TDC Development Documentation

一、Dependency injection

```
<-- Core dependency ->
<dependency>
    <groupId>io.nuls.v2</groupId>
    <artifactId>sdk4j-jdk8</artifactId>
    <version>1.1.10.RELEASE</version>
</dependency>

<-- RPC dependency ->
<dependency>
    <groupId>com.github.briandilley.jsonrpc4j</groupId>
    <artifactId>jsonrpc4j</artifactId>
    <version>1.1</version>
</dependency>
```

二、Address creation

```
ECKey key = new ECKey();
Address address = new Address(CHAIN_ID, CoinType,
BaseConstant.DEFAULT_ADDRESS_TYPE,
SerializeUtils.sha256hash160(key.getPubKey()));

// Account address
String walletAddress =
AddressTool.getStringAddressByBytes(address.getAddressBytes(),
address.getPrefix());
// Private key
String privateKey = key.getPrivateKeyAsHex();
```

三、Transaction

3.1 Main coin transfer

```
/** Transfer */
public void transfer() {
    String randomSeedNodeUrl = "http://47.241.101.72:8004/"; // Node address
    String privateKey = ""; // Send address private key
    String fromAddress = ""; // Send address
    String toAddress = "";  // Receive address;
    BigInteger amount = BigInteger.valueOf(10); // Transfer amount
    BigInteger scale = BigInteger.valueOf(8); // Precision
    String nonce = getNonce(randomSeedNodeUrl, fromAddress);
    BigInteger value = NumberUtil.mul(amount, scale ).toBigInteger();

    List<CoinFromDto> inputs = new ArrayList<>();
    List<CoinToDto> outputs = new ArrayList<>();
    TransferDto transferDto = new TransferDto();
    CoinFromDto coinFromDto = new CoinFromDto();
    CoinToDto coinToDto = new CoinToDto();
    coinFromDto.setAssetChainId(chainId);
    coinFromDto.setAssetId(assetId);
    coinFromDto.setAddress(fromAddress);
    coinFromDto.setAmount(value);
    coinFromDto.setNonce(nonce);
    inputs.add(coinFromDto);

    coinToDto.setAssetChainId(chainId);
    coinToDto.setAssetId(assetId);

    coinToDto.setAddress(toAddress);
    coinToDto.setAmount(value);
    outputs.add(coinToDto);

    transferDto.setInputs(inputs);
    transferDto.setOutputs(outputs);
    transferDto.setTime(DateUtil.currentSeconds());
    Result transferTxOffline1 =
NulsSDKTool.createTransferTxOffline(transferDto);

    Map<String, String> transferTxOffline = (Map) transferTxOffline1.getData();

    JSONObject j1 = new JSONObject();
```

```
    j1.put("txHex", transferTxOffline.get("txHex"));
    j1.put("address", fromAddress);
    j1.put("priKey", privateKey);
    RestFulResult<Map<String, Object>> resultSingResult = post(randomSeedNodeUrl
+ "api/account/priKey/sign", j1);

    JSONObject jsonObject1 =
JSONObject.parseObject(JSONObject.toJSONString(resultSingResult.getData()));
    SDKContext.wallet_url = randomSeedNodeUrl;
    Result<Map> result =
NulsSDKTool.broadcast(jsonObject1.get("txHex").toString());
}
```

