每次不重样，带你收获最新测试技术！

目录

CONTENTS

微信扫一扫关注我们

投稿邮箱：
editor@51testing.com

DDT 模块中@ddt 和@data 装饰器源码分析

◆ 作者：Tynam

DDT 库运行原理猜想

DDT（Data-Driven Tests）库是一个 Python 模块，它通过装饰器扩展了 unittest 测试框架，使得可以在单个测试方法上应用多组参数进行测试。DDT 提供了@ddt 类装饰器和@data、@unpack、@file_data 方法装饰器，用于实现数据驱动测试。

下面是一个 DDT 的使用示例，在测试类上添加 @ddt 装饰器，测试用例方法上添加 @data 装饰器并传入需要的数据：

```
# ddt_demo1.py
import unittest
from ddt import ddt, data
@ddt
class TestDDTDemo(unittest.TestCase):
    @data("江雨桐", "叶知秋", "柳如烟", "顾清歌")
    def test_ddt_data(self, value):
        print(value)
if __name__ == '__main__':
    unittest.main(verbosity=2)
```

我们先来运行一下 ddt_demo1.py 文件下测试用例，查看执行结果：

```
PS E:\LearnProject> python  .\ddt_demo1.py
test_ddt_data_1_江雨桐 (__main__.TestDDTDemo) ... 江雨桐
ok
```

```
test_ddt_data_2_叶知秋 (__main__.TestDDTDemo) ... 叶知秋
ok
test_ddt_data_3_柳如烟 (__main__.TestDDTDemo) ... 柳如烟
ok
test_ddt_data_4_顾清歌 (__main__.TestDDTDemo) ... 顾清歌
ok

----------------------------------------------------------------------
Ran 4 tests in 0.001s
OK
```

从结果中可以看到，一共执行了 4 条测试用例，依次是：test_ddt_data_1_江雨桐、test_ddt_data_2_叶知秋、test_ddt_data_3_柳如烟、test_ddt_data_4_顾清歌。刚好对应@data 装饰器传入的参数，测试用例的名称为：{测试用例方法名}_{参数序号}_{参数}。从中不难猜出 DDT 库的可能实现原理：首先获取传入的数据，并根据一定的规则将数据转换为一个个参数值和一条条测试用例方法，每一个参数值对应一条测试用例方法，其中测试用例名为“{测试用例方法名}_{参数值在数据中的序号}_{参数值}”；然后将参数值传入测试用例方法并执行。

如果不使用 DDT 库，则 ddt_demo1.py 的测试用例脚本实质等同于如下的测试用例脚本，或者说下面的脚本是上面测试用例执行过程中的一个中间状态：

```
# ddt_demo2.py
import unittest
class TestDDTDemo(unittest.TestCase):
    def __test_ddt_data(self, value):
        print(value)
    def test_ddt_data_1_江雨桐(self):
        self.__test_ddt_data("江雨桐")
    def test_ddt_data_2_叶知秋(self):
        self.__test_ddt_data("叶知秋")
    def test_ddt_data_3_柳如烟(self):
        self.__test_ddt_data("柳如烟")
    def test_ddt_data_4_顾清歌(self):
```

```
            self.__test_ddt_data("顾清歌")
if __name__ == '__main__':
    unittest.main(verbosity=2)
```

执行 ddt_demo2.py 脚本文件，结果如下：

```
PS E:\LearnProject> python ?.\ddt_demo2.py
test_ddt_data_1_江雨桐 (__main__.TestDDTDemo) ... 江雨桐
ok
test_ddt_data_2_叶知秋 (__main__.TestDDTDemo) ... 叶知秋
ok
test_ddt_data_3_柳如烟 (__main__.TestDDTDemo) ... 柳如烟
ok
test_ddt_data_4_顾清歌 (__main__.TestDDTDemo) ... 顾清歌
ok
----------------------------------------------------------------------
Ran 4 tests in 0.001s
OK
```

单从结果上来观察，ddt_demo1.py 和 ddt_demo2.py 两个脚本是等量的。ddt_demo1.py 只不过是使用了 DDT 库，写法更简洁一些，可以说是 ddt_demo2.py 脚本的一个简洁版、升级版的写法。

DDT 库源码分析

从上面示例脚本中可以看到，我们只使用了两个装饰器 @ddt 和 @data 就成功实现了参数化。下面我们对这两个装饰器进行源码刨析。

@data 源码分析

当我们运行测试用例首先会执行测试用例上的装饰器，即 @data。因此，我们先来分析 data 函数。

data 函数：

```
def data(*values):
    return idata(values)
```

data 函数接收一个参数，并返回 idata 函数。我们接着再来查看 idata 函数：

```
DATA_ATTR = '%values'
INDEX_LEN = '%index_len'
def idata(iterable, index_len=None):
    if index_len is None:
        # Avoid consuming a one-time-use generator.
        iterable = tuple(iterable)
        index_len = len(str(len(iterable)))
    def wrapper(func):
        setattr(func, DATA_ATTR, iterable)
        setattr(func, INDEX_LEN, index_len)
        return func
    return wrapper
```

idata 中用到了一个内置函数 setattr，它是将一个属性赋值给对象。这个函数接受三个参数：对象、属性名（字符串形式）和属性值。例如有一个 person = Person("Alice") 实例，使用 setattr(person, 'age', 30) 操作就是给 person 动态添加一个 age=30 的属性，后续可使用 person.age 调用获取其值。

从代码中可以看到 idata 是一个装饰器，最终返回的是内置函数 wrapper，而 wrapper 只干了两件事，就是给被装饰的函数 func 添加了 DATA_ATTR（变量的值为'%values'）和 INDEX_LEN（变量的值为'%index_len'）两个属性，其中 DATA_ATTR 的值为装饰器传入的参数； INDEX_LEN 为传入参数数量的位数数，例如参数数量是 3，则位数就是 1，参数数量是 30，则位数就是 2，参数数量是 300，则位数就是 3，这样做的目的是为了在设置测试用例名称”{测试用例方法名}_{参数序号}_{参数}“时，参数序号的位数保持一致，例如有 300 个参数，第一条用例名称就是{测试用例方法名}_001_{参数}，最后一条用例名称就是{测试用例方法名}_300_{参数}，参数序号始终为 3 位数。

再来看 ddt_demo1.py 脚本示例中的 @data("江雨桐", "叶知秋", "柳如烟", "顾清歌"), 此处就是给 test_ddt_data 测试用例方法添加了 %values=('江雨桐', '叶知秋', '柳如烟', '顾清歌') 和 %index_len=1 的两个属性。

@ddt 源码分析

了解了 @data 的作用后，我们再来看 @ddt 的作用，它都做了些那些事情。先看一下源码：

```
def ddt(arg=None, **kwargs):
    fmt_test_name = kwargs.get("testNameFormat", TestNameFormat.DEFAULT)
    def wrapper(cls):
        for name, func in list(cls.__dict__.items()):
            if hasattr(func, DATA_ATTR):
                index_len = getattr(func, INDEX_LEN)
                for i, v in enumerate(getattr(func, DATA_ATTR)):
                    test_name = mk_test_name(
                        name,
                        getattr(v, "__name__", v),
                        i,
                        index_len,
                        fmt_test_name
                    )
                    test_data_docstring = _get_test_data_docstring(func, v)
                    if hasattr(func, UNPACK_ATTR):
                        pass
                    else:
                        add_test(cls, test_name, test_data_docstring, func, v)
                delattr(cls, name)
            elif hasattr(func, FILE_ATTR):
                pass
        return cls
    # ``arg`` is the unittest's test class when decorating with ``@ddt`` while
```

```
        # it is ``None`` when decorating a test class with ``@ddt(k=v)``.
        return wrapper(arg) if inspect.isclass(arg) else wrapper
```

注：代码中 if hasattr(func, UNPACK_ATTR) 和 elif hasattr(func, FILE_ATTR)分别为 @unpack 和 @file_data 两个装饰器分支上的内容，不在本次分析范围内，故为了方便大家观看，分支代码在此使用 pass 做了处理。

先分析一下 ddt 函数结构，首先是设置变量 fmt_test_name，然后是一个内置函数 wrapper，最后将内置函数 wrapper 返回。

1.fmt_test_name = kwargs.get("testNameFormat", TestNameFormat.DEFAULT) 是用来设置测试用例名称格式的。测试用例名称格式有 TestNameFormat.DEFAULT 和 TestNameFormat.INDEX_ONLY 两种，其中 DEFAULT 是默认值，设置后的用例名称格式为{测试用例方法名}_{参数序号}_{参数}；INDEX_ONLY 是只显示参数序号，设置后的用例名称格式为{测试用例方法名}_{参数序号}。用法如下：

```
@ddt(testNameFormat=TestNameFormat.DEFAULT)
class TestDDTDemo(unittest.TestCase):
    pass


@ddt(testNameFormat=TestNameFormat.INDEX_ONLY)
class TestDDTDemo(unittest.TestCase):
    Pass
```

2.获取到用例名称格式后，程序会先执行返回语句 wrapper(arg) if inspect.isclass(arg) else wrapper，当使用@ddt 装饰器装饰一个测试类时，arg 是被装饰的 unittest 测试类。在这种情况下，arg 不是 None；当使用 @ddt(k=v) 装饰一个测试类时，arg 是 None。这种用法允许用户通过关键字参数来传递配置给 ddt 装饰器。示例如下：

```
@ddt
class TestDDTDemo1(unittest.TestCase):
    pass


@ddt(testNameFormat=TestNameFormat.INDEX_ONLY)
```

```
class TestDDTDemo2(unittest.TestCase):
    Pass
```

@ddt 写法，代码执行到 “wrapper(arg) if inspect.isclass(arg) else wrapper” 时，其中的 arg 就为被装饰的类 TestDDTDemo1，最终将被装饰的类当作参数带入内置函数 wrapper(arg)；@ddt(testNameFormat=TestNameFormat.INDEX_ONLY) 写法，代码执行到 “wrapper(arg) if inspect.isclass(arg) else wrapper” 时，其中的 arg 就为 None，最终返回的是内置函数 wrapper。

3.然后进入内置函数 wrapper 执行其代码，经过一番操作后最终将被装饰的类返回。首先使用了 for name, func in list(cls.__dict__.items()) 遍历被装饰类的所有属性和方法（此处主要目的是获取测试类下的所有测试方法），接着进入了一个条件判断 if hasattr(func, DATA_ATTR): pass elif hasattr(func, FILE_ATTR): pass 如果测试用例方法含有 DATA_ATTR 属性，则走 if 分支；如果测试用例方法含有 FILE_ATTR 属性，则走 elif 分支。回顾 @data 源码分析 ，会动态给测试方法添加一个 DATA_ATTR 属性，而 FILE_ATTR 属性也正是使用 @file_data 装饰器时动态添加的一个属性。因此，此处判断是逻辑是根据使用的装饰器来走不同的处理逻辑。由于我们使用的是 @data 装饰器，因此进入 if 判断逻辑。

4.if 判断逻辑下的第一行代码是 index_len = getattr(func, INDEX_LEN) ，获取传入参数数量的位数数。

5.接着就是 for i, v in enumerate(getattr(func, DATA_ATTR)) 语句，遍历所有的参数，变量 i 为当前参数索引，变量 v 为当前参数。for 循环语句下一共走了三个步骤：test_name = mk_test_name()、 test_data_docstring = _get_test_data_docstring() 以及一个判断语句 if hasattr(func, UNPACK_ATTR)。if 判断语句判断的是测试用例方法是否有属性 UNPACK_ATTR，即判断是否使用了 @unpack 装饰器，本文示例不曾使用 @unpack 装饰器，也不在本次讲解范围内，故代码运行的是 else 分支，执行了一个 add_test() 函数。故 for 循环下实际上依次调用了 mk_test_name()、_get_test_data_docstring() 和 add_test() 三个函数。

6.mk_test_name()

mk_test_name() 是用来设置测试用例名称的一个函数，源码如下：

```
def mk_test_name(name, value, index=0, index_len=5, name_fmt=TestNameFormat.DEFAULT):
    # Add zeros before index to keep order
    index = "{0:0{1}}".format(index + 1, index_len)
    if name_fmt is TestNameFormat.INDEX_ONLY or not is_trivial(value):
        return "{0}_{1}".format(name, index)
    try:
        value = str(value)
    except UnicodeEncodeError:
        # fallback for python2
        value = value.encode('ascii', 'backslashreplace')
    test_name = "{0}_{1}_{2}".format(name, index, value)
    return re.sub(r'\W|^(?=\d)', '_', test_name)
```

mk_test_name 函数很简单，就是根据给出的 name_fmt 值返回一个测试用例用例名称。如果 name_fmt=TestNameFormat.INDEX_ONLY，则返回 "{0}_{1}".format(name, index)；反之返回 "{0}_{1}_{2}".format(name, index, value)。其中 name 为被装饰的测试用例方法/函数名称，index 为”索引+1“值，value 就是对应的参数，之间使用下划线连接。故最终的测试用例名称为 {测试用例方法名}_{参数序号}_{参数} 或 {测试用例方法名}_{参数序号}。

7._get_test_data_docstring()

_get_test_data_docstring() 是用来获取测试方法/函数的描述文档，源码如下：

```
def _get_test_data_docstring(func, value):
    if not _is_primitive(value) and value.__doc__:
        return value.__doc__
    else:
        return None
```

如果 value 不是基本数据类型并且属性 __doc__ 有值，则使用本身的 __doc__。反之则返回 None，即使用测试名称。

8.add_test()

add_test() 是用来动态添加测试用例的一个函数，源码如下：

```
def add_test(cls, test_name, test_docstring, func, *args, **kwargs):
    setattr(cls, test_name, feed_data(func, test_name, test_docstring, *args, **kwargs))
```

setattr 在此处就是给测试类 cls 动态添加一条方法名为 test_name ，方法为 feed_data(func, test_name, test_docstring, *args, **kwargs) 的测试用例方法。实际上就是基于现有的测试用例方法/函数功能，给它一个新的测试用例名称。

feed_data 函数的作用是将测试数据项提供给测试。

分析到此处，相信大家都已经清楚了，将多组参数动态转换成多条测试用例方法的过程。

9.for 语句下 add_test() 函数执行完成后，内置函数 wrapper 还有最后一行代码 delattr(cls, name)。

当我们运行示例 ddt_demo1.py 脚本中的代码，for 语句下 add_test() 函数执行完成后，脚本实质上已经转换成了如下代码：

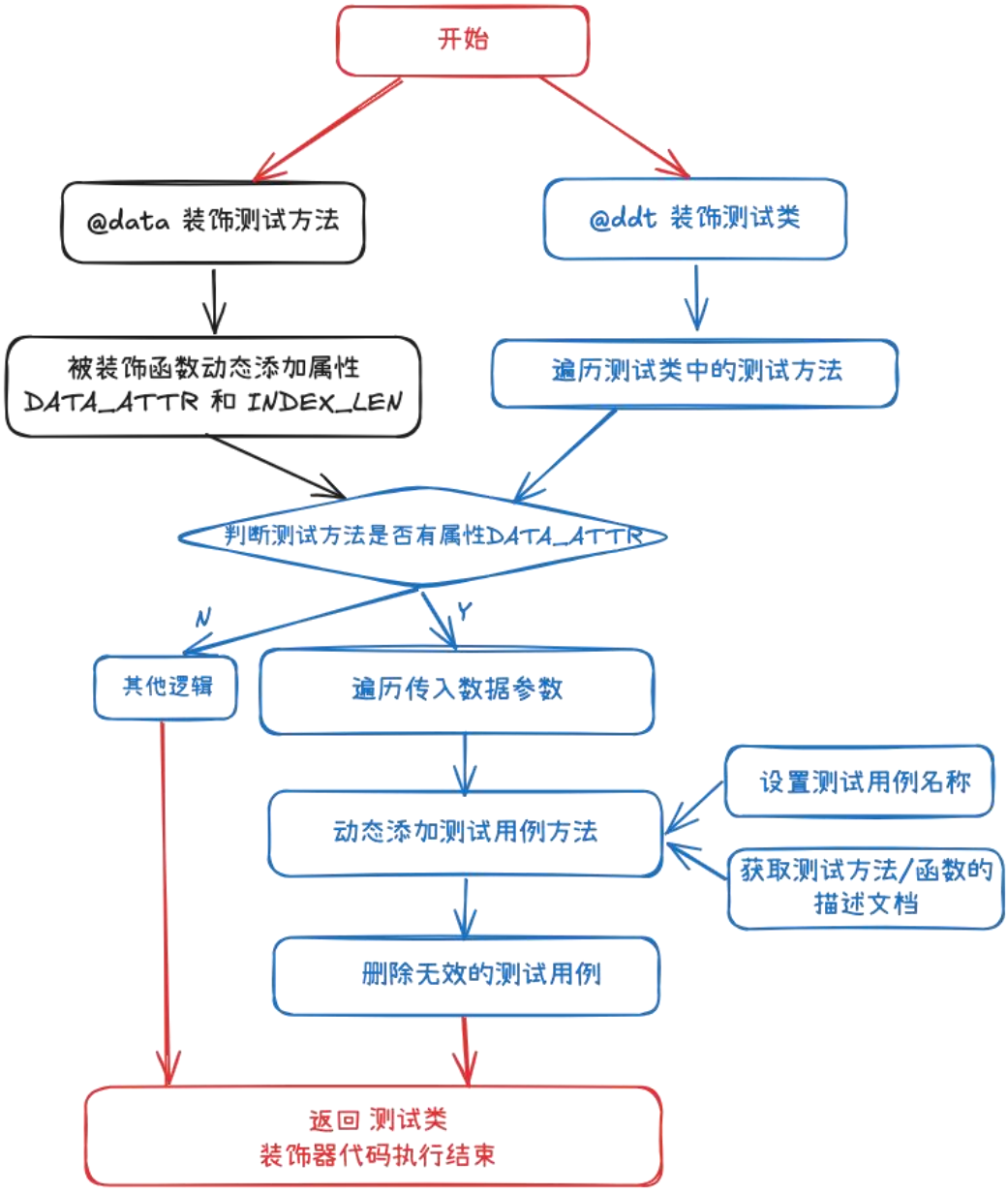
```
class TestDDTDemo(unittest.TestCase):
    def test_ddt_data(self, value):
        print(value)
    def test_ddt_data_1_江雨桐(self):
        self.test_ddt_data("江雨桐")
    def test_ddt_data_2_叶知秋(self):
        self.test_ddt_data("叶知秋")
    def test_ddt_data_3_柳如烟(self):
        self.test_ddt_data("柳如烟")
    def test_ddt_data_4_顾清歌(self):
        self.test_ddt_data("顾清歌")
```

