

Ontology JavaSDK 自动测试工具介绍

一、Ontology JavaSDK 自动测试工具简介

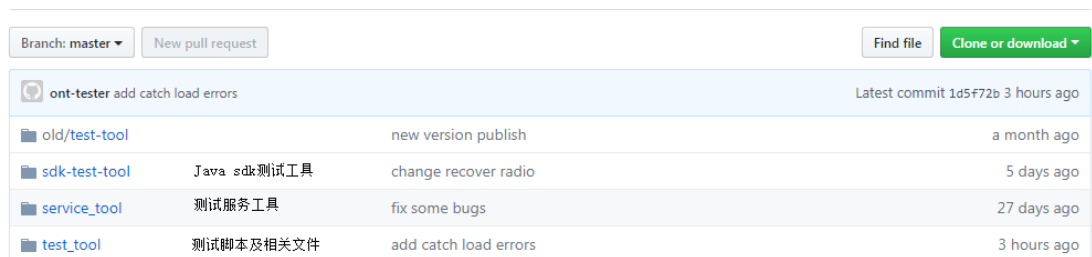
本工具是面向 ontology Java SDK 开发的自动化测试工具。工具能够收集测试结果并记录测试过程，形成测试报告。其测试内容涵盖 Ontology JavaSDK 的 RPC API、Restful API、Websocket API、Digital asset account、Digital identity、ONT Native、ONG Native、ontid、Claim、Claim record、Invoke 和助记词这十二个部分。

二、测试环境及自动测试工具框架简介

平台：windows

Java version：1.7 及以上

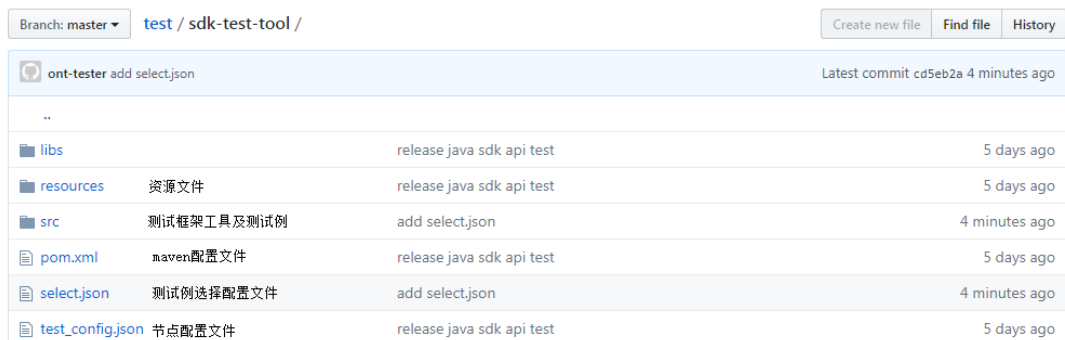
在 GitHub 上下载 ontology 测试相关文件 test-master，下载文件包含内容如图 2-1 所示。



Branch: master		New pull request	Find file	Clone or download
ont-tester add catch load errors		Latest commit 1d5f72b 3 hours ago		
old/test-tool		new version publish	a month ago	
sdk-test-tool	Java sdk测试工具	change recover radio	5 days ago	
service_tool	测试服务工具	fix some bugs	27 days ago	
test_tool	测试脚本及相关文件	add catch load errors	3 hours ago	

图 2-1

其中 sdk-test-tool 即为 ontology java sdk 的自动测试工具所在路径。sdk-test-tool 中内容如下图 2-2 所示：



Branch: master		test / sdk-test-tool /	Create new file	Find file	History
ont-tester add select.json		Latest commit cd5eb2a 4 minutes ago			
..					
libs		release java sdk api test	5 days ago		
resources	资源文件	release java sdk api test	5 days ago		
src	测试框架工具及测试例	add select.json	4 minutes ago		
pom.xml	maven配置文件	release java sdk api test	5 days ago		
select.json	测试例选择配置文件	add select.json	4 minutes ago		
test_config.json	节点配置文件	release java sdk api test	5 days ago		

