自动化测试代码解析

版本记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 编写/修订说明 | 修订人 | 修订日期 | 备注 |
| 1.0.0 | 创建 | 范训山 | 20210922 | 首次创建 |

目录

[一、研测自动化使用 3](#_Toc4840)

[1、打开研测平台 3](#_Toc10460)

[2、新建自动化测试任务 4](#_Toc12951)

[3、设置测试信息 4](#_Toc20814)

[4、关联测试用例 4](#_Toc9568)

[5、设置任务数据 5](#_Toc11445)

[6、选择运行设备 6](#_Toc3976)

[7、查看测试结果 6](#_Toc12873)

[8、获取服务器log 7](#_Toc1861)

[二、自动化测试开发环境搭建 8](#_Toc13254)

[1、搭建开发环境的准备 8](#_Toc16985)

[2、树莓派环境搭建 8](#_Toc22281)

[3、测试工程环境搭建 8](#_Toc8620)

[4、Python基本能力疏通 9](#_Toc2524)

[5、嵌入式代码编程规范 9](#_Toc4725)

[三、代码结构讲解 9](#_Toc10462)

[1、自动化测试配置信息 9](#_Toc14838)

[2、目录结构简介 11](#_Toc6887)

[3、分层信息 12](#_Toc18393)

[4、用例管理层介绍（testcase） 12](#_Toc27742)

[5. 用例逻辑层介绍（controller） 14](#_Toc27823)

[6、环境数据介绍 （meta） 14](#_Toc4689)

[7、数据层介绍 （dao） 14](#_Toc7452)

[8 业务逻辑层介绍（service） 15](#_Toc31577)

[9 组件层介绍 (component) 16](#_Toc20406)

[四、自动化测试用例新增 18](#_Toc1166)

[1、基本服务功能解析 18](#_Toc4780)

[（1）基本用例新增 18](#_Toc23312)

[（2）协议接口定义 18](#_Toc32670)

[（3）协议调试接口定义 18](#_Toc22261)

[（4）定义获取内存的协议解析类 19](#_Toc1882)

[（5）设置memory对应的log类 19](#_Toc673)

[2、协议交互项用例新增 19](#_Toc7253)

[（1）测试用例新增 19](#_Toc22990)

[（2）实现协议接口 20](#_Toc14873)

[（3）实现协议调试接口 21](#_Toc26054)

[（4）定义测试指令的协议解析类 21](#_Toc30979)

[（5）修改接收数据处理函数 21](#_Toc18604)

[（6）修改comment的处理函数 23](#_Toc6402)

[3、网络交互的用例新增 25](#_Toc6079)

[（1）测试用例新增 25](#_Toc821)

[（2）controller中实现对应的校验 25](#_Toc9469)

[4、网络状态操作 26](#_Toc412)

[（1）网络环境设置 26](#_Toc6797)

[（2）网络操作切换到目标状态 26](#_Toc9531)

[（3）网络状态获取 26](#_Toc7903)

[五、pycharm代码本地修改后的更新上传 26](#_Toc22107)

[1、编写调整修改本地代码 26](#_Toc11873)

[2、输入commit信息 27](#_Toc10190)

[3、代码切换到主分支代码 29](#_Toc53)

[4、拉取主分支代码 29](#_Toc3430)

[5、切换原来分支 30](#_Toc13652)

[6、将主分支合并到子分支的代码中，该步可能有代码冲突需要确认 30](#_Toc22263)

[7、将本地的子分支代码push上去 31](#_Toc19962)

[8、代码分支切换到主分支，后续步骤需要稳定后再操作 32](#_Toc7325)

[9、拉取主分支代码 33](#_Toc32600)

[10、将子分支向主分支合并 34](#_Toc28204)

[六、压测平台使用 34](#_Toc5078)

[1、压测平台测试设备信息的编辑 34](#_Toc26331)

[（1）打开产测平台 34](#_Toc4719)

[（2）进入设备列表界面 34](#_Toc4027)

[（3）编辑个人设备的设备信息 35](#_Toc3567)

[2、压测平台用例库的创建或编辑 36](#_Toc3704)

[（1）进入用例库管理界面 36](#_Toc17179)

[（2）新增用例库 36](#_Toc19308)

[（3）输入用例库信息 36](#_Toc9474)

[（4）用例库同步用例 37](#_Toc20755)

[（5）选择需要的用例项进行同步 37](#_Toc12565)

# 一、研测自动化使用

## 1、打开研测平台

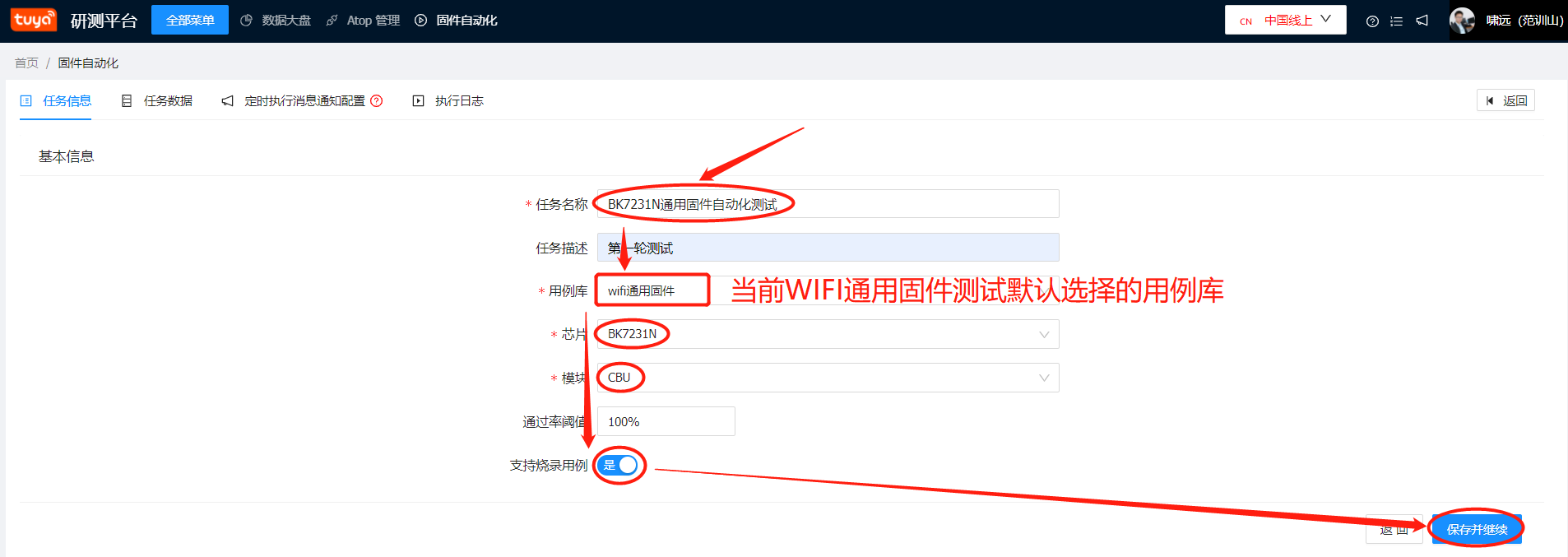
自动化平台链接：<https://galaxy.tuya-inc.com:7799/firmware/auto/task/list>

