

Ontology JavaSDK 自动测试工具接口说明

一、整体框架

1. 测试框架工具及测试例在 ontology JavaSDK 自动测试工具 test-master\sdk-test-tool 文件夹下的 src 中。

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 1-1

2. 测试框架工具及测试例 src 下结构如下图 1-2 所示：

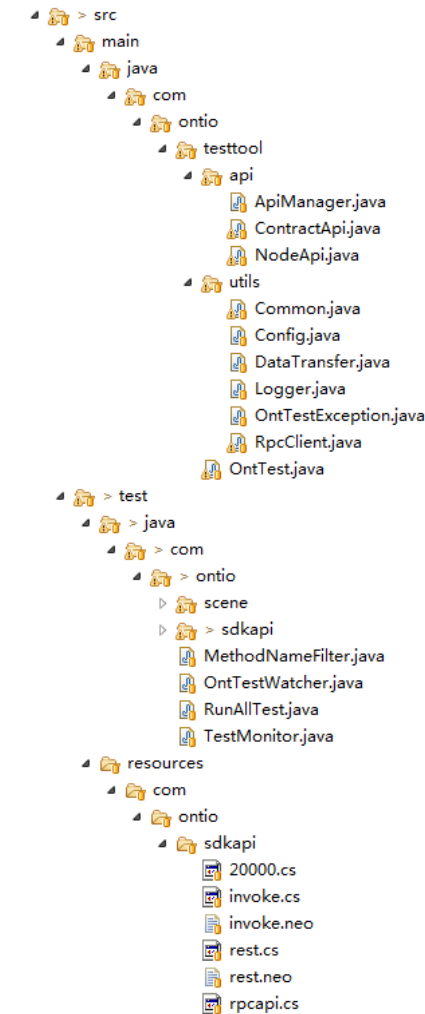


图 1-2

其中 `src\main\java\com\ontio\testtool` 中包含测试框架相关 api 接口，`src\test\java\com\ontio` 中包含测试例相关工具和测试例集合，路径 `src\test\java\com\ontio` 下的 `sdkapi` 文件夹用于存放所有的测试例集合，而路径 `src\test\java\com\ontio` 下的 `scene` 文件夹存放了 `sample.java`，路径 `src\test\resources\com\ontio` 下的 `sdkapi` 文件夹用于存放测试例所需要的智能合约文件。测试例集合文件夹 `src\test\java\com\ontio\sdkapi` 下文件如图所示，文件名称与 javaSDK 测试例 excel 表格的 sheet 名称对应：

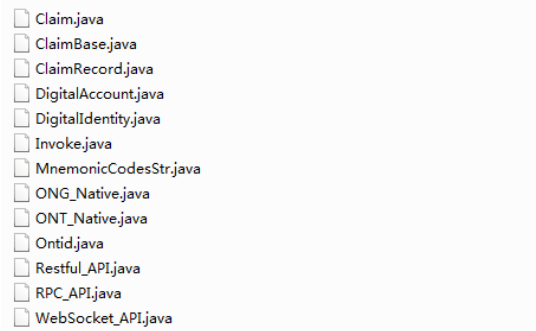


图 1-3

3. 从第二章开始将会按照以下结构对各文件及接口进行详细说明，可以按住 `ctrl` 并单击文件名或方法名跳转至相应内容：

1) [src/main/java](#)



- └─[restartAll\(String program, String config, String nodeargs\)](#)
- └─[restart\(int\[\] nodeindexs, String program, String config, String nodeargs\)](#)
- └─[restart\(int nodeindex, String program, String config, String nodeargs\)](#)
- └─[replaceConfigAll\(String path\)](#)（预留接口）
- └─[replaceConfig\(int\[\] nodeindexs, String path\)](#)（预留接口）
- └─[replaceConfigAll\(JSONObject cfg\)](#)（预留接口）
- └─[replaceConfig\(int\[\] nodeindexs, JSONObject cfg\)](#)（预留接口）
- └─[checkNodeSync\(\)](#)
- └─[getStatesMd5\(\)](#)
- └─[getBlockMd5\(\)](#)
- └─[getLedgereventMd5\(\)](#)
- └─[exec\(\)](#)（预留接口）
- └─[initOntOng\(\)](#)
- └─com.ontio.testtool.utils
 - └─[Common.java](#)
 - └─[TxStates](#) 类
 - └─[TxEvent](#) 类
 - └─Common 类
 - └─[getAccount\(\)](#)
 - └─[getDefaultAccount\(\)](#)
 - └─[loadJson\(\)](#)
 - └─[waitGenBlock\(\)](#)
 - └─[waitTransactionResult\(\)](#)
 - └─[waitWsResult\(\)](#)
 - └─[Config.java](#)
 - └─[nodeIp\(\)](#)
 - └─[nodeWallet\(\)](#)
 - └─[rpcUrl\(\)](#)
 - └─[restfulUrl\(\)](#)
 - └─[wsUrl\(\)](#)
 - └─[cliUrl\(\)](#)
 - └─[testServiceUrl\(\)](#)
 - └─[json\(\)](#)
 - └─[DataTransfer.java](#)
 - └─[Logger.java](#)
 - └─[getInstance\(\)](#)
 - └─[Logger\(\)](#)
 - └─[getPrefixPath\(\)](#)
 - └─[logfile\(\)](#)
 - └─[print\(\)](#)
 - └─[error\(\)](#)
 - └─[warning\(\)](#)
 - └─[description\(\)](#)
 - └─[step\(\)](#)