从脚本中可以看到，除过将四组参数转换为测试用例方法外，还有一个 test_ddt_data() 方法，根据 unittest 单元测试框架运行规则，会运行测试类下以 test 开头的所有方法，

故 test_ddt_data() 也会被当作一条测试用例来运行。但是 test_ddt_data() 缺少必要的参数 value，故运行它肯定会报错，且将此方法当作一条测试用例来运行也不符合预期，因此最后需要使用 delattr(cls, name)代码将其从测试类中移除。最终只剩下四条参数化的测试用例。

总结

了解了 @data 和 @ddt 内部运行逻辑，我们就能得到 ddt 参数化的一个过程图，如下图所示：



当测试方法被添加 @data 装饰器后，该方法会被动态添加 DATA_ATTR 和 INDEX_LEN 两个属性。

当测试类被添加 @ddt 装饰器后，首先会遍历测试类的所有测试方法和属性，当测试方法中含有 DATA_ATTR 属性时，便会遍历传入的数据参数，然后根据测试方法、数据参数、测试方法描述给测试类动态添加一条新的测试方法，最后删除无效的测试用例。

最终将含有新测试方法的测试类返回。

unittest 在收集测试用例时，根据一定的规则将新测试方法当作测试用例执行。

至此，@ddt 和 @data 源码分析结束。根据以上的逻辑，读者可以自行实现一个简单的参数化装饰器。

拓展学习

[1] 【Python 自动化测试学习交流群】学习交流

咨询：微信 atstudy-js 备注：学习群

Selenium Grid 分布式入门（Mac）

◆作者：枫叶

一．环境安装

1.安装 pycharm，安装社区版

2.安装 python 3.11.0

3.控制台安装 selenium，命令

pip3 install selenium

pingguo@192 / % pip3 install selenium

Collecting selenium

Downloading selenium-4.9.1-py3-none-any.whl (6.6 MB)

(■■■■■■■■■■■■■■■■■■■■I 6.6 kB/s

Collecting certifi>=2021.10.8

Downloading certifi-2023.5.7-py3-none-any.whl (156 kB)

(■■■■■■■■■■■■■■■■■■■■I 156 kB 32 kB/s

Collecting trio-websocket~=0.9

Downloading trio_websocket-0.10.2-py3-none-any.whl (17 kB)

Collecting urllib3[socks]<3,>=1.26

Downloading urllib3-2.0.2-py3-none-any.whl (123 kB)

I■■■■■■■■■■■■■■■■■■■■ I 123 44 kB/s

Collecting trio-=0.17

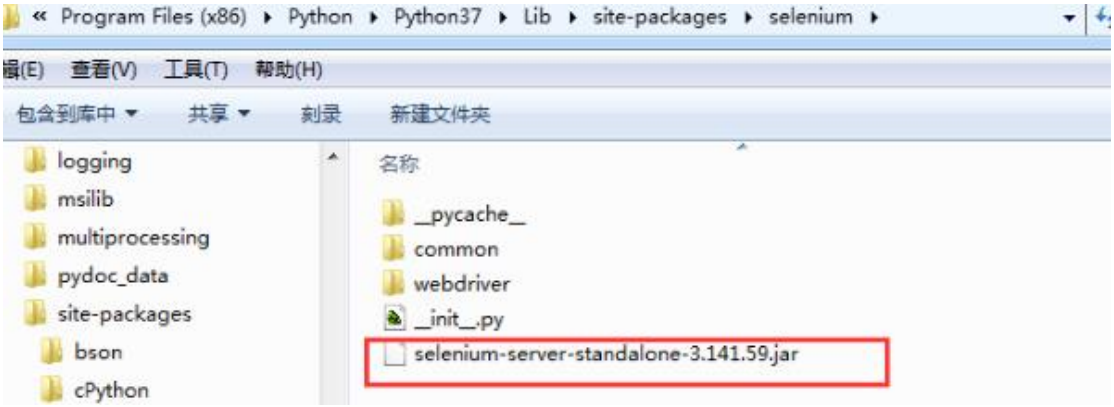
Downloading trio-0.22.0-py3-none-any.whl (384 kB)

(■■■■■■■■■■■■■■■■■■■■I 384 kB 68 kB/s

Collecting attrs>=19.2.0

Downloading attrs-23.1.0-py3-none-any.whl (61 kB)

完成安装，查看 selenium grid2 已经集成到 selenium server 中了（即 selenium-server-standalone-XXX.jar 包中）



二．知识点

1.selenium server 包含 grid

2.运行 grid 需要 java 环境，jdk 和 jre

3.下载 jar 包，https://www.selenium.dev/downloads/，放到本地 webdriver 文件夹下

4.grid 不分角色运行 java -jar selenium-server-standalone-3.141.59.jar

5.分配 hub 角色，java -jar selenium-server-standalone-3.141.59.jar -role hub

6.控制远程节点：

- 相互可以 ping 通
- java 环境和 selenium-server-xxx.jar
- 浏览器和驱动，设置 path

三．配置

1.java -jar selenium-server-standalone-3.141.59.jar

```
d:\dev\webdriver>dir
 驱动器 D 中的卷是新加卷
 卷的序列号是 18B3-90C3
d:\devRebdriver 的目录
```

2024/11/30 10:13 <DIR> .

2024/11/30 10:13 <DIR> ..

2024/11/30

10:13 10, 649, 948 selenium-server-standalone-3. 141. 59. jar

1 个文件 10, 649, 948 字节

2 个目录 125,290,975,232 可用字节

d:\dev\webdriver>java -jar selenium-server-standalone-3. 141. 59. jar

10:43:42.818 INFO [GridLauncherV3. parse] - Selenium server version: 3. 141. 59, revision: e82be7d358

10:43:42. 907 INFO [GridLauncherV3. lambda$buildLaunchers$3] - Launching a standalone Selenium Server on port 4444

2024/11/30 10:43:42. 962:INFO::main: Logging initialized ?467ms to org. seleniumhq. jetty9. util. log. StdErrLog

10:43:43.240 INFO [WebDriverServlet. <init>] - Initialising WebDriverServlet

10:43:44.811 INFO [SeleniumServer. boot] - Selenium Server is up and running on port 4444

java -jar selenium-server-standalone-3.141.59.jar -role hub

d:\dev\webdriver

2. java -jar selenium-server-standalone-3.141.59.jar -role hub

11:27:07.912 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358

11:27:08.002 INFO [GridLauncherV3?lambda$buildLaunchers$5] - Launching Selenium Grid hub on port 4444

2024/11/30 11:27:08.359:INFO::main: Logging initialized @727ms to org.seleniumhq.jetty9.util.log.StdErrLog

11:27:11.009 INFO [Hub.start] - Selenium Grid hub is up and running

11:27:11.010 INFO [Hub.start] - Nodes should register to http://192.*.*.*:4444/grid/register/

11:2 7:11.010 INFO [Hub.start] - Clients should connect to http:/192.*.*.*： 4444/wd/hub

3.java -jar selenium-server-standalone-3.141.59.jar -role node

d:\dev\webdriver

X java -jar selenium-server-standalone-3.141.59.jar -role node

11:37:22.531 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59 revision: e82be7d3S8

11:37:22.671 INFO [GridLauncherV3.1ambda$buildLaunchers$7] - Launching a Selenium Grid node on port 1928

2024/11/30 11:37:26.016:INFO::main: Logging initialized @3768ms to org.seleniumhq.jetty9.util.log.StdErrLog

11:37:26.213 INFO [WebDriverServlet.<init>] - Initialising MebDriverServlet

11:37:26.285 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 1928 \

11:37:26.285 INFO [GridLauncherV3.1afflbda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub

11:37:27.374 INFO [SelfRegisteringRemote$l.run] - Starting auto registration thread. Will try to register every 5000 ms.

11:37:28.884 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://localhost:4444/grid/re

gister

11:37:30.615 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use

4.又分配一个节点

D:\dev\webdriver

A java -jar selenium-server-standalone-3.141.59.jar -role node

111:40:12.228 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358

11:40:12.356 INFO [GridLaunche|fV3.lambda$buildLaunchers$7] - Launching a Selenium Grid node on port 7371

2024/11/30 11:40:15.808:INFO::main: Logging initialized @3868ms to org.seleniumhq.jetty9.util.log.StdErrLog

11:40:16.055 INFO [WebDriverServlet.<init>] - Initialising WebDriverServlet

ill:40:16.148 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 7371

11:40:16.149 INFO [GridLauncherV3.1ambda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub

ill:40:17.355 INFO [SelfRegisteringRemote$l.run] - Starting auto registration thread. Will try to register every 5000 ms.

： 11:40:18.853 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://localhost:4444/grid/re

gister

11:40:19.962 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use

回到 hub 看，有 2 个节点 node 注册了

d:\dev\webdriver

入 java -jar selenium-server-standalone-3.141.59.jar -role hub

11:27:07.912 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358

11:27:08.602 INFO [GridLauncherV3.lambda$buildLaunchers$5] - Launching Selenium Grid hub on port 4444

2024/11/30 11:27:08.359:INFO::main: Logging initialized @727ms to org.seleniumhq.jetty9.util.log.StdErrLog

11:27:11.009 INFO [Hub.start] - Selenium Grid hub is up and running

11:27:11.010 INFO [Hub.start] - Nodes should register to http://192.*.*.*:4444/grid/register/

11:27:11.010 INFO [Hub.start] - Clients should connect to http://192.*.*.*:4444/wd/hub

11:37:30.015 INFO [DefaultGridRegistry.add] - Registered a node http://192.*.*.*:1928

11:40:19.962 INFO [DefaultGridRegistry.add] - Registered a node http://192.*.*.*:7371

四．控制远程节点的运行

1.相互之间 ping 通

D\dev\driver

来 自 Ping 192.*.*.*具有 32 字节的数据：

来 自 192.*.*.* 的回复：字节=32 时间＜ lms TTL=64

来 自 192.*.*.* 的回复：字节=32 时间＜ lms TTL=64

来 192.*.*.* 的回复：字节=32 时间 Cms TTL=64

192.*.*.* 的回复：字节=32 时间＜ lms TTL=64

192.*.*.* 的 Ping 统计信息：

数据包：已发送=4,已接收=4,丢失=0 （ 0%丢失），

往返行程的估计时间（以毫秒为单位）：

最短=0ms,最卫=0ms.平应=0ms

2.Java 环境和 java_selenium_xxx.jar

远程虚拟机 Ubuntu 安装 java

执行命令

sudo apt install openjdk-8-jre-headless

验证 java 已装

```
fengye@ubuntu:~/Desktop$ java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
    -zero         to select the "zero" VM
    -dcevm        to select the "dcevm" VM
                  The default VM is server.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A : separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release.
```

安装 javac，执行命令

sudo apt install openjdk-8-jdk-headless

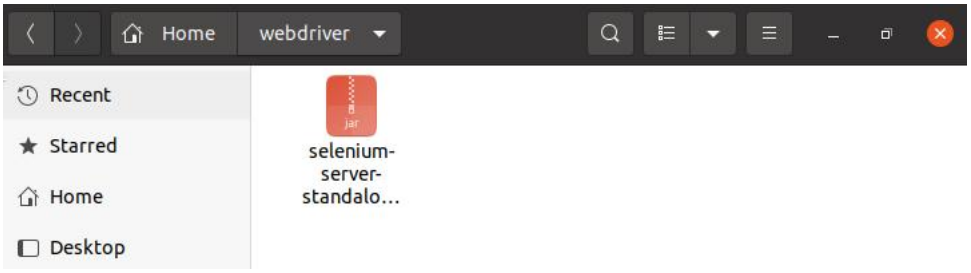
确认 javac 已安装

```
fengye@ubuntu:~/Desktop$ javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are
used
  -classpath <path>          Specify where to find user class files and annotat
ion processors
  -cp <path>                 Specify where to find user class files and annotat
ion processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compi
lation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors
 to run; bypasses default discovery process
```

获取 java 和 javac 的版本号

```
fengye@ubuntu:~/Desktop$ java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~20.04-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
fengye@ubuntu:~/Desktop$ javac -version
javac 1.8.0_292
```

在 ubuntu 系统中拷贝了之前 windows 系统中的 jar 包



3.浏览器 chrome/firefox 和对应的驱动，而且驱动要设置环境变量 path。

五．selenium grid2

先说明两个角色，hub 和 node。hub 负责主点，node 负责 node 节点。

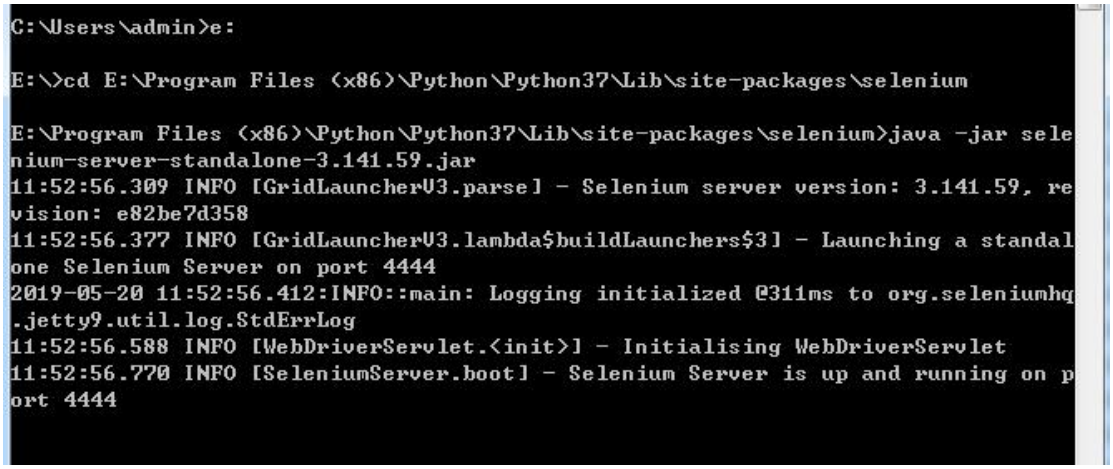
selenium-server-standalone-3.141.59.jar

首先，下载，在官网下载 https://www.selenium.dev/downloads/

再次，配置 java 的环境 jdk 和 jre，加入到 path 中；

最后，找到 selenium 的安装路径，一般在\Python\Python37\Lib\site-packages\selenium，把第一步得到的 jar 包放到该路径下，并执行

java -jar selenium-server-standalone-3.11.0.jar



备注：由于配置 javaappium 环境需要用到 selenium 类库，导入了该版本 java -jar selenium-server-standalone-3.11.0.jar。截止目前稳定版是 selenium-server-4.27.0.jar。

```
C:\Users\fengye>d:

d:\dev\Python\Python37\Lib\site-packages\selenium>java -jar selenium-server-standalone-3.11.0.jar
10:02:35.916 INFO [GridLauncherV3.launch] - Selenium build info: version: '3.11.0', revision: 'e59cfb3'
10:02:35.917 INFO [GridLauncherV3$1.launch] - Launching a standalone Selenium Server on port 4444
2020-10-29 10:02:35.987:INFO::main: Logging initialized @303ms to org.seleniumhq.jetty9.util.log.StdErrLog
10:02:36.148 ERROR [SeleniumServer.boot] - Port 4444 is busy, please choose a free port and specify it using -port optio
n

d:\dev\Python\Python37\Lib\site-packages\selenium>
```

提示 port4444 is busy，通过执行命令

netstat -aon|findstr 4444

得到 PID，这里是 4476

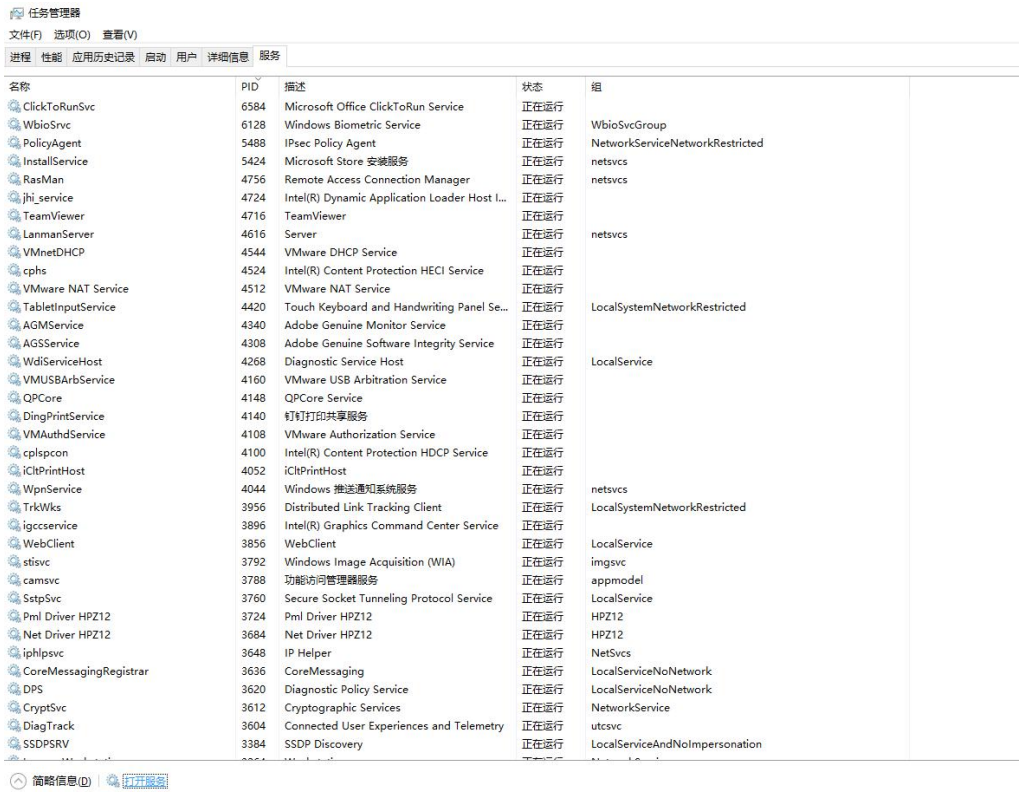
```
C:\Users\fengye>netstat -aon|findstr 4444
  TCP    0.0.0.0:4444           0.0.0.0:0              LISTENING       4476
  TCP    [::]:4444              [::]:0                 LISTENING       4476

C:\Users\fengye>_
```