## 2、新建自动化测试任务



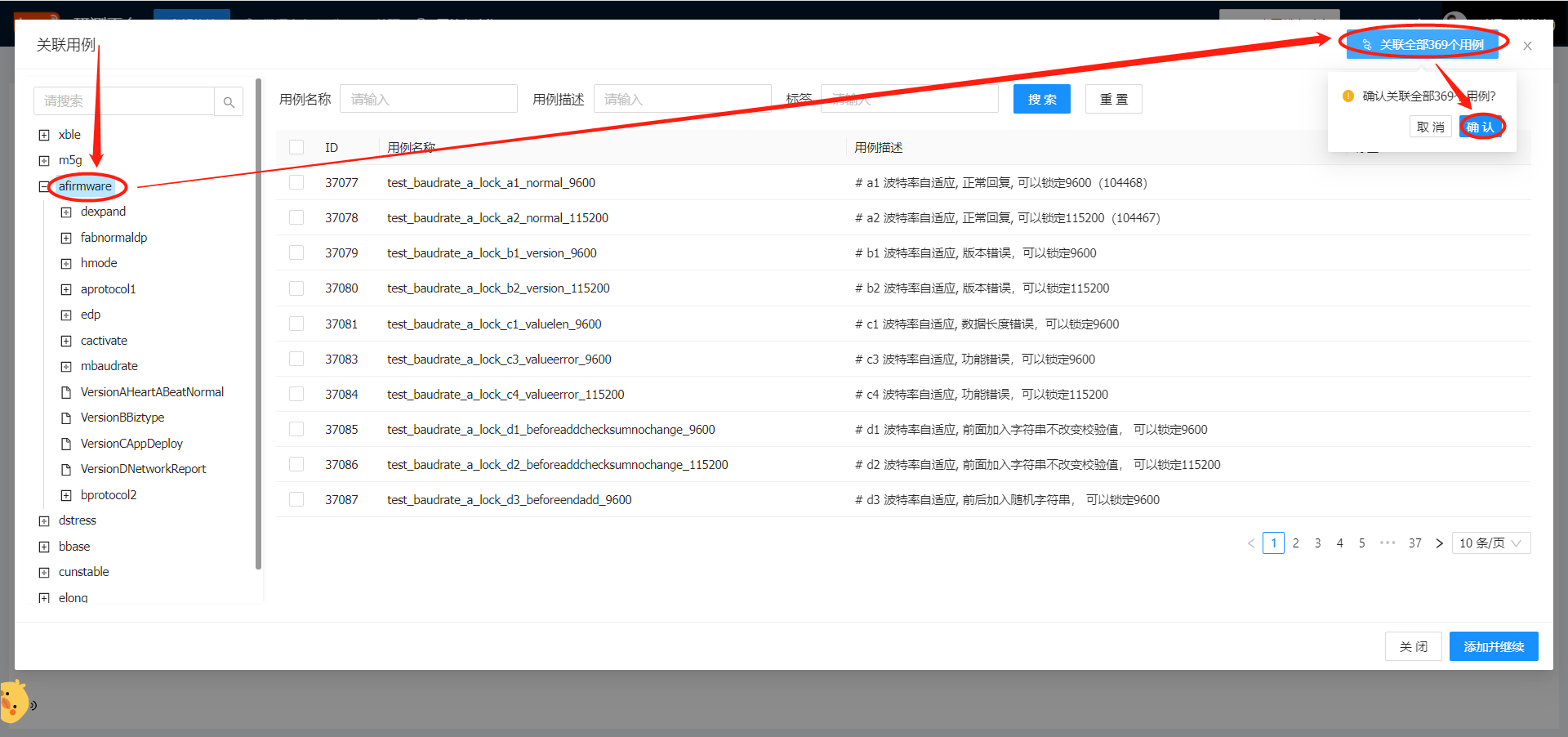
## 3、设置测试信息

设置任务信息：选择WIFI通用固件、平台、模块



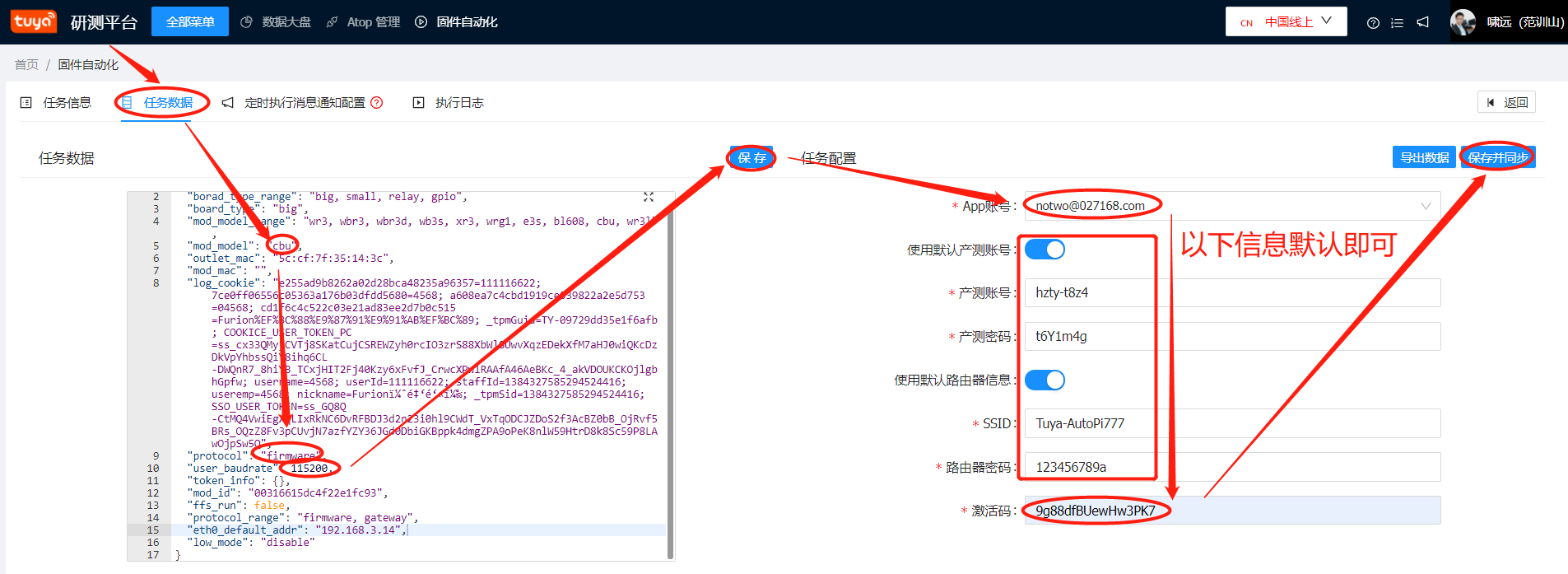
## 4、关联测试用例





## 5、设置任务数据

激活码随便输入一个即可：9g88dfBUewHw3PK7

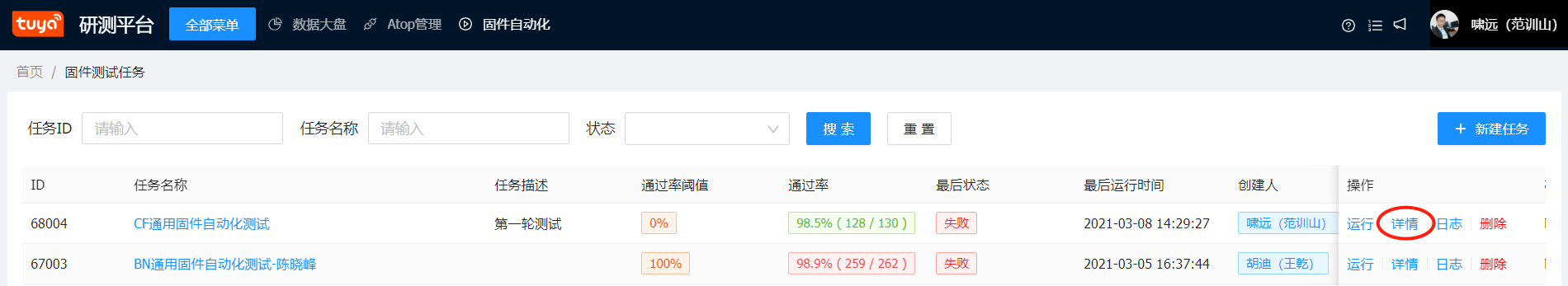


## 6、选择运行设备

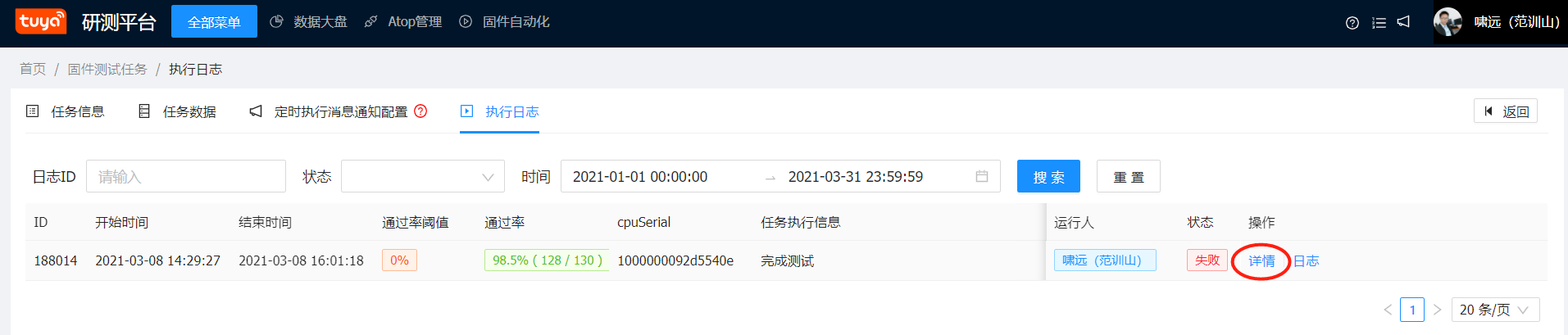


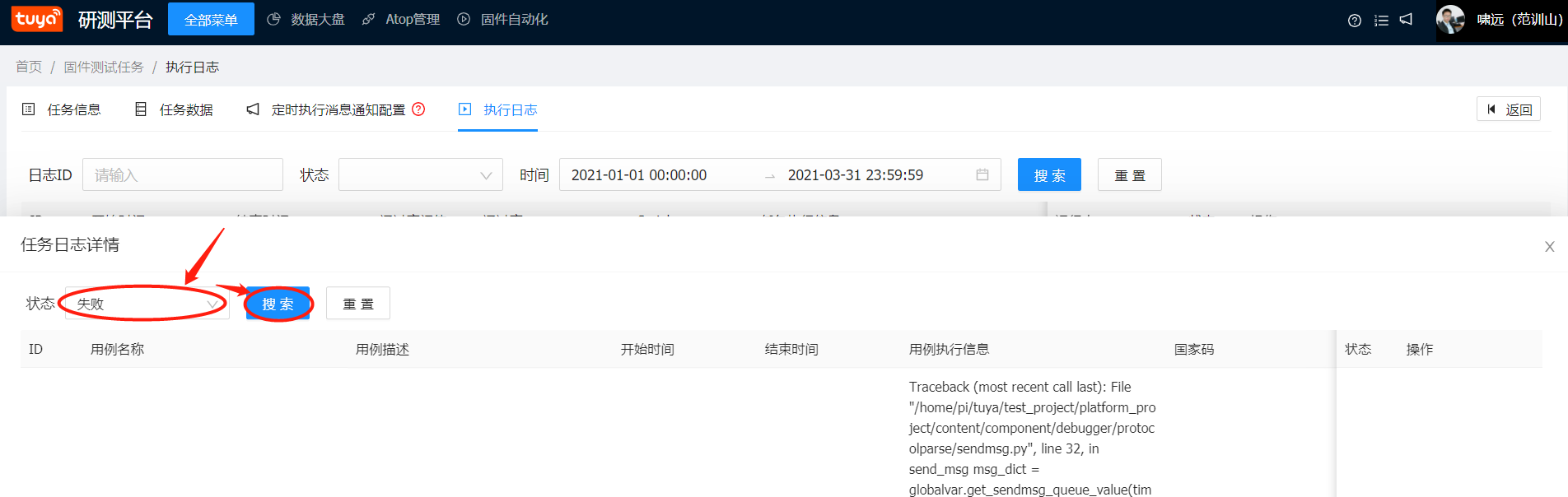


## 7、查看测试结果







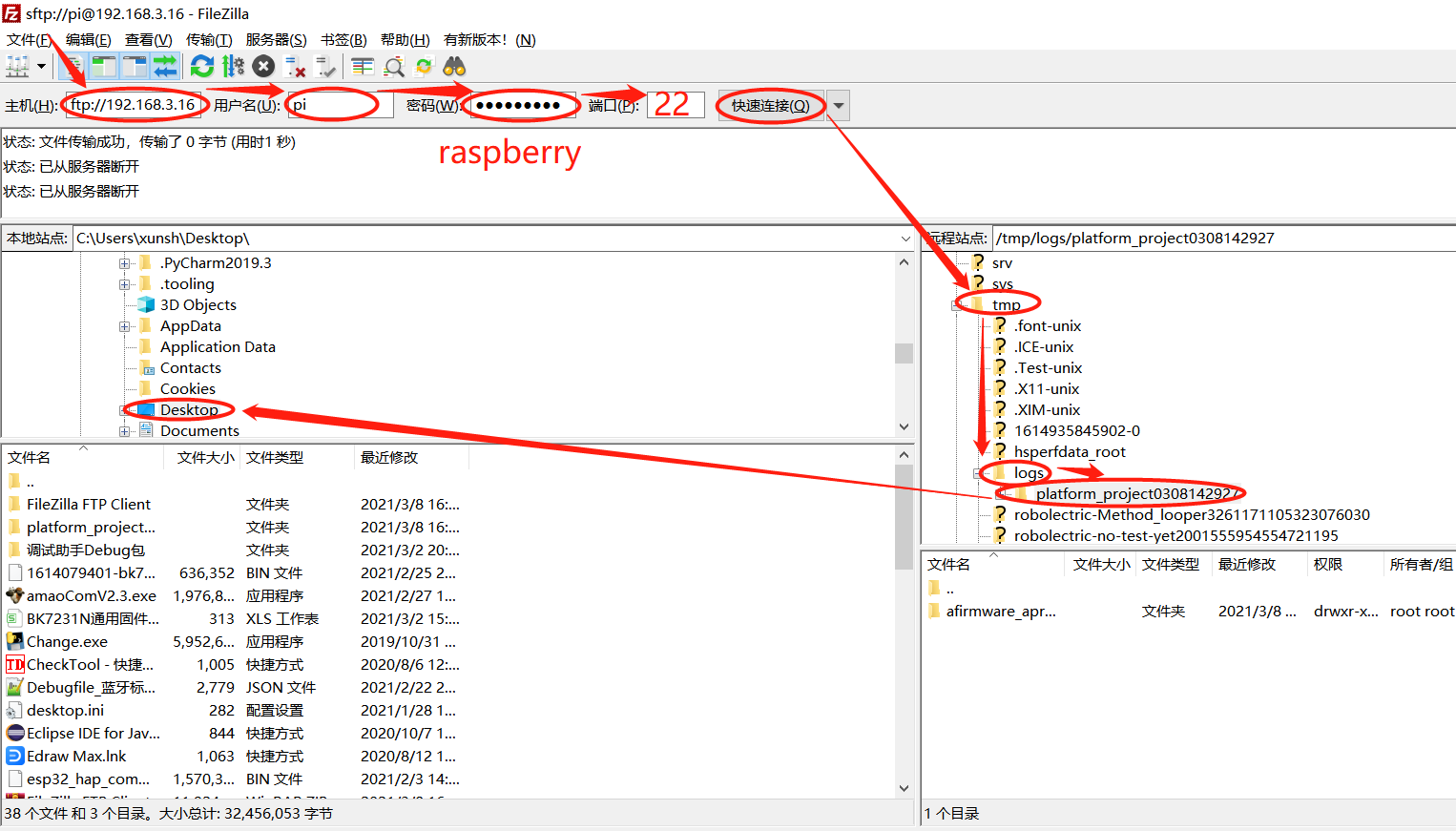


## 8、获取服务器log

连接屏蔽房路由器wifi，也可以使用网线连接路由器，访问192.168.3.14来局域网下载log



打开sftp工具，推荐winscp工具，百度可以直接获取



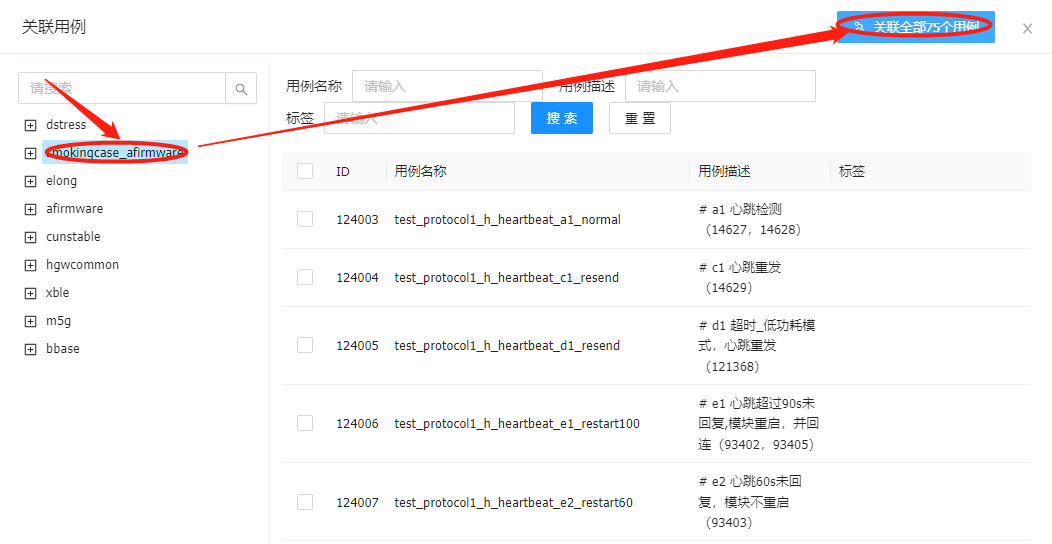
# 二、通用固件冒烟用例的选择

基本步骤同一

# 用例库的选择

# 2、冒烟用例的关联





# 三、自动化测试开发环境搭建

## 1、搭建开发环境的准备

需要设备：树莓派一个、SD卡一张、开发板一个、电源线一个

## 2、树莓派环境搭建

参考链接：<https://wiki.tuya-inc.com:7799/page/51792620>

## 3、测试工程环境搭建

gitlab账号可在网址：https://registry.code.tuya-inc.top/ 根据提示申请账号：

测试工程代码权限找测试经理小码(李军)开通

测试工程gitlab地址：

https://registry.code.tuya-inc.top/tuya\_embedded\_automation/project/testwififirmware

git代码下载地址：

https://registry.code.tuya-inc.top/tuya\_embedded\_automation/project/testwififirmware.git

提示：git客户端下载，参考百度

使用pycharm对测试工程远程调试：

提示：只有pycharm专业版才有远程调试功能，如何安装，参考百度

远程调试参考：<https://wiki.tuya-inc.com:7799/page/53841973>

环境安装完毕，可以通过pycharm deployment->upload to 树莓派，进入testcase/afirmware/aprotocol1/test\_protocol1\_a\_function.py 找个用例，右键run一把，如果返回成功，表示环境安装完成

## 4、Python基本能力疏通

参考链接：<https://wiki.tuya-inc.com:7799/page/51799482>

