链滴

# 使用 Java SDK 实现离线签名

作者：bytom

原文链接：https://ld246.com/article/1553674372005

来源网站：链滴

严格来说，tx-signer并不属于SDK，它是bytomd中构建交易、对交易签名两大模块的java实现版。此，若想用tx-signer对交易进行离线签名，需要由你在本地保管好自己的私钥。

如果你的目的是完全脱离于bytomd全节点，可能需要自己做更多额外的工作。比如，在构建交易时需要花费若干个utxo（Unspent Transaction Output）作为交易的输入，如果没有全节点则需要自来维护utxo。当使用tx-signer构建完成一笔交易并签名后，若没有全节点的帮助，也需要自己实现P2网络协议将交易广播到其他节点。

本文不会对以上技术细节进行讨论，而是利用bytomd全节点查询可用的utxo构建交易，对交易进行名并序列化后，同样使用bytomd提交交易。

# 准备工作

## 将Maven依赖引入到你的项目中

1. 获取SDK源码

```
git clone https://github.com/Bytom/bytom-java-sdk.git
```

2. 打包成JAR包并安装到本地的Maven仓库

```
$ mvn clean install -DskipTests
```

3. 在项目的POM文件中添加依赖。其中，第一个依赖是bytomd api的封装，可用于查询可用的utxo及提交交易；第二个依赖用于构建交易以及对交易进行离线签名。

```
<dependency>
  <groupId>io.bytom</groupId>
  <artifactId>java-sdk</artifactId>
  <version>1.0.0</version>
</dependency>

<dependency>
  <groupId>io.bytom</groupId>
  <artifactId>tx-signer</artifactId>
  <version>1.0.0</version>
</dependency>
```

# 构建交易

## 普通交易

1. 查询可用的utxo

在本文中，以下将全部使用全节点来查询可用的utxo，你也可以构建一套自己的utxo维护方案。

```
Client client = Client.generateClient();
UnspentOutput.QueryBuilder builder = new UnspentOutput.QueryBuilder();
builder.accountAlias = "bytom";
List<UnspentOutput> outputs = builder.list(client);
```

利用SDK只需要四行代码就能查询可用的utxo（SDK具体文档详见java-sdk documentation）。在QeryBuilder中可以指定是否为未确认的utxo（默认false），也可以通过from和count来进行分页查询默认查询所有）。

假设在当前账户下查询得到这样一个utxo：


```
{
    "account_alias": "bytom",
    "id": "ffdc59d0478277298de4afa458dfa7623c051a46b7a84939fb8227083411b156",
    "asset_id": "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff",
    "asset_alias": "BTM",
    "amount": 41250000000,
    "account_id": "0G1R52O1G0A02",
    "address": "sm1qxls6ajp6fejc0j5kp8jwt2nj3kmsqazfumrkrr",
    "control_program_index": 1,
    "program": "001437e1aec83a4e6587ca9609e4e5aa728db7007449",
    "source_id": "2d3a5d920833778cc7c65d7c96fe5f3c4a1a61aa086ee093f44a0522dd499a34",
    "source_pos": 0,
    "valid_height": 4767,
    "change": false,
    "derive_rule": 0
}
```