查看是哪个进程或程序使用它

执行命令 tasklist|findstr “4476”

```
C:\Users\fengye>tasklist|findstr "4476"
javaw.exe                     4476 Console                    1      4,900 K

C:\Users\fengye>
```

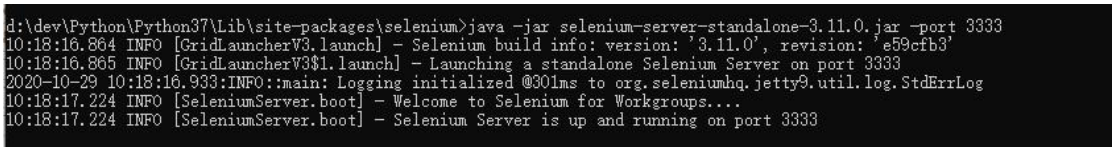
可以看到是 javaw.exe，打开任务管理器，服务尝试查找该进程或程序并关闭它。



回忆起 javaw.exe 在软件安装目录下 androidstudio 下，强制关闭它。

那么换 hub 的 port：3333，执行命令

java -jar selenium-server-standalone-3.11.0.jar -port 3333



启动 node，设置 port 为 5555

执行命令 java -jar selenium-server-standalone-3.11.0.jar -port 5555

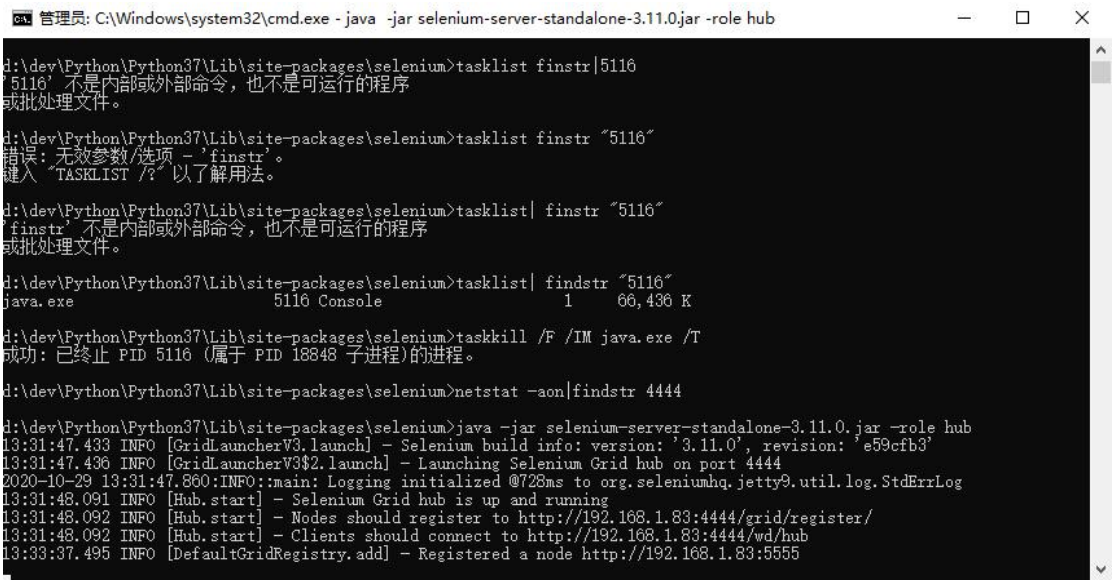
换完之后 node 启动报错，因此得关闭 javaw.exe

强制关闭端口的命令：

TASKKILL /F /IM javaw.exe /T

关闭所有 4444 端口的程序或进程，再次启动 hub，

Java -jar selenium-server-standalone-3.11.0.jar - role hub



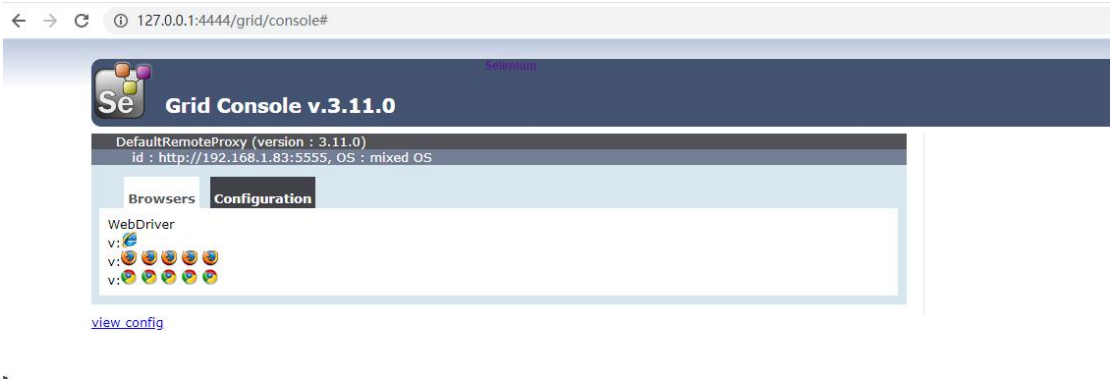
重开一个 cmd 窗口，同样地，进入到 jar 所在路径，本地路径是 D:\dev\Python\Python37\Lib\site-packages\selenium，执行命令 java -jar selenium-server-standalone-3.11.0.jar -role node

```
C:\Users\fengye>cd d:/dev/python/python37/Lib/site-packages/selenium

C:\Users\fengye>d:

d:\dev\Python\Python37\Lib\site-packages\selenium>java -jar selenium-server-standalone-3.11.0.jar -role node
13:33:36.516 INFO [GridLauncherV3.launch] - Selenium build info: version: '3.11.0', revision: 'e59cfb3'
13:33:36.521 INFO [GridLauncherV3$3.launch] - Launching a Selenium Grid node on port 5555
2020-10-29 13:33:36.896:INFO::main: Logging initialized @711ms to org.seleniumhq.jetty9.util.log.StdErrLog
13:33:37.017 INFO [SeleniumServer.boot] - Welcome to Selenium for Workgroups....
13:33:37.017 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 5555
13:33:37.017 INFO [GridLauncherV3$3.launch] - Selenium Grid node is up and ready to register to the hub
13:33:37.023 INFO [SelfRegisteringRemote$1.run] - Starting auto registration thread. Will try to register every 5000 ms.

13:33:37.023 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://localhost:4444/grid/re
gister
13:33:37.502 INFO [SelfRegisteringRemote.registerToHub] - Updating the node configuration from the hub
13:33:37.565 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use
```

扩展：

可见 hub 默认的端口是 4444，node 默认端口是 5555

浏览器访问 http://127.0.0.1:4444/grid/console#，可以查看启动的端口。



安装 nodejs，本机已安装，截止到 2020/10/29 最新的长期支持版为 node-v14.15.0-x64.msi

在 appium 下找到 js 启动脚本路径，本机是 C:\Program Files\Appium\resources\app\node_modules\appium\build\lib

执行命令 node main.js -a 192.168.1.83 -p 4723 -bp 4725 -U 3DN4C16505000633 --nodeconfig e:\ats\test_grid\test1huawei.json

拓展学习

[2] 云端全链路测试技术全栈学习

咨询：微信 atstudy-js 备注：全栈

当测试有了编码能力，开发插件提效就能成为核心竞争力

◆ 作者：海盐甜甜

从事软件测试六年了，平时业务重迭代快，如何增效便成为了测试的 KPI 之一。为此，我做过接口自动化、UI 自动化、数据工厂（一键生成测试数据），目标是希望在测试过程中节约造测试数据的时间以及借助自动化辅助回归。几年下来，我发现我对 java 是比较擅长的，所有的小工具也都是用 java 实现的，照葫芦画瓢，渐渐的大家也都会了。今年以来，我又陷入了沉思：当一个技能到了人人都会的程度，那我的核心竞争力是什么？

一次偶然的机会，我找到了我新的突破点。那天，我是要验证前端的一个展示，于是打开了 Charles 开始篡改返回，前端同事突然问我：你不觉得很重吗？测试一个点打开一个工具。然后，他推荐了前端常用的 mock 的方法，直接在 Console 就执行完了。随后我提出如果有多个接口要 mock 且这个结果要保持较长时间，这显然不是好方法，于是他又给我推荐了一个插件 Ajax Modifier，果然非常好用。在那一刻，我感受到了插件的力量，轻便又高效。在了解到一些简单的插件的实现对前端代码能力的要求并不是很高的情况下，我开始自学前端并尝试在测试工作中挖掘插件的需求，尝试写插件。

今年以来，我们团队进入了快速迭代的阶段，一个月有 2-3 个迭代且几个迭代并行，在各个迭代的范围确认之后便根据上线时间倒排确认提测时间、编写用例时间、技术评审时间、用例评审时间，整个过程十分紧凑，基本上没有 buff。还会经常遇到一些“困难”，如研发阶段发现任务难度、影响范围超出之前预研的结果；如有紧急需求插入（一般是不可控的第三方影响或者用户退费威胁）。而当问题来了的时候，我们就是解决问题，绝对不会影响版本的正常发布。那基本上逃不掉的就是加班，我估算过因为这样的变动导致一整周的每个晚上都在加班，有时甚至周末也需要加班。加班的内容除了因为延期提

测和扩大测试范围，还包括管理迭代以及迭代任务分配，根本目的保障质量。既然加班是逃不掉的，那么就只能从少加一点班去思考有没有什么既能提效又能保障质量的方法了。于是，插件的需求就被挖掘出来了。

一、开发用于管理迭代验证策略的插件：禅道复制链接及文字

如何让迭代任务更加清晰明了，团队成员明确责任意识，共同为上线质量保驾护航，产品侧、开发侧、测试侧都有自己的方案，我们测试侧做的是维护验证策略，即在一轮测试完成后会做验证策略的评审。验证策略评审主要讲的是针对这个任务测试点包含的内容，主要目的一是请产品和开发一起查缺补漏（测试环境和回归环境是最为详尽的，预发布环境和生成环境是支持验证的场景）；二是测试之间做交叉测试，验证策略可作为测试依据。为此，我做个一个拷贝迭代任务的插件，方便一键复制，从而根据任务把对应的验证策略维护在在线文档里。以禅道为例：

代码实现

```
// ==UserScript==
// @name         复制禅道迭代计划
// @namespace    http://tampermonkey.net/
// @version      2024-05-29
// @description  try to take over the world!
// @author       You
// @match        https://zentao.dc.[redacted]/pro/productplan-view-*.html
// @icon         https://zentao.dc[redacted]/pro/favicon.ico
// @grant        GM.setClipboard
// ==/UserScript==

(function () {
  // 确保页面加载完成后执行
  window.addEventListener('load', () => {
    // 检查当前URL是否匹配
    if (window.location.href.match(/https:\/\/zentao\.dc\.[redacted]\/pro\/productplan-view-.*\.html/)) {
      // 添加复制按钮到页面
      const copyBtn = document.createElement('button');
      copyBtn.textContent = '复制链接及文字';
      copyBtn.style.top = '0px';
      copyBtn.style.left = '-110px';
      copyBtn.style.height = '33px';
      copyBtn.style.position = 'absolute';
      copyBtn.style.background = '#16a8f8';
      copyBtn.style.color = '#fff';
      copyBtn.style.border = 'none';
      copyBtn.style.borderRadius = '4px';
      const btnGroup = document.getElementById('stories').getElementsByClassName('btn-group')[0];
      btnGroup.insertBefore(copyBtn, btnGroup.firstChild);
```

```
        // 复制功能
        copyBtn.addEventListener('click', () => {
          const links = Array.from(document.querySelectorAll('tr td:nth-child(5) a'));
          console.log(links);
          const linksText = links.map(link => {
            return `=HYPERLINK("${link.href}","${link.textContent}")`;
          }).join('\n');
          GM.setClipboard(linksText);
          alert('链接及对应文字已复制到剪贴板，可以粘贴到Excel中使用！');
        });
      } else {
        // 如果不在匹配的页面，删除按钮
        const existingButton = document.querySelector('button[textContent="复制链接及文字"]');
        if (existingButton) {
          existingButton.parentNode.removeChild(existingButton);
        }
      }
    });
})();
```

二、开发一键复制已格式化的入参插件：list 入参（复制数据的同时中间带英文格式的逗号）

接口入参为 list，测试性能的时候，往往会选择从数据库捞取有效数据，然后在所有复制出来的数据里手动输入英文格式的逗号，这显得十分繁琐，于是就在可视化的数据库里做了一个插件，便于在捞测试数据的同时支持复制格式正确的内容，一键生成真实有效的测试数据，可以节省不少时间。以数据库拷贝测试数据为例：

代码实现

```
// ==UserScript==
// @name         复制列
// @namespace    http://tampermonkey.net/
// @version      2024-10-25
// @description  try to take over the world!
// @author       You
// @match        https://[redacted].com/
// @icon         https://www.google.com/s2/favicons?sz=64&domain=servyou-it.com
// @grant        GM_getValue
// @grant        GM_setClipboard
// ==/UserScript==

(function () {
  "use strict";

  // 监听 DOM 变化
  const observer = new MutationObserver((mutations) => {
    mutations.forEach((mutation) => {
      if (mutation.addedNodes) {
        mutation.addedNodes.forEach((node) => {
          if (
            node.nodeType === 1 &&
            node.classList.contains("bgrid-contextmenu")
          ) {
            // 检查是否为元素节点并且包含指定类名
            const layer = node.querySelector(".bgrid-contextmenu-layer");
            if (layer && layer.children.length >= 2) {
              const firstChild = layer.children[0];
              const newItem = document.createElement("div");
              newItem.className = "bgrid-contextmenu-item";
              const icon = document.createElement("p");
              icon.className = "icon";
              icon.innerHTML =
                '<svg width="1em" height="1em" viewBox="0 0 48 48" fill="none"><path d="M13 12.4316V7.8125C13 6.2592 14.2592 5 15.8125
                5H40.1875C41.7408 5 43 6.2592 43 7.8125V32.1875C43 33.7408 41.7408 35 40.1875 35H35.5163"></path><path d="M32.1875
                13H7.8125C6.2592 13 5 14.2592 5 15.8125V40.1875C5 41.7408 6.2592 43 7.8125 43H32.1875C33.7408 43 35 41.7408 35
                40.1875V15.8125C35 14.2592 33.7408 13 32.1875 13Z"></path></svg>';
              const label = document.createElement("p");
              label.className = "label";
              label.textContent = "复制, 间隔";
              const desc = document.createElement("p");
              newItem.appendChild(icon);
              newItem.appendChild(label);
              newItem.appendChild(desc);
              firstChild.parentNode.insertBefore(
                newItem,
                firstChild.nextSibling
              );
              newItem.addEventListener("click", async function () {
                firstChild.click();
                const text = await navigator.clipboard.readText();
                GM_setClipboard(text.split("\n").join(","));
              });
            }
          }
        });
      }
    });
  });

  // 配置观察选项:
  const config = { childList: true, subtree: true };

  // 开始观察:
  observer.observe(document.body, config);
})();
```

这两个插件的开发和使用对我的测试工作有着不一样的意义，是在工作中的反思，是在反思后的成果。我们每天会有很多重复性的工作，也会有很多惯用的方式方法，当你觉得工作累的时候不妨停下来思考一下是不是能有更好的方法让自己轻松一点？什么是核心竞争力，我想勤于思考，勇敢实践算吧！

Python 接口自动化测试之操作 Excel

◆作者：渔民呀

大家在写接口自动化测试框架的时候，都会需要一些测试数据，比如说接口的入参、断言数据的等等，往往需要我们将这些数据提前准备好。我在之前准备测试数据的方式都是手动将测试数据转换成 json 文件，这种方式有两个很明显的弊端：

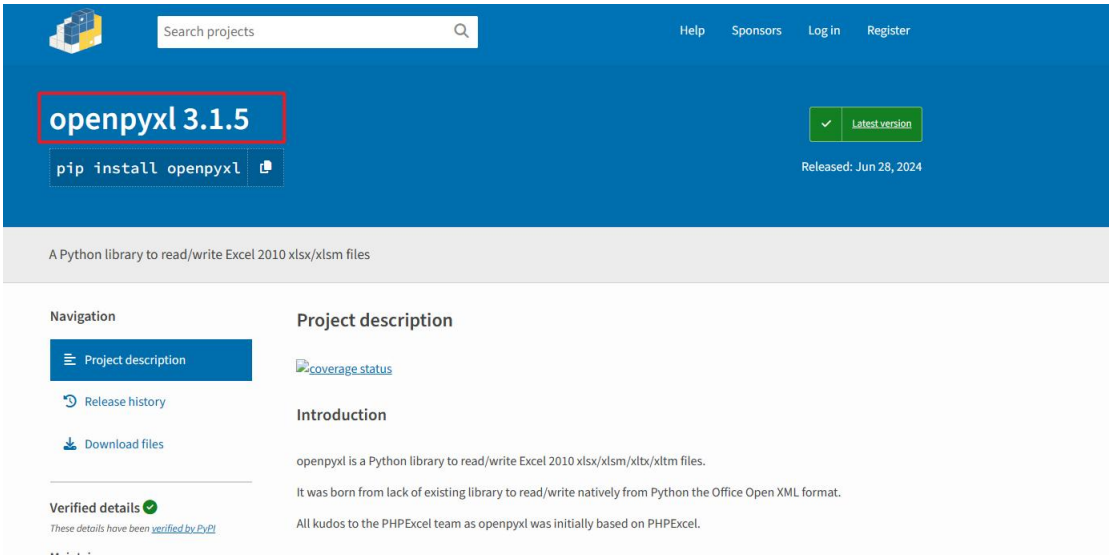
一、工作量大，效率低。当测试用例和测试数据比较多的时候，手动转化的工作量特别大，而且容易遗漏、出错，效率也很低。