## 5、嵌入式代码编程规范

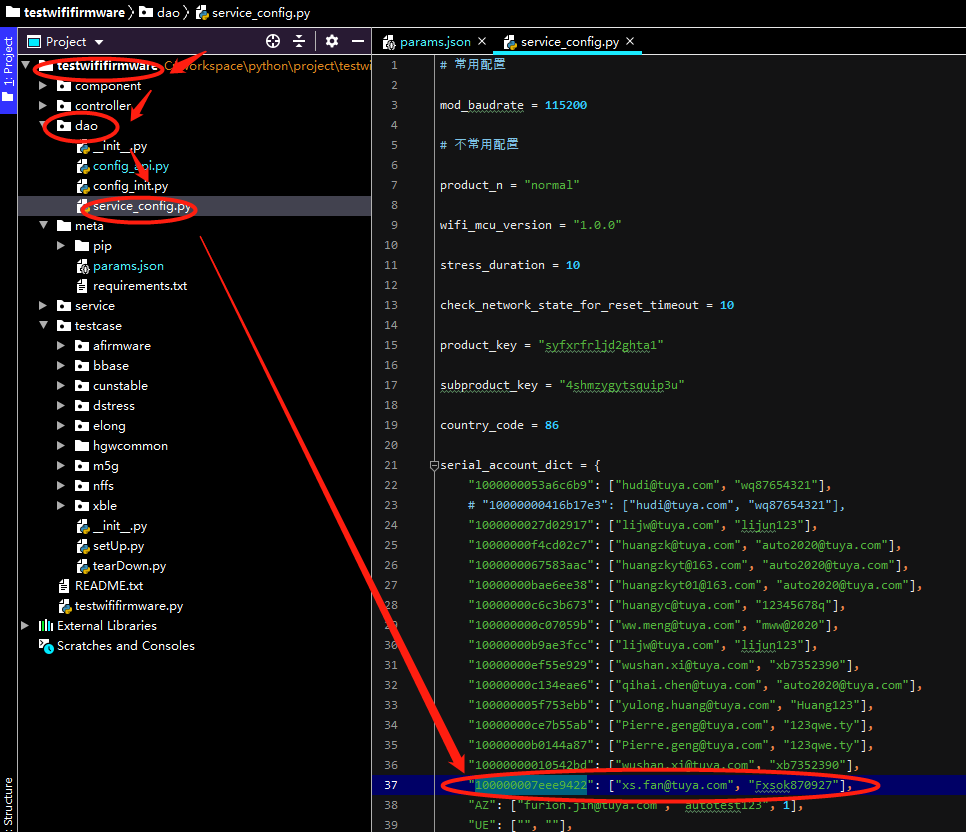
参考连接：<https://wiki.tuya-inc.com:7799/page/51781263>

# 四、代码结构讲解

## 1、自动化测试配置信息

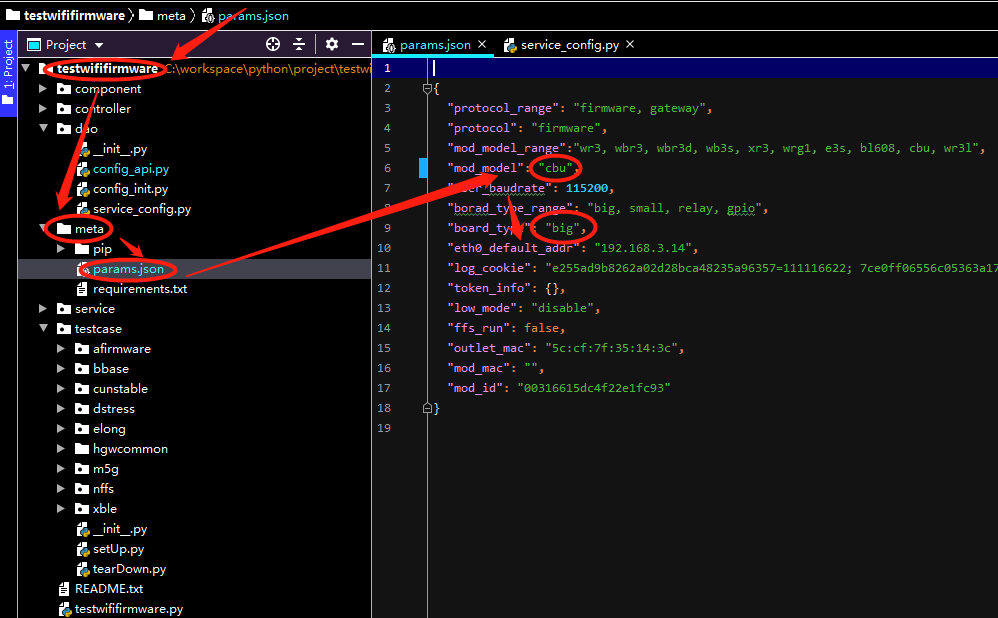
（1）开发板编号、对应开发账户信息、Json信息

testwififirmware> dao> service\_config.py



（2）测试模块配置信息

testwififirmware> meta> params.json



## 2、目录结构简介

项目名称：testwififirmware

主文件：testwififirmware.py

说明文档：README.txt

测试用例目录：testcase

用例目录下固件用例：afirmware

用例目录下压测用例：dstress

测试用例函数实现调用目录：testcase> controller\_\*\*> protocol\_service\_api> protocol\_debugapi> protocolparse\_\*\*\*

目录对应关系：

Controller> controller\_\*\*

Server> protocol\_service\_api

Component> protocolparse\_\*\*\*

## 3、分层信息

包括组件(component)、业务（service）、数据（dao）、用例逻辑（controller）、用例管理（testcase）、环境数据（meta）

## 4、用例管理层介绍（testcase）

用于存放管理用例，组织用例顺序，也是用例调试执行入口。使用unittest, 可以以包，模块，类，函数来多层次管理用例。

unittest根据ascii码来调节用例顺序。本工程根据unittest.TestCase类的assertEqual来判断用例执行结果。

setUp为用例设置运行环境

tearDown为用例清理运行环境

行数首行的注释，基本能反馈该用例的测试步骤

（1）afirmware （通用固件用例集）

aprotocol1 主要包括协议类测试

bprotocol2 根据模块状态划分的协议类测试

cactivate 配网测试

dexpand 通用固件扩展类协议

edp dp点测试

fabnormaldp dp点异常测试

hmode wifi模块各种模式切换

mbaudrate 波特率自适应功能

vsersion 版本兼容测试

（2）base (基线用例集)

aactivate 基线配网测试

bdp dp点测试

eremove 移除测试

fmode 模式切换测试

hreconnect 回连测试

irouter1 路由器兼容测试1

jrouter2 路由器兼容测试2

ktimer 定时测试

ologseq 日志序测试

（3）cunstable (不稳定用例)