3.2 Token transfer

```
public void tokenTransfer() {
    String randomSeedNodeUrl = "http://47.241.101.72:8004/"; // Node address
    String privateKey = ""; // Send address private key
    String fromAddress = ""; // Send address
    String toAddress = "";  // Receive address;
    BigInteger amount = BigInteger.valueOf(10); // Transfer amount
    BigInteger scale = BigInteger.valueOf(8); // Precision
    String contractAddress = ""; //Contract address
    BigInteger tokenBalance = BigInteger.valueOf(10);// Token balance of the
sending address

    BigInteger value = NumberUtil.mul(amount, scale ).toBigInteger();
    //Get trading gasLimit
    int gasLimit = getGasLimit(fromAddress, contractAddress, "transfer", new
Object[]{toAddress, value});
    String nonce = getNonce(randomSeedNodeUrl, fromAddress);

    //Assemble transaction
    JSONObject param = new JSONObject();
    param.put("fromAddress", fromAddress);
    param.put("senderBalance", tokenBalance);
    param.put("nonce", nonce);
    param.put("toAddress", toAddress);
    param.put("contractAddress", contractAddress);
    param.put("gasLimit", gasLimit);
    param.put("amount", value);
    String resultSing = HttpRequest.post(randomSeedNodeUrl +
"api/contract/tokentransfer/offline").body(param.toJSONString()).execute().body
();
```

```
    // String resultSing = HttpUtil.URLPostByJsonParameter(main +
"api/contract/tokentransfer/offline", param.toJSONString());
    JSONObject jsonObject = JSONObject.parseObject(resultSing);
    JSONObject jsonObject1 = (JSONObject) jsonObject.get("data");

    //Sign transaction
    param = new JSONObject();
    param.put("txHex", jsonObject1.get("txHex"));
    param.put("address", fromAddress);
    param.put("priKey", privateKey);
    RestFulResult<Map<String, Object>> resultSingResult = post(randomSeedNodeUrl
+ "api/account/priKey/sign", param);

    JSONObject jsonObject3 =
JSONObject.parseObject(JSONObject.toJSONString(resultSingResult.getData()));
    //Broadcast transaction
    SDKContext.wallet_url = randomSeedNodeUrl;
    Result<Map<String, Object>> result =
NulsSDKTool.broadcast(jsonObject3.get("txHex").toString());
}
```

3.3 Auxiliary methods used in transaction:

```
public static String getNonce(String baseUrl, String address) {
    Result<Map<String, String>> addressByPriKey =
getAccountBalance(randomSeedNodeUrl, address, chainId, assetId);
}

public static Result getAccountBalance(String baseUrl, String address, int chainId,
int assetsId) {
    validateChainId();

    Map<String, Object> params = new HashMap<>();
    params.put("assetChainId", chainId);
    params.put("assetId", assetsId);

    Result result;
    RestFulResult restFulResult = post(baseUrl + "api/accountledger/balance/" +
address, params);
    if (restFulResult.isSuccess()) {
        result = Result.getSuccess(restFulResult.getData());
    } else {
        ErrorCode errorCode =
ErrorCode.init(restFulResult.getError().getCode());
```

```
        result =
Result.getFailed(errorCode).setMsg(restFulResult.getError().getMessage());
    }
    return result;

}

public static RestFulResult<Map<String, Object>> post(String url, Map<String,
Object> params) {
    try {
        String resultStr =
HttpRequest.post(url).body(JSON.toJSONString(params)).execute().body();
        RestFulResult<Map<String, Object>> result = toResult(resultStr);
        return result;
    } catch (Exception e) {
        return RestFulResult.failed(CommonCodeConstanst.DATA_ERROR.getCode(),
e.getMessage(), null);
    }
}

private static RestFulResult toResult(String str) throws IOException {
    Map<String, Object> resultMap = JSONUtils.json2map(str);
    RestFulResult result = null;
    Boolean b = (Boolean) resultMap.get("success");
    if (b) {
        result = RestFulResult.success(resultMap.get("data"));
    } else {
        Object dataObj = resultMap.get("data");
        if (dataObj instanceof Map) {
            Map<String, Object> data = (Map<String, Object>)
resultMap.get("data");
            if (data != null) {
                result = RestFulResult.failed(data.get("code").toString(),
data.get("msg").toString());
            }
        } else {
            result =
RestFulResult.failed(CommonCodeConstanst.SYS_UNKOWN_EXCEPTION.getCode(),
resultMap.toString());
        }
    }
    return result;
}
```

```
public static int getGasLimit(String address, String contractAddress, String method,
Object[] args) {
    ImputedGasContractCallForm gasContractCallForm = new
ImputedGasContractCallForm();
    gasContractCallForm.setSender(address);
    gasContractCallForm.setContractAddress(contractAddress);
    gasContractCallForm.setMethodName(method);
    gasContractCallForm.setArgs(args);
    Result<Map> gasResult =
NulsSDKTool.imputedContractCallGas(gasContractCallForm);
    // Parse and obtain gasLimit
    return 0;
}
```