二、灵活性低，测试数据不易维护。一旦后期测试用例改动或更新都需要手动去修改 json 文件，非常不方便，增加了对人工的依赖，降低了测试数据的可维护性。

手动准备测试数据的方式肯定是有待优化的，最近学习到一种将测试用例转换成测试数据的方法，就是使用 openpyxl 库，读取 Excel 文件中的测试用例，转化成测试数据。相比之前手动准备测试数据的方式，这种方式更加便捷和灵活。可以轻松的完成测试用例到测试数据的转换，非常好用。

openpyxl 库是一个用于读写 Excel 2010 及更高版本的.xlsx/.xlsm 文件的 Python 库。它提供了强大的接口，使得开发者可以轻松地操作 Excel 文档，进行数据读写和格式化处理。最新的版本是 openpyxl 3.1.5，在 Python 官网可以清楚的看到 openpyxl 版本和一些编码示例

三方库地址：https://pypi.org/project/openpyxl/



下面来介绍下如何通过 openpyxl 三方库操作 Excel 读取和写入数据。

一、编写好接口测试用例，示例如下

接口名称	请求URL	请求方式	请求头	请求参数类型	请求参数	预期结果	是否执行	测试结果	备注
新增站点	/environment/station/add	POST	{"Content-Type":"application/json","Authorization":"eyJhbGciOiJIUzUxMiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VyX2tleSI6Ijc4OGVlY2QwLTVmNGUtNDE5OS1hNTFjLTQwZTM5MjAzYTQ3MyIsInVzZXJuYW1lIjoiYWRtaW4ifQ.ugP1KWS4OrAITMGo-_WgwT4hLDPnEgzDAqnCftTP-Cr8X1e7FJhxOpAAxo0IGSGDVnqFqkK6YjeSfmefI7UUtA"}	json	{ "address": "美丽路", "areaId": 10016, "attachment": None, "cityCode": None, "cityName": None, "customerId": 1, "districtCode": None, "districtName": None, "latitude": None, "latitudeWgs": None, "longitude": None, "longitudeWgs": None, "provinceCode": None, "provinceName": None, "remark": "站点一", "stationName": "测试01", "stationStatus": 0, "stationType": 3, "streetCode": None, "streetName": None }	{ "code": 200, "data": True, "msg": None }	是		

二、编写 openpyxl 操作 Excel 数据的方法

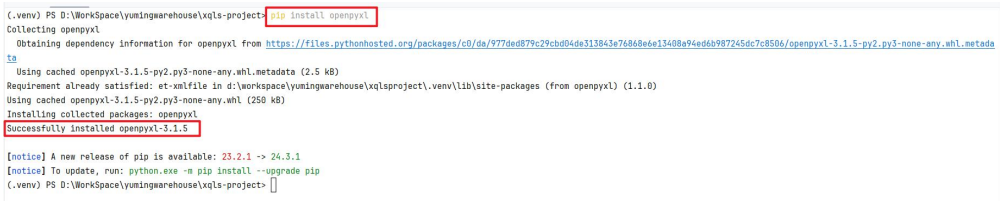
2.1 导包

```
#1、导包
import os
import json
import openpyxl
from config import cell_config
from config import BASE_DIR
```

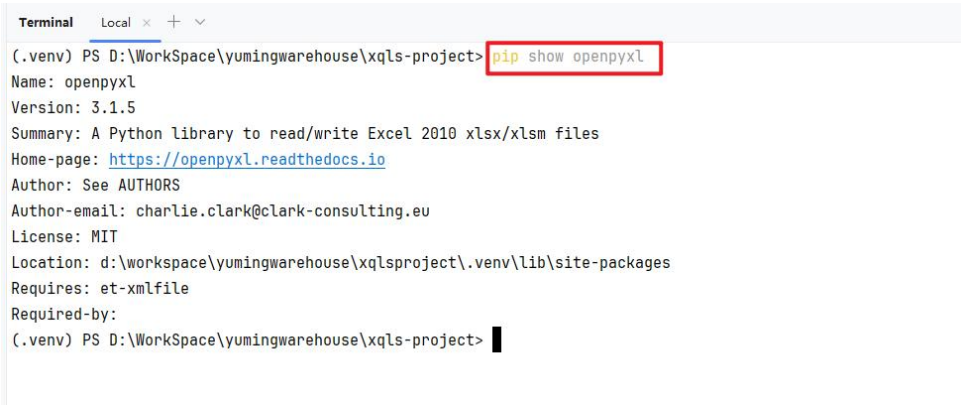
导包示例

其中 os、json 是 Python 的标准库，不需要下载，直接导入即可；openpyxl 因为是第三方库，使用前需要进行下载，使用 pip 包管理器下载即可，可以在命令行中输入以下命令：

pip install openpyxl



验证是否安装成功，可输入以下命令：



cell_config、BASE_DIR 是我在配置文件中自定义的变量，用于后续代码中，后续会介绍到。

2.2 定义 FileTool 类，类中定义__init__()初始化方法打开制定 Excel 文件，获取表单对象。

```
class FileTool:
    # 1、初始化
    @classmethod
    def __init__(cls, filename):
        # 2、动态文件路径
        cls.filename = BASE_DIR + os.sep + "data" + os.sep + filename
        print("要打开的文件是:", cls.filename)
        # 3、打开文件，获取workbook对象
        cls.workbook = openpyxl.load_workbook(cls.filename)
        # 4、获取sheet表单对象
        cls.sheet = cls.workbook[cls.workbook.sheetnames[0]]
        # 5、获取总行数
        cls.row = cls.sheet.max_row
        print("总行数：", cls.row)
```

cls.filename = BASE_DIR + os.sep + "data" + os.sep + filename 是设置对象的文件路径，就是所读取的 excel 文件的地址

BASE_DIR：自定义的变量，用于存放当前编码项目所在的根路径。

os.sep：这是 Python 的 os 模块中的一个变量，表示当前操作系统的路径分隔符。在 Windows 系统中，路径分隔符通常是反斜杠 \，而在 Unix/Linux/macOS 系统中，路径分隔符是正斜杠 /。使用 os.sep 可以使你的代码更加可移植，因为它会根据运行代码的操作系统自动选择正确的路径分隔符。

data:自定义的用于存放测试用例 Excel 文件的目录。

2.3 定义 read_excel 方法，读取 Excel 文件数据

```
# 2、读取excel数据
@classmethod
def read_ecxel(cls):
    # 1、新建空列表（存储每行测试数据）
    case = list()
    # 2、遍历每行数据
    for i in range(2, cls.row + 1):
        # 新建空字典（存储每行测试数据）
        data = dict()
        #判断是否执行该用例
        if cls.sheet.cell(i, cell_config.get('is_run')).value == "是":
            try:
                # 读取数据，追加到字典中
                data['path'] = cls.sheet.cell(i, cell_config.get("path")).value
                data['method'] = str(cls.sheet.cell(i, cell_config.get("method")).value).lower()
                data['headers'] = eval(cls.sheet.cell(i, cell_config.get("headers")).value)
                data['params_type'] = cls.sheet.cell(i, cell_config.get('params_type')).value
                data['params'] = eval(cls.sheet.cell(i, cell_config.get("params")).value)
                data['expect'] = eval(cls.sheet.cell(i, cell_config.get("expect")).value)
                # 记录每行数据的 行与列，执行完写入结果使用
                data['x_y'] = [i, cell_config.get("result")]
                # 将字典追加到列表中
                case.append(data)
                # 将读取结果写入到excel中
                cls.write_excel( x_y: [i, cell_config.get("desc")],  msg: "数据已读取完成！")
            except Exception as e:
                cls.write_excel( x_y: [i, cell_config.get("desc")], e)
    # 将读取的数据写入json文件中
    cls.write_json(case,  filename: "测试用例.json")
    print("读取的数据为:", case)
```

For 循环中下标从 2 开始是因为 Excel 文件中的第一行是表头名称，所以需要从第 2 行开始读取数据，其中读取字典的语句中，所有的字典 key 都是在上文 2.1 中提到的 config 配置文件中定义好的 cell_config，value 值分别对应 excel 文件中表头所在的列，第一列的序号是 1，以此类推。这样就可以将对应列的数据与定义的参数名关联起来。

```
# excel数据对应列
cell_config = {
    "path": 2,
    "method": 3,
    "headers": 4,
    "params_type": 5,
    "params": 6,
    "expect": 7,
    "is_run": 8,
    "result": 9,
    "desc": 10
}
```

2.4 定义 write_excel 方法，向 excel 中写入数据

```
# 3、写入excel
@classmethod
def write_excel(cls, x_y, msg):
    try:
        # x_y参数的格式为列表
        cls.sheet.cell(x_y[0], x_y[1]).value = msg
    except Exception as e:
        cls.sheet.cell(x_y[0], x_y[1]).value = e
    finally:
        # 保存excel
        cls.workbook.save(cls.filename)
```

x_y 是 data 字典中的键，x_y 的 value 保存的是 excel 文件中备注列“desc”所在的行号，列号，x_y[0], x_y[1]代表取列表中第一个元素和第二个元素，即行号、列号。每条测试用例数据读取完后，向 excel 中对应的备注列中写入数据读取完成的标识。

2.5 定义 write_json 方法，将读取的数据写入到 json 文件中

```
# 4、写入json
@classmethod
def write_json(cls, case, filename):
    filename = BASE_DIR + os.sep + "data" + os.sep + filename
    with open(filename, "w", encoding="utf-8") as f:
        json.dump(case, f, indent=4, ensure_ascii=False)
```

在 2.3 中定义的 read_excel 方法内，会在 for 循环执行完成后，也就是读取到所有需要执行的测试用例的数据并存入列表后，调用 write_json 方法将保存在列表中的数据最终写入到 json 文件中。

2.6 定义 read_json 方法，读取 json 中的测试数据

```
# 读取json数据
@classmethod
def read_json(cls, file_name):
    with open(file_name, "r", encoding="utf-8") as f:
        json_data = json.load(f)
        return json_data
```

这部分是为了将来在执行测试用例时，通过读取数据自动生成测试用例，实现参数化，这部分会在今后分享文章中介绍。

三、调用方法，验证执行结果

```
7
8  if __name__ == '__main__':
9      F1 = FileTool("测试用例.xlsx")
0      F1.read_ecxel()
1
2
```

执行结果：

- 成功读取到数据，并存放在列表中

```
D:\WorkSpace\yumingwarehouse\xqlsProject\.venv\Scripts\python.exe D:\WorkSpace\yumingwarehouse\xqls-project\common\readTestCase.py
要打开的文件是: D:\WorkSpace\yumingwarehouse\xqls-project\data\测试用例.xlsx
总行数： 3
读取的数据为: [{'path': '/environment/station/add', 'method': 'post', 'headers': {'Content-Type': 'application/json', 'Authorization': 'eyJhbGciOiJIUzUxMiJ9
.eyJ1c2VyX2lkIjoxLCJ1c2VyX2tleSI6Ijc4OGVlY2QwLTVmNGUtNDE5OS1hNTFjLTQwZTM5MjAzYTQ3MyIsInVzZXJuYW1lIjoiYWRtaW4ifQ
.ugP1KWS40rAXTM8o-_WgwY4hLOPnfgzDAqnCftTF-Cr8X1e7FJhxOpAAxoO1GSGDVnqFqkK6YjeSfmefI7UUtA'}, 'params_type': 'json', 'params': {'address': '美丽路', 'areaId': 10016, 'attachment': None, 'cityCode':
None, 'cityName': None, 'customerId': 1, 'districtCode': None, 'districtName': None, 'latitude': None, 'latitudeWgs': None, 'longitude': None, 'longitudeWgs': None, 'provinceCode': None,
'provinceName': None, 'remark': '站点一', 'stationName': '测试01', 'stationStatus': 0, 'stationType': 3, 'streetCode': None, 'streetName': None}, 'expect': {'code': 200, 'data': True, 'msg':
None}, 'x_y': [2, 9]}, {'path': '/environment/station/add', 'method': 'post', 'headers': {'Content-Type': 'application/json', 'Authorization': 'eyJhbGciOiJIUzUxMiJ9
.eyJ1c2VyX2lkIjoxLCJ1c2VyX2tleSI6Ijc4OGVlY2QwLTVmNGUtNDE5OS1hNTFjLTQwZTM5MjAzYTQ3MyIsInVzZXJuYW1lIjoiYWRtaW4ifQ
.ugP1KWS40rAXTM8o-_WgwY4hLOPnfgzDAqnCftTF-Cr8X1e7FJhxOpAAxoO1GSGDVnqFqkK6YjeSfmefI7UUtA'}, 'params_type': 'json', 'params': {'address': '富强路', 'areaId': 10016, 'attachment': None, 'cityCode':
None, 'cityName': None, 'customerId': 1, 'districtCode': None, 'districtName': None, 'latitude': None, 'latitudeWgs': None, 'longitude': None, 'longitudeWgs': None, 'provinceCode': None,
'provinceName': None, 'remark': '站点一', 'stationName': '测试01', 'stationStatus': 0, 'stationType': 3, 'streetCode': None, 'streetName': None}, 'expect': {'code': 200, 'data': True, 'msg':
None}, 'x_y': [3, 9]}]
```

- 数据读取完成后，成功像 Excel 文件中写入数据

接口名称	请求URL	请求方式	请求头	请求参数类型	请求参数	预期结果	是否执行	测试结果	备注
新增站点	/environment/station/add	POST	{"Content-Type":"application/json","Authorization":"eyJhbGciOiJIUzUxMiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VyX2tleSI6Ijc4OGVlY2QwLTVmNGUtNDE5OS1hNTFjLTQwZTM5MjAzYTQ3MyIsInVzZXJuYW1lIjoiYWRtaW4ifQ.ugP1KWS40rAXTM8o-_WgwY4hLOPnfgzDAqnCftTF-Cr8X1e7FJhxOpAAxoO1GSGDVnqFqkK6YjeSfmefI7UUtA"}	json	{ "address": "美丽路", "areaId": 10016, "attachment": None, "cityCode": None, "cityName": None, "customerId": 1, "districtCode": None, "districtName": None, "latitude": None, "latitudeWgs": None, "longitude": None, "longitudeWgs": None, "provinceCode": None, "provinceName": None, "remark": "站点一", "stationName": "测试01", "stationStatus": 0, "stationType": 3, "streetCode": None, "streetName": None }	{ "code": 200, "data": True, "msg": None }	是		数据已读取完成！

- 成功将数据转化成 json 数据

readTestCase.py　测试用例.json　config.py

```
[
    {
        "path": "/environment/station/add",
        "method": "post",
        "headers": {"Content-Type": "application/json"...},
        "params_type": "json",
        "params": {"address": "美丽路"...},
        "expect": {
            "code": 200,
            "data": true,
            "msg": null
        },
        "x_y": [ 2 elements… ]
    },
    {
        "path": "/environment/station/add",
        "method": "post",
        "headers": {"Content-Type": "application/json"...},
        "params_type": "json",
        "params": {"address": "富强路"...},
        "expect": {
            "code": 200,
            "data": true,
            "msg": null
        },
        "x_y": [ 2 elements… ]
    }
]
```

从以上的实践过程来看，我们可以感受到，通过使用 openpyxl 三方库控制 Excel 将测试用例转化成测试数据的方法确实非常实用，这样也提高了测试用例的使用价值，增加了测试数据乃至测试用例的可维护性。

在实际操作读取 Excel 数据将其最终转换成 json 数据的时候，我也遇到了一些问题，如果没处理好，否则可能会导致一些文件读写失败或者其他的问题，我也将我实操过程的经验整理下来，可供大家参考。

如果要操作的 Excel 文件在本地已经打开，在执行代码的过程中可能会无法写入数据，需要先将本地打开的文件关闭，再执行代码。

```
Traceback (most recent call last):
  File "D:\WorkSpace\yumingwarehouse\xqls-project\common\readTestCase.py", line 50, in read_ecxel
    cls.write_excel([i, cell_config.get("desc")], "数据已读取完成！")
  File "D:\WorkSpace\yumingwarehouse\xqls-project\common\readTestCase.py", line 67, in write_excel
    cls.workbook.save(cls.filename)
  File "D:\WorkSpace\yumingwarehouse\xqlsProject\.venv\lib\site-packages\openpyxl\workbook\workbook.py", line 386, in save
    save_workbook(self, filename)
  File "D:\WorkSpace\yumingwarehouse\xqlsProject\.venv\lib\site-packages\openpyxl\writer\excel.py", line 291, in save_workbook
    archive = ZipFile(filename, 'w', ZIP_DEFLATED, allowZip64=True)
  File "D:\Python\lib\zipfile.py", line 1249, in __init__
    self.fp = io.open(file, filemode)
PermissionError: [Errno 13] Permission denied: 'D:\\WorkSpace\\yumingwarehouse\\xqls-project\\data\\测试用例.xlsx'
```