因为模块返回不确定导致的不稳定用例集合

（4）dstress （压测）

adpstress dp压测,基础协议压测

egwdpstress dp正常异常压测

h5gstress 5g回连，多次重试，以及dp压测

还包含配网，重置，定时，快联，蓝牙等相关压测

（5）elong (长运)

包含配网后长时间监测，异常后长时间监测，5g长时间后回连

（6）hgwcommon (网关相关)

agwprotocol 网关基础指令测试

bsubdevice 网关子设备测试

cgwbraudrate 网关波特率自适应

dgwdpissueandreport 网关dp测试

（7）m5g （5g相关用例）

包括5g配网，模式切换，加密，dp功能

（8）nffs （ffs配网用例）

ffs配网相关用例

（9）xble (蓝牙相关用例)

adeplopy 蓝牙配网

bcontrol 蓝牙发送dp

plug\_play plug&play功能

## 5. 用例逻辑层介绍（controller）

用例相关步骤在该层进行组织。

该层会调用很多service层的一些单功能接口，组织成真实的测试场景，对照手工测试用例，通过对service接口的拼接，能快速组织用例。

部分用例步骤较为相似，为减少重复代码，做了一定的抽象与功能组合。

该层代码，分模块根据业务内容，与基线还是通用固件无关

包的命令以controller\_业务名方式，根据业务名，模块名，函数名，可以快速与业务功能联系出对应的测试用例场景

## 6、环境数据介绍 （meta）

pip 包含难下载的包，跟内部包

param.json参数解释

protocol: 协议类型，目前适配通用固件跟网关协议

mod\_model: 模组型号，有根据不同模组，做特殊处理的情况

user\_baudrate: 用户口波特率，非自适应的模块，需要填写

board\_type： 开发板类型，big：就是那个比板砖还大的板子，small：陆渴大佬做的板子，relay:接继电器，gpio：陈远理老板上下电方案

eth0\_default\_addr: 给树莓派绑定默认ip

log\_cookie: 日志序测试前，需要先去 涂鸦运营平台> 日志平台，f12搞个cookie, 如果不知道怎么操作，求助百度吧

token\_info: 对接研测平台的一个保留参数

low\_mode: 打开这个功能，相当于协议中模式的low字段为1，不清楚通用固件协议的，自己去要份

ffs\_run: 是否有ffs功能，ffs需要搭建专门的环境才能测试

outlet\_mac: ffs用的插座的mac地址，配网成功后，自己从app端搞，当然每个环境只需搞定一次，自己拿小本本记着就行

requirements.txt里面有测试工程依赖包的具体版本，在平台执行用例时候用到

## 7、数据层介绍 （dao）

一些基础参数配置，并不直接来自外部参数，而是来自此处。

config\_api: 其他模块需要使用外部参数时从中获取

config\_init: 需要根据平台查询的配置信息，以及初始化配置

service\_config: 账号信息，产品schema等一些可以固定不变的参数，来自此文件

## 8 业务逻辑层介绍（service）

很多单功能步骤会在该进行生成。可以理解为用例逻辑层的一些接口。用例逻辑层就是根据，service\_api中函数进行拼接形成的。新接入业务，可能会需要增加service\_api中的接口。想要速成用例，必须合理利用service\_api中的接口。了解目前已经实现的接口，就能明白目前自动化工程所具备的用例覆盖能力。除了对外的service\_api, 该层很多内容就是为实现service\_api而展开的。也有很多service\_api来自组件能力。

service\_api 介绍:

activate\_serviceapi: 主要实现了app配网、串口配网、插座配网功能接口

dp\_serviceapi: 主要实现了，dp上报, dp下发， dp查询功能，还包括插座dp点下发

excel\_seriveapi: 主要实现存储重置结果excel表

expand\_serviceapi: 主要实现清理扩展协议的残留数据清理

globalvar\_serviceapi: 主要实现了获取模块重启次数

heatbeat\_serivceapi: 主要实现了获取心跳次数，获取心跳状态，打开心跳重试等心跳相关接口

iot\_seriveapi: 主要实现了登录账号密码设置

mode\_serviceapi: 主要实现了，模组的模式切换

modinfo\_seriveapi: 主要实现了 快联日志，模组连接状态等从模组日志中能获取的信息

modstate\_serviceapi: 主要实现了模组处于启动，初始化，工作，配网状态获取

netstate\_serviceapi: 主要实现了，模组网络状态的设置，与查询

network\_serviceapi: 主要实现了，打开关闭路由，打开关闭网络等修改路由器相关功能

power\_serviceapi: 主要实现了，上下电功能

product\_serviceapi: 主要实现了，模组产品信息的获取与设置功能以及工作模式

protocol\_serviceapi: 主要实现了，获取内存，mac,信号强度，重置等协议指令功能

query\_serviceapi: 主要实现了，设置获取一些测试工程系统内部状态功能

queue\_serviceapi: 主要实现了，串口服务使用队列实现的数据交互方法

remove\_serviceapi: 主要实现了，app对模块移除，恢复出厂设置

sdk\_serviceapi: 主要实现了，app sdk的登录参数设置与启动

serialvar\_serviceapi: 主要实现了，模块与树莓派之间的串口信息记

sysinit\_serviceapi: 主要实现了，测试工程启动时的一些初始化

tread\_serviceapi: 主要实现了，串口服务线程的启动与停止

timer\_serviceapi: 主要实现了，定时器增加查询，删除功能

weather\_serviceapi: 主要实现了，天气数据获取，下发，以及天气状态获取

wiring\_serviceapi: 主要实现了，大板子自动接线功能

## 9 组件层介绍 (component)

该层主要是一些独立功能模块，不涉及多端交互，可独立使用。

ble: wifi蓝牙双模，包含蓝牙的基础功能

debugger:类似于桌面串口调试助手,模拟MCU与模块之间进行数据交互。

1）clearlogs : 为防止日志文件过大的日志清理模块

2) comment: 与调试助手类似，模块在接收与发送的指令都有对应中文提示，为了方便查看

3）common: 串口服务的通用处理

4）dealmodinfo: 模组日志信息在此进行处理

5) dealuserinfo: 接收到的串口数据在此进行处理

6）debugger\_api: 串口服务的对外功能，service层调用的基础协议指令入口

7）modmanage: 模块所处的状态查询与监控

8）protocolparse: 对接收的串口指令进行解析，检测是否符合预期

9）serialoperate: 串口数据收发读取，写入

10）threadmanage: 串口服务，日志等线程管理

11）var: 串口服务的内部变量

iotcloud:云端打通的一些atop接口。主要包括定时，移除功能。

logplaform:日志平台接口，在获取日志序时使用，查询日志信息。但是需要配置cookies

logs:测试用例在调用testlog, modlog等日志方法后运行后会在/tmp/logs目录记录各种运行日志。 包括测试日志，模块日志，hostapd日志，串口交互日志，串口原始收发日志。

network:通过网络软件实现的对树莓派网络wlan0的操作。 主要包括打开断开路由器以及网络。在service使用时调用network\_api接口

power:实现的多种电源接入方式,来控制模块的上下电功能，调用入口power\_api

sdk:由画十移植的树莓派上的sdk版本，实现app上的功能。测试工程在初始化时，会启动sdk。主要使用其中的下发dp点，以及查询dp功能。

testplatform:研测平台对接入口。研测平台提供的查询接schema口，当前版本未使用.

wiring大板子接线功能。可以根据模块mod\_model确定不同的接线方法。设置接口来自于autopi树莓派平台。

10、网络状态解释

ez: 处于ez配置状态

ap: 处于ap配置状态

ezap: 处于ezap共存状态（需要产品信息中设置该模式）

norouter: 处于未连上路由状态

router: 处于连上路由，未连上云端状态

cloud: 处于连上云端状态

衍生的模块网络状态:

因为部分模块在配网前与配网后，虽然模块处于同一状态，但是表现形式不一致，为了保证测试完整性，添加各种状态

norouter0： 未经连云激活成功的模块（重置过的也是未激活状态，例如：配网过程中有该状态），当前处于未连上路由状态

norouter1: 已连云激活成功的模块（激活后，未经重置的模块，例如：连云之后断开路由器有该状态），当前处于未连上路由状态

router0:未经连云激活成功的模块（重置过的也是未激活状态，例如：配网过程中有该状态），当前处于连上路由，未连上云端状态

router1:未经连云激活成功的模块（激活后，未经重置的模块，，例如：连云之后断开外网有该状态），当前处于连上路由，未连上云端状态

serialcloud: 通过串口配网成功，当前处于连上云端状态

serialcloud\_ez: 模块处于ez状态，通过串口配网成功，当前处于连上云端状态

serialcloud\_ap: 模块处于ap状态，通过串口配网成功，当前处于连上云端状态

appcloud: 通过app配网成功，当前处于连上云端状态

appcloud\_ez: 模块处于ez状态，通过app配网成功，当前处于连上云端状态

appcloud\_ap: 模块处于ap状态，通过app配网成功，当前处于连上云端状态

cloud\_reconnect： 激活成功后，模块因为环境或者自身机制，导致状态不是连云。但是在未经过串口配网或者app配网的情况下，再次连上了云端。

# 五、自动化测试用例新增

## 1、基本服务功能解析

### （1）基本用例新增

文件：testwififirmware> testcase> afirmware> afirmware> aprotocol1> test\_protocol1\_a\_function.py

|  |
| --- |
| # 模块设备的基本信息获取 class ProtocolBDevInfo(unittest.TestCase):  def setUp(self) -> None:  controller\_setup\_common.setup\_wifi(self)   def tearDown(self) -> None:  controller\_teardown\_common.teardown\_sleep()   # @unittest.skip("abc")  def test\_protocol1\_a\_function\_b1\_memory(self):  *"# b1 获取wifi模块剩余内存（14646）"* resp = protocol\_serviceapi.get\_memory()  self.assertEqual(resp > **0,** True**,** msg=resp) |

### （2）协议接口定义

文件：testwififirmware> service> service\_api> afirmware> protocol\_serviceapi.py

|  |
| --- |
| # 获取内存 # 实现步骤：1.发送获取内存指令 2.捞取模块发送的指令，看是否存在返回内存指令 3.解析返回指令，解析出内存值 def get\_memory():  return protocol\_debuggerapi.get\_memory() |

