

Ontology python 测试框架 api 及测试用例详细介绍

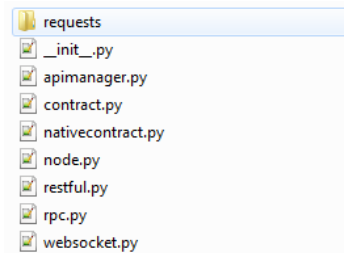
一、客户端测试框架及测试用例详细介绍：

在 test_tool 下包含文件如下：

api	测试脚本相关API
monitor	检测恢复测试环境
resource	自检文件相关
test	测试脚本
tools	测试框架相关工具
utils	测试框架代码
config.json	配置文件
config3.json	
config16.json	
install.sh	环境安装工具
README.md	
run_all.sh	

1. 其中 api 和 utils 中包含了测试脚本相关 API 和测试框架相关代码：

1) api 中目录如下：



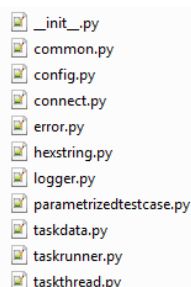
contract.py、nativecontract.py、node.py、restful.py、rpc.py、websocket.py 分别为 contract, nativecontract, node, restful, rpc 和 websocket 相关 API, apimanager.py 整合了这些 API, 通过 from api.apimanager import API 即可调用以上六个文件中的方法。

- a) contract.py: 其中包含 deploy_contract(), deploy_contract_full(), sign_transction(), call_signed_contract(), call_contract(), sign_multi_transction(), call_multisig_contract(), init_admin(), invoke_function()等方法, 用于实现部署合约, 调用合约, 多签等。
- b) nativecontract.py: 其中包含 init_ont_ong(), transfer_multi(), transferFrom_multi(), approve_candidate(), register_candidate(), commit_dpos(), black_node(), quit_node(), update_global_param(), withdraw_ont(), unvote_for_peer(), vote_for_peer(), withdraw_user_role(), delegate_user_role(), bind_user_role(), bind_role_function(), regid_with_publickey(), transfer_ong(), approve_ong(), allowance_ong(), transfer_ont(), approve_ont(), allowance_ont()等方法, 用于对 ont 和 ong 进行转账和回收, 拉黑节点和撤销拉黑, 注册和审核通过节点申请 (成为候选节点), 投票、关联和更新全局参数等。
- c) node.py: 其中包含 wait_gen_block(), get_current_node(), findSystemNode(), _check_md5(), check_node_state(), check_node_ledgerevent(), check_node_block(), check_node_all(), start_nodes(), start_node(), stop_all_nodes(), stop_nodes(), stop_node(), replace_configs(), replace_config(), transfer_ont(), transfer_ong(), withdrawong(), exec_cmd(), stop_sigsvr(), start_sigsvr()等方法, 用于实现等待区块生成、获取当前节点地址、检查 md5 是否一致、检查节点服务器数据库是

否一致、开启和关闭节点服务器、替换 nodecontroll.py 相关 config、ong 和 ont 转账及收回 ong、开启和关闭签名服务。

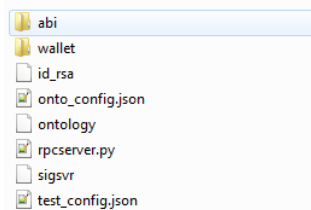
- d) restful.py: 其包含 getgenerateblocktime(), getconnectioncount(), getblocktxsbyheight(), getblockbyheight(), getblockbyhash(), getblockheight(), getblockhashbyheight(), gettransactionbytxhash(), postrawtx(), getstorage(), getbalance(), getcontract(), getsmartcodeeventbyheight(), getsmartcodeeventbyhash(), getblockheightbytxhash(), getmerkleproofbytxhash(), getgasprice(), getallowance() 等方法, 用于通过 restful 接口实现获取 gasprice, allowance, balance, storage 和根据 hash, txhash, height 去获取相关区块信息等。
- e) rpc.py: 通过 rpc 接口实现获取 gasprice, allowance, balance, storage 和根据 hash, txhash, height 去获取相关区块信息等。
- f) websocket.py: 通过 websocket 接口实现获取 gasprice, allowance, balance, storage 和根据 hash, txhash, height 去获取相关区块信息等。

2) utils 中目录如下:

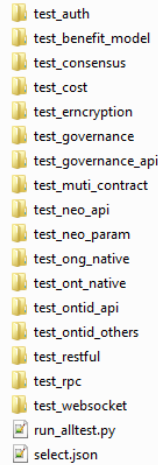


- a) common.py 中包含 cmp(), pause(), bl_reserver(), base58_to_address(), bl_address() 等方法, 这些方法用于实现比较 str、dict、list 数据是否一致, 中止测试, base58 编码和 bl_reserver 的字符串两两转序。
- b) config.py 中包含了 case 中及测试代码中会使用到的部分变量和路径。
- c) connect.py 中包含类 WebSocket(), 在这个类中存在一个名叫 exec 的方法将其内线程 ws_thread 和 ws_heartbeat_thread 打开, 然后监听输入内容并根据内容发送请求。除此以外还存在以下七个方法:
 - i. multithread_run(logger, cfg_request, cfg_response) : 比较期望的 cfg_response 与 cfg_request 所得到的回复是否一致, 若一致则令变量 result 为 True 反之则为 False, 同时令 response 等于 request 后得到的回复结果, 方法最后返回 result 和 response。
 - ii. con(itype, ip, request) :
根据 itype (可以为"RPC","CLI","RESTFUL","WS","ST") 选择调用接口。
itype 为"RPC","CLI","RESTFUL","WS"和"ST"时分别调用 con_rpc, con_cli, con_restful, con_ws 和 con_test_service。
 - iii. con_cli(ip, request) : 若参数 ip 存在输入, url 为"http://" + ip + ":20000/jsonrpc", 若没有则从上层目录中的 config.json 中读取 url, 从该 url 读取 response 返回。itype 为"CLI"时调用。
 - iv. con_rpc(ip, request): 同上用途, 但有 ip 时端口号为 20336。itype 为"RPC"时调用。
 - v. con_test_service(ip, request): 同上用途, 但有 ip 时端口号为 23636。itype 为"ST"调用。
 - vi. con_restful(ip, api_request): 根据 api_request (是字典, 可以包含"api" (必

- 有), "command", "params") 内部的内容。若有 ip 输入, url 为 "http://" + ip + ":20334/jsonrpc", 若没有则从上层目录中的 config.json 中读取 url。根据 command 内容发送或者获取信息。itype 为 "RESTFUL" 调用。
- vii. con_ws(ip, request): 若有 ip 输入, url 为 "http://" + ip + ":20335 ", 若没有则从上层目录中的 config.json 中读取 url, 使用该 url 发送 request 并等待 response 返回。itype 为 "WS" 时调用。
 - d) error.py 用于捕捉测试用例时可能遇到的异常。
 - e) hexstring.py 文件中主要包含了一些数据转换的方法:
 - i. FileToHex(path): 将参数 path 路径中文件内容 ASCII 值对应的十六进制数作为返回值。
 - ii. ByteToHex(bytes_value): 将参数 bytes_value 中内容 ASCII 值对应的十六进制数作为返回值。
 - iii. HexToByte(hexStr): 将参数 hexStr 中的十六进制数转换为对应 ASCII 值。
 - iv. recoveryAddress(m, hexArr, checkMultiSig="ae"): 返回原 pubkey 经过一系列处理后的散列值 (最后返回结果为 40 位十六进制数), 同时 printsha256 与 ripemd160 的值。
 - v. numSeq(m): 当 m 为数字 0 时 return "00", 当 m 为大于等于 1 且小于等于 16 的正整数时返回八十加 m 的十六进制数, 当 m 大于 16 后返回 (m+1) 的十六进制后两两倒序排列后的数据。
 - f) logger.py 用于记录 log 信息。
 - g) parametrizedtestcase.py 用于实现参数化。
 - h) taskdata.py 类 Task 用于返回 task 中相应的数据; 类 TaskData 用于将 path 路径下.json 文件中的数据作为 task 添加到 ret, 最后返回 ret。
 - i) taskrunner.py 包含两个基础的方法 run_single_task() 和 run_pair_task(), 前者用于执行单个 webapi 并将返回的 response 与期望 response 作对比, 一致时输出 OK 相关语句, 不一致时输出 Failed 相关语句 (包括执行所使用时间), 最终返回 result 和 response; 后者当参数 compare_src_key 为 bool 型的 True 时返回调用 run_single_task(task1) 和 run_single_task(task2) 结果 response 的比较结果。
 - j) taskthread.py 用于初始化进程, 也包括启动进程和获取结果。
2. monitor 中包含了一个 monitor.py 文件, 该文件用于检测和恢复测试环境 (包括重启各节点服务器和签名服务器 sigserver), 在每次执行测试脚本时会调用 monitor 以保证测试环境。当使用调用 run_alltest.py 时 -m 后跟 0 时不再调用 monitor。
 3. resource 包含了自检文件中用于参考的各类文件, 其中 onto_config.json 和 test_config.json 根据 wallet 文件夹中的 16 个 wallet.dat 进行配置, 其他文件都需要为测试中使用的文件 (一般为最新测试版本的文件), 其目录如下所示:



4. test 中放置了所有测试脚本:



```

test_auth
test_benefit_model
test_consensus
test_cost
test_erncryption
test_governance
test_governance_api
test_muti_contract
test_neo_api
test_neo_param
test_ong_native
test_ont_native
test_ontid_api
test_ontid_others
test_restful
test_rpc
test_websocket
run_alltest.py
select.json

```

- 1) 所有测试文件中框架如下，其中 resource 和 test_api 根据测试实际情况会存在出入：

test_xxxx

```

|--resource  ##一般包含合约.neo 文件和 neo 相关配置文件
|--__init__.py
|--test_api.py  ##包含 test_main 需要调用的一些方法
|--test_config.py  ##包含 test_main 所用到的配置
|--test_main.py  ##包含所有相关测试用例

```

- 2) 以下是测试文件与包含用例的对应关系，测试文件 test_main.py 中测试用例的命名如下例子：test_base_001_initContractAdmin 其中 test 后的 base, normal, abnormal 分别代表测试该方法的第一条 case，期望结果为正常的 case 和期望结果为异常的 case，数字 001 与表格中测试用例序号相对，initContractAdmin 一般为测试的方法名（部分为实际测试文件中调用的方法名）。

- a) test_auth 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的合约调用授权管理。
- b) test_benefit_model 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的分润模型测试。
- c) test_consensus 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的共识及帐本。
- d) test_cost 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的消费模型测试。
- e) test_erncryption 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的加密算法。
- f) test_governance 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 Native 合约_治理合约相关。
- g) test_governance_api 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 Native 合约_治理合约 API。其中包含了 RegisterCandidate、ApproveCandidate、RejectCandidate、BlackNode、WhiteNode、QuitNode、VoteForPeer、UnVoteForPeer、Withdraw、updateConfig、UpdateGlobalParam、UpdateSplitCurve、TransferPenalty 等测试内容。
- h) test_muti_contract 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的跨合约调用。
- i) test_neo_api 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的智能合约 API_NEO。
- j) test_neo_param 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 neo_参数列表及返回值。
- k) test_ong_native 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 ONG Native

合约 API。

- l) test_ont_native 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 ONT Native 合约 API。
- m) test_ontid_api 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 Native 合约 _ontid。
- n) test_ontid_others 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 Native 合约 _ontid 合约相关。
- o) test_restful 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 Restful API。
- p) test_rpc 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 RPC API。
- q) test_websocket 中的 case 对应 ontology 自动测试工具介绍.xlsm 中的 WebSocket API。
- r) run_alltest.py 以及 select.json 在第四节测试环境搭建及运行测试脚本中的第九点中已经详细说明，此处不再赘述。

