券商客户端自动化测试库

（0.0.4）

谁的谁（41715399@qq.com）

基础教程

一、简介

该项目是一个实验项目，旨在深度学习和挖掘 python pywinauto 库的功能和潜力，实现对中国境内券商客户端的自动化操作，完成自动化交易。该项目采用“客户端驱动型工厂模式”设计，所以从理论上讲，该项目具备了支持所有券商客户端及其未来版本的能力。

二、困难

编写这种 GUI 自动化测试是困难的，原因在于不确定性，主要表现在以下三方面：

1、环境不确定。客户端所处的运行环境不确定，如有一些软件经常会遮挡客户端，客户端运行的电脑性能会影响客户端的稳定。

2、自身不确定。客户端运行本身很不稳定，控件或隐或现，弹窗类型不确定，行为不固定。

3、未来不确定。券商的客户端很明显会升级，这会带来最大的不确定性，往往是调试好一个版本时，客户端就升级了。

三、实盘

正是由于上述不确定性，所以不建议将该软件应用于实盘交易，可以用来学习参考 pywinauto 库的应用技巧。

四、版本

券商客户端自动化测试库（pytradecn0.0.4）

谁的谁（41715399@qq.com）

五、声明

本软件遵守“MIT License”开源协议开源，仅供学习和参考。您可以自由使用或修改源代码或二进制文件，但必须保留上述版权声明。该软件旨在深度学习和挖掘 python pywinauto 库的功能和潜力，由于环境的不确定性和该软件的不可靠性，请不要将该软件应用于实盘交易。如您确需量化交易实盘功能，请使

用券商提供的量化交易平台，否则由于您使用该软件实盘交易所造成的账户损失或政策风险，开源软件提供者或插件提供者均不承担任何责任。同时，无论是直接的、间接的、偶然的、潜在的因使用该软件所造成的账号安全损失、数据安全损失、账户资产损失或其他任何责任事故，开源软件提供者或插件提供者均不承担任何责任。请不要将该软件应用于商业活动，否则由于把该软件应用于商业活动所造成的一切损失或法律责任，开源软件提供者或插件提供者均不承担任何责任。

六、项目特色

该项目是始于 2023 年初的一个实验项目，由于个人的原因而编写。该项目通过使用 python 第三方库 pywinauto，实现对券商客户端的自动化操作测试，包括自动登录、验证码识别、买卖、撤单、查询等功能。该项目深度应用 pywinauto 库，代码中有许多 pywinauto 库的应用技巧，包括该库存在的 BUG 也已在代码中标明。该项目具有以下特点：

1、该项目采用一种“客户端驱动型工厂模式”设计，或者叫“客户定制式工厂模式”。所以，从理论上讲，该项目具备了支持所有券商客户端及其未来版本的能力。

2、由于交易的严谨性和严肃性以及客户端行为的不确定性，所有的操作均有返回值，要么成功要么失败，不会因运行时错误而“卡”在半路。

3、自动化软件测试受内外环境的影响较大，该项目以最大的可能减少内外环境的变化对软件自动化的影响。

4、由于采用“客户端驱动型工厂模式”，所以项目可扩展性高、可根据不同的券商版本制作不同的交易模型也可以制作不同的部件适应不同的场景，如制作不同的登录引擎以适应不同的登录方式等。

5、调用接口简单，一行代码就能完成对客户端的调用，同时能够完全隐藏您的客户端信息，实现隐式调用。

七、技术交流

如果您有任何使用上的问题，也可以加入 QQ 群或联系作者 QQ

QQ 群号：5045383 ，我的 qq：41715399

八、安装

第一步：安装 pytradecn

运行 pip install pytradecn

该项目需要以下依赖库：

pywinauto-0.6.8

pillow

第二步：下载 Tesseract

下载并拷贝 Tesseract 文件夹到 pytradecn.utils 包内或修改 ocr.path

下载地址：https://github.com/tesseract-ocr/tesseract

注意：不要忘记下载简体中文支持包。

九、使用之前

使用之前应该尽量净化您的客户端运行环境，以保证您的券商客户端能够稳定运行。应先手动登录您的客户端保证其能正常运行，同花顺的客户端登录后应该关闭它的悬浮窗。

十、插件支持

pytradecn 内部集成了银河双子星的客户端支持，但可喜的是 pytradecn 允许以插件的形式支持不同的券商客户端。如您需要同花顺客户端的插件或银河双子星的插件，请发邮件索取或在 test 文件夹中。41715399@qq.com

十一、开始使用

我们以同花顺的客户端为例，将同花顺客户端的插件保存在您项目的任何位置，例如保存在包 mypacket 中。

十二、创建交易对象

首先应该创建交易对象，使用以下代码：

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

trader = Trader(client=THS92030)  # 创建交易对象
print(trader.query('当日委托')[1])
trader.close()  # 关闭交易是一个好的习惯
```

```

```

事实上，Trader 方法不需要显式输入参数，pytradecn 可以自动识别您定义的客户端，如下方法：

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

trader = Trader()  # pytradecn 会自动识别您的客户端 THS92030
print(trader.query('当日委托')[1])
trader.close()  # 关闭交易是一个好的习惯
```

或者，彻底隐藏客户端的导入，如把导入客户端放在包 mypacket 的 __init__.py 中，如下：

```
# 在 mypacket.__init__.py 中
from . import ths92030
```

在您的主代码中：

```
# 导入 pytradecn
from pytradecn import Trader

trader = Trader()  # pytradecn 会自动识别您的客户端 THS92030
print(trader.query('当日委托')[1])
trader.close()  # 关闭交易是一个好的习惯
```

十三、交易功能

默认状态下，pytradecn 内部集成的银河双子星或同花顺客户端只提供了买（buy）、卖（sell）、撤单（cancel）、查询（query）四个功能，可喜的是您可以修改或编写插件实现任何您想要的功能。

十四、处理错误

由于 GUI 程序运行很不稳定，鉴于此，pytradecn 设计成完成任何功能均会有输出，不会因运行时错误卡死代码，所以上述四个功能或者您自定义的功能的返回值永远是一个二元组，元组的第一项是标识成功与否的布尔值，元组的第二项是真正的返回值，如发生错误，第二项为发生错误的原因。所以在您的代码中一定要有容错处理。