### （3）协议调试接口定义

文件：testwififirmware> component> debugger> debugger\_api> protocol\_debuggerapi.py

|  |
| --- |
| # 获取内存 # 实现步骤：1.发送获取内存指令 2.捞取模块发送的指令，看是否存在返回内存指令 3.解析返回指令，解析出内存值 def get\_memory():  testlog("info"**,** "[debugger ] get\_memory start ...")  result = protocolparse\_function.WifiMemory.run()  testlog("info"**,** "[debugger ] get\_memory success.")  return result |

### （4）定义获取内存的协议解析类

文件：testwififirmware> component> debugger> protocolparse> protocolparse\_function.py

|  |
| --- |
| class WifiMemory(ProtocolParseBase):  biztype = "memory"   @classmethod  def check\_resp(cls**,** resp\_data):  memory\_size = resp\_data[**0**] \* pow(**256, 3**) + resp\_data[**1**] \* pow(**256,2**) + resp\_data[**2**] \* pow(**256, 1**) +resp\_data[**3**] \* pow(**256, 0**)  if (**0** <= memory\_size <= **100** \* **1024** \* **1024**):  return memory\_size  reason = "[debugger ] wifi memory\_size must during [0, 100M]. memory\_size:{memory\_size}".format(memory\_size=memory\_size)  testlog("error"**,** reason)  raise Exception(reason) |

### （5）设置memory对应的log类

文件：testwififirmware> component> debugger> comment\_modreply.py

|  |
| --- |
| def memory\_comment(moddata):  memory\_size = moddata[**0**] \* pow(**256, 3**) + moddata[**1**] \* pow(**256, 2**) + moddata[**2**] \* pow(**256, 1**) + moddata[**3**] \* pow(**256, 0**)  return "应答内存 {memory\_size}".format(memory\_size=memory\_size) |

## 2、协议交互项用例新增

### （1）测试用例新增

文件：testwififirmware> testcase> afirmware> dexpand> test\_expand\_e\_getWifiModInfo.py

|  |
| --- |
| import unittest  from service.service\_api import protocol\_serviceapi from controller.controller\_env import controller\_setup\_common**,** controller\_teardown\_common   # 获取Wifi模块相关信息 class ExpandAGetWifiModInfo\_AP(unittest.TestCase):  def setUp(self) -> None:  controller\_setup\_common.setup\_wifi(self)   def tearDown(self) -> None:  controller\_teardown\_common.teardown\_sleep()   # @unittest.skip("abc")  def test\_expand\_e\_getWifiModInfo\_a1\_apssid(self):  *"# a1 获取AP ssid名称"* resp = protocol\_serviceapi.getWifiModeInfo\_apssid()  self.assertEqual(resp**,** True**,** msg=resp)   # @unittest.skip("abc")  def test\_expand\_e\_getWifiModInfo\_a2\_countryCode(self):  *"# a2 获取国家码"* resp = protocol\_serviceapi.getWifiModeInfo\_countryCode()  self.assertEqual(resp**,** True**,** msg=resp)   # @unittest.skip("abc")  def test\_expand\_e\_getWifiModInfo\_a3\_sn(self):  *"# a3 sn"* resp = protocol\_serviceapi.getWifiModeInfo\_sn()  self.assertEqual(resp**,** True**,** msg=resp)   # @unittest.skip("abc")  def test\_expand\_e\_getWifiModInfo\_a4\_all(self):  *"# a4 获取所有当前已支持的数据内容"* resp = protocol\_serviceapi.getWifiModeInfo\_all()  self.assertEqual(resp**,** True**,** msg=resp) |

### （2）实现协议接口

文件：testwififirmware> server> service\_api> protocol\_serviceapi.py

|  |
| --- |
| # WIFI模块相关信息apssid # 实现步骤： 1.发送命令，捞取返回指令，并校验返回指令格式 2.校验返回，01 00 表示打开成功，其他返回错误 def getWifiModeInfo\_apssid():  return protocol\_debuggerapi.getWifiModeInfo\_apssid() |

### （3）实现协议调试接口

文件：testwififirmware> component> debugger> debugger\_api> protocol\_debuggerapi.py

|  |
| --- |
| # 获取WIFI模块相关信息apssid def getWifiModeInfo\_apssid():  testlog("info"**,** "[debugger ] getWifiModeInfo start ...")  result = protocolparse\_wifiModeInfo.GetWifiModeInfo\_apSsid.run()  testlog("info"**,** "[debugger ] getWifiModeInfo success ...")  return result |

### （4）定义测试指令的协议解析类

文件：testwififirmware> component> debugger> protocolparse> protocolparse\_wifiModeInfo.py

|  |
| --- |
| from component.debugger.protocolparse.protocolparse\_base import ProtocolParseBase from component.debugger.common.command import \*  class GetWifiModeInfo\_apSsid(ProtocolParseBase):  biztype = "expandservice"   @classmethod  def get\_mcu\_data(cls**,** \*\*kwargs):  return [**7, 1**]   @classmethod  def check\_resp(cls**,** resp\_data):  mcu\_hexcmd = " ".join([hex(x).replace("0x"**,** "").zfill(**2**) for x in resp\_data])  mcu\_string = hexToString(mcu\_hexcmd)  if resp\_data[**0**:**2**] == [**7, 0**]:  reason = "[debugger ] the open debuglog:{opendebug}".format(opendebug=mcu\_string)  testlog("info"**,** reason)  return True  else:  reason = "[debugger ] the open debuglog:{opendebug}".format(opendebug=mcu\_string)  testlog("error"**,** reason)  return False |

### （5）修改接收数据处理函数

定文件：testwififirmware> component> debugger> dealuserinfo> dealuserinfo\_expand.py

|  |
| --- |
| from component.logs.testlog import testlog from component.debugger.serialoperate import writeuserinfo from component.debugger.var import globalvar**,** expand\_var from component.debugger.dealuserinfo import dealuserinfo\_weather  class DealExpandService:   @classmethod  def run(cls**,** \*\*kwargs):  #self.save\_readdata()  # 获取回复命令  moddata = kwargs["moddata"]  biztype = kwargs["biztype"]  subbiztype = moddata[**0**]  if (subbiztype == **1**) or (subbiztype == **3**) or (subbiztype == **4**) or(subbiztype == **7**) or (subbiztype == **8**):  globalvar.put\_sendmsg\_queue({"biztype": biztype**,** "moddata": moddata})  elif subbiztype == **2**:  writeuserinfo.writedata(biztype**,** [**2**])  if moddata[**1**] == **0**:  expand\_var.put\_expand\_queue("greentime\_report"**,** moddata)  elif moddata[**1**] == **1**:  expand\_var.put\_expand\_queue("localtime\_report"**,** moddata)  else:  reason = "[debugger ] DealExpandService subbiztype 2, Unknow time notice reutrn moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason)  elif subbiztype == **5**:  writeuserinfo.writedata(biztype**,** [**5**])  if moddata[**1**] == **0**:  expand\_var.put\_expand\_queue("report\_localreset"**,** moddata)  elif moddata[**1**] == **1**:  expand\_var.put\_expand\_queue("report\_appreset"**,** moddata)  elif moddata[**1**] == **2**:  expand\_var.put\_expand\_queue("report\_appfactory"**,** moddata)  else:  reason = "[debugger ] DealExpandService subbiztype 5, Unknow report resetnotice reutrn moddata:{moddata}".format(  moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason)  elif subbiztype == **8**:  # 模块debug日志打开  if moddata[**1**] == **0**:  expand\_var.put\_expand\_queue("open\_debuglog"**,** moddata)  else:  testlog("error"**,** "[debugger ] DealExpandService subbiztype 8, Debuglog open fail. moddata:{moddata}".format(moddata=moddata))  else:  reason = "[debugger ] Unknow time notice reutrn moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason) |

### （6）修改comment的处理函数

定文件：testwififirmware> component> debugger> comment> comment\_mod.py

|  |
| --- |
| def expandservice\_comment(moddata):  subbiztype = moddata[**0**]  if subbiztype == **1**:  if moddata[**1**] == **0**:  return "[mod\_reply] 应答打开时间服务通知成功"  elif moddata[**1**] == **1**:  return "[mod\_reply] 应答打开时间服务通知失败"  else:  reason = "[debugger ] subbiztype 1 Unsupport moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason)  elif subbiztype == **2**:  if moddata[**1**] == **0**:  time\_info = comment\_utils.intlist\_to\_datestr\_noweek(moddata[**2**:])  return "[mod\_send] 上报格林时间 {time\_info}".format(time\_info=time\_info)  elif moddata[**1**] == **1**:  time\_info = comment\_utils.intlist\_to\_datestr\_noweek(moddata[**2**:])  return "[mod\_send] 上报本地时间 {time\_info}".format(time\_info=time\_info)  else:  reason = "[debugger ] subbiztype 2 Unsupport moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason)  elif subbiztype == **3**:  if moddata[**1**] == **0**:  return "[mod\_reply] 响应请求天气服务成功"  elif moddata[**1**] == **1**:  return "[mod\_reply] 响应请求天气服务失败"  else:  reason = "[debugger ] subbiztype 3 Unsupport moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason)  elif subbiztype == **4**:  if moddata[**1**] == **0**:  return "[mod\_reply] 响应打开重置状态成功"  elif moddata[**1**] == **1**:  return "[mod\_reply] 响应打开重置状态失败"  else:  reason = "[debugger ] Unsupport moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason)  elif subbiztype == **5**:  if moddata[**1**] == **0**:  return "[mod\_send] 上报本地重置状态"  elif moddata[**1**] == **1**:  return "[mod\_send] 上报app重置状态"  elif moddata[**1**] == **2**:  return "[mod\_send] 上报app恢复出厂设置状态"  else:  reason = "[debugger ] subbiztype 5 Unsupport moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason)  elif subbiztype == **7**:  if moddata[**1**] == **0**:  return "[mod\_reply] 获取Wifi模块相关数据信息成功"  else:  return "[mod\_reply] 获取Wifi模块相关信息信息失败"  elif subbiztype == **8**:  if moddata[**1**] == **0**:  return "[mod\_reply] 设置模块debug日志成功"  else:  return "[mod\_reply] 设置模块debug日志失败"  else:  reason = "[debugger ] Else subbiztype Unsupport moddata:{moddata}".format(moddata=moddata)  testlog("error"**,** reason)  raise Exception(reason) |

## 3、网络交互的用例新增

### （1）测试用例新增

# 防误触模式串口配网-mt时间设置，时间设置为10min，确认时间在【3,10】min按对应数据处理  
class ProtocolWifiFuncTest\_palmRejection\_10(unittest.TestCase):  
 def setUp(self) -> None:  
 controller\_setup\_common.setup\_wifi(self**,** mode="m2,n,low"**,** mt=**10**)  
  
 def tearDown(self) -> None:  
 controller\_teardown\_common.teardown\_sleep()  
  
 # @unittest.skip("abc")  
 def test\_mode\_a\_deploy\_f1\_serial\_deploy\_palmRejection\_10(self):  
 *"# f1 防误触模式，mt时间设置为10min，确认重置后10min，状态会回到先前的组网状态（346167）"* resp = controller\_serialactivate\_palmRejection.controller\_serialactivate\_palmRejection(deploy\_type="appcloud\_ez"**,** timeout=**10**)  
 self.assertEqual(resp**,** True**,** msg=resp)