- └─[open\(\)](#)
- └─[appendRecord\(\)](#)
- └─[close\(\)](#)
- └─[state\(\)](#)
- └─[setBlock\(\)](#)
- └─[write\(\)](#)
- └─[OntTestException.java](#)
- └─[RpcClient.java](#)
 - └─[RpcClient\(\)](#)
 - └─[getHost\(\)](#)
 - └─[call\(\)](#)
 - └─[makeRequest\(\)](#)
 - └─[send\(\)](#)

2) [src/test/java](#)

- └─com.ontio
 - └─[MethodNameFilter.java](#)
 - └─[MethodNameFilter\(\)](#)
 - └─[shouldRun\(\)](#)
 - └─[OntTestWatcher.java](#)
 - └─[RunAllTest.java](#)
 - └─[TestMonitor.java](#)
- └─com.ontio.scene(例子 sample.java, 不进行详细说明)
- └─com.ontio.sdkapi

二、 测试框架及相关 api 接口说明

1. OntTest.java 详细说明

OntTest 类下首先声明了五个静态变量：ontSdk, api, logger, common 和 wsLock。随后七个有静态方法，分别为 init(), wsLock(), bindNode(int index), sdk(), api(), logger(), common()。各方法具体说明如下：

- 1) init(): 使得使用 system.out.println 输出便能够在日志文件中记录相关信息。建立钱包文件的临时文件 wallet.dat.tmp。
- 2) wsLock(): 当变量 wsLock 有值返回原值，若其为 null 则定义一个 Object 的 wsLock 并返回。
- 3) bindNode(int index) : 根据输入参数 index 建立 Rpc、Restful、Websocket 连接设置，设置签名服务，设置默认连接类型，打开临时钱包文件。
- 4) sdk(): 实例化 com.github.ontio 下 OntSdk 的 getInstance()类。
- 5) api(): 实例化 com.ontio.testtool.api 下的 ApiManager()类。
- 6) logger(): 实例化 com.ontio.testtool.utils 下的 Logger()类。
- 7) common(): 实例化 com.ontio.testtool.utils 下的 Common()类。

2. api 下的 ApiManager.java 详细说明

ApiManager 用于 api 接口管理，内部有 contract()和 node ()这两个方法，说明如下：

- 1) contract(): 实例化同文件路径中的 ContractApi()类，用于调用合约相关 API 接口。
- 2) node ()：实例化同文件路径中的 NodeApi()类，用于调用节点相关 API 接口。

3. api 下的 ContractApi.java 详细说明

ContractApi()中仅含一个用于部署合约的方法 deployContract()，该方法包含两个参数，分别为 codePath(String), options(JSONObject)。codePath 为合约存放路径。options 为预留参数，在方法中暂未使用。

deployContract()会读取路径 codePath 下智能合约的内容，存于变量 codeContent 中。随后根据 {"code":codeContent,"type":"CSharp"}生成 request，最后将 request 发送至 smartx.ont.io 处进行编译并返回结果 response (url: http://139.219.97.24:8080/api/v1.0/compile)。得到 response 后获取其中的 avmcode 和 abi，根据 avmcode 调用方法 AddressFromVmCode(avmcode)得到合约地址。再利用 avmcode 调用方法 makeDeployCodeTransaction 生成交易，获取交易 tx。使用 tx 调用方法 sendRawTransaction 实际执行交易，并等待相应交易结果。若部署成功，返回合约地址 address 和 abi。(若失败则会返回 null 并记录 log)。