十五、买入

创建了交易对象后就可以执行买入操作，使用下列代码：

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

trader = Trader()  # pytradecn 会自动识别您的客户端 THS92030

success, text = trader.buy(code='601002', price='4.18', count='100')
if success is True:
    print('委托编号：', text)
else:
    print('错误：', text)

trader.close()  # 关闭交易是一个好的习惯
```

注意：由于计算机的浮点数危机，如果采用 float 型，在金融和财务领域是致命的，所以 pytradecn 对所有的输入和输出均为字符串，特别是价格和数量等，这样便于在您的代码中使用 Decimal 类进行精确计算。

十六、卖出

同买入一致。

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

trader = Trader()

success, text = trader.sell(code='601002', price='5.18', count='100')
if success is True:
    print('委托编号：', text)
else:
    print('错误：', text)

trader.close()
```

十七、撤单

先参考如下代码：

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

trader = Trader()

success, text = trader.sell(code='601002', price='5.18', count='100')
if success is True:
    print('委托编号：', text)
    trader.cancel(合同编号=text)  # 撤销掉刚才的委托单
else:
    print('错误：', text)

trader.close()
```

事实上，cancel 功能非常强大，它可以撤销掉您任何想要的组合单。以下是可能的使用场景：

```
# 1、撤销全部委托单
trader.cancel()
# 2、使用一个关键字参数过滤委托单
trader.cancel(证券名称='农业银行')  # 将证券名称为农业银行的委托单撤销
trader.cancel(操作='买入')  # 将所有的买入委托单撤销
trader.cancel(合同编号='123456')  # 将合同编号为 123456 的委托单撤销
# 3、使用多个关键字参数过滤委托单
trader.cancel(证券名称='农业银行', 操作='买入')  # 将农业银行的买入单撤销
# 4、使用一个关键字参数，多值过滤委托单
trader.cancel(证券名称=('农业银行', '平安银行'))  # 将农业银行和平安银行的委托单撤销
trader.cancel(合同编号=('123456', '654321'))  # 将合同编号为‘123456’和‘654321’的委托单撤销
# 5、使用多关键字参数，多值过滤委托单
trader.cancel(证券名称=('农业银行', '平安银行'), 操作='买入')  # 将农业银行和平安银行的买入单撤销
```

十八、查询

客户端查询功能中的所有查询项，pytradecn 均可以完成查询，注意，

pytradecn 默认未提供日期选择功能，您可以自行编写代码添加。参考以下代码：

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

trader = Trader()

success, table = trader.query('当日委托')
if success is True:
    print(table)
else:
    print('错误: ', table)

trader.close()
```

如果查询成功，则会返回表格对象，table 对象是一个只读列表，您可以像操作列表一样操作它，包括遍历、索引以及一些有关列表的方法，但这个列表是只读的，不可以修改。列表中的每一个项就是表格中的一行，是一个只读字典，可以像字典一样去遍历、索引它，但不可以修改。

表格对象有一个重要的方法 items()，它可以按条件过滤表格，返回一个新表格，可能的应用场景如下：

```
# 1、返回表格副本
table.items()
# 2、使用一个关键字参数过滤表格
table.items(证券名称='农业银行')  # 过滤出证券名称为农业银行的数据
table.items(操作='买入')  # 过滤出所有的买入数据
table.items(合同编号='123456')  # 过滤出合同编号为 123456 的数据
# 3、使用多个关键字参数过滤表格
table.items(证券名称='农业银行', 操作='买入')  # 过滤出将农业银行的买入数据
# 4、使用一个关键字参数，多值过滤表格
table.items(证券名称=('农业银行', '平安银行'))  # 将农业银行和平安银行的数据过滤出来
table.items(合同编号=('123456', '654321'))  # 将合同编号为‘123456’和‘654321’的数据过滤出来
# 5、使用多关键字参数，多值过滤表格
table.items(证券名称=('农业银行','平安银行'), 操作='买入')  # 将农业银行和平安银行的买入单过滤出来
```

表格对象还有一个 item()方法，与 items()一样的参数，只不过 item()方法只返回唯一的一个项（字典类型），如果为空或有多个项则会报错。

十九、关闭交易

在您完成交易后，关闭交易是一个好的习惯，关闭的方法有两种

一种是使用 close()方法关闭，如下代码：

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

trader = Trader() # pytradecn 会自动识别您的客户端 THS92030
# 您的代码
trader.close()  # 关闭交易
```

一种是使用 with 关键字，如下代码：

```
# 导入 pytradecn 和同花顺客户端插件，导入顺序不分先后
from pytradecn import Trader
from mypacket.ths92030 import THS92030

with Trader() as trader:  # 使用 with 关键字会自动关闭交易
    # 您的代码
```

二十、编写插件

往往会编写插件以支持某一券商客户端或者某一特定的版本,需要编写的内容包括客户端配置、登录引擎、交易模型，或许可能还有一些特殊的自定义控件。pytradecn 已经内置了三种登录引擎，分别是不需要验证码的普通登录（DEFAULT）、验证码登录（VERIFYCODE）、主动刷新验证码登录（VERIFYCODEPLUS），内置的默认模型（DEFAULT）实现了 buy、sell、cancel、query 四种功能。如果内置的登录引擎、交易模型或者一些控件无法满足要求，则需要新建这些登录引擎、交易模型或者控件。最好的方法是针对某一券商的某一版本新建全套的插件，ths92030 是一个案例，它是同花顺 9.20.30 版本的支持插件;yh55891 是银河双子星 5.58.91 版本的插件支持。插件的组织方法有两种，一种是集中模式，一种是分散模式。ths92030 和 yh55891 采用的是集中模式。

集中模式

集中模式指将客户端配置、登录引擎、交易模型，或自定义控件集中编写在一个 py 文件中，在您项目的任何位置将该 py 文件引入，pytradecn 即可自动设别。这样做的好处是便于理解和传播，缺点是比较凌乱。

分散模式

分散模式即将客户端配置、登录引擎、交易模型，或自定义控件分别编写在不同的 py 文件中，放在同一个包内（事实上不同的包也是可以的），在包的__init__.py 文件中将这些 py 文件引入即可。

二十一、远程桌面

很多人喜欢把交易的代码部署在远程的服务器（计算机）上，自己的工作电脑通过远程桌面连接服务器（计算机）进行维护，这会出现问题。由于微软的远程桌面连接远程服务器（计算机）时，远程服务器（计算机）会自动锁屏，导致模拟鼠标键盘的操作出错，解决此问题的方法请参考：

https://www.kancloud.cn/gnefnuy/pywinauto_doc/1193039

进阶教程

一、Trader 方法的使用

pytradecn 有一个顶级方法 Trader，该方法是一个交易类，实例化 Trader 就可以创建一个交易对象。在同一个代码中可以多次实例化 Trader,通过 Trader 方法的 client 参数传入不同的客户端，以建立不同的交易对象，这些交易对象之间互相不干扰。也就是说在同一个代码中可以通过不同的 Trader 对象操控不同的券商客户端，即使 client 传入同一个客户端，也不会出现问题，只不过他们操纵的是同一个客户端（有谁会这样用呢？）。

Trader 方法的完整调用如下：