### （2）controller中实现对应的校验

def controller\_serialactivate\_mt(timeout**,** \*\*kwargs):  
 testlog("warn"**,** "[controller] controller\_serialactivate\_lowpower start... kwargs:{kwargs}".format(kwargs=kwargs))  
 network\_serviceapi.set\_network\_ok()  
 netstate\_serviceapi.set\_netstate("ap")  
 delt = **10** # 预留时间避免网络延迟引起用例不通过  
 if timeout < **3** or timeout > **10**:  
 sleeptime = **3** \* **60** + delt  
 else:  
 sleeptime = timeout \* **60** + delt  
 time.sleep(sleeptime)  
 new\_state = protocol\_debuggerapi.get\_network\_state()  
 testlog("warn"**,** "[controller] controller serialactivate\_lowpower status:{netstatus}".format(netstatus=new\_state))  
 if new\_state == "timeout":  
 return True  
 else:  
 return False

## 4、网络状态操作

### （1）网络环境设置

network\_serviceapi.set\_network\_ok()

### （2）网络操作切换到目标状态

netstate\_serviceapi.set\_netstate("ap")

状态包括：ez、ap、norouter0、norouter1、router0、router1、cloud、serialcloud、serialcloud\_ez、5g\_serialcloud\_ez

、serialcloud\_ap、5g\_serialcloud\_ap、appcloud、appcloud\_ez、5g\_appcloud\_ez、appcloud\_ap、5g\_appcloud\_ap、timeout、ezap

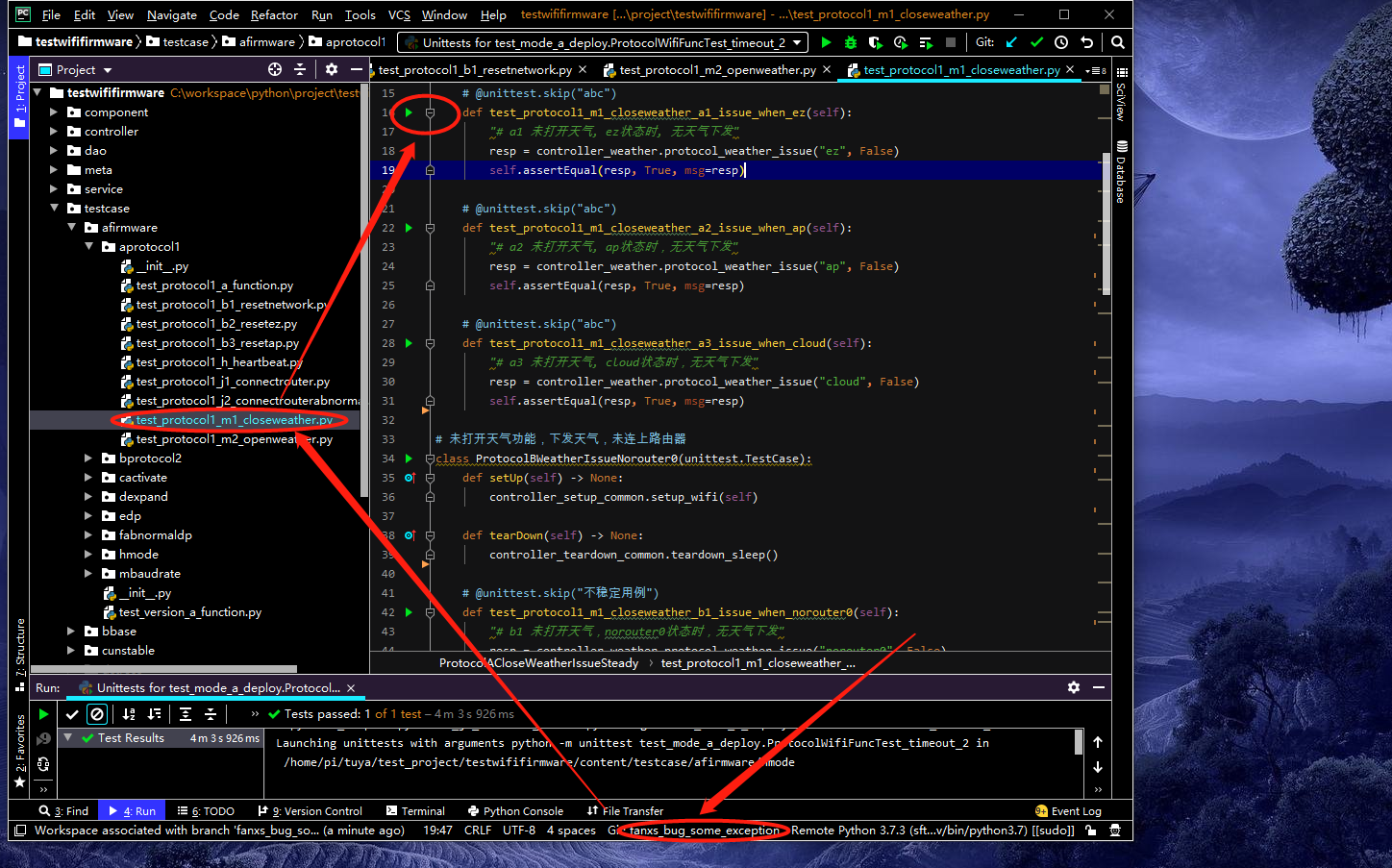
### （3）网络状态获取

new\_state = protocol\_debuggerapi.get\_network\_state()