参数值是空的时候，我们需要填写 None，不要填写为“”或者 null，写“”代表的是空字符串而不是空值；写 null 在执行代码的时候会报错，因为 Python 中没有定义这个值，Pythonz 中的控制是 None，我们在文件中填写 None，读取到 json 中会自动转换成 null。

```
  File "D:\WorkSpace\yumingwarehouse\xqlsProject\.venv\lib\site-packages\openpyxl\cell\cell.py", line 218, in value
    self._bind_value(value)
  File "D:\WorkSpace\yumingwarehouse\xqlsProject\.venv\lib\site-packages\openpyxl\cell\cell.py", line 187, in _bind_value
    raise ValueError("Cannot convert {0!r} to Excel".format(value))
ValueError: Cannot convert NameError("name 'null' is not defined") to Excel
```

操作数据的行和列时，行号和列号都是从 1 开始，这是 openpyxl 库设计时的一个特点，它使得操作 Excel 文件时更符合用户的直觉，因此在使用时候需要留心。

实际上 openpyxl 三方库的功能还是挺丰富的，围绕 Ecxel 文件操作，还有比如说创建文件、删除文件、复制文件、设置标题等等。我以上介绍的内容可能比较有限，主要是我过去使用 openpyxl 三方库控制 Excel 实操过程的记录和总结，希望能够给到有需要的伙伴一些参考，欢迎大家指正。

拓展学习

[3] 【Python 自动化测试学习交流群】学习交流

咨询：微信 atstudy-js 备注：学习群

测试的流程有哪些，每一步该怎么做好？

◆ 作者：蓝羽

软件发版（有的公司也称为上线或者更新）的流程大致可分为两种：

第一种：软件的开发周期和测试周期是完全独立的。假如某公司这个月要做一次软件的发版，设定任务周期为一个月，给开发的开发周期为两周，给测试的测试周期为两周。也就是说，给开发的两周时间，除非任务真的完不成，一般在这个截止日期开发要做好既定任务的开发，并提交测试。提交测试以后，测试要在两周内把既定的任务做好测试，并保证稳定。

当然，这只是一个例子，具体的还要看公司怎么执行。但是大体是一样的，这种公司有严格的开发和测试周期，这样的好处就是开发和测试都能在各自的时间里合理安排时间，做好自己的工作。

第二种：软件的开发周期和测试周期不是独立的。同样假设某公司要做一次软件的发版，但是这次发版任务有好几个模块要修改，又有几个模块要增加。那这种公司采取的策略就是开发做一个模块就提交一个模块，测试就开始测试这个已经做好的模块。开发再做一个模块测试再测试一个。开发和测试的时间是不独立的。

但是无论上面哪种类型：在软件发版的时候，测试想要保证软件的质量，都要经历下面的流程。

第一步：独立测试的内容

这部分内容就是分给你的，由你负责的测试内容。因为是分配给你的任务，所以如果这个部分出问题了，那除了这部分的开发之外，你就是直接的负责人。想要做好这部分的测试，请参考——不拍漏测！花 2 年总结的这个测试模版太全了这个文章。

第二步：交叉和兼容性测试

因为我主要做的是黑盒测试，我只说下我目前公司的情况。在第一步做完以后，觉得自己负责的这个功能你已经测试稳定了。假如你用的手机是小米手机，ios 系统用的是苹果 6 手机。那么，和你同在测试组的人，就会被安排用别的手机再测试一下你负责的这个功能。他们会被安排用别的设备去测试你负责的这个功能，一般就测试一遍。从而检查这个功能在别的设备上运行的情况。

在这个步骤中，往往是容易被追责的时候。为什么？

原因是这个功能是你自己负责的，会先经过你的第一遍测试。如果你在测试的时候，能把可以发现的 bug 都找出来，当别人被安排做交叉和兼容测试的时候，他们就基本测试不出来 bug。那么恭喜你，你的领导虽然不会觉得你做的很好，但至少不会觉得你做的差。但是如果你的同事拿到测试文档后，测试这个你负责的功能，他找出了很多个 bug。那么最直接的你的领导肯定会问你：这个功能你有认真测试吗？这就会留给领导你工作做的不好的印象。

也可能你会说，我自己已经测试的很好了，觉得没问题了，但是别人真的测试出来好多 bug。其实这种情况基本不会发生。如果你做的是黑盒测试，那么除非别人测试的手机屏幕比你的大或者小，否则一般很少出兼容测试的问题。大多数都是你没测试到的功能性的问题。

你可能也觉得开发中间改代码了怎么办？其实对于已经稳定的功能，开发也是不愿意再碰这些代码的，毕竟出问题了他自己也会有责任。

所以，如果在这个环节不想出问题，那就只能踏踏实实的做好第一步。

你负责的内容，别人会做交叉和兼容性测试。那么别人负责的内容，你也会做交叉兼容性测试。这个也得好好完成，其实要想做好就参照第一步的方法，把需求文档的内容，完完全全的好好测试一遍。如果你没发现 bug，那么按既定流程，改任务进度就好了。但是如果你发现了 bug，发现的 bug 还挺多的情况，怎么办？我的建议是，如果是确定的兼容性问题，可以直接提 bug。但如果是功能性的问题，建议你直接给相关的负责人说，由那个人提 bug。

这样，你能很好的做了兼容性测试，从而领导也不会觉得他做的不好。测试是可以互相帮助的，说不定哪一天，你有好多地方没做好，别人发现了你的 bug，也给你小窗提

醒，这样由你自己提到 bug 系统后，就还是你自己好好工作的表现。

在这里，我不是教你怎么偷奸耍滑，就是给你介绍了我的工作方式。因为一般公司的同事相处的都挺好的，所以互相帮助真的很好。

也许你会觉得这样领导就看不到你的工作能力了。其实不是的，在第一步独立测试阶段是最容易看一个人能力的。在第二步，你发现特别多的 bug 时，你可以提一部分，然后给同事说一部分，这样，既体现了你的工作能力，又帮助了同事。

第三步：发版前自己负责的内容全部测试一遍

做好了第一步和第二步，是为了保证这个版本，新增的功能或者是新要修改的功能稳定。但是以往的老功能呢，该怎么办呢？还是需要测试一遍的。在我们公司这个第三步叫上架测试，就是发版前把自己负责的内容全部测试一遍。这一步不同于第一步，不需要你测试各种细节，但是要测试你负责的主要功能。这一步想做好，那就得对自己负责的内容特别熟悉，这样你负责的内容才不会出问题。具体该怎么记忆自己负责的内容，以后会更新文章介绍。

这里说明一下：第一步，第二步，第三步都是在测试服务器上完成的。

第四步：发版后的检查测试

前三步做好以后，软件就会提交审核并发版，更新完正式服务器后，像我们公司，需要对线上的版本进行检查。目的是查看，从测试服务器更新到正式服务器的过程中是否有漏掉或者错误的内容，如果有就要及时更正。一般软件发版的时间都会避开用户使用的时间，这样就可以在不影响用户使用的情况下更新系统。这一步其实跟第三步的做法是一样的，只是第三步在测试服务器上，这一步在正式服务器上。

第五步：完成涉及的测试大纲

前四步完成后，整个发版流程基本结束了。但是你负责的内容，还有一项工作要做，那就是写负责内容的测试大纲。这个测试大纲主要的作用是以后你可能会忘记某些功能，那查看测试大纲就可以弄清楚功能的整个流程。还有一点，当这个内容不是你负责的时

候，另外的人接手这个功能时，查看测试大纲，可以更好的熟悉功能。每个公司有自己测试大纲书写的模式，但最基本的是写好每一个测试用例。

这里再多说几句。

有的公司因为开发时间和测试时间是独立的，所以上面几步测试完成的时候都可以按计划有顺序的进行。而且一般好点的公司，开发自己写完代码后都会自己跑一遍代码，看看自己写的代码是不是有大问题，然后再提交给测试进行测试。这样，测试做工作也不至于太耽误时间。

但是我所在的公司不是这种，是第二种。开发时间和测试时间是交叉的。而且公司也不统计开发的 bug 率。所以开发一般写完代码后都是直接连看都不看一遍，都堆给测试（为什么知道开发没看一眼，因为我每次刚点开就是 crash 的，但凡开发看一眼就会发现，甚至有的功能都没做就堆给测试了）。如果碰到这种的，作为测试，最主要的是要有个好心态。因为碰到这样的开发，可能有时候你要等待的时间很长。当然，有的时候也会很急，比如都快发版了，开发才提交代码。我们这种小点的公司，开发的权限比较大，所以想做好测试，那就得自己有自己的规划，明确知道每一步怎么做的又快又好。

每一个工作都不容易，但是如果你掌握了这个工种应有的技能，那就不难。希望看文章的你精进测试技术，把测试做的越来越好。

从一个测试人员的角度看如何击败骇客

◆作者：小猪

前言

我们测试人员在执行安全测试或者渗透测试的时候，大部分都会遵循一个流程化的方案，打个不太恰当的比喻，基本上就是流水线的工作，先做什么，再做什么，最后在出个报告，问题解决了，再迭代一次收尾。这样看来总觉得哪里不对劲，按理说安全（及渗透）测试的任务是发现系统潜在的漏洞，目的是防止骇客入侵，这个任务能不能达成最终目的，骇客是不是也和我们一样工作，笔者有把握说一定不是。

为什么笔者有把握说一定不是？因为如果我们的工作真的可以防止骇客入侵，那么就不会有红蓝对抗，护网行动了，而这两项工作又和我们测试人员关系不大。

尤其是现在的生产环境的前后端软硬件防御方案，已经把系统围的和铁桶似的了。在这种环境下如果还有哪个骇客敢正面进攻，只有两种情况，一是勇，二是愣。

勇，是人家有信心，也许防御方案中的某个关键部件就是人家做的，这个很好理解，造锁的一定能开锁，我加密的数据我一定能解，开源项目这么多，大部分公司都唯开发论，拿过来就当成自己的用，不重视测试，最终的系统防愣不防勇，也算活该。

笔者作为一个测试人员，想谈谈如何“击败”骇客，这个“击败”有两个层面的意思，一是挡住骇客的攻击，二是在挡住骇客攻击的同时给他一击。思考这个问题实际上可以给我们测试人员一个思维上的提升，并且在讨论系统部署方案，甚至设备选型的过程中有一定的发言权。

单纯的讲这个问题可能会很枯燥，那我们就围绕一个案例场景去展开，可能会有意思的多。

思考这么一个特殊的场景：

目标人物在某咖啡店喝咖啡，手机放在咖啡桌上，在不接触目标人物的前提下，通过技术手段入侵目标人物的手机并成功发送一条短信。如果你是一名黑客，你的入侵思路是什么？

这是一个看起来非常戏剧化的场景，但却是真实世界有可能发生的事件，想象一下案例中的目标人物有可能是运维人员，甚至公司的 CEO，事情是不是变得有趣起来了，围得和铁通似的系统，在内部被瓦解了。

其实这个案例场景也可以告诉我们现在的骇客团队的攻击方向是很多元化的，以前那种愣的做法，直接正面进攻打穿系统，植入反向通道的方法，成功的概率已经越来越低了。

在接下来内容中，我们首先会从多个角度分析骇客的行为，看骇客可以通过什么方法来达成案例中目的，接着简单讨论一下被动防御策略，就是挡住骇客攻击，然后再讨论一下主动诱捕策略，也就是给骇客一击，这块内容中笔者会以自己开发的模拟系统为例，为大家简单直观的展示诱捕策略的应用。

一、骇客的行为分析

我们在分析这个案例之前，必须先明确三个问题。

第一个问题就是：骇客的目的是什么？如下图 1-1 所示。

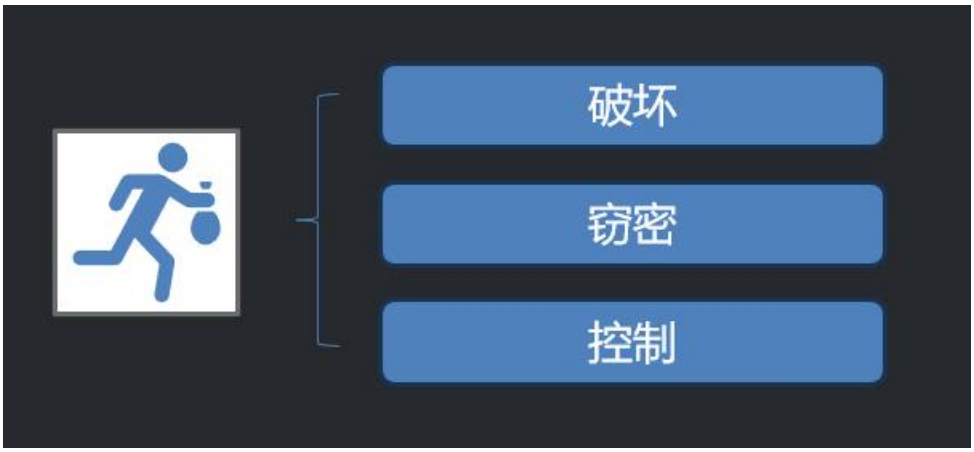


图 1-1 骇客的目的

所有骇客的目的总结一下无非就是上图中的三个，即破坏、窃密和控制。结合本案例，骇客要想利用目标人物的手机发送一条短信，首先就要控制目标人物的手机。

第二个问题：骇客的能力是什么。如下图 1-2 所示。

图 1-2 骇客的能力

通过本案例，骇客必然要进行信息收集和信息分析，才能确定目标人物，并进行有效的策略设计，最终达成目的。这里讲的策略设计不是简简单单的黑客工具的使用流程，更复杂，原因就是现在的网络环境以及终端设备的复杂性，导致几十年前的那种孤狼行为的成功率越来越低，现在都是团队化的骇客，为了一个高价值的目标，长期持续地进行隐秘的试探和攻击，也就是所谓的 APT（Advanced Persistent Threat，高级持续威胁）。

接下来是技术研发，这也是必不可少的能力。这里所说的技术研发可不单单指使用某个开发语言的编程能力，这是属于软件方向的能力，还有硬件方向的能力，有些在软件方向解决不了的技术问题，可能要靠一些硬件工具去实现。举个例子吧，像是破解一个设备，这个设备可能是玩游戏的，通过软件层面去破解可能非常复杂，也许根本不可能，但是通过更改设备主板上的某个电路，就可以轻易的绕过它的加密屏障，但这也是破解设备不耐用的原因，所以大家要支持正版。

最后是资源配置和资源利用，资源是广义的，一个匿名的服务器，一个网站等等都可以是资源。

第三个问题：骇客的原则是什么？如下图 1-3 所示。

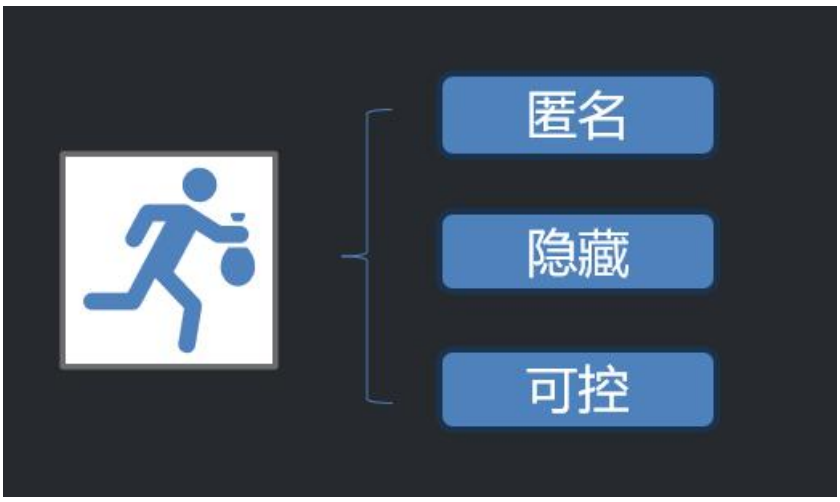


图 1-3 骇客的原则

这三个原则除了可控以外其他应该都很好理解，可控举个例子讲就是你别发动一个洪水攻击，或者叫什么高中低端离子炮，把关键路由都堵塞了，造成了大范围的影响，这个时候就只能卷铺盖跑路了，意思就是别做不能控制的事情，别得罪不能得罪的人。其余两项都是保证他安全的，毕竟骇客行为可不是什么好行为，本质上和偷、盗没什么区别，所以羡慕骇客的都是小孩子，受哪些无聊影视剧的影响，成年人没人羡慕他们。

接下来我们看一下骇客的思维，也就是他是怎么想的，如图 1-4 所示。

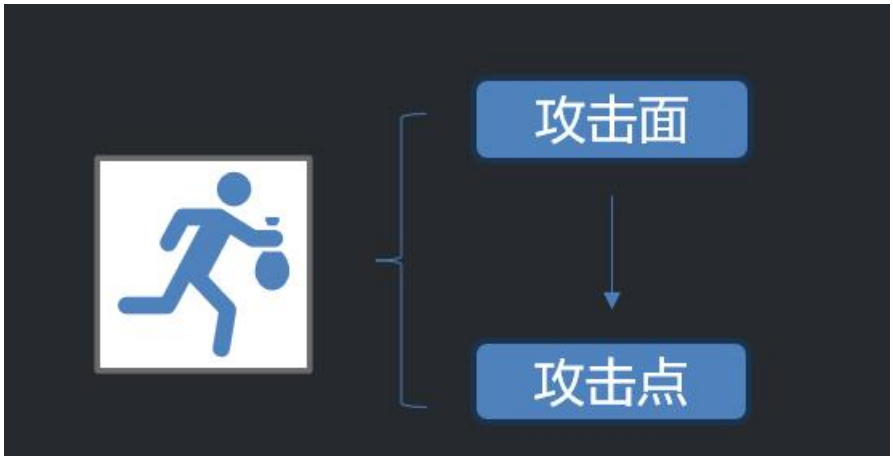


图 1-4 骇客的思维

骇客的思维简单的归纳就是确定攻击面，细分攻击点，实际上根据骇客的思维就能确定相应的防御方案。

好了，现在我们知道了骇客的目的，骇客的能力以及骇客的原则，并且知道了骇客的思维，所有的准备工作都就绪了，已经可以解决案例中提出的问题了。

大家可以展开想象，但记住要用技术手段，不能是暴力手段，不能把目标人物按在

地上，逼着他用手机发一条短信，因为要站在骇客的角度，所以必须坚持骇客的原则。

其实根据骇客的思维，手机就是一个显而易见的攻击面，根据案例场景要在不接触目标人物的情况下控制他的手机，那就要确定手机有哪些对外通信交互的功能，这些功能就是攻击点，第一个就是 WIFI，第二个就是蓝牙，而且有些型号的手机 WIFI 和蓝牙走的还是同一个电路，当然这涉及到硬件，就不展开太多了，除了这两个，还有 GPS，NFC（这个距离最短），基带（收发射频信号的，就是管通话的）等等。