```
trader = Trader(client=None, user=None, psw=None, second=None, **account)
```

参数的含义如下：

client　　：客户端，默认为最后一次使用 import 引入的客户端

user　　：账号，默认为客户端中定义的账号

psw　　：密码，默认为客户端定义的密码

second　：第二密码，默认为客户端定义的密码

**account ：账号中的其他信息

完成交易后，随时关闭交易是一个好的习惯，可以使用 Trader 类中的 close() 方法关闭交易，也可以使用 with 关键字实现隐式关闭。

```
trader = Trader(client=None, user=None, psw=None, second=None, **account)
# ...您的代码
trader.close()  # 关闭交易

# 或者
with Trader(client=None, user=None, psw=None, second=None) as trader:
    # 您的代码
```

二、编写客户端类

不同的券商客户端或不同的客户端版本需要编写一个特定的客户端类，该类可以做为 Trader 方法的 client 参数传入，以实现交易对象和具体客户端的绑定。

编写一个客户端类的方法如下（以同花顺 9.2.30 为例）：

1、首先引入客户端的基类 BaseClient（或 Client）

```
# 先引入客户端的基类
from pytradecn import BaseClient
```

2、正确命名类名

```
# 客户端类的命名以大写字母和其版本号组成，区别不同的券商和版本
class THS92030(BaseClient):
```

3、设置正确的参数

账号信息参数（user、psw、second、account）

```
    # 账号信息
    user = '123456'  # 资金账号、客户号、上 A 股东、深 A 股东等
    psw = '654321'   # 第一密码
```

```
second = None    # 第二密码:通讯密码、认证口令等，没有时请设置为 None
account = {}     # 账户中的其他自定义信息
```

客户端安装位置参数（path）

```
# 客户端安装位置，大小写敏感，且盘符为大写
path = r'D:\Users\Administrator\xiadan.exe'
```

客户端信息参数（version、name、key）

```
# 客户端信息
version = '9.20.30_01'  # 也可以用来记录此代码的版本
name = '同花顺'
key = 'ths92030'  # 客户端设别符，重要关键参数，一定保持唯一
```

注意：参数 key 是一个重要参数，在 pytradecn 内部标识不同的客户端，如果您定义并 import 了多个客户端，应始终保持 key 值的唯一性，最好的取值方法是用其客户端名称小写字母和版本号组成。

客户端窗口规范参数（loginwindow、mainwindow）

loginwindow 是客户端登录窗口的规范，mainwindow 是客户端主窗口的规范，pytradecn 依靠这两个参数识别客户端目前所处的状态。loginwindow 和 mainwindow 的类型为一个字典（或包含字典的列表），字典中的 key 为描述窗口的特征，包括以下项：

title	：有这个标题的窗口，inspect.exe 下的 name 属性
title_re	：标题与此正则表达式匹配的窗口
class_name	：具有此窗口类的窗口
class_name_re	：类与此正则表达式匹配的窗口
control_type	：具有此控件类型的窗口
auto_id	：具有此自动化 ID 的窗口
framework_id	：具有此框架 ID 的窗口
predicate_func	：自定义验证函数
control_count	：该窗口在无任何弹窗时的子项数，*必填项*

以上参数使用 inspect.exe 查看，只要能把两个窗口区别开即可，没必要都填写（除 control_count 外）。注意：control_count 参数必须填写，该参数会用来设别弹窗，该参数应尽量取最小值。

下面是客户端 THS92030 的设置值：

```
   loginwindow = dict(title='', control_type='Pane', control_count=16)
   mainwindow = dict(title='网上股票交易系统 5.0', control_type='Window',
control_count=4)
```

值得一提的是，即使集成在行情软件中的交易客户端（例如通达信），loginwindow 和 mainwindow 也是支持的，只是要设置成列表即可。

部件选择参数（loginengine、trademodel）

loginengine 参数用来选择登录引擎，pytradecn 内置了三种登录引擎，分别是 DEFAULT（默认）、VERIFYCODE、VERIFYCODEPLUS。DEFAULT 实现了不使用验证码的通用登录方法，包括账号、密码、第二密码（或通讯密码）的登录方式；VERIFYCODE 实现了使用验证码的登录方式，包括账号、密码、第二密码（或通讯密码）、验证码；VERIFYCODEPLUS 登录方式与 VERIFYCODE 登录方式一样，只不过验证码输入前会主动刷新验证码。登录引擎可以自定义，以实现个别特殊的登录方式，有关自定义登录引擎的方法，请参考“编写登录引擎类”章节。

trademodel 参数用来选择交易模型，交易模型用来定义并实现您想要的功能。pytradecn 内置了一种交易模型 DEFAULT（默认），它实现了 buy（买）、sell（卖）、cancel（撤单）、query（查询）四个功能，注意：它实现的是银河双子星客户端的功能。您应该针对您的客户端自定义交易模型，以实现特定客户端的交易功能。有关自定义交易模型的方法，请参考“编写交易模型”章节。

下面是客户端 THS92030 的部件设置值，它使用自定义的登录引擎和交易模型。