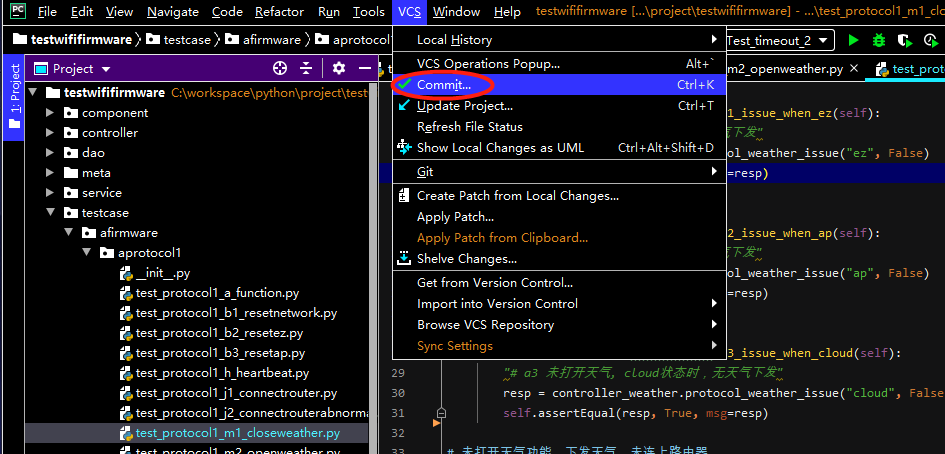
# 六、pycharm代码本地修改后的更新上传

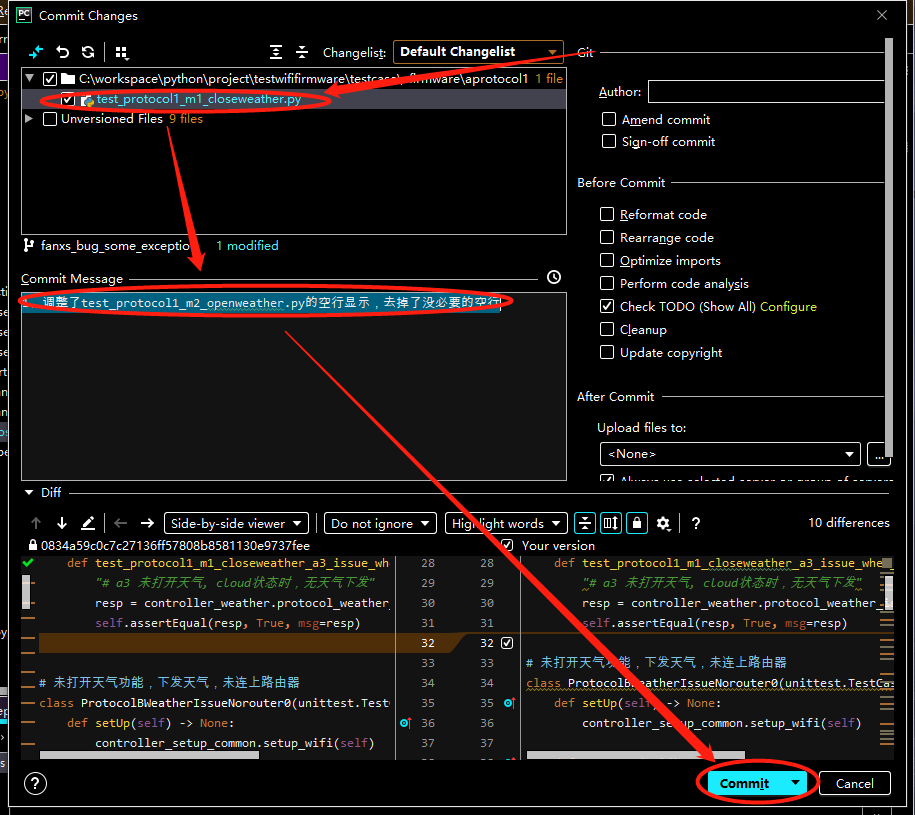
## 1、编写调整修改本地代码

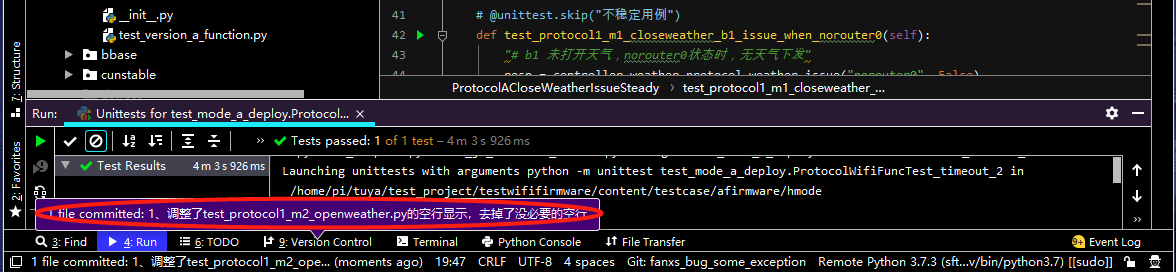
切到自己的分支开发修改代码



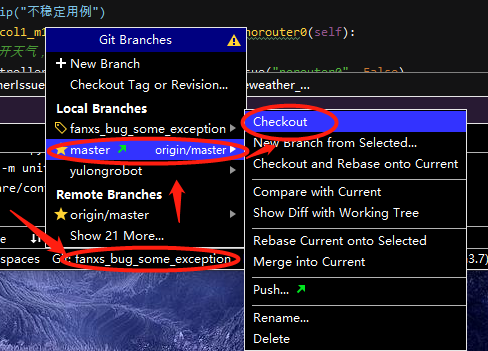
## 2、输入commit信息



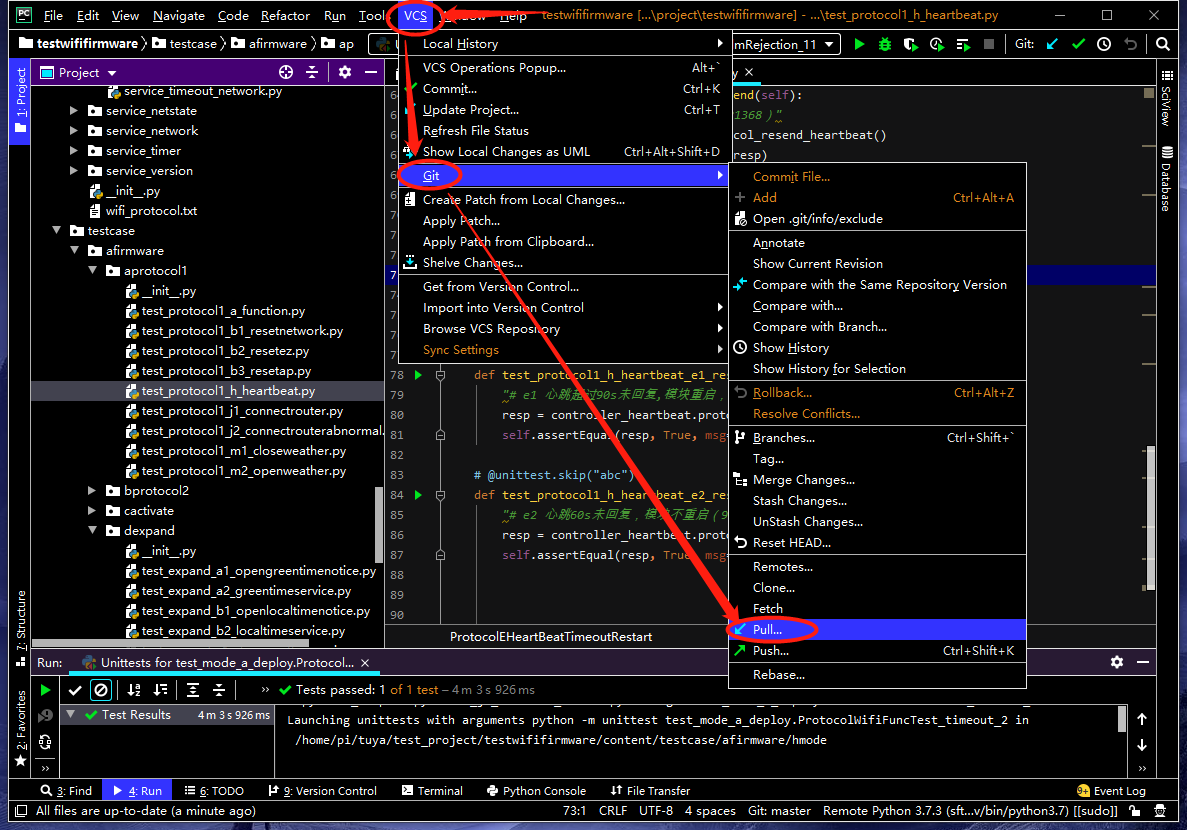




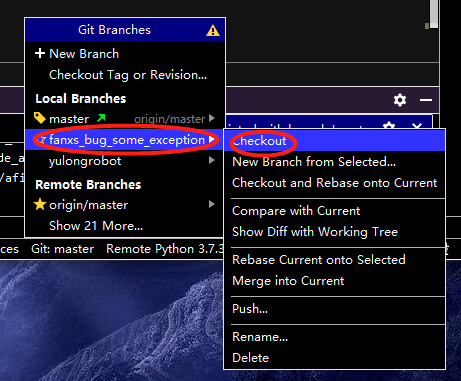
## 3、代码切换到主分支代码



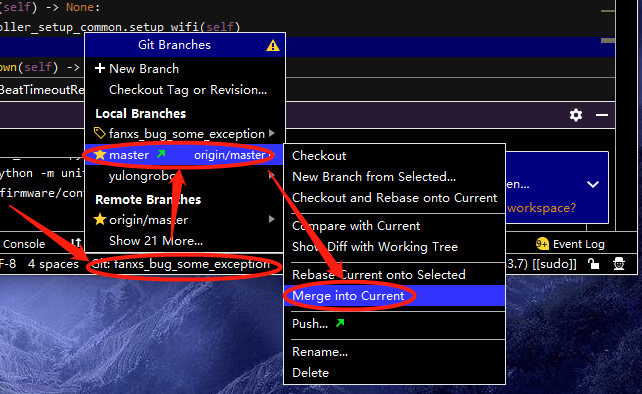
## 4、拉取主分支代码



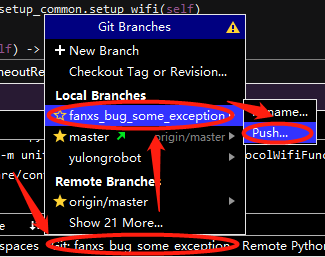
## 5、切换原来分支

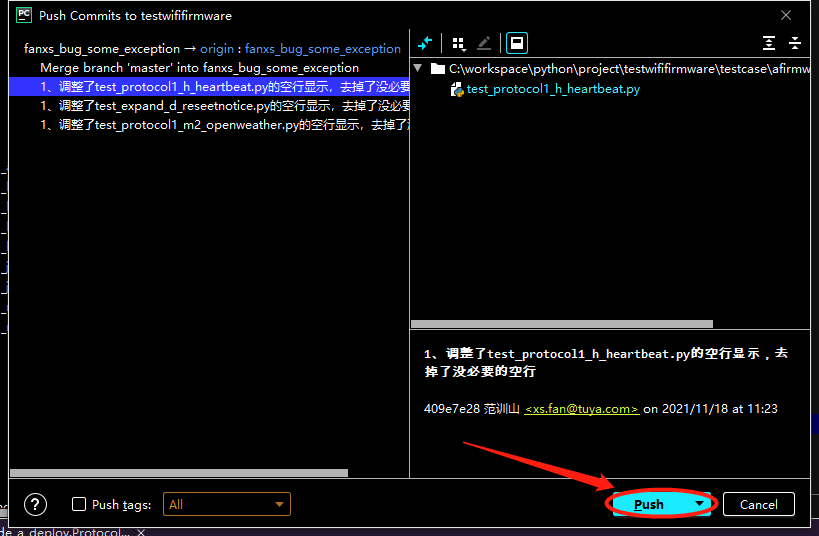


## 6、将主分支合并到子分支的代码中，该步可能有代码冲突需要确认

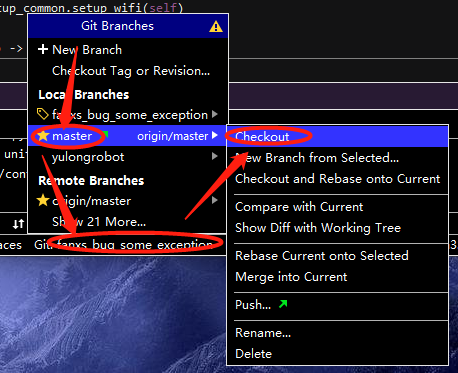


## 7、将本地的子分支代码push上去

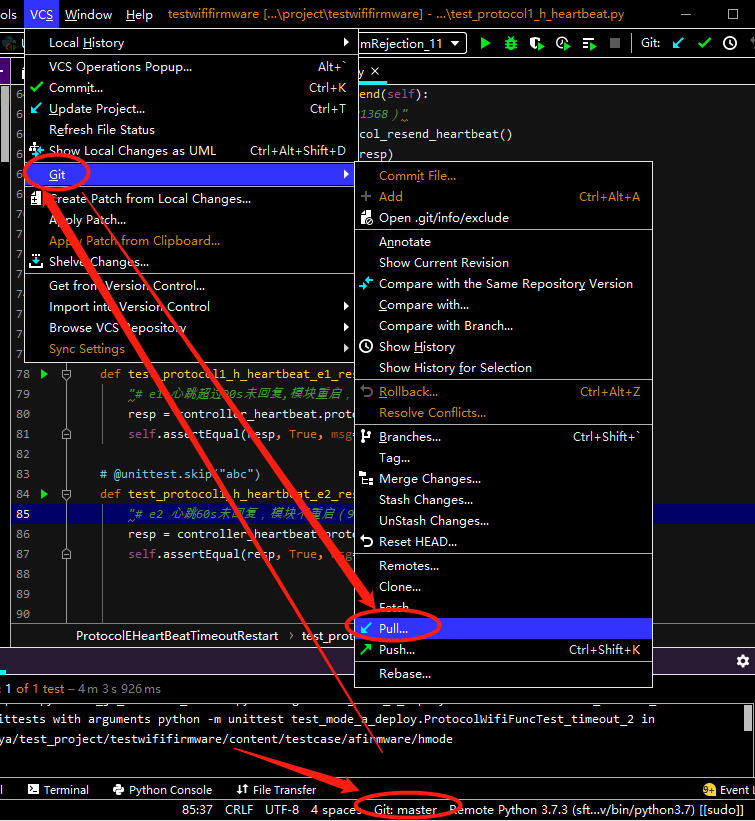