案例中的场景还有一个特殊要求，就是控制他的手机还要发送一条短信，我能想到的有三个方向。

第一个方向，就是根据骇客的能力进行信息收集和分析，确定目标人物所使用的手机操作系统及其版本，再通过特定的技术手段向目标操作系统植入一个特定的 APP 来达到目的，所以说陌生的 APP 不要装，申请过多权限的 APP 要警惕。

第二个方向，这个方向要求骇客的技术研发能力非常高，估计要配合技术数段和社工手段才能做到，就是让目标人物的手机刷入一个定制的操作系统，开源的操作系统大家都可以下载编译，只要在操作系统的内核中做做文章，别说发送一条短信了，他在手机上的所有操作都在骇客的掌控之下，所以说刷机也要小心，尤其是找别人远程刷机。

第三个方向就涉及到硬件，当然不是说让你做个手机了，也许是一个 U 盘，这里就不展开讲了，陌生的 U 盘尽量不要用。

以上说的只是按照骇客的行为分析，根据案例场景得到的一个思路，实际执行起来非常的复杂也不一定能成功，大家也不需要过渡小心，除非你真的觉得自己是特别有价值的目标人物。

二、被动防御策略

针对骇客的被动防御策略就非常简单了，大家在工作当中也应该都接触过，如果经历过等保测评的话，了解的就更深。

基于系统层面的被动防御策略无外乎就是通过部署各协议层防火墙和入侵检测设备，基于特征和行为对可疑流量进行监控和阻断，这个时候溯源骇客主要靠日志记录，我们根据上一节内容中讲的骇客原则可以知道，骇客会使用各种跳板和匿名资源来保护自己，

因此这种基于事后的防御策略进行溯源的难度较大。

基于系统层面的被动防御策略虽然溯源的难度较大，但是完整的防御策略已经涵盖了安全的方方面面，系统层面只是其中的一小部分，完整的安全包括：物理安全，网络安全，主机安全，应用安全，数据安全及备份恢复，这里除了物理完全，其余几个安全基本上是骇客攻击的四个方向，其余还包括安全管理制度的建立，这个就是管人的了，是骇客社工的方向。

安全管理制度与我们测试人员关系不大，而且也太冗长不好总结，但是被动防御策略里面引入的相关安全设备，我们需要了解一下，在执行安全测试，尤其是渗透测试的时候会接触到这些设备。

物理安全里面的防盗窃和防破坏的设备、防火设备、温湿度控制、物理访问控制设备、防水和防潮设备和电力供应（就是 UPS 不间断供电）了解一下就好。

其余的包括：

访问控制设备（如：防火墙，如果是网站系统，需部署 WAF 即 WEB 应用防火墙，及防篡改系统等）。

边界性完整性检查设备，就是防内网用户私开通道连外网，像准出控制设备。

入侵防范，如：IDS 入侵检测和 IPS 入侵防御等设备。

安全审计设备，如：日志审计系统，数据库审计系统，日志服务器等。

恶意代码防范设备，如：网络版的杀毒软件。

备份和恢复设备，如：数据备份系统，异地容灾，虚拟化的漂移。

网络和系统安全设备，如：相关的漏洞扫描设备。

结构安全设备，这是系统部署方面的，如：负载均衡设备。

资源控制设备，如：运维管理系统。

网络设备防护，身份鉴别设备，如：堡垒机+Ukey 认证。

除了以上的这些设备，现在还引入了人工智能。大家觉得谁还会拿个工具就正面进攻，那就只剩下愣的了，网上愣的人还不少，下载一个装满黑客工具的操作系统，就觉得天下无敌了，殊不知那些黑客工具的流量早就被防火墙标识了，最终就玩了个寂寞。

因此笔者在这里建议对这方面技术感兴趣的朋友，本身又从事的测试工作，可以系统的学习一下安全测试，使用正规的测试工具，自己编写测试脚本，这里和黑客工具的区别就是，本身不对系统产生任何危害，点到即止，而且是在授权的情况下进行，安全可控。

话说回来，网上愣的人这么多，虽然对系统造成不了太严重的危害，可是也会消耗一些资源，那就通过下一节要讲的内容，给他一击，“鼓励”一下技术不到家还这么有勇气的行为。

三、主动诱捕策略

主动诱捕策略就是部署蜜罐系统。

对于骇客来讲，蜜就是漏洞，蜜罐就是装满漏洞的罐子。

一个优秀的蜜罐系统，可以长时间的拖住骇客，在设计好的路线上，让骇客一个蜜一个蜜的吃，尽可能多的留下痕迹也就是有效信息，最终奖励骇客蜜信(伪造的机密文件含有回溯代码)，把骇客钓出来。