```
loginengine = 'THS_9_20_30'
trademodel = 'THS_9_20_30'
```

速度模式参数（TRADE_SPEED_MODE）

TRADE_SPEED_MODE 参数用来设置操作客户端的速度，由 5 个值可供选择，分别是 turbo（极速,仅 0.0.3 以后版本）、fast（快速）、defaults(默认)、slow（慢速）、dull（极慢,仅 0.0.3 以后版本），默认值为 fast。

弹窗设置参数（PROMPT_TITLE_ID、PROMPT_CONTENT_ID、PROMPT_OKBUTTON_TITLE、PROMPT_CANCELBUTTON_TITLE、PROMPT_CLOSE_BUTTON）

弹窗是指客户端主动弹出的公告框、点击按钮触发的提示框、右下角公告框、或其他一些选择框，它浮在客户端的上面。一般的弹窗都有一些共同点，它们都包含一个标题（title）、一个内容（content）、一个确定按钮（ok）、一个取消按钮（cancel）、一个关闭按钮（close），但并不是所有的弹窗都具备以上功能，比如右下角的弹窗就只有关闭按钮和内容框。弹窗控件是 pytradecn 内置的一种自定义控件，它拥有 title 属性、content()方法、ok()方法、cancel()方法、close()方法。有关弹窗控件的详细内容请参考“弹窗控件”章节。

PROMPT_TITLE_ID 参数用来设置弹窗内 title 的原始控件 autoid。

PROMPT_CONTENT_ID 参数用来设置弹窗内 content 的原始控件 autoid。

一个客户端所有弹窗的 PROMPT_TITLE_ID 或 PROMPT_CONTENT_ID 不一定是唯一的，幸运的是只有在使用弹窗控件的 title 属性或 content()方法时才会使用这两个参数，所以只有按您的代码要求设置这两个值就可以，如果您没有使用弹窗控件的 title 属性或 content()方法，那么完全可以忽略这两个参数。

PROMPT_OKBUTTON_TITLE 参数用来设置弹窗“确定”按钮的文字描述，它是一个正则表达式。到目前为止，作者也不知道到底有多少种弹窗，您应该按照实际情况不断添加它。

PROMPT_CANCELBUTTON_TITLE 参数用来设置弹窗“取消”按钮的文字描述，它是一个正则表达式，您应该按照实际情况不断添加它。

PROMPT_CLOSE_BUTTON 参数是一个元组，里面是“关闭”按钮的 autoid，因

为可能有多个不同的 autoid。

下面是默认的弹窗设置

```
    # 提示框弹出框相关
    PROMPT_TITLE_ID = '1365'
    PROMPT_CONTENT_ID = '1004'
    PROMPT_OKBUTTON_TITLE = '(确定|是|现在测评|我知道了|保存|立即升级).*'
    PROMPT_CANCELBUTTON_TITLE = '(取消|否|稍后测评|我知道了|以后再说).*'
    PROMPT_CLOSE_BUTTON = ('1008', '1003')  # 关闭按钮可能有不同的 id
```

下面是以同花顺 9.2.30 版本的完整客户端配置类

```
# 先引入客户端的基类
from pytradecn import BaseClient

# 类名的命名以大写字母和其版本号组成，区别不同的券商和版本
class THS92030(BaseClient):
    # 账号信息
    user = '123456'  # 资金账号、客户号、上 A 股东、深 A 股东等
    psw = '654321'   # 第一密码
    second = None    # 第二密码:通讯密码、认证口令等，没有时请设置为 None
    account = {}     # 账户中的其他自定义信息

    # 客户端安装位置，大小写敏感，且盘符为大写
    path = r'D:\Users\Administrator\xiadan.exe'

    # 客户端信息
    version = '9.20.30_01'  # 也可以用来记录此代码的版本
    name = '同花顺'
    key = 'ths92030'  # 客户端设别符，重要关键参数，一定保持唯一

    # 窗口规范
    loginwindow = dict(title='', control_type='Pane', control_count=16)
    mainwindow = dict(title='网上股票交易系统 5.0', control_type='Window',
control_count=4)

    # 部件选择
    loginengine = 'THS_9_20_30'  # 登录引擎名
    trademodel = 'THS_9_20_30'  # 交易模型名
```

```
    # 设置交易的速度模式
    TRADE_SPEED_MODE = 'fast'

    # 提示框弹出框相关
    PROMPT_TITLE_ID = '1365'
    PROMPT_CONTENT_ID = '1004'
    PROMPT_OKBUTTON_TITLE = '(确定|是|现在测评|我知道了|保存|立即升级).*'
    PROMPT_CANCELBUTTON_TITLE = '(取消|否|稍后测评|我知道了|以后再说).*'
    PROMPT_CLOSE_BUTTON = ('1008', '1003')  # 关闭按钮可能有不同的 id
```

4、自定义参数

您可以在客户端类中定义任何您想要定义的参数，它们可以在登录引擎、交易模型等任何可能的地方被访问，常见的自定义参数可能有两种：一种是一些设置信息，比如登录引擎中的试错次数；一种是控件的描述，在后续的代码中依靠这种描述找到客户端中真实的控件。

三、编写登录引擎

如果 pytradecn 内置 DEFAULT、VERIFYCODE、VERIFYCODEPLUS 三种登录引擎无法满足要求时，需要编写一个自制的登录引擎类。编写登录引擎只要以 BaseEngine（或 Engine）为基类，并实现 login 方法即可。

1、登录引擎的标准代码格式（以同花顺 9.2.30 为例）