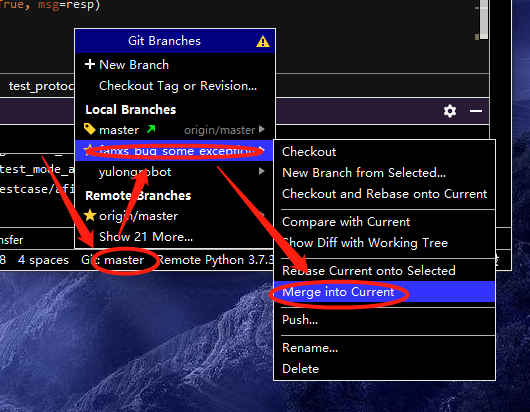
## 8、代码分支切换到主分支，后续步骤需要稳定后再操作



## 9、拉取主分支代码



## 10、将子分支向主分支合并



# 七、压测平台开发使用

## 1、压测平台测试设备信息的编辑

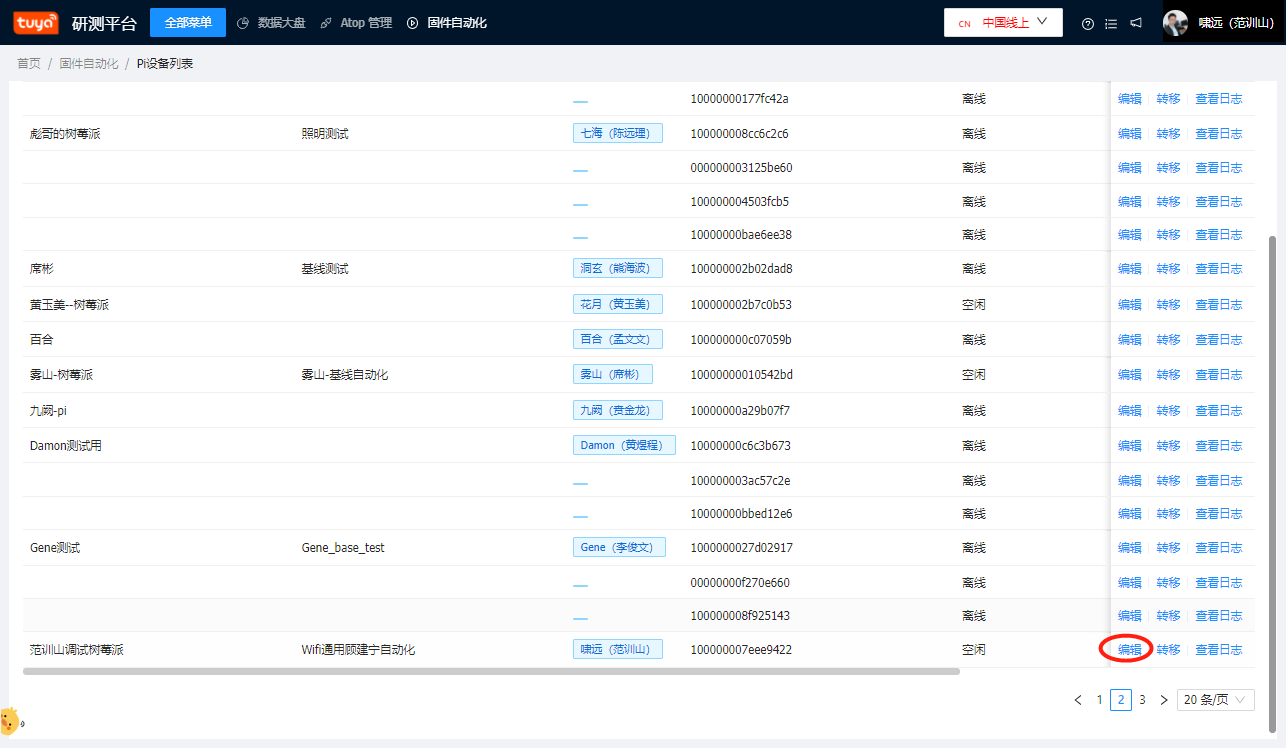
### （1）打开产测平台

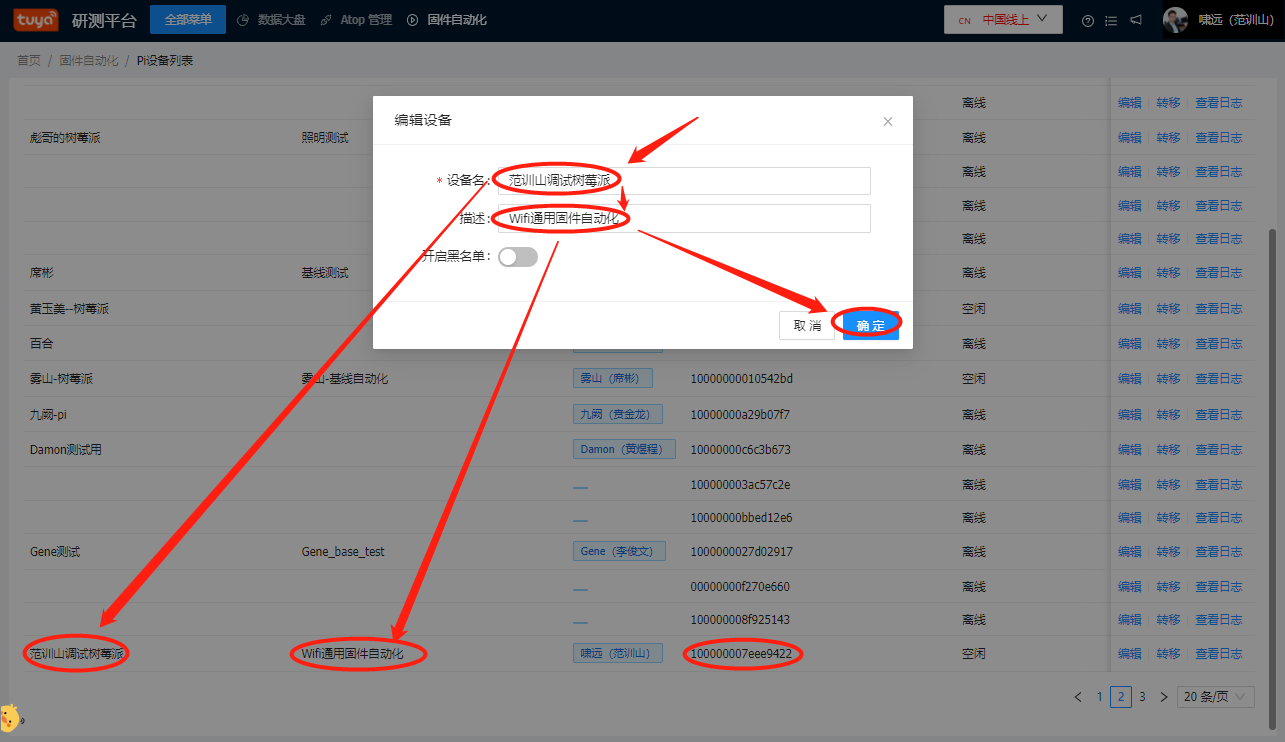
链接：<https://galaxy.tuya-inc.com:7799/firmware/auto/task/list>

### （2）进入设备列表界面



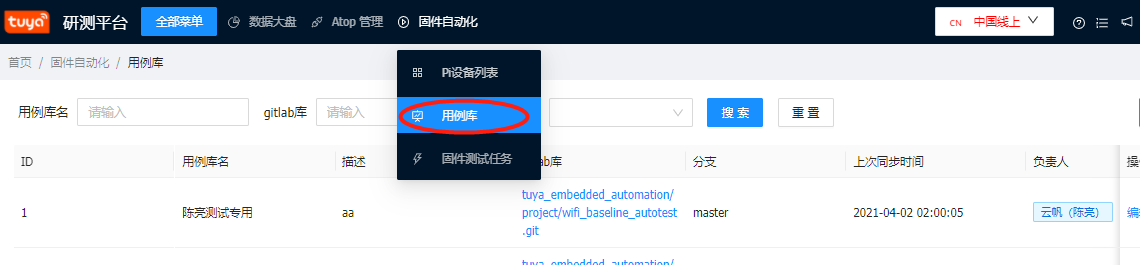
### （3）编辑个人设备的设备信息



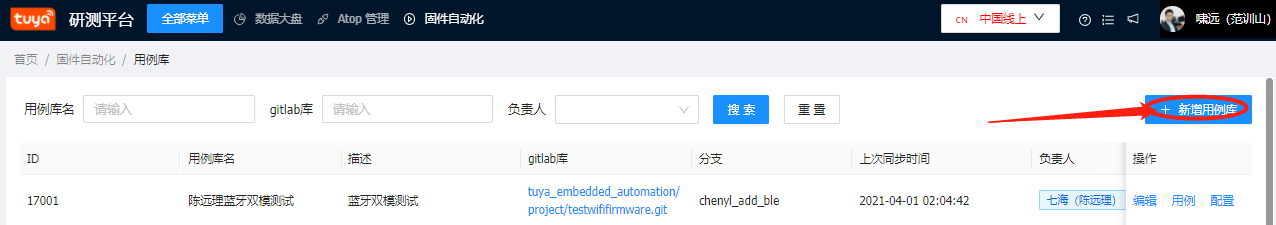


## 2、压测平台用例库的创建或编辑

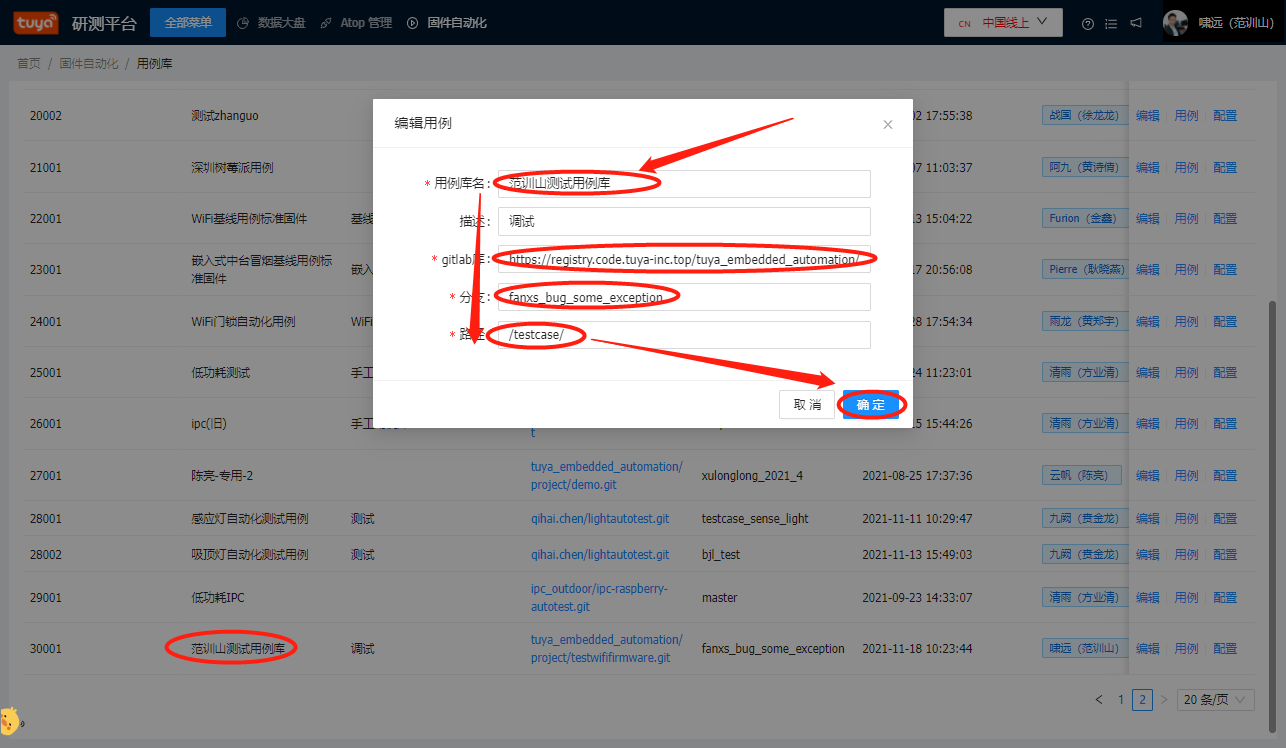
### （1）进入用例库管理界面



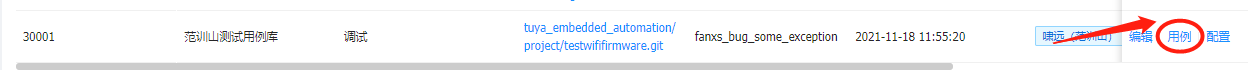
### （2）新增用例库



### （3）输入用例库信息



### （4）用例库同步用例



### （5）选择需要的用例项进行同步