3.4 Main coin offline transfer

```
public void createTxSimpleTransferOfTDC() throws Exception {
    String fromAddress = "";  // Send address
    String toAddress = "";  // Receive address
    NulsSDKBootStrap.initMain("http://47.241.101.72:8004/");
    SDKContext.addressPrefix = "TDC";
    SDKContext.main_chain_id = 66;

    String value = "1.5";
    int tokenDecimals = 8;
    BigInteger amount = new
BigDecimal(value).multiply(BigDecimal.TEN.pow(tokenDecimals)).toBigInteger();

    Result<Map> result = NulsSDKTool.createTxSimpleTransferOfNuls(fromAddress,
toAddress, amount);
    String txHex = (String) result.getData().get("txHex");
    //Signature
    String prikey = "";  // Send address private key
    result = NulsSDKTool.sign(txHex, fromAddress, prikey);
    txHex = (String) result.getData().get("txHex");
    String txHash = (String) result.getData().get("hash");
    //Broadcast
    result = NulsSDKTool.broadcast(txHex);
    System.out.println(String.format("hash: %s", txHash));
}
```

3.5 Token offline transfer

```
public void tokenTransferTxOffline() throws Exception {
    String fromAddress = "";   // Send address
    String privateKey = "";  // Send address private key

    NulsSDKBootStrap.initMain("http://47.241.101.72:8004/");
    SDKContext.addressPrefix = "TDC";
    SDKContext.main_chain_id = 66;

    // Online interface (cannot be skipped, must be invoked) - String account balance
information
    Result accountBalanceR = NulsSDKTool.getAccountBalance(fromAddress,
SDKContext.main_chain_id, SDKContext.main_asset_id);
    String s = JSONUtils.obj2PrettyJson(accountBalanceR);
    Map balance = (Map) accountBalanceR.getData();
    BigInteger senderFeeBalance = new
BigInteger(balance.get("available").toString());
    String nonce = balance.get("nonce").toString();

    String toAddress = "";  // Receive address
    String contractAddress = "";  // Contract address

    int tokenDecimals = 8;
    // Transfer amount of tokens
    String tokenAmount = "10";
    BigInteger amount = new
BigDecimal(tokenAmount).multiply(BigDecimal.TEN.pow(tokenDecimals)).toBigIntege
r();

    String methodName = "transfer";
    String methodDesc = "";
    Object[] args = new Object[]{toAddress, amount};

    ImputedGasContractCallForm iForm = new ImputedGasContractCallForm();
    iForm.setSender(fromAddress);
    iForm.setContractAddress(contractAddress);
    iForm.setMethodName(methodName);
    iForm.setMethodDesc(methodDesc);
    iForm.setArgs(args);

    Result iResult = NulsSDKTool.imputedContractCallGas(iForm);
    Assert.assertTrue(JSONUtils.obj2PrettyJson(iResult), iResult.isSuccess());
```

```
    Map result = (Map) iResult.getData();
    Long gasLimit = Long.valueOf(result.get("gasLimit").toString());

    Result<Map> map = NulsSDKTool.tokenTransferTxOffline(fromAddress,
senderFeeBalance, nonce, toAddress, contractAddress, gasLimit, amount,
"tokenTransferTxOffline");
    String txHex = map.getData().get("txHex").toString();
    System.out.println("txHex : {}" +  txHex);

    // Signature

    Result res = NulsSDKTool.sign(txHex, fromAddress, privateKey);
    Map signMap = (Map) res.getData();
    // Online interface - broadcast transaction
    System.out.println("signMap.get(\"txHex\").toString() : {}" +
signMap.get("txHex").toString());
    Result<Map> broadcaseTxR =
NulsSDKTool.broadcast(signMap.get("txHex").toString());
    Assert.assertTrue(JSONUtils.obj2PrettyJson(broadcaseTxR),
broadcaseTxR.isSuccess());
    Map data = broadcaseTxR.getData();
    String hash1 = (String) data.get("hash");
    System.out.println(String.format("hash: %s", hash1));
}
```