```
# 先导入登录引擎的基类
from pytradecn import BaseEngine

# 正确命名您的类名，以大写字母和其版本号组成，区别不同的券商和版本
class THS92030Engine(BaseEngine):

    # 为这个登录引擎取一个唯一的名字，他将作为客户端参数 loginengine 的值
    name = 'THS_9_20_30'

    # 初始化可有可无，按需要
    def __init__(self, client):
        super(THS92030Engine, self).__init__(client)

    # 实现 login 方法
    def login(self):
```

```
    # 您的代码
```

2、登录引擎的基类 BaseEngine（或 Engine）

基类中存在的一些属性或方法可以辅助您高效实现您的相关功能。

客户端属性（_client）

使用“self._client”可以访问指定客户端类中的参数，例如“self._client.user”可以访问客户端类中定义的用户账号。

注意：

self._client.loginwindow 和 self._client.mainwindow 返回的不再是字典，而是一个 WindowSpecification 对象，请参考“WindowSpecification 类”章节。

弹窗管理器属性（_prompt）

使用“self._prompt”可以用来管理各类弹窗或提示框，如存在性判断、捕捉提示框、关闭弹窗等，有关弹窗管理器的详细用法，请参考“PromptManager 类”章节。

获取控件方法（_get_control）

使用“self._get_control(control_define)”返回控件描述对象（UIAControlSpecification），可以对控件进行存在性判断、点击等各种操作，有关 UIAControlSpecification 类的详细介绍，请参考“UIAControlSpecification 类”章节。

参数 control_define 为一个字典，用来描述客户端实际控件的特征，使用 inspect.exe 可以查看各种控件特征。此外，pytradecn 支持自定义控件，也可以通过 control_define 传入自定义控件的设置值，例如自定义的验证码控件有一个白名单设置 whitelist。

完整的控件特征如下：

title　　：有这个标题的控件，inspect.exe 下的 name 属性

title_re　　：标题与此正则表达式匹配的控件

class_name　　：具有此窗口类的控件

class_name_re ：类与此正则表达式匹配的控件

handle ：具有此句柄的控件

control_id ：具有此 id 的控件

control_type ：具有此类型的控件

auto_id ：具有此自动化 ID 的控件

framework_id ：具有此框架 ID 的控件

depth ：从窗口开始搜索的深度，默认为全部后代

predicate_func ：自定义验证函数

control_key ：指定此控件的包装器，默认使用 control_type 的值

found_index ：控件按条件过滤后，此索引的控件

**config ：对此控件的设置参数，只适用于自定义控件

control_type 和 control_key 是什么关系？control_type 可以通过 inspect.exe 来查看，就是控件的类型。正常情况下，pytradecn 在找到控件后会按照 control_type 指定的类型返回控件的包装器。例如类型为 Edit 的 Windows 标准控件有一个 set_text()方法，可以在编辑框内输入文字，但是中国的 GUI 的软件设计师经常会设计一些非标准的控件，导致无法使用标准控件的方法，或者使用一些技巧实现某种类标准控件，例如使用几张图片组合成一种类 Tab 控件。为了解决这个问题，pytradecn 支持自定义控件，并使用 control_key 参数强制指定成自定义的控件。例如当发现标准的 Edit.set_text()无法输入文字时，可以将 control_key 设置为 Editor，Editor 是 pytradecn 内部自定义的编辑器控件，同样拥有 set_text()方法，Editor.set_text()可以解决标准 Edit 控件无法输入的问题。

上文中的控件特征没有必要都使用，经常使用只有 auto_id、control_type、control_key、found_index、class_name，我认为有这几个参数足以准确定位一个控件。

下面是一个登录验证码控件的参数定义

```
LOGIN_VERIFYCODEIMAGE_ID = {
    'auto_id': '1499',
    'control_type': 'Image',
    'control_key': 'Verifycode',
    'box': (None, None, None, None),
```

```
    'whitelist': '0123456789',
    'refresh': None
}
```

验证码本身就是一幅图像，所以其 control_type 为 Image，标准的 Image 控件没有太多的可用方法，pytradecn 内部自定义了一个 Verifycode 控件，该控件拥有一个 window_text()方法，可以返回图像上面的字符，还有一个 refresh()方法，用来刷新验证码。所以将 control_key 设置为 Verifycode。后面的 box、whitelist、refresh 为 Verifycode 控件可以接受的参数，box 指验证码在图像上的范围，whitelist 为白名单，refresh 指刷新验证码的按钮，默认为图像本身。

control_define 有一种简写方式，可以将 auto_id、control_type、control_key、found_index 四个参数合并成一个字符串：“auto_id|control_type|control_key|found_index”，配置给 control 参数。例如上面的验证码图像参数也可以写成：

```
LOGIN_VERIFYCODEIMAGE_ID = {
    'control': '1499|Image|Verifycode',
    'box': (None, None, None, None),
    'whitelist': '0123456789',
    'refresh': None
}
```

甚至，直接将 control 字符串赋值给验证码控件

```
LOGIN_VERIFYCODEIMAGE_ID = '1499|Image|Verifycode'
```

只是这种方式无法给自定义控件提供额外的设置参数，例如 whitelist。

3、中止登录

在您的代码中抛出任何错误都会中止登录，这也是退出登录的方法。在 pytradecn.error 中有一个 LoginError，正常情况下以抛出该错误作为中止登录的方法。

4、通用的登录引擎

首先，将相对复杂的控件定义写在客户端类中

```
# 在上文讲的客户端类中定义一个验证码图像的控件
```

```
LOGIN_VERIFYCODEIMAGE_ID = {
    'control': '1499|Image|Verifycode',
    'box': (None, None, None, None),
    'whitelist': '0123456789',
    'refresh': None
}
```

编写登录引擎

```
# 先导入登录引擎的基类
from pytradecn import BaseEngine
# 导入错误
from pytradecn.error import LoginError, TimeoutError

# 正确命名您的类名，以大写字母和其版本号组成，区别不同的券商和版本
class YourNameEngine(BaseEngine):

    # 为这个登录引擎取一个唯一的名字，它将作为客户端参数 loginengine 的值
    name = '命名一个唯一的名称'

    # 初始化可有可无，按需要，也可以在初始化中定义控件
    def __init__(self, client):
        super(YourNameEngine, self).__init__(client)

    # 实现 login 方法
    def login(self):
        for i in range(5):  # 试错 5 次
            # 关闭可能存在的提示框或登录失败后的提示框
            self._prompt.close()
            # 输入用户账号
            self._get_control('1001|Edit|Editor').set_text(self._client.user)
            # 输入用户密码
            self._get_control('1012|Edit|Editor').set_text(self._client.psw)
            # 如果存在第二密码
            if self._get_control('1181|Edit|Editor').exists():
                self._get_control('1181|Edit|Editor').set_text(self._client.second)
            # 如果存在验证码图像
            if self._get_control(self._client.LOGIN_VERIFYCODEIMAGE_ID).exists():
                self._get_control('1003|Edit|Editor').set_text(
                    self._get_control(self._client.LOGIN_VERIFYCODEIMAGE_ID).window_text()
```

```
            )
        # 点击登录按钮
        self._get_control('1006').click()
        try:
            # 等待登录窗口消失，7 秒内
            self._client.loginwindow.wait_not('exists', timeout=7)
            break  # 登录成功退出循环
        except TimeoutError:
            # 登录窗口超时未消失
            continue  # 登录不成功，重新登录
    else:
        # 达到最大试错次数，登录不成功
        self._prompt.close()  # 关闭可能存在的登录错误提示框
        # 抛出错误，中止登录
        raise LoginError('登录不成功！')
```

四、编写交易模型

交易模型用来定义并实现主窗口中需要操作的功能，在您自定义的模型中应至少定义以下 2 个方法：initialization（初始化主窗口）、reset（复位主窗口），因为这两个方法在模型的基类中被声明为接口，所以必须有定义，如您不需要可以写一个空方法。

编写交易模型只需要以 BaseModel（或 Model）为基类，在定义 initialization、reset 的基础上，增加您想添加的任何功能。

1、交易模型的标准代码格式

```
# 首先导入模型基类
from pytradecn import BaseModel
# 或许您还需要抛出一些错误
from pytradecn.error import StockCodeError, StockPriceError, StockCountError, TradeFailFError

