如返回值是int8：0101022c
说明：01(1byte version)01 (vector size) 02(uint8 类型)2c (合约函数返回值)
```

- 当transaction_receipt.result != SUCCESS时，output为错误信息字符串的datastream编码(当result为EXECUTOR_PROGRAM_ERROR，错误信息为合约主动调用运行时函数abort触发)

```
例如：01010c196f626a65637420697320616c7265616479206372656174656564
说明：01(1byte version)01 (vector size) 0c (string类型) 196f626a65637420697320616c7265616479206372656174656564 (错误信息)
```

5.3 receipt结构

属性	取值	描述
	0: SUCCESS	交易调用成功
	3001: EXECUTOR_RUNTIME_ERROR	平台虚拟机异常
	3002: EXECUTOR_PROGRAM_ERROR	合约异常
	3003: EXECUTOR_UNKNOWN_ERROR	未知异常，请联系链平台开发者定位问题
	4001: READ_WRITE_SCOPE_LOAD_CCB_FAILED	读写集分析异常，一般为合约CCB读写集元数据或者是交易参数错误
	4002:	

result			READ_WRITE_SCOPE_CCB_NOT_VALID	
			4003： READ_WRITE_SCOPE_ARGS_DECODE_FAILED	
			4004： READ_WRITE_SCOPE_ARGS_ENCODE_FAILED	
			4005： READ_WRITE_SCOPE_INVALID_SENDER	
			4006： READ_WRITE_SCOPE_INVALID_CONTRACT	
			4007： READ_WRITE_SCOPE_INVALID_ARGUMENT	
			4010： READ_WRITE_SCOPE_UNKNOWN	
			10204: VM_METHOD_NOT_EXIST	合约方法不存在，可能是递归合约调用过程中找不到方法
			10205: VM_PARAMETER_NOT_MATCH	调用合约方法时，参数不匹配
			其他错误码	请联系链平台开发者定位问题
output			见上述说明	见上述说明
logs	只有在result==0时可能有，由合约执行过程中调用运行时函数Log产生	log0		
		log1		
		…		
		logn		