2. 构建交易

现在需要往0014c832e1579b4f96dc12dcfff39e8fe69a62d3f516这个control program转100个BT。代码如下：

```
String btmAssetID = "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff";
// 下面的字段与utxo中的字段一一对应
SpendInput input = new SpendInput();
input.setAssetId(btmAssetID);
input.setAmount(41250000000L);
input.setProgram("001437e1aec83a4e6587ca9609e4e5aa728db7007449");
input.setSourcePosition(0);
input.setSourceID("2d3a5d920833778cc7c65d7c96fe5f3c4a1a61aa086ee093f44a0522dd499a
4");
input.setChange(false);
input.setControlProgramIndex(1);
// 选择使用BIP32还是BIP44来派生地址，默认BIP44
input.setBipProtocol(BIPProtocol.BIP44);
// 账户对应的密钥索引
input.setKeyIndex(1);
// 自身本地保管的私钥，用于对交易进行签名
input.setRootPrivateKey("4864bae85cf38bfbb347684abdbc01e311a24f99e2c7fe94f3c071d9c
3d8a5a349722316972e382c339b79b7e1d83a565c6b3e7cf46847733a47044ae493257");

Transaction tx = new Transaction.Builder()
            .addInput(input)
            // 加入需要转入的output
            .addOutput(new Output(btmAssetID, 10000000000L, "0014c832e1579b4f96dc12dcff
39e8fe69a62d3f516"))
            // 剩余的BTM用于找零
            .addOutput(new Output(btmAssetID, 31250000000L, "0014bb8a039726df1b649738
```

```
9973db14a4b4fd4becf"))
                .setTimeRange(0)
                .build();

String rawTransaction = tx.rawTransaction();
```

对交易调用build方法后，自动会对交易进行本地的验证和签名操作。注意，在本地只是做简单的字验证，本地验证通过并不代表交易合法。最后对交易调用rawTransaction方法返回交易序列化后的字串。

3. 提交交易

本文利用bytomd全节点来提交交易：

```
HashMap<String, Object> body = new HashMap<>();
body.put("raw_transaction", "070100010160015e4b5cb973f5bef4eadde4c89b92ee73312b940
84164da0594149554cc8a2adeaffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff80c480c124020
160014cb9f2391bafe2bc1159b2c4c8a0f17ba1b4dd94e6302401cb779288be890a28c5209036
a1a27d9fe74a51c38e0a10db4817bcf4fd05f68580239eea7dcabf19f144c77bf13d3674b5139aa
1a99ba58118386c190af0e20bcbe020b05e1b7d0825953d92bf47897be08cd7751a37adb95d6
2e5224f55ab02013dffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff80b095e42001160014a82f
2bc37bc5ed87d5f9fca02f8a6a7d89cdd5c000149ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
80d293ad03012200200824e931fb806bd77fdcd291aad3bd0a4493443a4120062bd659e64a3e
bac6600");
Transaction.SubmitResponse response = client.request("submit-transaction", body, Transacti
n.SubmitResponse.class);
```

交易提交成功后，response返回交易ID。

# 发行资产交易

1. 查询可用的utxo

发行资产时，需要使用BTM作为手续费，因此第一步同样需要查询当前账户下可用的utxo，由于上面经提到，这里不再赘述。

2. 查询需要发行的资产信息

例如，需要发行的资产id为7b38dc897329a288ea31031724f5c55bcafec80468a546955023380af2aad14

```
Asset.QueryBuilder builder = new Asset.QueryBuilder();
builder.setId("7b38dc897329a288ea31031724f5c55bcafec80468a546955023380af2faad14");
List<Asset> assets = builder.list(client);
```

假设查询得到的资产信息如下：

```
{
        "type": "asset",
        "xpubs": [
            "5ff7f79f0fd4eb9ccb17191b0a1ac9bed5b4a03320a06d2ff8170dd51f9ad9089c4038e
7280b5eb6745ef3d36284e67f5cf2ed2a0177d462d24abf53c0399ed"
        ],
```

```
        "quorum": 1,
        "key_index": 3,
        "derive_rule": 0,
        "id": "7b38dc897329a288ea31031724f5c55bcafec80468a546955023380af2faad14",
        "alias": "棒棒鸡",
        "vm_version": 1,
        "issue_program": "ae20db11f9dfa39c9e66421c530fe027218edd3d5b1cd98f24c826f4dc0cd131a475151ad",
        "raw_definition_byte": "7b0a202022646563696d616c73223a20382c0a2020226465736726970746 96f6e223a207b7d2c0a2020226e616d65223a2022222c0a20202273796d626f6c2232022220a7d",
        "definition": {
            "decimals": 8,
            "description": {},
            "name": "",
            "symbol": ""
        }
}
```