# 正确命名您的类名，以大写字母和其版本号组成，区别不同的券商和版本
class YourNameModel(BaseModel):

    # 为这个模型取一个唯一的名字，他将作为客户端参数 trademodel 的值
    name = '命名一个唯一的名称'

    # 初始化可有可无，依需要
```

```
def __init__(self, client):
    super(YourNameModel, self).__init__(client)

# 完成对 initialization、reset 的定义
def initialization(self):
    # 在您的客户端处于“已登陆”状态时，initialization 会被访问
    pass

def reset(self):
    # 不管您的客户端是处于“已登录”或“未登录”，reset 都会被访问
    pass

# 在这里添加您想要的其他功能
def myfunction(self, *args, **kwargs):
    pass
```

initialization 方法和 reset 方法不需要返回值，您的自定义方法一定要带有返回值。不要在您的主代码中直接访问 initialization 方法和 reset 方法，他们会被 pytradecn 内部隐式访问。

2、交易模型的基类 BaseModel(或 Model)

基类中存在的一些属性或方法可以辅助您高效实现您的相关功能。

客户端属性（_client）

使用“self._client”可以访问指定客户端类中的参数，例如“self._client.user”可以访问客户端类中定义用户账号。

注意：

self._client.loginwindow 和 self._client.mainwindow 返回的不再是字典，而是一个 WindowSpecification 对象，请参考“WindowSpecification 类”章节。

弹窗管理器属性（_prompt）

使用“self._prompt”可以用来管理各类弹窗或提示框，如存在性判断、捕捉提示框、关闭弹窗等，有关弹窗管理器的详细用法，请参考“PromptManager 类”章节。

获取控件方法（_get_control）

使用“self._get_control(control_define)”返回控件描述对象（UIAControlSpecification），可以对控件进行存在性判断、点击等各种操作，有关 UIAControlSpecification 类的详细介绍，请参考“UIAControlSpecification 类”章节。

参数 control_define 同上。

3、中止交易

在您的代码中抛出任何错误都会中止交易，这也是退出交易的方法。在 pytradecn.error 中有 StockCodeError, StockPriceError, StockCountError, TradeFailFError，正常情况下以抛出这些错误作为中止交易的方法。

五、类参考

1、弹窗管理器类 PromptManager

class pytradecn.prompt.PromptManager(client)

基类：object

弹窗或提示窗管理器，可以对弹窗进行存在性判断、关闭或捕捉提示框。管理弹窗时，不应该将此类实例化，而是使用登录引擎基类或模型基类中提供的 _prompt 属性。即使您将此类实例化，您实例化的对象也与_prompt 属性指向同一个对象（单例模式）。

close(**kwargs)

关闭当前可见的弹窗

参数 kwargs 是一个用来过滤弹窗的关键字列表：

title ：有这个标题的弹窗，支持正则表达式，此值依赖客户端中的 PROMPT_TITLE_ID 设置

content ：有这个内容的弹窗，支持正则表达式，此值依赖客户端中的 PROMPT_CONTENT_ID 设置

text ：有这个文本的弹窗，支持正则表达式，这是一个万能参数，所有 inspect.exe 中的可见文字字符串。

func ：自定义过滤函数

返回值：无

举例

```
self._prompt.close()  # 关闭当下可见的所有弹窗
self._prompt.close(text='请输入您的交易密码')  # 关闭当下弹框中有'请输入您的交易密码'这个文本的弹窗
```

tooltip(timeout=None, retry_interval=None, **kwargs)

捕捉提示框

提示框往往在点击一个按钮后弹出，按钮点击后 GUI 程序会有一些运算，弹窗弹出时甚至会有肉眼不可见的动画，这个过程需要一定的时间，所以我们才需要“捕捉”。timeout 参数设定捕捉的时长，默认情况下取决于客户端类中的 TRADE_SPEED_MODE 参数设定，当 TRADE_SPEED_MODE 设定为 defaults 时，捕捉时长为 0.6 秒。将 timeout 设置为 0，即只捕捉当前已存在的弹窗。

参数：

timeout ： 设定捕捉时长，默认与 TRADE_SPEED_MODE 设定有关

retry_interval ： 设定查询间隔，默认与 TRADE_SPEED_MODE 设定有关

kwargs ： 过滤弹窗的关键字列表，同上

返回值：

当成功捕捉到弹窗时，返回弹窗控件，有关弹窗控件的详细介绍，请参考“弹窗控件”章节；当超时 timeout 无法捕捉到弹窗时，返回 None。

举例：

```
# 捕捉带有“委托确认”文本的弹窗
pane = self._prompt.tooltip(text='委托确认')
if pane is not None:
    pane.ok()
```

exists(timeout=None, retry_interval=None, **kwargs)

判断弹窗是否存在

注意 timeout 参数默认为 0，即只判断当前是否存在，如果您想让 exists

具备捕捉功能，应将 timeout 参数设置为一个不为 0 的固定值。

参数：

timeout　　：　设定捕捉时长，默认为 0

retry_interval ：　设定查询间隔，默认与 TRADE_SPEED_MODE 设定有关

kwargs　　：　过滤弹窗的关键字列表，同上

返回值：当存在弹窗时，返回 True，否则返回 False。

举例：

```
if self._prompt.exists(text='请输入您的交易密码'):
    # 客户端被锁屏
    pane = self._prompt.tooltip(text='请输入您的交易密码')
    pane.child('1039|Edit|Editor').set_text(self._client.psw)
    pane.ok()
```

下面的代码效果是一样的

```
pane = self._prompt.tooltip(timeout=0, text='请输入您的交易密码')
if pane is not None:
    # 客户端被锁屏
    pane.child('1039|Edit|Editor').set_text(self._client.psw)
    pane.ok()
```

为什么将 timeout 设置为 0，timeout=0 意即捕捉“当前”，与默认下的 exists 是一样的。如果 timeout 不为 0 会发生什么？，不为 0 的话，假设根本就不存在这个弹窗，那么 tooltip 会不停的检测，直到 timeout 超时，这会让您的代码运行时感觉会“卡”一下。

```
start_monitor(delay=0, **kwargs)
```

开启一个新线程，监视 UGI 程序是否弹出弹窗，如果弹出弹窗就关闭。请谨慎使用此方法，有可能会关闭掉正常的提示框。

参数：

delay　：　监视时长，默认为 0，即关闭当前的所有弹窗

kwargs ：　过滤弹窗的关键字列表，同上

返回值：无

举例：