四、Query method

4.1 Balance inquiry

Dependency injection:

```
public void getBalanceRpc() {
    String randomSeedNodeUrl =  "http://47.241.101.72:18003/jsonrpc";  // RPC
connection address
    String address = ""; // Address
    try {
        JsonRpcHttpClient client = new JsonRpcHttpClient(new
URL(randomSeedNodeUrl));
        Map result = client.invoke("getAccountBalance", new Object[]{CHAIN_ID,
TDCUtil.CHAIN_ID, 1, address}, Map.class, new HashMap<>(1));

    } catch (Throwable throwable)
```

```
    }
}
```

4.2 token balance inquiry

```
public void getTokenBalanceRpc() {
    String address = ""; // Address
    String contractAddress = ""; // Contract address
    String randomSeedNodeUrl =  "http://47.241.101.72:18003/jsonrpc";  // RPC
connection address

    BigDecimal balance = BigDecimal.ZERO;
    try {
        JsonRpcHttpClient client = new JsonRpcHttpClient(new
URL(randomSeedNodeUrl));
        JSONObject result = client.invoke("getAccountTokens", new
Object[]{CHAIN_ID, 1, 100, address}, JSONObject.class, new HashMap<>(1));
        if (result != null && result.containsKey("list") &&
result.getJSONArray("list").size() > 0) {
            List<JSONObject> list =
JSON.parseArray(result.getJSONArray("list").toJSONString(), JSONObject.class);
            for (JSONObject jsonObject : list) {
                if
(contractAddress.equals(jsonObject.getString("contractAddress"))) {
                    balance =
jsonObject.getBigDecimal("balance").movePointLeft(jsonObject.getInteger("decima
ls"));
                    break;
                }
            }
        }
    } catch (Exception e) {

    }
}
```

4.3 Query transaction details by hash

```
public Result tryRpcQueryTX() {
    String txHash = ""; // Transaction hash
```

```
    String randomSeedNodeUrl =  "http://47.241.101.72:18003/jsonrpc";  // RPC
connection address    try {
        Result result = null;
        Map<String, String> headers = new HashMap<String, String>(1);
        JsonRpcHttpClient client = new JsonRpcHttpClient(new
URL(randomSeedNodeUrl), headers);
        Map resultMap = client.invoke("getTx", new Object[]{CHAIN_ID, txHash},
Map.class, headers);
        if (resultMap != null) {
            result = Result.getSuccess(null);
            result.setData(resultMap);
            return result;
        }
    } catch (Throwable ex) {

    }
}
```

五、Query the interface list

Function explanation	Method Name	Parameters	Parameter description
Query blockchain information	getChainInfo	without	
Query general information about the blockchain operation	getInfo	[chainId]	
Query latest block header	getBestBlockHeader	[chainId]	
Query the block header by height	getHeaderByHeight	[chainId, blockHeight]	
Query the block header by hash	getHeaderByHash	[chainId, blockHash]	
Query the complete block by height	getBlockByHeight	[chainId, blockHeight]	
Query the complete block	getBlockByHash	[chainId, blockHash]	

by hash			
Query the block header list	getBlockHeaderList	[chainId, pageNumber, pageSize, isHidden, packedAddress]	chainId: int //Chain id pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value [1-1000] isHidden:boolean //Whether Hide blocks with only consensus reward transactions packedAddress:string //Filter by block packing address, not required
Query account details	getAccount	[chainId, address]	
Query account details by alias	getAccountByAlias	[chainId, alias]	
Query account on-chain asset List	getAccountLedgerList	[chainId, address]	
Query account single asset balance	getAccountBalance	[chainId, assetChainId,assetId, address]	chainId: int //Chain ID assetChainId: int //Asset chain ID assetId: int // Asset ID address: string //Account address
Query Availability of Alias	isAliasUsable	[chainId, alias]	
Query the transaction details	getTx	[chainId, txHash]	
Query transaction list	getAccountTxs	[chainId, pageNumber, pageSize, address, txType, isHidden]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] txType:int //Transaction type(txType),query all transactions when type=0 isHidden:boolean //Whether to hide consensus reward transactions(transaction when txType=1)