图 2-2

其中 bin 文件夹下为编译后的文件，libs 下为 ontology-sdk-java-1.0.jar，logs 下存放了测试例执行后的日志文件。resources 中包含 neo 和 wallets，neo 中为合约的 sample.cs 和 sample.neo 文件，wallets 中包含以“wallet”+编号命名的钱包文件。src 下为测试框架工具及测试例集合，pom.xml 为 Maven 配置文件，select.json 用于按类型选择测试例集合（详见见第四章的第 2 节中介绍），test_config.json 则为节点配置文件（详见见第三章的第 2 节中介绍）。

三、运行测试脚本

运行测试脚本步骤：

1. 启动整个区块链。使用远程工具连接节点，利用基于 python3 的自检文件启动区块链节点（与 ontology 自动化测试工具介绍中一致，若已经配置好相关配置可以略过该步骤，需要在节点服务器环境下运行以下步骤）：

在 GitHub 上下载 ontology 测试相关文件，下载文件 test-master 打开后如图 2-1 所示。首先将下载的 test-master 中的 service_tool 和 test_tool 配置到 node 文件夹同级目录/home/ubuntu/ontology/下（必须同级目录），然后按照以下步骤安装所需 python 模块：

```
$ cd /home/ubuntu/ontology/service_tool/
```

```
$ chmod 777 install.sh
```

在/home/ubuntu/ontology/test_tool/tools/init_selfcheck 路径下包含了自检文件及其配置文件，另外在 test-master/test_tool/resource 下有自检文件中需要使用的最新文件。在实现自检之前需要修改配置文件中的内容（路径为/home/ubuntu/ontology/test_tool/tools/init_selfcheck/config.json），配置修改示例如下：

```
{
  "resource":{
    "root":"/home/ubuntu/ontology/test_tool/resource",
    "ontology_source_name": "ontology",
    "ontology_dbft_1_source_name": "ontology-bft_1",
    "ontology_dbft_2_source_name": "ontology-bft_2",
    "ontology_dbft_3_source_name": "ontology-bft_3",
    "wallet_source_name": "wallet",
    "onto_config_source_name": "onto_config.json",
    "test_config_source_name": "test_config.json",
    "sigsvr_source_name": "sigsvr",
    "abi_source_name": "abi",
    "test_service_source_name": "rpcserver.py"
  },
  "node":{
    "root":"/home/ubuntu/ontology/node/",
    "onto_name": "ontology",
    "ontology_dbft_1_name": "ontology-bft_1",
    "ontology_dbft_2_name": "ontology-bft_2",
    "ontology_dbft_3_name": "ontology-bft_3",
    "wallet_name": "wallet.dat",
    "onto_config_name": "config.json",
    "sigsvr_name": "sigsvr",
    "abi_name": "abi"
  },
  "test_config_path": "/home/ubuntu/ontology/test_tool/config.json ",
  "default_node_args": " --ws --rest --loglevel=0 --enableconsensus --networkid=6666 --
gasprice 0 --gaslimit 20000 ",
  "test_service_path": "/home/ubuntu/ontology/service_tool/rpcserver.py"
}
```

该配置文件内容的 resource 中的 root 为自检文件相关 resource 文件夹的路径。node 中的 root 为 node 文件夹路径。当然如若节点服务器内 node 文件夹中的 config.json，sigsvr，abi，ontology，wallet.dat 等作出命名上改动，在配置文件中"node"下的"onto_name"，"wallet.dat"，"config.json"，"sigsvr_name"，"abi_name"的内容都需要做相应变化。其中 test_config_path 为 test_tool 下 test_config.json 所在路径，而 test_service_path 为测试服务 rpcserver.py 文件所在路径