```
# 监视 5 秒内出现的所有弹窗并关闭
self._prompt.start_monitor(delay=5)
```

stop_monitor()

关闭上面的监视器

参数：无

返回值：无

2、窗口描述类 WindowSpecification

WindowSpecification 类是 pywinauto 库的原生类，在自定义的登录引擎或交易模型中使用 self._client.loginwindow 和 self._client.mainwindow 后会返回 WindowSpecification 对象。有关 WindowSpecification 类的详细介绍，请查看 pywinauto 官方网址。

https://www.kancloud.cn/gnefnuy/pywinauto_doc/1193049

该类常用的方法有：

exists(timeout=None, retry_interval=None)

检查窗口是否存在，如果窗口存在则返回 True，否则返回 False

参数:

timeout ： 捕捉窗口存在的时长，默认值取决于 TRADE_SPEED_MODE 设置

retry_interval ： 检查窗口存在的事件间隔

举例：

```
# 判断登录窗口是否存在
if self._client.loginwindow.exists(timeout=0)
    # 未登录
else:
    # 已登录
```

wait(wait_for, timeout=None, retry_interval=None)

等待窗口处于特定状态。

参数:

wait_for ：等待窗口进入的状态。可以是以下任何状态，也可以按空格间隔的组合状态。 exists 表示窗口是有效的句柄， visible 表示窗口未隐藏， enabled 表示窗口未被禁用， ready 表示窗口可见并已启用，active 表示窗口处于活动状态。

timeout ：如果窗口在此秒数后未处于适当状态，则引发 pytradecn.error.TimeoutError 错误。 默认值取决于 TRADE_SPEED_MODE 设置。

retry_interval ： 重试间隔，默认值取决于 TRADE_SPEED_MODE 设置。

返回：等待成功后返回窗口的包装器

举例：等待主窗口存在并准备就绪（即等待登录动作完成）

```
# 等待主窗口在 15 秒内存在并准备就绪
self._client.mainwindow.wait('exists ready', timeout=15)
```

wait_not(wait_for_not, timeout=None, retry_interval=None)

等待窗口不处于特定状态

参数:

wait_for_not ：等待窗口不在的状态。可以是以下任何状态，也可以按空格间隔的组合状态。 exists 表示窗口是有效的句柄， visible 表示窗口未隐藏， enabled 表示窗口未被禁用， ready 表示窗口可见并已启用，active 表示窗口处于活动状态。

timeout ：如果窗口在此秒数后还处于状态，则引发 pytradecn.error.TimeoutError 错误。 默认值取决于 TRADE_SPEED_MODE 设置。

retry_interval ： 重试间隔，默认值取决于 TRADE_SPEED_MODE 设置。

返回：无

举例：

```
# 等待登录窗口在 7 秒内消失，否则抛出错误
try:
    self._client.loginwindow.wait_not('exists', timeout=7)
    break  # 登录成功退出循环
except TimeoutError:
    continue  # 登录不成功
```

wrapper_object()

返回窗口的控件包装器

控件包装器可以对窗口做各种操作，详细说明请查看“控件参考”。

例如：

```
# 模拟鼠标点击
self._client.loginwindow.wrapper_object().click_input()
```

wrapper_object()可以省略

```
# 省略 wrapper_object()
self._client.loginwindow.click_input()
```

3、控件描述类 UIAControlSpecification

class pytradecn.uiacontrol.UIAControlSpecification(criteria)

基类：object

控件描述类与 WindowSpecification 类非常类似，只不过 UIAControlSpecification 类更侧重于窗口内的具体控件。您不应该实例化该类，而是通过登录引擎基类或交易模型基类的_get_control(control_define)方法返回控件描述对象。该类的主要方法有：

exists(timeout=None, retry_interval=None)

检查控件是否存在，如果控件存在则返回 True，否则返回 False

参数:

timeout ： 捕捉控件的时长，默认值为 0，即仅检测当下。注意此值与 WindowSpecification 类的 exists 方法不同

retry_interval ： 检查窗口存在的事件间隔

举例：

```
# 判断第二密码框是否存在
if self._get_control('1181|Edit|Editor').exists():  # 第二密码
    self._get_control('1181|Edit|Editor').set_text(self._client.second)
```

wait(wait_for, timeout=None, retry_interval=None)

等待控件处于特定状态。

参数:

wait_for ：等待控件进入的状态。可以是以下任何状态，也可以按空格间隔的组合状态。 exists 表示窗口是有效的句柄， visible 表示窗口未隐藏， enabled 表示窗口未被禁用， ready 表示窗口可见并已启用，active 表示窗口处于活动状态。

timeout ： 如果控件在此秒数后未处于适当状态，则引发 pytradecn.error.TimeoutError 错误。 默认值取决于 TRADE_SPEED_MODE 设置。

retry_interval ： 重试间隔，默认值取决于 TRADE_SPEED_MODE 设置。

返回：等待成功后返回控件的包装器

举例：

```
# 等待撤单按钮变为可用后点击它（wait 方法会返回控件的包装器）
self._get_control('1099').wait('enabled').click()
```

wait_not(wait_for_not, timeout=None, retry_interval=None)

等待控件不处于特定状态

参数:

wait_for_not ：等待控件不在的状态。可以是以下任何状态，也可以按空格间隔的组合状态。 exists 表示窗口是有效的句柄，visible 表示窗口未隐藏， enabled 表示窗口未被禁用，ready 表示窗口可见并已启用，active 表示窗口处于活动状

态。

timeout ： 如果窗口在此秒数后还处于状态，则引发 pytradecn.error.TimeoutError 错误。 默认值取决于 TRADE_SPEED_MODE 设置。

retry_interval ： 重试间隔，默认值取决于 TRADE_SPEED_MODE 设置。

返回：无

wrapper_object()

返回控件的包装器

控件包装器可以对控件做各种操作，详细说明请查看“控件参考”。

例如：

```
# 输入资金账号
self._get_control('1001|Edit|Editor').wrapper_object().set_text(self._client.user)
```

wrapper_object()可以省略