这对于技术不到家，还特别有勇气，非常愣的骇客非常有效，他对于这些蜜就像蜜獾一样，越吃越开心，根本不会考虑会被蛰到。

如图 1-5 所示为简单的蜜罐系统部署图。

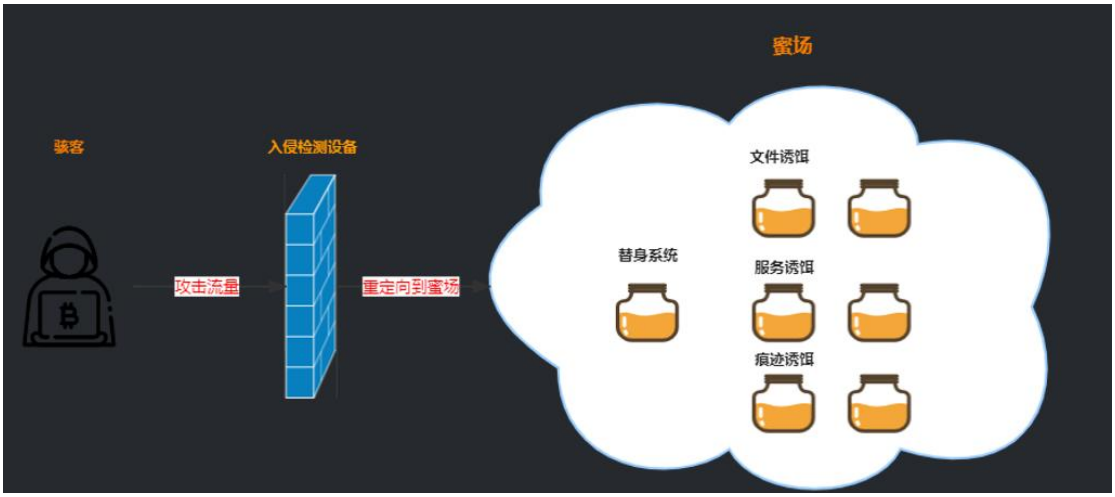


图 1-5 蜜罐系统部署图

接下来就是使用蜜罐系统演示如何诱捕骇客，使用笔者自行开发的安全测试工具进行演示，如图 1-6 所示启动安全测试工具的服务平台。

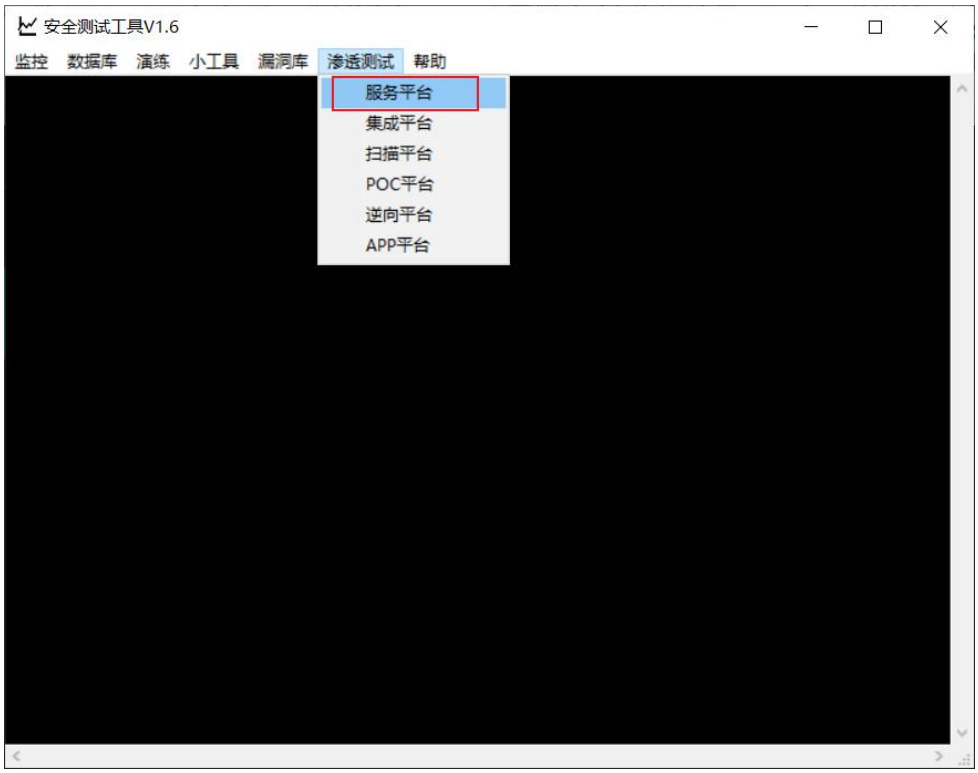


图 1-6 启动安全测试工具的服务平台

接着在服务平台的窗口中，输入命令“start http”启动蜜罐模拟系统，如图 1-7 所示。

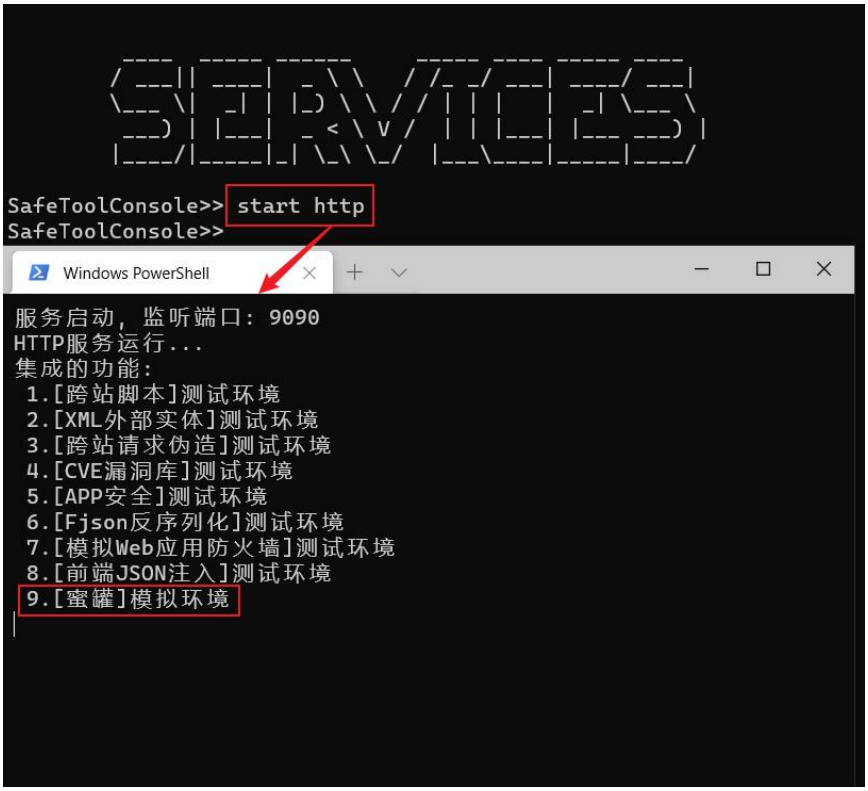


图 1-7 启动蜜罐模拟系统

浏览器打开模拟系统，这时模拟骇客进行 SQL 注入，如图 1-8 所示。

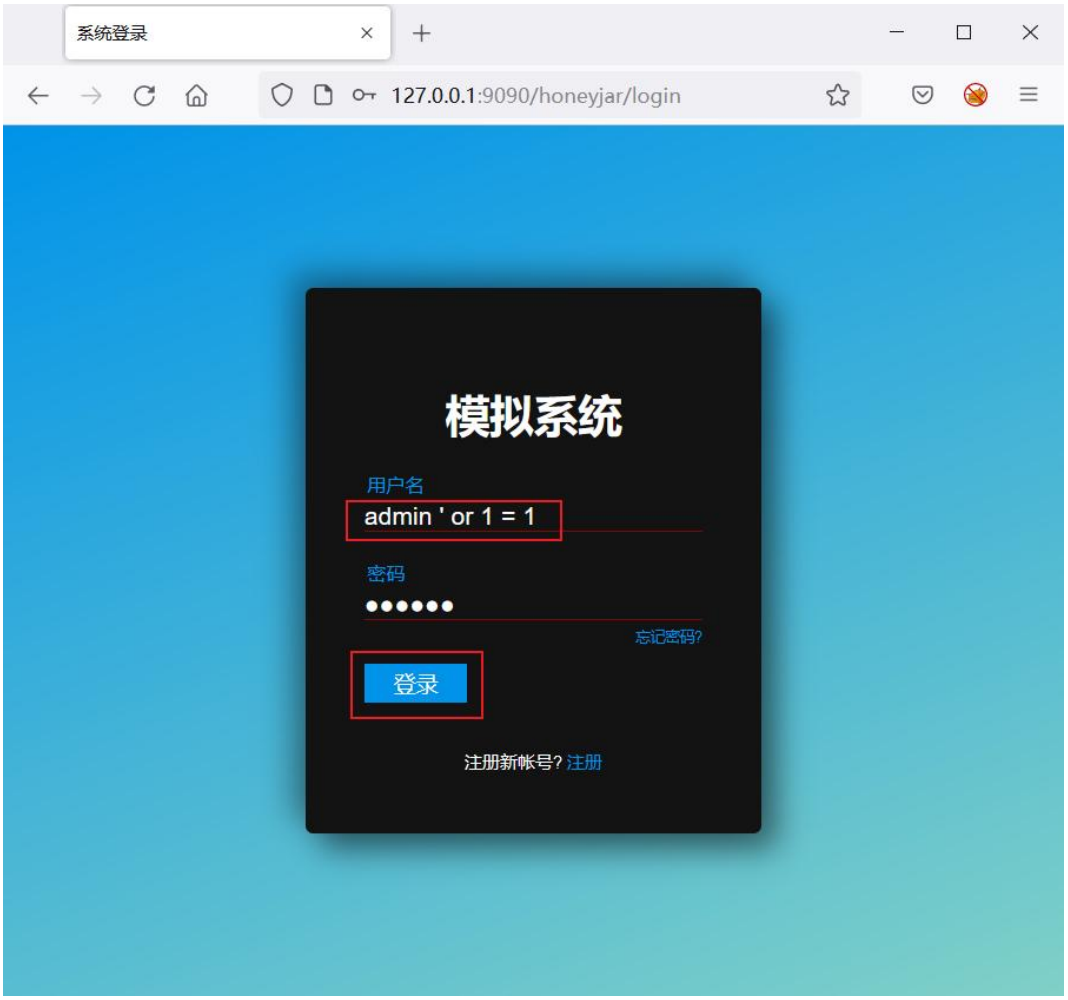


图 1-8 模拟骇客进行 SQL 注入

后台系统检测到 SQL 注入流量,重定向到蜜场,并返回蜜信账号，如图 1-9 所示。

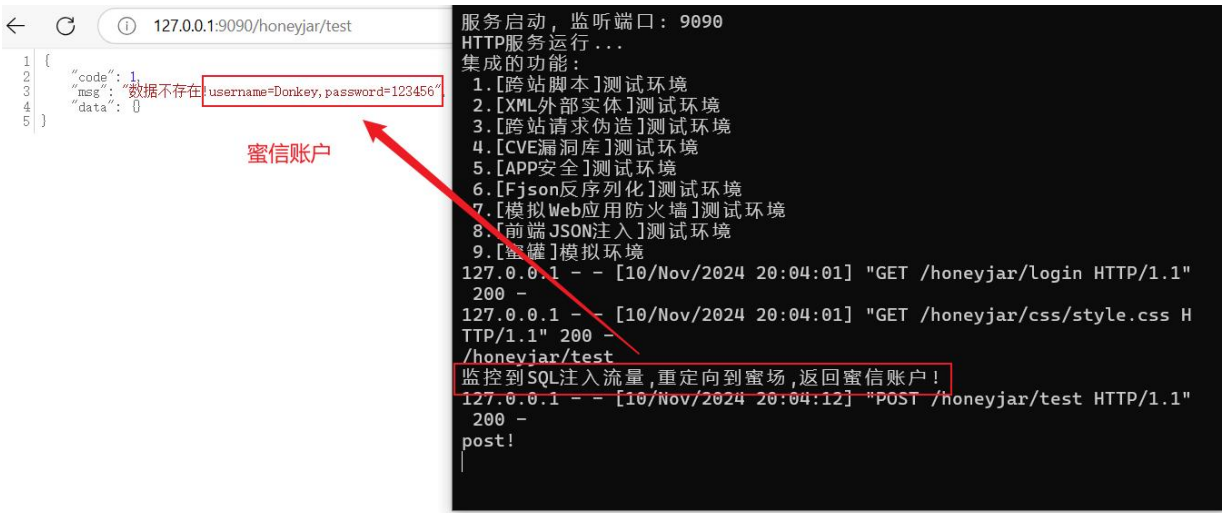


图 1-9 返回蜜信账号

骇客以为攻击成功，使用得到的蜜信账户登录系统，如图 1-10 所示。

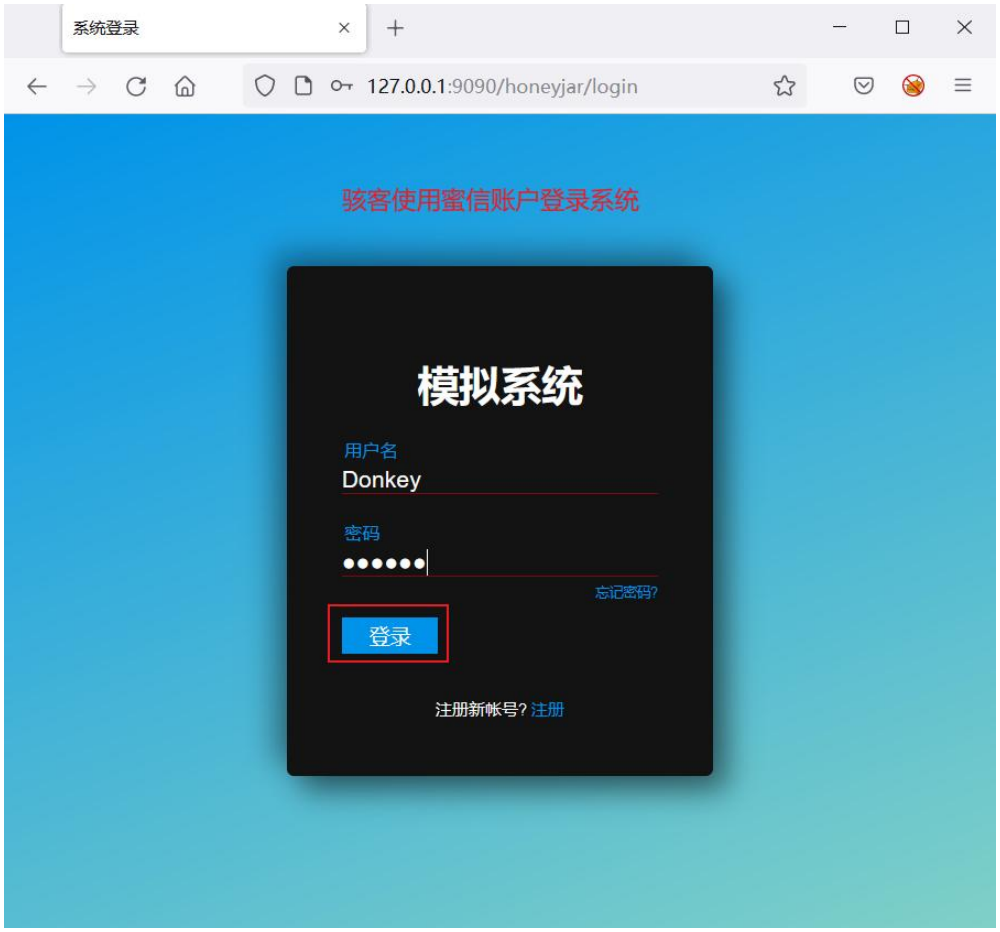


图 1-10 蜜信账户登录系统

后台系统检测到蜜信账户,重定向到蜜场,返回替身页面,并投放含有回溯代码的文件诱饵，如图 1-11 所示。

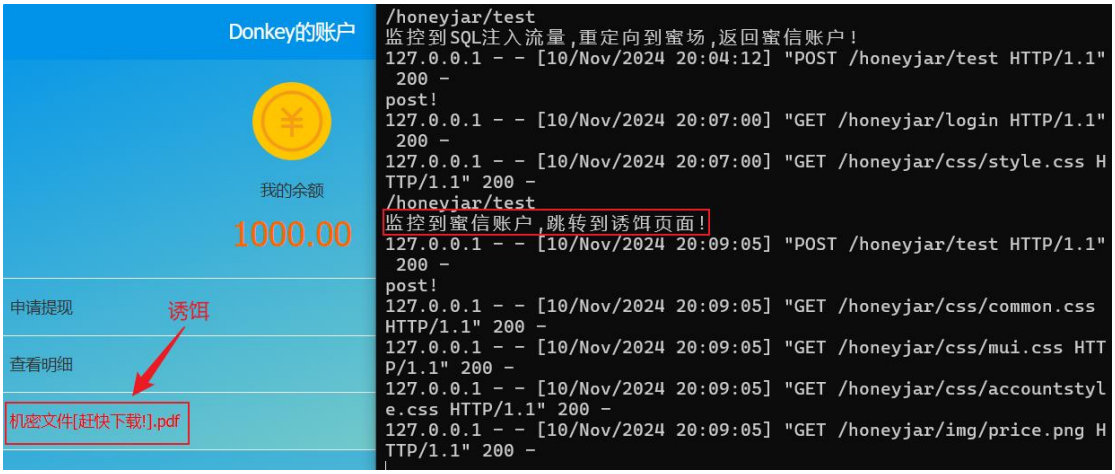


图 1-11 投放文件诱饵

到了这一步，如果骇客下载了文件诱饵，并且打开，那就会得到一个非常深刻的教训。

本节内容中所使用的安全测试工具，在 51testing 测试圈中有详细的安装和使用教程。

有兴趣的读者可以访问：http://quan.51testing.com/pcQuan/lecture/117

四、总结

以上就是笔者从一个测试人员的角度出发，分析骇客的行为，接着再总结被动防御和主动诱捕策略，这在执行安全测试或者渗透测试的时候可以更灵活及有针对性的设计测试用例，并在相关的系统安全会议上作为测试人员提出更有针对性的问题，而且提出的问题也有可靠的依据。

拓展学习

[4] 【安全测试学习交流群】学习交流

咨询：微信 atstudy-js 备注：学习群

测试工程师利用 AI 搭建智能体的实例

◆ 作者：张亚洲

一、前言

近两年 AI 快速发展，给各行各业带了了机遇和挑战，软件测试行业也面临着同样的问题，如何快速拥抱 AI，不被 AI 所淘汰，是所以测试工程师要面对的问题。所以我们需要了解和学习 AI 在工作中应用，提高工作效率。

二、目前测试在 AI 方面的应用

据我所了解，目前在测试领域 AI 可以落地的有四个方面：

1、单元测试

通过现有的代码，在 IDE 里面使用 copilot、?comate 等 AI 辅助编码插件形式生成单元测试用例。可以快速创建用例，方便快捷，提高工作效率。

2、测试用例

给大模型输入需求和输出的用例格式要求，生成所需格式的测试用例。也可以扩展接入用例系统自动添加到用例库。目前对小需求还可以很好的生成用例，大的需求和很多版本后的迭代需求，目前还是有很多问题。

3、接口用例

给接口的请求方法，URL，参数和类型取值等，再给一条可以调通的接口参数进行用例扩展，目前是可以的。可以把生成的接口用例和数据接入接口测试平台中。

4、测试小助手

如果把测试人员日常工作中的事情，放到一个智能体中，比如可以方便快捷的部署服务，打包，创建测试数据等。这样可以节省很多测试人员的时间。

可以想象一个场景，以后只需要和小助手聊天就可以部署服务，打包，日常工作任务的处理。是不是很科幻呢，可以告诉你，现在就可以实现了。下面是搭建智能体的实例。

三、使用工具 Dify

Dify 是一款开源的大语言模型(LLM) 应用开发平台。它融合了后端即服务（Backend as Service）和 LLMOps 的理念，使开发者可以快速搭建生产级的生成式 AI 应用。即使你是非技术人员，也能参与到 AI 应用的定义和数据运营过程中。

由于 Dify 内置了构建 LLM 应用所需的关键技术栈，包括对数百个模型的支持、直观的 Prompt 编排界面、高质量的 RAG 引擎、稳健的 Agent 框架、灵活的流程编排，并同时提供了一套易用的界面和 API。这为开发者节省了许多重复造轮子的时间，使其可以专注在创新和业务需求上。

为什么使用 Dify？

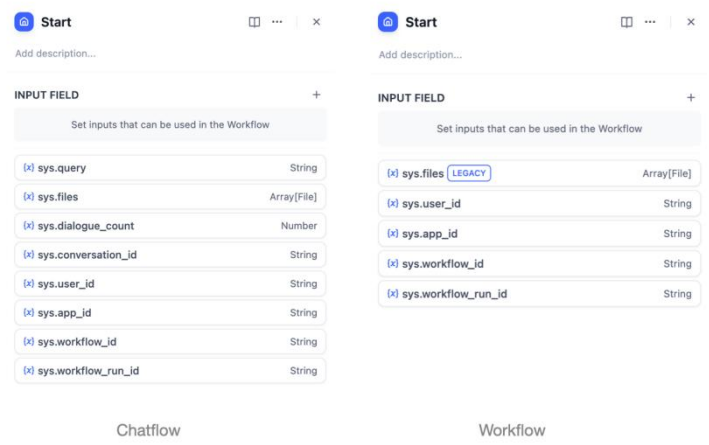
你或许可以把 LangChain 这类的开发库（Library）想象为有着锤子、钉子的工具箱。与之相比，Dify 提供了更接近生产需要的完整方案，Dify 好比是一套脚手架，并且经过了精良的工程设计和软件测试。

需要借助 Dify 的 chat-workflow 工作流编排，实现测试智能体。

1. 部署 Dify 很简单，通过 docker 部署只需要一个命令，可以通过官方文档找到部署的教程。

2. chat-workflow 工作流常用的组件

开始："开始" 节点是每个工作流应用（Chatflow / Workflow）必备的预设节点，为后续工作流节点以及应用的正常流转提供必要的初始信息，例如应用使用者所输入的内容、以及上传的文件等。



核心就是 sys.query 这个用户输入变量，后面经常使用。

问题分类器:通过定义分类描述，问题分类器能够根据用户输入，使用 LLM 推理与之相匹配的分类并输出分类结果，向下游节点提供更加精确的信息。

常见的使用情景包括客服对话意图分类、产品评价分类、邮件批量分类等。

在一个典型的产品客服问答场景中，问题分类器可以作为知识库检索的前置步骤，对用户输入问题意图进行分类处理，分类后导向下游不同的知识库查询相关的内容，以精确回复用户的问题。

http 请求:允许通过 HTTP 协议发送服务器请求，适用于获取外部数据、webhook、生成图片、下载文件等情景。它让你能够向指定的网络地址发送定制化的 HTTP 请求，实现与各种外部服务的互联互通。

该节点支持常见的 HTTP 请求方法：

GET，用于请求服务器发送某个资源。

POST，用于向服务器提交数据，通常用于提交表单或上传文件。

HEAD，类似于 GET 请求，但服务器不返回请求的资源主体，只返回响应头。

PATCH，用于在请求-响应链上的每个节点获取传输路径。

PUT，用于向服务器上传资源，通常用于更新已存在的资源或创建新的资源。

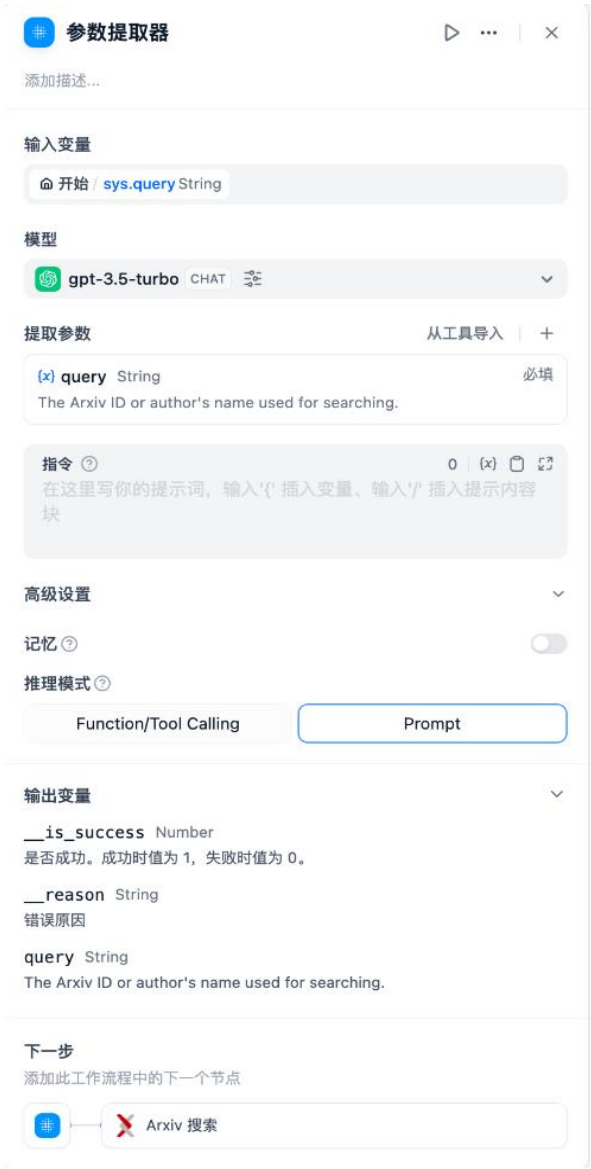
DELETE，用于请求服务器删除指定的资源。

参数提取器

利用 LLM 从自然语言推理并提取结构化参数，用于后置的工具调用或 HTTP 请求。

Dify 工作流内提供了丰富的工具选择，其中大多数工具的输入为结构化参数，参数提取器可以将用户的自然语言转换为工具可识别的参数，方便工具调用。

工作流内的部分节点有特定的数据格式传入要求，如迭代节点的输入要求为数组格式，参数提取器可以方便的实现结构化参数的转换。



LLM

调用大语言模型的能力，处理用户在 “开始” 节点中输入的信息（自然语言、上传的文件或图片），给出有效的回应信息。

应用场景

LLM 节点是 Chatflow/Workflow 的核心节点。该节点能够利用大语言模型的对话/生成/分类/处理等能力，根据给定的提示词处理广泛的任务类型，并能够在工作流的不同环节使用。

意图识别，在客服对话情景中，对用户问题进行意图识别和分类，导向下游不同的

流程。

文本生成，在文章生成情景中，作为内容生成的节点，根据主题、关键词生成符合的文本内容。

内容分类，在邮件批处理情景中，对邮件的类型进行自动化分类，如咨询/投诉/垃圾邮件。

文本转换，在文本翻译情景中，将用户提供的文本内容翻译成指定语言。

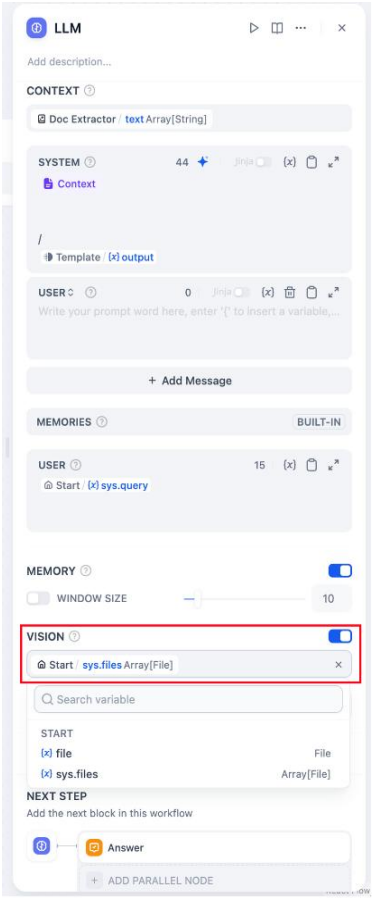
代码生成，在辅助编程情景中，根据用户的要求生成指定的业务代码，编写测试用例。

RAG，在知识库问答情景中，将检索到的相关知识和用户问题重新组织回复问题。

图片理解，使用具备 vision 能力的 LLM，理解与问答图像内的信息。

文件分析，在文件处理场景中，使用 LLM 识别并分析文件包含的信息。

选择合适的模型，编写提示词，你可以在 Chatflow/Workflow 中构建出强大、可靠的解决方案。



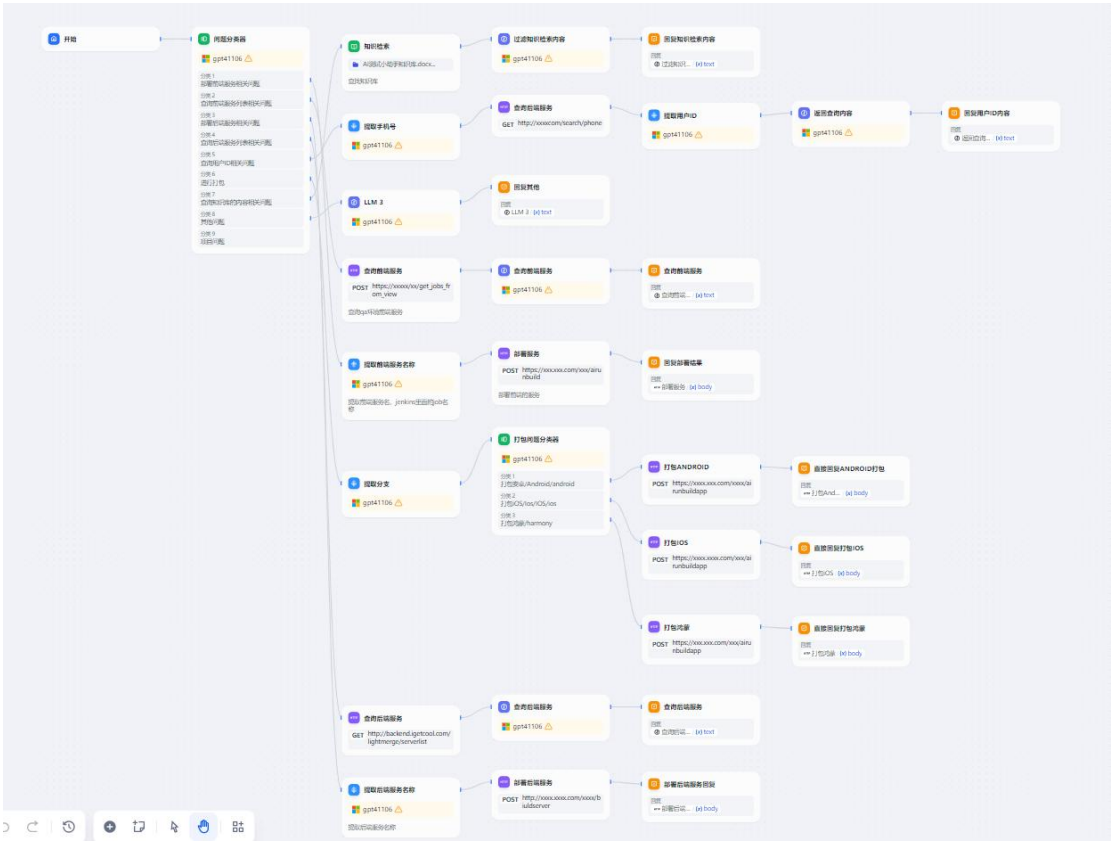
四、实例

搭建一个智能测试小助手,帮忙部署和打包，要实现的功能，部署前端服务。

实现这么多功能需要一个问题分类器，把用户输入的问题就行分类，实现自己的功能。下面是一个问题分类器实例，可以参考一下。

查询 uid

通过上面分类器可以发现，主要是前端，后端服务查询和部署，打包，查询用户 ID，查询知识库等。通过分类器后会把任务分配给后续的操作节点。



上面是一个完整的测试智能体工作流编排，通过编排可以实现打包，部署，创建测试数据或者日常查询知识库的任务。

举个例子:用户输入 部署后端 user 服务，智能体就可以帮你自动部署 user 服务了。

目前还存在一些问题，需要调节提示词实现稳定的功能。相信以后会越来越智能。

拓展学习

[5] 【AI 测试学习交流群&软件测试专用提示词包】

咨询：微信 atstudy-js 备注：AI

Android 平台上开源测试框架总结

◆作者： Erich

一、Android 平台主流开源框架简介

在 Android 平台上，有多个开源且好用的自动化测试框架。以下是几个被广泛使用和认可的框架：

1.1 Appium[https://appium.io/docs/en/latest/intro/]

Appium 是一个跨平台的移动测试工具，支持 iOS 和 Android 上的原生、混合及移动 Web 应用。

它使用了供应商提供的 UI 框架（如 iOS 中的 XCTest 和 Android 中的 UIAutomator 或 Instrumentation）来实施测试，并将这些框架封装到 Selenium WebDriver 中，允许开发者使用多种编程语言编写测试脚本。