4. api 下的 NodeApi.java 详细说明

NodeApi()类中包含构造方法 NodeApi(), default 类型方法_start()和其他公有方法。详细说明如下：

- 1) NodeApi(): NodeApi()为构造方法，在方法 NodeApi()中初始化了 ontSdk 和 rpcs。ontSdk 继承 OntSdk 的 getInstance(), 而 rpcs 中包含各节点 TEST_SERVICE 的 url。
- 2) _start(): _start()为 NodeApi 的私有方法，其调用需要六个参数分别为 nodeindex(int), ontology(String), config(String), args(String), clean_chain(boolean)和 clean_log(boolean)。_start()方法会取 nodeindex 对应节点的 url 信息，生成包含标志参数 clear_chain, 标志参数 clear_log, name, node_args 和 config 相关信息的 params。最后通过调用 RpcClient 的方法 rpc.call("start_node", params)开启单个节点。
- 3) stopAll(): 方法 stopAll()会构建一个包含配置文件中所有节点 index 的整型数组 nodeindexes,

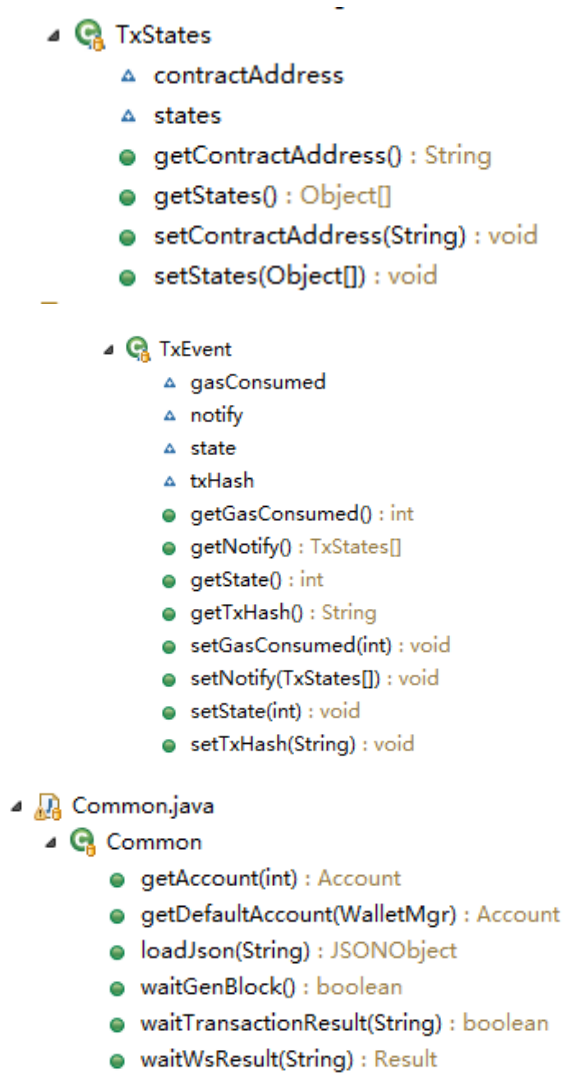
然后调用方法 `stop(int[] nodeindexes)`。

- 4) `stop(int nodeindex)`: 参数 `nodeindex` 为需要开启的节点 `index`。方法 `stop(int nodeindex)` 会通过 `rpcs.get(nodeindex)` 获取相关 url 信息, 再调用 `rpc.call("stop_node", null)` 关闭该节点。
- 5) `stop(int[] nodeindexes)`: 方法 `stop(int[] nodeindexes)` 会根据参数中的 `index` 依次循环调用方法 `stop(int nodeindex)` 来开启相应节点。
- 6) `startAll()`: 调用方法 `startAll(含参)`。其中参数 `program` 为 "ontology", 参数 `config` 为 "config.json", 参数 `nodeargs` 为配置文件 `test_config.json` 中的 `DEFAULT_NODE_ARGS`。
- 7) `startAll(String program, String config, String nodeargs)`: `startAll(含参)` 会先构建一个包含配置文件中所有节点 `index` 的整型数组 `nodeindexes`。
- 8) `start()`: 含有参数 `nodeindexes(int[])`, `program(String)`, `config(String)` 和 `nodeargs(String)`。参数 `nodeindexes` 为需要打开的节点 `index`, 通过 `startAll()` 调用时 `program`, `config` 和 `nodeargs` 分别为 "ontology", "config.json" 和配置文件 `test_config.json` 中的 `DEFAULT_NODE_ARGS`。方法 `start()` 根据 `index` 循环调用 `_start()` 依次开启节点。
- 9) `restartAll()`: 直接调用 `restartAll(含参)` 并返回调用结果。参数 `program`, `config` 与 `nodeargs` 分别为 "ontology", "config.json" 和配置文件 `test_config.json` 中的 `DEFAULT_NODE_ARGS`。
- 10) `restartAll(String program, String config, String nodeargs)`: 方法 `restartAll(含参)` 首先调用方法 `stopAll()` 关闭所有节点, 再根据配置文件内容生成包含各节点 `index` 的整型数组 `nodeindexes`, 最后调用方法 `restart(nodeindexes, program, config, nodeargs)`。