```
# 省略 wrapper_object()
self._get_control('1001|Edit|Editor').set_text(self._client.user)
```

六、控件参考

1、控件基类 BaseWrapper

class pywinauto.base_wrapper.BaseWrapper(element_info)

基类：onject

控件基类是 pywinauto 库的原生类，所有的控件包装器均继承自控件基类，包括 pytradecn 的自定义控件，控件基类中定义了很多的方法，可用方便的操作控件，包括模拟鼠标点击、键盘输入等。控件基类的详情请参考：

https://www.kancloud.cn/gnefnuy/pywinauto_doc/1193057

2、UIA 控件基类 UIAWrapper

class pywinauto.controls.uiawrapper.UIAWrapper(element_info)

基类：BaseWrapper

UIAWrapper 类继承自 BaseWrapper 类，是 pywinauto 库的原生类提供了一些 UIA 控件特有的方法。UIA 控件的详情请参考：

https://www.kancloud.cn/gnefnuy/pywinauto_doc/1193062

3、标准控件库

pywinauto 提供了很多标准的控件包装器，他们都继承自 UIAWrapper，比如 EditWrapper 控件包装器，它是 Edit 控件的标准包装器。更多的标准控件请参考：

https://www.kancloud.cn/gnefnuy/pywinauto_doc/1193063

4、自定义控件基类 BaseUIAWrapper

class pytradecn.control.baseuiawrapper.BaseUIAWrapper(element_info)

基类：UIAWrapper

由于官方标准的控件库无法满足我们的要求，所以 pytradecn 拥有自定义控件的能力，BaseUIAWrapper 提供了一些自定义控件的公共方法。

```
config(key)
```

返回自定义控件的设置值，例如验证码控件有一个白名单设置 whitelist

```
standard()
```

返回自定义控件的 pywinauto 标准控件

```
own()
```

返回自定义控件的另一个副本

```
child(control_define)
```

返回自定义控件的后代控件描述对象

参数 control_define 同上

```
image_text(box=None, whitelist=None)
```

返回自定义控件上面的可见文本字符

```
exists(timeout=None)
```

判断自定义控件是否还存在

5、弹窗控件 PromptWrapper

基类：BaseUIAWrapper

类型：Prompt

弹窗控件 PromptWrapper 是 pytradecn 的自定义控件

```
title
```

属性，返回弹窗的标题，此属性依赖客户端中的 PROMPT_TITLE_ID 设置

```
content()
```

返回弹窗的内容文本，此方法依赖客户端中的 PROMPT_CONTENT_ID 设置

```
ok()
```

点击“确定”按钮

```
cancel()
```

点击“取消”按钮

```
close()
```

点击“关闭”按钮

6、表格控件 GridWrapper

基类：BaseUIAWrapper

类型：GridCSV

表格控件 GridWrapper 是 pytradecn 的自定义控件，自定义的表格控件本质上是一个自读列表，您可用像列表一样索引、遍历等操作。此列表一些自己独特的方法：

refresh()

刷新并返回表格控件，一定要先使用此方法才能返回真实的表格数据

items(**kwargs)

依关键字过滤表格后返回一个新表格

参数 kwargs 是客户端实际表格中的表头字段，如：

委托时间	证券代码	证券名称	操作	备注	委托数量	成交数量	委托价格	成交均价	撤消数量	合同编号	交易市场

item(**kwargs)

依关键字过滤表格，并返回唯一的行，返回类型为一个只读字典

如果有多个行或没有行，则会抛出错误

参数 kwargs 同上

GridCSV 控件接收 5 个参数：

headHeight ：表头的行高

lineHeight ：普通的行高

offset ：表格左侧功能的偏移量

saveto ：另存为对话框

savetofile ：另存为对话框中保持地址的编辑框

下面是一个典型的 GridCSV 控件设置

```
GRID_DEFAULT_ID = {
    'control': '1047|Pane|GridCSV|1',
    'headHeight': 24,
    'lineHeight': 23,
    'offset': 6,
    'saveto': '|Window|Prompt',
    'savetofile': '1152|Edit'
}
```

表格中的“行”返回一个 GridItem 对象，GridItem 对象是一个只读字典，您可以像字典一样索引、遍历它，但 GridItem 也有自己的方法：

```
click()
```

点击行

```
double_click()
```

双击行

```
select()
```

选择行左侧的功能

7、编辑器控件 EditorWrapper

基类：BaseUIAWrapper

类型：Editor

pytradecn 自定义的 Editor 控件用来解决标准 Edit 控件无法输入的问题

```
set_text(text)
```

输入文本

text ： 要输入的文本

8、验证码控件 VerifycodeWrapper

基类：BaseUIAWrapper

类型：Verifycode

验证码控件 VerifycodeWrapper 是 pytradecn 的自定义控件

```
window_text()
```

返回验证码表面的文本字符

refresh()

刷新验证码并返回一个新的验证码控件

验证码控件可以接收 3 个设置参数：

box ：(左，上，右，下)四元组，表示验证码字符在图像上的位置，默认为整个图像

whitelist ：验证码的白名单，默认为 None

refresh ：刷新验证码的按钮，默认为 None，表示图像本身

下面是一个典型的验证码控件设置

```
LOGIN_VERIFYCODEIMAGE_ID = {
    'control': '1499|Image|Verifycode',
    'box': (None, None, None, None),
    'whitelist': '0123456789',
    'refresh': None
}
```

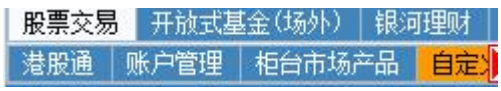
9、标签控件 TabPaneWrapper

基类：BaseUIAWrapper

类型：Tabpane

标签控件 TabPaneWrapper 是 pytradecn 的自定义控件。有一些类似 Tab 控件的结构，我们可以把它强制指定为 Tabpane，以实现 tab 控件的功能，比如下面的结构：

或者



select(index)

依索引选择对应的标签并返回其 link 的控件

Tabpane 控件有一个设置参数 tabs，它是一个列表，列表中的项是表示每个标签属性的字典，下面是一个典型的 Tabpane 控件：

```
PRODUCT_MENU_ID = {
    'control': '1001|Pane|Tabpane',
    'class_name': 'CCustomTabCtrl',
    'tabs': [
        {'name': '股票交易', 'rect': (1, 2, 64, 19), 'link': '129|Tree'},
        {'name': '开放式基金(场外)', 'rect': (65, 2, 176, 19), 'link':
'240|Tree'},
        {'name': '银河理财', 'rect': (177, 2, 240, 19), 'link': '909|Tree'},
        {'name': '证券出借', 'rect': (241, 2, 304, 19), 'link': '2037|Tree'},
        {'name': '港股通', 'rect': (1, 22, 52, 39), 'link': '5199|Tree'},
        {'name': '账户管理', 'rect': (53, 22, 116, 39), 'link': '830|Tree'},
        {'name': '柜台市场产品', 'rect': (117, 22, 204, 39), 'link':
'5040|Tree'}
    ]
}
```