二、测试服务端及相关工具文件

netervice	
release	
sdk-test	
test_service	测试服务工具
test_tool	测试脚本及相关文件

1. 测试脚本及相关文件已在第一节客户端测试框架及测试用例中详细介绍，此处不再赘述。
2. 测试服务工具中的服务会在测试前配置好环境以及启动测试服务（见第四节第三点）。其中的 install.sh 用于安装所需要的 python 包，配置文件 config.py 中包含了 PORT 和 NODE_PATH，run.sh 用于启动测试服务，rpcserver.py 即为 run.sh 所启动的测试服务。启动该服务后方可实现多签和请求其他节点数据库数据。rpcserver.py 包含方法如下：
 - 1) calc_md5_for_file(file): 返回文件的 MD5。
 - 2) calc_md5_for_folder(folder): 返回文件夹的 MD5。
 - 3) get_db_md5(db_name): 获取 db_name 所对应的数据，选取部分 key 和 value 来更新 md5，最终返回 md5.hexdigest()(十六进制数据字符串值)。
 - 4) get_host_ip(): 调用后返回主机地址。
 - 5) con_cli(request): 往 url 为'http://127.0.0.1:20000/cli'处发送请求，请求信息根据参数 request 得到。
 - 6) get_states_md5(**kwargs): 调用后返回 NODE_PATH+'/Chain/states'的 md5 值。
 - 7) get_block_md5(**kwargs): 调用后返回 NODE_PATH+'/Chain/block'的 md5 值。
 - 8) get_ledgerevent_md5(**kwargs): 调用后返回 NODE_PATH+'/Chain/ledgerevent'的 md5 值。
 - 9) siginvoctx(**kwargs): 返回调用接口 con_cli(request)的结果。
 - 10) stop_node(**kwargs): kill 所有 ontology、ontology-bft_1、ontology-bft_2、ontology-bft_3 同名进程，结束后返回 True。
 - 11) replace_node_config(**kwargs): 打开 NODE_PATH+"/config.json"权限可写，写入参数信息关闭该 json 文件，结束后返回 True。
 - 12) transfer_ont(**kwargs): 删除目录下.tmp 后缀文件，根据参数生成实现 ONT 转账的 cmd 并生成 tmp 文件，返回生成的 tmp 中的内容。
 - 13) transfer_ong(**kwargs): 删除目录下.tmp 后缀文件，根据参数生成实现 ONG 转账的 cmd

并生成 tmp 文件，返回生成的 tmp 中的内容。

- 14) `withdrawong(**kwargs)`: 删除目录下 tmp 后缀文件，根据参数生成实现回收 ONG 的 cmd 并生成 tmp 文件，返回生成的 tmp 中的内容。
- 15) `start_node(**kwargs)`: 调用后实现强制删除 `NODE_PATH + "/Chain"` 下所有文件，获取参数中的具体信息。若其中 `clear_chain` 和 `clear_log` 为 `True` 则再次强制删除 `config.NODE_PATH + "/Chain"` 和 `config.NODE_PATH + "/Log"` 下的所有文件。根据 `node_args` 生成相应的 cmd 并执行该命令，结束后返回 `True`。
- 16) `exec_cmd(**kwargs)`: 执行参数中所包含的命令，结束后返回 `True`。
- 17) `stop_sigsvr(**kwargs)`: stop 签名服务。
- 18) `start_sigsvr(**kwargs)`: start 签名服务。
- 19) `heart_beat(**kwargs)`: 调用直接返回 "I'm OK."。
- 20) `check_xmode_ontology(**kwargs)`: 当 ontology 没有执行权限时，给其所有权限。
- 21) `check_xmode_sigsvr(**kwargs)`: 当 sigsvr 没有执行权限时，给其所有权限。
- 22) `check_xmode_tools(**kwargs)`: 当 `test_tool/tools` 下的 `base58ToAddress` 没有执行权限时，给其全部权限。
- 23) `get_version_ontology(**kwargs)`: 查询 ontology 文件的版本，计算 MD5，返回 `result={"md5":ontology_md5,"version":ontology_version}`。
- 24) `get_version_wallet(**kwargs)`: 返回 wallet 的 MD5。
- 25) `get_version_onto_config(**kwargs)`: 返回配置文件 `onto_config.json` 的 MD5。
- 26) `get_version_test_config(**kwargs)`: 返回配置文件 `test_config.json` 的 MD5。
- 27) `get_version_sigsvr(**kwargs)`: 查询 sigsvr 文件的版本，计算 MD5，返回 `result={"md5": sigsvr_md5,"version": sigsvr_version}`。
- 28) `get_version_abi(**kwargs)`: 返回 abi 文件夹的 MD5。
- 29) `get_version_test_service(**kwargs)`: 返回测试服务 `test_service/rpcserver.py` 的 MD5。
- 30) `stop_test_service(**kwargs)`: 关闭 rpcserver 服务。
- 31) `application(request)`: 调用以上的 dispatcher，返回调用结果，并且在没有 error 时在结果中添加 "error":0，在有 error 时在结果中添加 "desc":error_message 和 "error":error_code