- 11) `restart(int[] nodeindexes, String program, String config, String nodeargs)`: 该方法会根据参数 `nodeindexes` 中包含的节点 `index` 依次调用 `_start()` 启动节点。注意这里调用方法 `_start()` 的标志参数 `clean_chain` 和 `clean_log` 都为 `true`, 会清除 `log` 和 `chain`。该方法用于重启多个节点。
- 12) `restart(int nodeindex, String program, String config, String nodeargs)`: 参数 `nodeindex` 为单个节点的 `index`, 该方法会调用方法 `stop()` 和 `_start` 重启该节点。同样, 这里调用方法 `_start()` 的标志参数 `clean_chain` 和 `clean_log` 都为 `true`, 会清除 `log` 和 `chain`。该方法用于重启单个节点。
- 13) `replaceConfigAll(String path)`: (预留接口)
- 14) `replaceConfig(int[] nodeindexes, String path)`: (预留接口)
- 15) `replaceConfigAll(JSONObject cfg)`: (预留接口)
- 16) `replaceConfig(int[] nodeindexes, JSONObject cfg)`: (预留接口)
- 17) `checkNodeSync()`: 方法 `checkNodeSync()` 中含有参数 `nodeindexes(int[])`, `nodeindexes` 为包含各节点 `index` 的整型数组。该方法会依次校对各节点调用 `getStatesMd5()` 的返回值 MD5, 若所有节点 `statesMD5` 一致便返回 `true`, 反之返回 `false`。该方法用于确认节点是否同步。
- 18) `getStatesMd5()`: 包含参数 `nodeindex(int)`, 根据节点 `index` 去调用 `rpc.call("get_states_md5", null)` 获取 `states` 数据库 MD5 并返回调用结果。
- 19) `getBlockMd5()`: 包含参数 `nodeindex(int)`, 根据节点 `index` 去调用 `rpc.call("get_block_md5", null)` 获取 `block` 数据库 MD5 并返回调用结果。
- 20) `getLedgereventMd5()`: 包含参数 `nodeindex(int)`, 根据节点 `index` 去调用 `rpc.call("get_ledgerevent_md5", null)` 获取 `Ledgerevent` 数据库 MD5 并返回调用结果。
- 21) `exec(int nodeindex, String cmd)`: (预留接口)
- 22) `initOntOng()`: 方法 `initOntOng()` 的作用分两种情况。当配置文件 `test_config.json` 中的 `TEST_MODE` 为 `false` 时, 首先方法 `initOntOng()` 会取前七个节点默认 `account` 中的 `publicKey` 并生成 `pubkeylist`, 然后调用方法 `addressFromMultiPubKeys(M, pubkeylist)` 获取多签地址, 其中参数 `M` 为 5, 接着生成 `ONT` 转账交易并执行交易和等待交易执行结果, 随后再使用与 `ONT` 转账一样的步骤实现 `withdrawONG` 和 `transferONG`; 当配置文件 `test_config.json` 中的 `TEST_MODE` 为 `true` 时, 方法 `initOntOng()` 会给节点上的若干个 `account` 分配 `ONT` 和

ONG。首先会先给自身（defaultaccount）转账，再去 withdrawONG，随后依次给非默认账户转 ONT 和 ONG。

5. utils 下的 Common.java 详细说明

Common.java 中包含三个类：TxStates、TxEvent、Common，，如图所示：



TxStates 定义了两个对象：States、ContractAddress，两个指的是 Txevents 下 notify 下其中的 States 和 ContractAddress。States 是 交易方式，ContractAddress 是合约地址，方法是定义的对象所对应的 get、set 方法。