(即/home/ubuntu/ontology/service_tool/rpcserve.py), 其他配置默认即可。

除此以外/home/ubuntu/ontology/test_tool/resource 下的 test_config.json 内容需要修改如下:

```
"RPC_URL": "http:// 127.0.0.1:20336/jsonrpc",
"CLIRPC_URL": "http://127.0.0.1:20000/cli",
"RESTFUL_URL": "http:// 127.0.0.1:20334",
"WS_URL": "ws:// 127.0.0.1:20335",
"NODES": [{
    "ip": "139.219.128.177",
    "wallet": "wallet00.dat"
},
{
    "ip": "139.219.141.110",
    "wallet": "wallet01.dat"
},
...
],
"MULTI_SIGNED_ADDRESS": "AJroHBLRsfpTPQocSKjpADp4AKybg8GZuz",
"INIT_AMOUNT_ONG": "1000000000000000",
"DEFAULT_NODE_ARGS": "--ws --rest --loglevel=0 --enableconsensus --networkid=299 --
gasprice 0 --gaslimit 20000 ",
"NODE_PATH": "/home/ubuntu/ontology/node"
}
```

其中 NODES 中的 ip 需要改为实际 16 个节点服务器的 ip 地址 (前七个默认为创世共识节点),
NODE_PATH 为节点中的 node 文件夹路径, 其他配置默认即可。RPC_URL, CLIRPC_URL,
RESTFUL_URL, WS_URL 修改如下:

```
"RPC_URL": "http:// 127.0.0.1:20336/jsonrpc",
"CLIRPC_URL": "http://127.0.0.1:20000/cli",
"RESTFUL_URL": "http:// 127.0.0.1:20334",
"WS_URL": "ws:// 127.0.0.1:20335",
```

/home/ubuntu/ontology/test_tool/resource 下的 onto_config.json 内容如下:

```
"ConsensusType": "vbft",
"VBFT": {
    "peer_handshake_timeout": 10,
    "max_block_change_view": 1000,
    "admin_ont_id": "did:ont:AG4pZwKa9cr8ca7PED7FqzUfcwnrQ2N26w",
    "peers": [
        {
            "address": "AG4pZwKa9cr8ca7PED7FqzUfcwnrQ2N26w",
            "peerPubkey":
"02e3ff56206295e71295aeaf47b7f3681f93fb98b692a66b103feb7b756667c74d",
            "initPos": 0,
            "index": 1
        },
        {
```

```

        "address": "AUSyZx1BoQbTy3avMKygvJrkW8mRbzt6Aa",
        "peerPubkey":
"02ca35fd7f3d78f01a423e17a85fe122715c2bcece7c5c8c851adcb265eb59486c",
        "initPos": 0,
        "index": 2
    },
    .....
],
    "vrf_value":
"1c9810aa9822e511d5804a9c4db9dd08497c31087b0daafa34d768a3253441fa20515e2f30f81741102af
0ca3cefc4818fef16adb825fbaa8cad78647f3afb590e",
    "min_init_stake": 10000,
    "hash_msg_delay": 10000,
    "l": 112,
    "k": 7,
    "c": 2,
    "n": 7,
    "block_msg_delay": 10000,
    "vrf_proof":
"c57741f934042cb8d8b087b44b161db56fc3ffd4ffb675d36cd09f83935be853d8729f3f5298d12d6fd28d
45dde515a4b9d7f67682d182ba5118abf451ff1988"
    },
    "SeedList": [
        "139.219.128.177:20338",
        "139.219.141.110:20338",
        "139.219.141.33:20338",
        "139.219.136.233:20338"
    ]
}

```

注意 onto_config.json 中 ConsensusType 为共识机制根据实际情况为“vbft”或“dbft”。

admin_ont_id 中的内容为“did:ont:”加上 index 为 1 的节点 address。peers 中填入七个节点的相关信息，address 为节点 wallet 中的 address，peerPubkey 为节点 pubkey，index 为节点 pubkey，initPos 为设置的 initpos。SeedList 中填入节点 index 为 1 到 4 的节点 ip 地址格式如上。