Query transactions packaged in block	getBlockTxList	[chainId, pageNumber, pageSize, blockHeight, txType]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] blockHeight:long //Block height txType:int //Transaction type (txType),query all transactions when type=0
Query the account transaction list	getAccountTxs	[chainId, pageNumber, pageSize, address, txType, isHidden]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] address: string //Account address txType:int //Transaction type(txType),query all transactions when type=0 isHidden:boolean //Whether to hide consensus reward transactions(transaction when txType=1)
Verify the legitimacy of offline assembled transactions	validateTx	[chainId, txHex]	chainId：int //Chain ID txHex：string // Hexadecimal string of Assembled transaction serialization
Broadcast offline assembly transactions	broadcastTx	[chainId, txHex]	chainId：int //Chain ID txHex：string // Hexadecimal string of Assembled transaction serialization
Query available delegated consensus node list	getConsensusNodes	[chainId, pageNumber, pageSize, type]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] type:int //Node type //0:All nodes,1:Normal Node,2:Developer node,3:Ambassador node

Query all delegated consensus node list	getAllConsensusNodes	[chainId, pageNumber, pageSize]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000]
Query the consensus node list delegated by the account	getAccountConsensus	[chainId, pageNumber, pageSize, address]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] address:string //Account address
Query the consensus node details	getConsensusNode	[chainId, txHash]	chainId: int //Chain ID txHash:string //The transaction hash when the node is created
Query the consensus node details created by the account	getAccountConsensusNode	[chainId, address]	
Query list Information in Node Delegation	getConsensusDeposit	[chainId, pageNumber, pageSize, txHash]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] txHash:string //The transaction hash when the node is created
Query Historical delegate list of node	getAllConsensusDeposit	[chainId, pageNumber, pageSize, txHash, type]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] txHash:string //The transaction hash when the node is created type:int //0:join delegate,1:Exit delegate,2:All
Query account delegate list	getAccountDeposit	[chainId, pageNumber,	chainId: int //Chain ID pageNumber:int //Page number

		pageSize, address, agentHash]	pageSize:int //Items number per page, retrieve Value[1-1000] address:string //Account address txHash:string //The transaction hash when the node is created,query all accounts if it is empty
Query total delegate amount of account	getAccountDepositValue	[chainId, address, agentHash]	chainId: int //Chain ID address:string //Account address txHash:string //The transaction hash when the node is created,query all accounts if it is empty
Query consensus penalties list	getPunishList	[chainId, pageNumber, pageSize, 0, agentAddress]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] type:int //Penalty type 0:All,1:yellow card,2:red card agentAddress:string //Delegate account address of consensus node
Query round list	getRoundList	[chainId, pageNumber, pageSize]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000]
Query contract details	getContract	[chainId, contractAddress]	chainId: int //Chain ID contractAddress:string //Smart contract address
Query contract list	getContractList	[chainId, pageNumber, pageSize, onlyNrc20, isHidden]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] Only Nrc20:boolean //Query only NRC 20 contracts isHidden: boolean //Whether to hide the nrc20 contract,this

			parameter is only valid if only NRC 20=false
Query contract-related transactions list	getContractTxList	[chainId, pageNumber, pageSize, txType, contractAddress]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] txType:int //Transaction type Query all transactions when the default is 0 contractAddress:string //Contract address
Query the NRC 20 contract transfer record list	getContractTokens	[chainId, pageNumber, pageSize, contractAddress]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] contractAddress:string //Contract address
Query the account NRC 20 transfer record list	getTokenTransfers	[chainId, pageNumber, pageSize, address, contractAddress]	chainId: int //Chain ID pageNumber:int //Page number pageSize:int //Items number per page, retrieve Value[1-1000] address:string //Account address contractAddress:string //Contract address
Transaction volume statistics	getTxStatistical	[chainId, type]	chainId: int //Chain ID type: int //0: Last 14 days, 1: Last week, 2: Last month, 3:Last year
Number statistics of consensus nodes	getConsensusNodeCount	[chainId]	
Consensus reward statistics	getConsensusStatistical	[chainId, type]	chainId: int //Chain ID type: int //0:14 days, 1:Weeks, 2: Months, 3: Years, 4: All
Annualized reward rate statistics	getAnnulizedRewardStatistical	[chainId, type]	type: int //0:14 days, 1:Weeks, 2: Months, 3: Years, 4: All