TxEvents 定义了四个对象：GasConsumed、TxHash 、State、Notify，指的是交易中的一些信息，GasConsumed 是 $gasprice * gaslimit$ 的数量，TxHash 是交易 Hash 值，State 是交易状态，状态只返回 1 和 0 两种，1 表示交易成功，0 表示交易失败，Notify 是一些交易的信息，返回的是交易方式、交易地址、交易金额、合约地址、交易状态等信息。方法是定义的对象对应的 get、set 方法。

Common 中有 6 个方法：getAccount(int index)、getDefaultAccount(WalletMgr walletmgr)、loadJson(String filepath)、waitGenBlock()、waitTransactionResult(String hash)、waitWsResult(String action)。

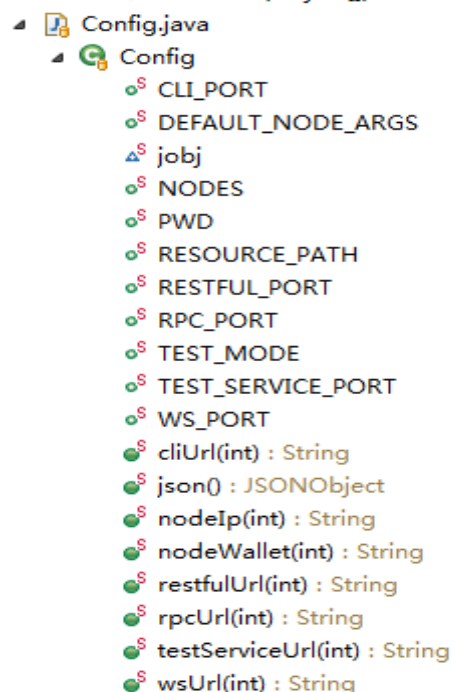
- 1) getAccount(int index)获取的是钱包，是 com.github.ontio.account.Account 类型，在单机

模式下，参数指的是一个钱包中创建钱包的编号，即 `getAccount(0)` 是当前钱包中的第一个 `account`，`getAccount(1)` 是当前钱包的第二个 `account`，以此类推；在非单机模式下，参数指的是按顺序的节点上的 `account`，即 `getAccount(0)` 指的是第一个节点钱包的默认 `account`，`getAccount(1)` 指的是第二节点钱包的默认 `account`，一次类推。

- 2) `getDefaultAccount(WalletMgr walletmgr)` 获取的是当前节点的默认钱包的 `account` 信息，写入的参数是 `WalletMgr` 类型。
- 3) `loadJson(String filepath)` 是对 `json` 文件的解析，参数是需要解析的 `json` 文件的路径，返回解析后的 `json` 文件中需要的数据。
- 4) `waitGenBlock()` 是等待一个区块生成。
- 5) `waitTransactionResult(String hash)` 是等待一笔交易结果，一笔交易产生后会有相应的 `hash` 值返回，将 `hash` 值作为参数写入，等待一笔交易结果，打印这笔交易的 `state`、`hash` 和 `GasConsumed`，返回交易结果 `true`（成功）或者 `false`（失败）。
- 6) `waitWsResult(String action)` 特别用于 `webSocket api` 的测试所写的方法，用于返回每一条测试用例最终的结果，参数填入每一条测试用例对应的 `action` 操作。

6. `utils` 下的 `Config.java` 详细说明

`Config.java` 主要是获取配置文件中的数据，然后写成静态方法可供调用。`RPC_PORT`、`RESTFUL_PORT`、`WS_PORT`、`CLI_PORT`、`TEST_SERVICE_PORT`、`NODES`、`DEFAULT_NODE_ARGS`、`PWD`、`RESOURCE_PATH`、`TEST_MODE` 是静态对象，获取的是 `test_config` 配置文件中对应的每一个数据。`Config.java` 中一共有 7 个方法，且都是静态方法：`nodeIp(int index)`、`nodeWallet(int index)`、`rpcUrl(int index)`、`restfulUrl(int index)`、`wsUrl(int index)`、`cliUrl(int index)`、`testServiceUrl(int index)`、`JSONObject json()`，如图所示。