最后根据如下操作完成自检：

```

$ cd /home/ubuntu/ontology/test_tool/tools/init_selfcheck
$ chmod 777 init_selfcheck.py
$ python3 init_selfcheck.py

```

配置相关文件后，即可使用自检文件重启节点。

（自检文件除了重启节点外，还会检查 ontology 的执行权限、版本和 MD5，abi 下所有文件 MD5，测试服务版本以及 sig 的执行权限、版本和 MD5，并将 test_config.json 和 onto_config.json 与 test-master\test_tool\resource 下的 wallet 依次进行确认和修正，修正结束后将 onto_config.json 和 wallet 文件夹覆盖到 16 个节点服务器下，最终重启所有节点服务器和 sig 服务并检查重启后节点数量是否正常）。

2. 配置 ont java sdk 测试框架（该框架需要在 windows 环境下运行，以下步骤均在 windows 环境

下执行)：

在 GitHub 上下载 ontology 测试相关文件，下载文件 test-master 打开后如图 2-1 所示。首先将下载的 test-master 中的 sdk-test-tool 配置到任意路径，介绍中使用 C 盘，如果路径有所改动请在后续步骤中相应修改。在 C:\sdk-test-tool 下文件如下图示：

Branch: master test / sdk-test-tool /

Create new fileFind fileHistory

ont-tester add select.jsonLatest commit cd5eb2a 4 minutes ago

..

libs

release java sdk api test

5 days ago

resources

资源文件

release java sdk api test

5 days ago

src

测试框架工具及测试例

add select.json

4 minutes ago

pom.xml

maven配置文件

release java sdk api test

5 days ago

select.json

测试例选择配置文件

add select.json

4 minutes ago

test_config.json

节点配置文件

release java sdk api test

5 days ago

其中的节点配置文件 test_config.json 中内容如下：

```
{
  "RPC_PORT": "20336",
  "CLI_PORT": "20000",
  "RESTFUL_PORT": "20334",
  "WS_PORT": "20335",
  "TEST_SERVICE_PORT" : "23635",
  "TEST_MODE":false,
  "DEFAULT_NODE_ARGS" : "--ws --rest --loglevel=0 --enableconsensus --networkid=377 --
gasprice=0 --gaslimit=20000",
  "MULTI_SIGNED_ADDRESS" : "",
  "INIT_AMOUNT_ONG" : "7000000000000000",
  "PWD" : "123456",
  "NODES": [
    {
      "ip": "139.219.138.201",
      "wallet" : "wallet00.dat"
    },
    {
      "ip": "42.159.154.145",
      "wallet" : "wallet01.dat"
    },
    ... ..
  ]
}
```

TEST_MODE 后的 true 或 false 代表是否为测试模式下的单节点，正常情况默认选择 false 即可。PWD 为 wallet 的 password。NODES 中需要填写测试对应 16 台节点的 ip 及对应的 wallet。其他配置默认即可

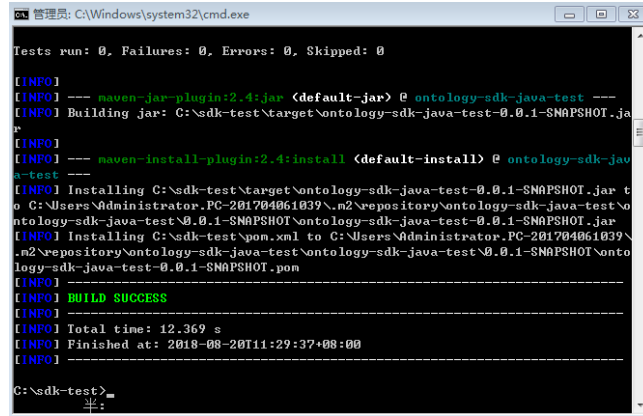
3. 配置相关文件后，打开 cmd 窗口，输入以下命令进行相关测试：
随后进入测试相关路径进行编译：

```
cd C:\sdk-test-tool
```

```
mvn install
```

```
mvn compile
```