3. 构建交易

现在需要发行1000个棒棒鸡资产：

```
IssuanceInput issuanceInput = new IssuanceInput();
issuanceInput.setAssetId("7b38dc897329a288ea31031724f5c55bcafec80468a546955023380affaad14");
issuanceInput.setAmount(100000000000L);
// issue program
issuanceInput.setProgram("ae20db11f9dfa39c9e66421c530fe027218edd3d5b1cd98f24c826f49c0cd131a475151ad");
// 可以不指定，不指定时将随机生成一个
issuanceInput.setNonce("ac9d5a527f5ab00a");
issuanceInput.setKeyIndex(5);
// raw definition byte
issuanceInput.setRawAssetDefinition("7b0a202022646563696d616c73223a20382c0a2020226657363726970746 96f6e223a207b7d2c0a2020226e616d65223a2022222c0a20202273796d6266c223a2022220a7d");
// 该资产对应的私钥
issuanceInput.setRootPrivateKey("4864bae85cf38bfbb347684abdbc01e311a24f99e2c7fe94f3071d9c83d8a5a349722316972e382c339b79b7e1d83a565c6b3e7cf46847733a47044ae493257");

// 创建一个spend input作为手续费，假设当前有一个100BTM的utxo，并且使用1BTM作为手续费则后续还要创建99BTM的找零地址
SpendInput feeInput = new SpendInput(btmAssetID, 10000000000L, "0014cb9f2391bafe2bc159b2c4c8a0f17ba1b4dd94e");
feeInput.setKeyIndex(1);
feeInput.setChange(true);
feeInput.setSourceID("4b5cb973f5bef4eadde4c89b92ee73312b940e84164da0594149554cc8aadea");
feeInput.setSourcePosition(2);
feeInput.setControlProgramIndex(457);
feeInput.setRootPrivateKey("4864bae85cf38bfbb347684abdbc01e311a24f99e2c7fe94f3c071dc83d8a5a349722316972e382c339b79b7e1d83a565c6b3e7cf46847733a47044ae493257");
```

```
Transaction tx = new Transaction.Builder()
                .addInput(issuanceInput)
                .addInput(feeInput)
                // 该output用于接收发行的资产
                .addOutput(new Output("7b38dc897329a288ea31031724f5c55bcafec80468a54695
023380af2faad14", 100000000000L, "001437e1aec83a4e6587ca9609e4e5aa728db7007449"))
                // 找零
                .addOutput(new Output(btmAssetID, 9800000000L, "00148be1104e04734e5edaba5
ea2e85793896b77c56"))
                .setTimeRange(0)
                .build();
```

4. 提交交易

提交交易的方式与普通交易一致。

## 销毁资产交易

销毁资产跟发行资产类似，同样需要BTM作为手续费。

1. 查询可用的utxo

查询方式与普通交易一致。

2. 构建交易

这里以销毁一个BTM为例，假设查询得到一个100BTM的utxo：

```
// 查询得到一个100BTM的utxo作为输入
SpendInput input = new SpendInput(btmAssetID, 10000000000L, "0014f1dc52048f439ac7fd7
f8106a21da78f00de48f");
input.setRootPrivateKey(rootKey);
input.setChange(true);
input.setKeyIndex(1);
input.setControlProgramIndex(41);
input.setSourceID("0b2cff11d1d056d95237a5f2d06059e5395e86f60e69c1e8201ea624911c0c
5");
input.setSourcePosition(0);

// 销毁资产时，可添加一段附加的文本
String arbitrary = "77656c636f6d65efbc8ce6aca2e8bf8ee69da5e588b0e58e9fe5ad90e4b896e
958c";
// 销毁99个BTM，剩余1个BTM作为手续费
Output output = Output.newRetireOutput(btmAssetID, 9900000000L, arbitrary);

Transaction transaction = new Transaction.Builder()
                .addInput(input)
                .addOutput(output)
                .setTimeRange(2000000)
                .build();

String rawTransaction = transaction.rawTransaction();
```

3. 提交交易

提交交易的方式与普通交易一致。

bytom java sdk：https://github.com/Bytom/bytom-java-sdk/