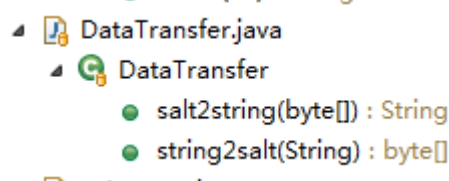


- 1) `nodeIp(int index)` 首先获取的是 `test_config` 中配置的 IP，参数 `index` 即为第几个节点的 IP 地址，`nodeIp(0)` 即返回第一台节点的 IP 地址。
- 2) `nodeWallet(int index)` 获取节点的临时钱包名称，参数 `index` 即第几个节点的钱包，`nodeWallet(0)` 获取的是第一个节点的节点钱包名称。

- 3) `rpcUrl(int index)`获取节点的 `RPC_PORT` 端口号, 参数 `index` 即第几个节点的端口号, `rpcUrl(0)`获取的是第一个节点的 `rpc` 地址。
- 4) `restfulUrl(int index)`获取节点的 `restful` 的端口号, 参数 `index` 即第几个节点的端口号, `restfulUrl(0)`获取的是第一个节点的 `restful` 的地址。
- 5) `wsUrl(int index)`获取节点的 `websocket` 的端口号, 参数 `index` 即第几个节点的端口号, `wsUrl(0)`获取的是第一个节点的 `websocket` 的地址。
- 6) `cliUrl(int index)`获取节点的 `cli` 的端口号, 参数 `index` 即第几个节点的端口号, `cli(0)`获取的是第一个节点的 `cli` 的地址。
- 7) `testServiceUrl(int index)`获取节点的 `testService` 的端口号, 参数 `index` 即第几个节点的端口号, `testService(0)`获取的是第一个节点的 `testService` 的地址。
- 8) `json()`用于解析 `test_config.json`, 获取 `json` 文件中的 `RPC_PORT`、`RESTFUL_PORT`、`WS_PORT`、`CLI_PORT`、`TEST_SERVICE_PORT`、`NODES`、`DEFAULT_NODE_ARGS`、`PWD`、`RESOURCE_PATH`、`TEST_MODE` 等数据, 为以上获取 IP 或者各个节点的端口号提供数据。

7. `utils` 下的 `DataTransfer.java` 详细说明

`DataTransfer.java` 包含两个方法 `string2salt(String salt)`和 `salt2string(byte[] salt)`, 如图所示。



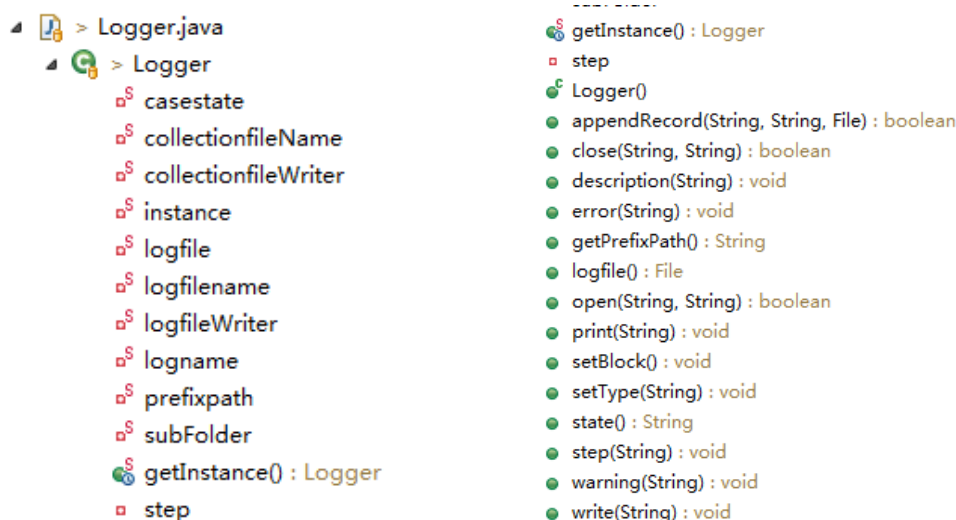
这两个方法是 `String` 和 `byte[]`数组型进行转换, 在 `account` 中 `salt` 属于 `byte[]`类型, 在测试时候需要将它转换为 `String` 类型。

`string2salt(String salt)`是把 `String` 类型转化为 `byte[]`类型。

`salt2string(byte[] salt)`是把获取的 `byte[]`型转化为 `String` 类型。

8. `utils` 下的 `Logger.java` 详细说明

`Logger.java` 包含多个静态对象和方法, 静态对象为下面的方法提供服务。方法包括 `Logger.getInstance()`、`Logger()`、`getPrefixPath()`、`setType(String type)`、`logfile()`、`print(String content)`、`error(String content)`、`warning(String content)`、`description(String content)`、`step(String content)`、