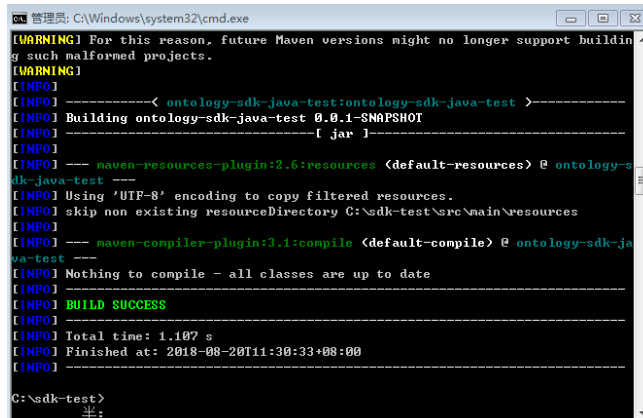
mvn install 的结果如图 3-1， mvn compile 的结果如图 3-2。



```
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ ontology-sdk-java-test ---
[INFO] Building jar: C:\sdk-test\target\ontology-sdk-java-test-0.0.1-SNAPSHOT.jar
[INFO] --- maven-install-plugin:2.4:install (default-install) @ ontology-sdk-java-test ---
[INFO] Installing C:\sdk-test\target\ontology-sdk-java-test-0.0.1-SNAPSHOT.jar to C:\Users\Administrator.PC-201704061039\.m2\repository\ontology-sdk-java-test\ontology-sdk-java-test\0.0.1-SNAPSHOT\ontology-sdk-java-test-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\sdk-test\pom.xml to C:\Users\Administrator.PC-201704061039\.m2\repository\ontology-sdk-java-test\ontology-sdk-java-test\0.0.1-SNAPSHOT\ontology-sdk-java-test-0.0.1-SNAPSHOT.pom
[INFO] --- BUILD SUCCESS ---
[INFO] Total time: 12.369 s
[INFO] Finished at: 2018-08-20T11:29:37+08:00
[INFO]
```

图 3-1



```
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO] ---< ontology-sdk-java-test:ontology-sdk-java-test >-----
[INFO] Building ontology-sdk-java-test 0.0.1-SNAPSHOT
[INFO] ---[ jar ]-----
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ ontology-sdk-java-test ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\sdk-test\src\main\resources
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ ontology-sdk-java-test ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- BUILD SUCCESS ---
[INFO] Total time: 1.107 s
[INFO] Finished at: 2018-08-20T11:30:33+08:00
[INFO]
```

图 3-2

注意：如果编译出现找不到符号的错误，请确认使用的jdk版本是否为1.7以上，确认后再检查pom文件配置并添加相应的jar包依赖（如遇其他编译问题，请根据具体的错误信息分析修改）。

最后输入以下命令测试：

```
mvn -e exec:java -Dexec.mainClass="com.ontio.RunAllTest" -Dexec.args="" -Dexec.classpathScope="test"
```

其中的-Dexec.args=""中可以填入C:\sdk-test-tool\src\test\java\com\ontio下RunAllTest.java所需的参数。-Dexec.args等于空或去掉“-Dexec.args=""”都为运行所有测试例，如需选择性测试详见第四章具体说明。


```

    "Ontid" : false
}

```

在上面的 config 配置表示，想要运行 ONT_Native 和 DigitalAccount 的 java 测试文件。

2. 打开文件夹后，运行要一定要在\test-master\sdk-test-tool 目录下打开命令窗口，然后输入以下命令：`mvn -e exec:java -Dexec.mainClass="com.ontio.RunAllTest" -Dexec.args="-c select.json" -Dexec.classpathScope="test"`

select.json 即为根据需要选择运行测试用例的 json 文件，如果在文件夹中，json 文件的名字不是 select.json，则在命令窗口中，名字也需要进行相应的修改，运行此命令，得到的结果是选择运行的所有文件运行一遍。命令窗口中，还提供了其他命令：