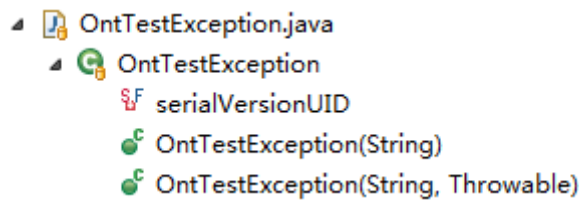


open(String logfilename, String logname)、appendRecord(String name, String status, File logfile)、close(String ret, String info)、String state()、setBlock()、write(String contents), 如图所示。

- 1) getInstance()是定义一个类的全局对象, 供其他类共同使用。
- 2) Logger()是默认构造函数, 这里设置了记录 log 文件夹的名称以时间命名和 log 文件夹的路径位置。
- 3) getPrefixPath()是定义的静态对象 prefixpath 的 get 方法, 获取的是存储 log 的文件夹, 即以时间命名的文件夹。
- 4) logfile()是 File 类定义的对象 logfile, 通过 get 方法获取。
- 5) print(String content)是写 log, 在写的 log 之前加一个[INFO] 表示是打印的信息。
- 6) error(String content)是将错误信息记录下来, 在写的 log 之前加上[ERROR], 表示是错误信息。
- 7) warning(String content)是将警告信息记录下来, 在写的 log 之前加上[WARNING], 表示是警告信息。
- 8) description(String content)是普通的描述信息, 描述测试用例时使用, 在写的 log 之前加上[DESCRIPTION], 表示是描述信息。
- 9) step(String content)是记录步骤信息, 测试用例每一步做什么用此方法记录下来, 在写 log 之前加入[STEP]-序号, 序号从 1 开始, 自动向下编号。
- 10) open(String logfilename, String logname)是打开文件, 参数 logfilename 和 logname 分别指的是 log 文件夹名称和 log 名称, 用于创建日志和打印日志。
- 11) appendRecord(String name, String status, File logfile)是记录每一条 log 的名称, log 状态和 log 文件名, 每一个模块的测试用例都会生成一张 CSV 的表格, 通过该方法向表格内不断添加每一条 log 的信息。
- 12) close(String ret, String info)是记录每一条 log 最终运行的结果, pass、fail 或者 block, 然后再向 CSV 文件中写入信息。
- 13) state()返回测试用例结果。
- 14) setBlock()是将 log 信息结果写成 block, 在外部调用使用。
- 15) write(String contents)是写 log 信息, 与 print 不同, 前面不加[INFO], 且以\n 作为分隔符, 取\n 之后的字符。

9. utils 下的 OntTestException.java 详细说明

OntTestException.java 包含两个构造函数: OntTestException(String message) 和 OntTestException(String message, Throwable ex), 是 OntTest 如果报错跑出的错误信息, 如图所示。



10. utils 下的 RpcClient.java 详细说明

RpcClient.java 包含一个定义为 final 的 URL, 和五个方法: RpcClient(String url)、getHost()、call(String method, Object params)、makeRequest(String method, Object params)、send(Object request)。

- 1) RpcClient(String url)是该类的构造函数。
- 2) getHost()返回的是 URL 的主机名和端口。
- 3) call(String method, Object params)是两个方法的调用, 首先调用 makeRequest 方法, 再

调用 send 方法，方法的具体介绍如下。

- 4) makeRequest(String method, Object params)是发出请求,要传入参数 method 和 params, 将 request 请求放入 map 集合中, 然后输出形式是 POST url=%s,%s, 即 URL 和 JSON 格式的 request 均转化为字符串格式。
- 5) send(Object request)是将参数 request 请求发送的方法, 首先打开连接, 设置提交模式为 post 模式, 请求格式为 JSON 格式, 连接后对象转换为 JSON 字符串。

三、测试例相关工具文件说明

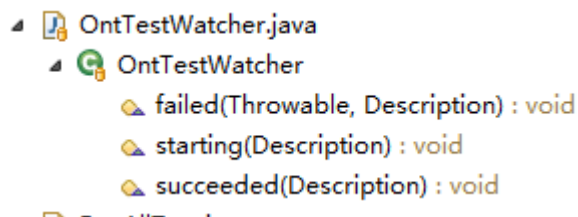
1. MethodNameFilter.java 详细说明

MethodNameFilter.java 文件的 MethodNameFilter 类下有构造方法 MethodNameFilter()和方法 shouldRun()。这两个方法说明如下：

- 1) MethodNameFilter()：该方法包含参数 excludedMethods(Set<String>)，includeSheets(Set<String>)，includeTypes(Set<String>)，filterCases(Set<String>) 和 includeClasses(Set<String>)。当参数不为空时，方法会将参数中的内容循环 add 到相应的成员变量中。该方法用于实现成员变量 excludedMethods，includeSheets，includeTypes，filterCases 和 includeClasses 的初始化。
- 2) shouldRun()：方法 shouldRun()包含参数 description(Description)，首先取参数中的方法名存于变量 methodName 中。当 methodName 为"test_init"时，结果返回 true。当 methodName 不为"test_init"时，若 excludedMethods 为空或者 excludedMethods 中包含 methodName 时返回 false。取参数 description 中包含测试例的文件名记于变量 m_file 中，当成员变量 filterCases 为空且 includeTypes 为空或 includeTypes 与 methodName 中的期望结果一致时，若成员变量 includeSheets 为空或者 includeSheets 包含 m_file 则返回 true。而当 includeTypes 为空并且 filterCases 为空或 filterCases 和 includeClasses 中包含了 methodName 和 m_file 时，若 includeSheets 为空或者包含 m_file 则返回 true。其他情况都返回 false。

2. OntTestWatcher.java 详细说明

OntTestWatcher.java 中包含三个方法：starting(Description description)、failed(Throwable e, Description description)、succeeded(Description description)，这个类主要用于每一个测试用例开始和结束时使用，如图所示：



starting(Description description)是打开 log，log 文件的生成的是以类名为文件夹名称，方法名为 log 名称的 log。

failed(Throwable e, Description description)是测试用例失败时，在 log 文件中写入 fail 字符串并将 log 错误信息输出。

succeeded(Description description)是当测试用例测试通过后，在 log 文件中写入 pass 并将该测试用例的描述（测试用例的方法名称和测试用例所在的位置）。

3. RunAllTest.java 详细说明

RunAllTest.java 用于实现测试例的选择并进行测试，其只有一个 main 函数不包含其他方法，main()中参数有 args(String[])。首先 main 函数会将参数 args 按“-c”(--config)、“-t”(--type)、“-f”(--filter)和“-e”(--exclude)分开依次存于变量 parameter_c，parameter_t，parameter_f 和 parameter_e。

当变量 parameter_c 不为空时，取 parameter_c 对应配置文件（select.json）中 value 为 true 所对应的 key 依次记于变量_files 中；

当变量 parameter_t 不为空时，若 parameter_t 为"base"则变量_types 中添加"base"，若 parameter_t 为"normal"则变量_types 中添加"base"和"normal"，若 parameter_t 为"abnormal"则变量_types 中添加"abnormal"；

当变量 parameter_f 不为空时，则将 parameter_f 中的类名与方法名分开，并将类名添加到变量_classes 中，将方法名添加到变量_methods 中；

当变量 `parameter_e` 不为空时,则将 `parameter_e` 中的方法名依次添加到变量 `_excludes` 中。

然后将各测试例集合对应的类依次添加到变量 `all_class` 中, 将 `JUnitCore()`实例化为 `junitRunner`。变量 `_class` 循环取 `all_class` 中的类, 当 `_class` 的类名不是要执行的测试例类时 `_class` 继续去取一个值; 当 `_class` 的类名是需要执行的测试例时, 记录当前开始时间, 运行相对应的测试例, 结束后记录结束时间, 将运行测试的时间信息记录于 `logs` 路径下该类名测试例集合的文件夹下的 `report.ini` 文件中。

最后获取测试例总数记于变量 `total_cases` 中, 记录结果为 `fail` 的测试例数量于变量 `failed_cases` 中, 输出 `case` 的失败率。

4. TestMonitor.java 详细说明

`TestMonitor` 中包含方法 `testRunStarted`, `testRunFinished`, `testStarted` 和 `testFinished`。
`testRunStarted` 用于在开始测试时输出 "Number of tests to execute: "加上即将测试的测试例数量,
`testRunFinished` 用于在测试结束后输出 "Number of tests executed: "加上最终所执行的测试例数目,
`testStarted` 用于在测试每一条测试例之前输出 "Starting: "加上用例的方法名。`testFinished` 则会读取 `logger` 信息, 并在出现 "connect error:"后重启节点, 最后在结束每一条测试后输出 "Finished: "加上用例方法名。