- 1) 使用-f 进行操作：根据需要选择某个单条的 case 进行测试，修改命令为 `mvn -e exec:java -Dexec.mainClass="com.ontio.RunAllTest" -Dexec.args="-c select.json -f 类名.方法名" -Dexec.classpathScope="test"`

例如，运行 DigitalAccount 中的第一个测试例，则在这段命令里需要这么输入：`mvn -e exec:java -Dexec.mainClass="com.ontio.RunAllTest" -Dexec.args="-c select.json -f DigitalAccount.test_base_001_importAccount" -Dexec.classpathScope="test"`

- 2) 使用-t 进行操作：根据需要选择不同的类型的测试用例进行测试，即根据 base normal abnormal 三种类型进行选择，通过-c select.json 选择需要运行的模块后，再使用-t 选择想要的类型。

例如，运行 DigitalAccount 下所有 normal 的测试例，则输入命令：`mvn -e exec:java -Dexec.mainClass="com.ontio.RunAllTest" -Dexec.args="-c select.json -t normal" -Dexec.classpathScope="test"`。

- 3) 使用-e 进行操作：通过-c select.json 选择需要运行的模块后，根据需要并不要按照 base、normal、abnormal 的分类测试，也不需要单条测试，目标是排除一些不需要运行的测试用例，可使用-e 来进行操作。

例如，在想运行 DigitalAccount 下排除第一条 test_base_001_importAccount 的测试用例，则输入：`mvn -e exec:java -Dexec.mainClass="com.ontio.RunAllTest" -Dexec.args="-c select.json -e DigitalAccount.test_base_001_importAccount" -Dexec.classpathScope="test"`

例如排除多条不需要运行的测试用例，则排除的每一条后面加一个逗号，例如想同时排除第一条和第二条：`mvn -e exec:java -Dexec.mainClass="com.ontio.RunAllTest" -Dexec.args="-c select.json -e DigitalAccount.test_base_001_importAccount,DigitalAccount.test_abnormal_002_importAccount" -Dexec.classpathScope="test"`

3. 运行过程中如果想要终止，则按 Ctrl+C，会提示是否终止批处理操作，选择 y 表示 yes，终止即可测试用例的继续运行。如下图所示：

```

2018-08-17 09:36:23: [-----]
2018-08-17 09:36:23: [ INFO ]
2018-08-17 09:36:23: [ DESCRIPTION ]
2018-08-17 09:36:23: [ STEP ]
2018-08-17 09:36:23: [ STEP ]
终止批处理操作吗(Y/N)?

```

图 4-2

4. 在 test-master\sdk-test-tool\logs 文件夹下，存放的是运行测试用例后记录的 log，文件夹的名称是以时间的方式进行命名，时间为开始运行该测试用例开始的时间。
点入该文件夹，文件夹以测试模块的不同进行命名，根据想要查看的 log 点入相应的文件夹

中。

在该文件夹下，存在两种类型的文件，文件格式为 CSV 的是运行测试用例的汇总表格，log 格式的是每一条运行的 log 文件。如图所示：

名称	修改日期	类型	大小
collection.csv	2018/8/16 14:41	XLS 工作表	2 KB
test_base_001_getNodeCount.log	2018/8/16 14:36	LOG 文件	1 KB
test_base_002_getBlock.log	2018/8/16 14:36	LOG 文件	1 KB
test_base_003_getBlockJson.log	2018/8/16 14:36	LOG 文件	1 KB
test_base_004_getBlock.log	2018/8/16 14:36	LOG 文件	1 KB
test_base_005_getBlock.log	2018/8/16 14:36	LOG 文件	1 KB
test_base_006_getBlockHeight.log	2018/8/16 14:36	LOG 文件	1 KB
test_base_007_getTransaction.log	2018/8/16 14:36	LOG 文件	2 KB
test_base_008_getStorage.log	2018/8/16 14:37	LOG 文件	6 KB
test_base_010_getBalance.log	2018/8/16 14:37	LOG 文件	1 KB

图 4-3

在 CSV 文件中，每一条运行的 log 和形成一张 CSV 的表格，表格中包含每一条测试用例的方法名，测试结果和测试 log 名称。该表格文件可以清晰明了的反映运行的测试用例运行结果如何，如图所示：

	A	B	C	D	E	F
1	NAME	STATUS	LOG PATH			
2	test_base_001_importAccount	pass	test_base_001_importAccount.log			
3	test_abnormal_002_importAccount	pass	test_abnormal_002_importAccount.log			
4	test_abnormal_003_importAccount	fail	test_abnormal_003_importAccount.log			
5	test_abnormal_004_importAccount	pass	test_abnormal_004_importAccount.log			
6	test_abnormal_005_importAccount	fail	test_abnormal_005_importAccount.log			
7	test_abnormal_006_importAccount	pass	test_abnormal_006_importAccount.log			
8	test_normal_007_importAccount	pass	test_normal_007_importAccount.log			
9	test_abnormal_008_importAccount	pass	test_abnormal_008_importAccount.log			
10	test_abnormal_009_importAccount	pass	test_abnormal_009_importAccount.log			
11	test_normal_010_importAccount	pass	test_normal_010_importAccount.log			
12	test_abnormal_011_importAccount	pass	test_abnormal_011_importAccount.log			
13	test_abnormal_012_importAccount	pass	test_abnormal_012_importAccount.log			
14	test_normal_013_importAccount	pass	test_normal_013_importAccount.log			
15	test_abnormal_014_importAccount	pass	test_abnormal_014_importAccount.log			
16	test_abnormal_015_importAccount	pass	test_abnormal_015_importAccount.log			
17	test_abnormal_016_importAccount	pass	test_abnormal_016_importAccount.log			
18	test_abnormal_017_importAccount	pass	test_abnormal_017_importAccount.log			
19	test_abnormal_018_importAccount	pass	test_abnormal_018_importAccount.log			
20	test_base_019_createAccount	pass	test_base_019_createAccount.log			
21	test_normal_020_createAccount	pass	test_normal_020_createAccount.log			
22	test_normal_021_createAccount	pass	test_normal_021_createAccount.log			
23	test_normal_022_createAccount	pass	test_normal_022_createAccount.log			
24	test_base_023_createAccountFromPriKey	pass	test_base_023_createAccountFromPriKey.			

图 4-4

5. log 文件夹下存在一个 xlsx 文件,打开文件，第一张 sheet 名为“java sdk 测试-总计”的，是所有测试用例运行结束后的结果进行的数据统计。

使用方法：

首先点击“清空目前记录”，清空表格中所有以往的记录。

再点击“填入目前所有记录”，在测试结果统计表中，会自动统计每一模块相应的数据，如图所示：

测试结果统计										
组件	总数	尚未执行	尚未执行[%]	通过	通过[%]	失败	失败[%]	锁定	锁定[%]	已完成[%]
交互接口	72	0	0.00%	67	93.06%	5	6.94%	0	0.00%	100.00%
Wallet manager	146	0	0.00%	125	85.62%	21	14.38%	0	0.00%	100.00%
Digit asset	158	0	0.00%	148	93.67%	10	6.33%	0	0.00%	100.00%
Digit identity	141	0	0.00%	112	79.43%	29	20.57%	0	0.00%	100.00%
Neo smart contract	26	0	0.00%	17	65.38%	9	34.62%	0	0.00%	100.00%
deploy and invoke										
助记词和keystore接口	58	0	0.00%	31	53.45%	27	46.55%	0	0.00%	100.00%
总计	601	0	0.00%	500	83.19%	101	16.81%	0	0.00%	100.00%

图 4-5

最后点击“填入目前消耗时间”，则在第 2 张 sheet “java sdk 测试”中，会统计执行时间，在“测试结果细分”表中，可知道每一个测试模块需要的时间和总时间。

从第 4 张 sheet 开始，是每一个模块的具体测试用例，在“log 信息”一列有每一条 log 信息的链接，点击可以查看，记录的是每一条 log 的具体内容。