优点包括不需要重新编译应用程序或安装额外的东西到设备上，拥有活跃的社区支持。

1.2 UI Automator[https://developer.android.com/training/testing/ui?automator?spm=5176.28103460.0.0.1c775d274LGU9M]:

这是另一个由 Google 提供的 Android UI 测试框架，它允许进行黑盒测试，即不需要了解应用内部结构。

支持跨应用功能测试，例如从一个应用跳转到另一个应用。

缺点是不支持 WebView 元素，所以对于混合应用的支持有限。

1.3 Espresso[https://developer.android.com/training/testing/espresso?spm=5176.28103460.0.0.1c775d274LGU9M]:

Espresso 是由 Google 开发的一个轻量级的 Android UI 测试框架，适合于白盒测试。

它与被测应用运行在同一进程中，因此可以访问应用内部状态，使得测试更加可靠快速。

主要用于单个应用内的交互测试，不直接支持跨应用测试，但可以通过联合测试解决这一限制。

1.4 Robotium[]:

Robotium 是一款成熟的 Android UI 测试框架，能够执行灰盒测试。

支持对本地和混合应用进行全面的测试，包括活动、按钮、菜单等 UI 组件以及手势操作。

测试可以在真实设备或模拟器上运行，并且可以作为持续集成的一部分，官网已经搜不到了，不确定是否还有人维护。

1.5 Selendroid[http://selendroid.io/?spm=5176.28103460.0.0.1c775d274LGU9M]:

Selendroid 是一个基于 Instrumentation 的框架，完全兼容 WebDriver 协议。

它可以在模拟器和实际设备上使用，并且可以集成网格节点以扩展测试规模。不过，随着 Appium 的发展，Selendroid 逐渐式微。

二、Appium 简介

2.1 发展历史

Appium 的发展历史可以追溯到 2012 年，它从一个简单的想法逐渐成长为最受欢迎的开源跨平台移动自动化测试框架之一。以下是 Appium 发展的关键里程碑：

- 早期灵感与创建

2011 年：Dan Cuellar 在担任 Zoosk 的测试经理时遇到了 iOS 应用程序测试的问题，意识到需要一种更好的方法来自动化测试过程。

2012 年：Dan 在 Selenium 大会上展示了他的项目，并进行了闪电演讲。他的演示引起了广泛关注，尤其是 Jason Huggins（Selenium 的共同创始人）的兴趣。

• 开源与社区支持

2012 年 8 月：Dan 在他的 GitHub 上发布了 C#版本的源代码，并上传了基于 Python 实现的新版本。

2012 年 9 月：Jason 添加了一个 Web 服务器，并开始通过 HTTP 实现 WebDriver 有线协议，使 iOSAuto 可以从任何语言的任何 Selenium WebDriver 客户端库编写脚本。

2013 年 1 月：Sauce Labs 决定全力支持 Appium，并提供更多的开发能力。团队选择使用 Node.js 作为框架，为 Appium 的基本架构奠定了基础。

• 成长与成熟

2014 年 5 月：发布了 Appium 1.0，这是 Appium 发展的一个重要里程碑。此时，Appium 已经成为最受欢迎的开源跨平台移动自动化框架。

2016 年末：为了保持 Appium 的开源状态，Sauce Labs 将 Appium 作为一个项目捐赠给了 JS Foundation（后来合并到了 OpenJS Foundation），确保其在社区中长期地发展。

• 生态系统扩展

2013 年初至 2016 年：期间，Appium 增加了对 Android 和 Selendroid 的支持，成为第一个真正的跨平台自动化框架。项目继续吸引用户和贡献者，功能不断丰富，错误得到修复，稳定性也得到了改善。

• Appium 2.0 发布

2023 年：Appium 2 发布，重点放在将 Appium 作为一个生态系统而不是单一项目上。任何人都可以开发和共享驱动程序和插件，这为实现 iOS 和 Android 之外的平台自动化开发开辟了无限的可能性。

• 持续发展

自成立以来，Appium 一直由不同的个人和组织指导，并且曾经用三种不同的编程语

言实现过。随着 Appium 加入 OpenJS Foundation，它持续获得来自全球开发者社区的支持，不断发展以满足新的需求和技术趋势。

Appium 的历史反映了它从一个小型实验性项目成长为一个强大的、广泛采用的工具的过程，同时也体现了开放源码的力量以及社区合作的重要性。今天，Appium 仍然是移动应用自动化测试领域的重要参与者，支持着数以千计的应用程序的开发和测试工作。

2.2 优点与缺点

优点

1. 跨平台支持：

Appium 支持 iOS、Android 和 Windows 应用程序的自动化测试，这意味着你可以用同一套 API 来编写针对不同平台的测试脚本。

2. 无需重新编译应用：

测试人员不需要访问应用程序的源代码或对应用程序进行任何修改就可以直接进行自动化测试，这使得它非常适合用于黑盒测试。

3. 使用 WebDriver 协议：

Appium 基于 W3C WebDriver 标准构建，允许用户利用现有的 Selenium WebDriver 客户端库来编写测试脚本，从而简化了学习曲线，并且可以与现有的 CI/CD 工具链无缝集成。

4. 丰富的社区支持：

由于其开源性质，Appium 拥有一个活跃且庞大的开发者社区，提供了大量的插件、扩展和其他资源，帮助解决常见问题并加速开发过程。

5. 灵活性和可扩展性：

支持多种编程语言（如 Java, Python, Ruby, JavaScript 等），并且可以通过自定义驱动程序和插件来适应特定需求。

6. 强大的 UI 元素识别能力：

利用内置的定位策略（例如通过 XPath、ID、Accessibility ID 等）来识别屏幕上的

UI 组件，提供稳定的元素查找机制。

7. 良好的文档和支持：

官方网站提供了详尽的文档和指南，包括安装说明、示例代码以及最佳实践建议。

缺点

1. 性能问题：

在某些情况下，尤其是在复杂的应用场景中，Appium 可能会表现出比原生工具更慢的速度，这是因为它依赖于多个层之间的通信（比如从客户端到服务器再到设备）。

2. 初期设置较为复杂：

尽管有详细的文档，但初次配置环境（特别是对于初学者）可能需要花费一定的时间去理解和解决问题，例如配置 Appium Server、正确安装依赖项等。

3. 对 WebView 的支持有限：

虽然 Appium 已经增强了对 WebView 内容的操作能力，但在处理混合模式（native+web）应用时仍然可能存在挑战，特别是在确保稳定性和速度方面。

4. 依赖外部服务和工具：

Appium 本身并不包含所有必要的功能；它经常需要与其他工具和服务一起工作，例如模拟器/真机管理工具、报告生成工具等，这增加了整体架构的复杂度。

5. 版本兼容性问题：

随着操作系统版本更新和技术栈的发展，可能会遇到新旧版本间的兼容性问题，导致测试脚本需要定期维护以保持最新状态。

6. 不稳定的行为：

在某些情况下，Appium 可能会出现不稳定的行为，比如偶发性的崩溃或无法可靠地执行某些命令，这些问题通常与底层平台的变化有关。

总的来说，Appium 是一个强大而灵活的工具，适合那些希望在一个平台上同时测试 iOS 和 Android 应用程序的企业和个人开发者。然而，了解它的优势和潜在限制可以帮助你做出更好的决策，并为可能出现的问题做好准备。

2.3 核心概念

Appium 作为一款强大的移动应用自动化测试框架，有几个核心概念对于理解和使用它是至关重要的。以下是 Appium 的一些关键概念：

1. WebDriver 协议

Appium 基于 W3C WebDriver 标准构建，这是一个浏览器自动化接口的标准。它允许开发者通过 HTTP 请求来控制浏览器或移动应用程序的行为。WebDriver API 提供了丰富的命令集，用于模拟用户与应用的交互，如点击、输入文本等。

2. Appium Server

Appium Server 是 Appium 的核心组件之一，它充当客户端（测试脚本）和移动设备/模拟器之间的中间件。服务器接收来自客户端的命令，然后将这些命令转换为适合目标平台的本地驱动程序（例如 XCUITest Driver for iOS, UiAutomator2 Driver for Android）可以理解的指令，并执行相应的操作。

3. 客户端库（Client Libraries）

为了方便地编写测试脚本，Appium 支持多种编程语言的客户端库，如 Java、Python、Ruby、JavaScript (Node.js) 等。这些库实现了 WebDriver 协议，让开发者可以用自己熟悉的语言编写自动化测试。

4. Desired Capabilities

Desired Capabilities 是一组键值对参数，用来配置测试会话的属性。它们告诉 Appium Server 关于要启动的应用程序的信息，例如平台名称（iOS 或 Android）、设备信息、应用程序路径等。这些设置在创建一个新的测试会话时提供给 Appium Server。

5. 驱动程序（Drivers）

驱动程序是负责与特定操作系统进行通信的部分。对于不同的平台（如 iOS、Android），Appium 使用不同的驱动程序来执行自动化任务。例如，UiAutomator2 用于 Android，而 XCUITest 则用于 iOS。每个驱动程序都有自己的特点和能力，以适应各自平台的独特需求。

6. 元素定位（Element Locators）

元素定位是指如何在应用程序界面上找到 UI 组件。Appium 提供了多种策略来定位

元素，包括但不限于 ID、XPath、Accessibility ID、类名等。选择合适的定位策略对于确保测试的稳定性和可靠性至关重要。

7. 插件系统（Plugin System）

从 Appium 2.0 开始引入了插件系统，使得扩展 Appium 功能变得更加容易。插件可以添加新的命令、改变现有行为或者集成第三方服务。这为定制化和增强 Appium 的能力提供了极大的灵活性。

8. 平台独立性（Platform Independence）

尽管 Appium 本身需要根据平台的不同使用不同的驱动程序，但它设计成尽可能保持跨平台的一致性。这意味着你可以在不同平台上使用相似的 API 调用，减少了学习成本并提高了代码复用率。

9. 混合应用支持（Hybrid Apps Support）

对于混合应用（即包含原生和 Web 内容的应用），Appium 能够切换上下文，在原生视图和 WebView 之间无缝切换，以便测试不同部分的功能。

理解上述核心概念有助于更好地掌握 Appium 的工作原理及其使用方法，从而更有效地进行移动应用的自动化测试。

三、UI Automator 简介

3.1 发展历史

UI Automator 是 Google 为 Android 平台开发的自动化测试框架之一，首次发布于 2012 年。它的设计目的是为了简化 Android 应用的 UI 测试流程，并允许开发者和测试人员在无需了解应用程序内部结构的情况下对应用进行黑盒测试。随着 Android 系统的不断发展，UI Automator 也经历了多次迭代，以适应新的 API 级别和功能需求。尽管如此，在 2021 年 Google I/O 大会上，Google 宣布了 UI Automator 将逐渐被 Espresso 取代作为官方推荐的 UI 测试工具，但 UI Automator 仍然可以用于跨应用测试等特定场景。

3.2 优点和缺点

优点

1. 黑盒测试：UI Automator 不依赖于应用程序的源代码或内部实现细节，这使得它非常适合用来测试第三方应用或者当您不想暴露内部逻辑时。

2. 跨应用测试：与仅限于单个应用内的 Espresso 不同，UI Automator 可以跨越多个应用边界执行操作，这对于测试涉及多个应用之间的交互的用例非常有用。

3. 稳定性：由于直接运行在设备上并使用系统级服务，UI Automator 提供的测试通常比较稳定，不容易受到应用重启等因素的影响。

4. 类型：UI Automator 能够识别并操作各种标准的 Android UI 组件，如按钮、文本框、列表等。

缺点

1. 性能问题：相较于 Espresso，UI Automator 可能会稍微慢一些，尤其是在复杂的布局中查找视图元素时。

2. WebView 限制：对于包含 WebView 的应用，UI Automator 不能直接与网页内容交互，需要额外的方法来处理这类情况。

3. 更新频率较低：因为不再是最主要的官方推荐测试工具，所以 UI Automator 的更新和支持可能不如以前频繁。

3.3 常用方法

UI Automator 提供了丰富的 API 来帮助用户创建强大的自动化测试脚本。以下是一些常用的 API 方法：

UiDevice: 提供了对整个设备状态的操作接口，例如点击屏幕上的任意位置、按返回键、获取当前窗口信息等。

```
UiDevice.getInstance().pressHome(); // 按下 Home 键
```

UiObject: 表示一个具体的 UI 元素，可以通过属性（如 resource-id, text）来定位，并且可以对其执行一系列动作（如点击、长按、输入文本等）。

```
UiObject button = new UiObject(new UiSelector().text("Submit"));
button.click();
```

UiCollection: 用于遍历和操作一组相似的 UI 元素，比如列表项。

```
UiCollection list = new UiCollection(new
UiSelector().className("android.widget.ListView"));
UiObject item = list.getChildByText(new
UiSelector().className("android.widget.TextView"), "Item Name");
```

UiScrollable: 当 UI 元素不在屏幕可见范围内时，可以使用此对象来滚动视图直到目标元素出现。

```
UiScrollable scrollable = new UiScrollable(new
UiSelector().scrollable(true));
UiObject element = scrollable.getChildByText(new
UiSelector().className("android.widget.TextView"), "Desired Text");
element.click();
```

UiSelector: 是一个构建器类，用于定义要查找的 UI 元素的选择条件，可以组合多个属性来精确定位。

```
UiObject checkBox = new UiObject(new
UiSelector().resourceId("com.example:id/checkbox").checked(false));
checkBox.click();
```

通过这些 API，大家可以构建出不同场景下的测试用例，来保证我们应用程序的稳定性。

四、UI Automator2 简介

4.1 发展历史

UIAutomator2 是基于原始 UI Automator 框架的升级版本，旨在解决原生 UI Automator 的一些局限性，并提供更强大的功能和更好的兼容性。它最初是为了满足 Appium 的需求而开发的，作为其 Android 自动化测试引擎的一部分。随着需求的增长和技术的进步，UIAutomator2 逐渐成为一个独立且重要的工具，特别是在需要与现代 Android 应用和特性（如 WebView 支持）进行交互时。

4.2 优点和缺点

优点

1. 增强的功能：相比传统的 UI Automator，UIAutomator2 引入了更多新特性，例如对 WebView 的支持、更高效的元素查找算法以及更稳定的长按操作等。

2. 更好的性能：通过优化内部实现，UIAutomator2 在执行速度上有显著提升，特别是在处理复杂 UI 结构时。

3. 持续更新和支持：由于它是 Appium 项目的一部分，因此得到了持续的维护和发展，确保与最新的 Android 版本保持同步。

4. 丰富的 API：提供了更加丰富的 API 接口，可以更容易地创建复杂的测试场景，包括但不限于滑动、拖拽、多点触控等高级手势。

5. 社区活跃度高：作为一个开源项目，UIAutomator2 拥有一个活跃的开发者社区，能够快速响应问题并分享最佳实践。

缺点

1. 依赖于特定环境：为了使用 UIAutomator2，必须安装某些特定的环境和配置，这可能对于初学者来说是一个门槛。

2. 学习曲线：尽管 API 相对友好，但对于那些不熟悉 Java 或 Android 开发的人来说，仍然存在一定的学习成本。

3. 部分旧版设备不支持：虽然大多数现代 Android 设备都能很好地运行 UIAutomator2，但在一些较老的设备上可能会遇到兼容性问题。

4.3 常用方法

UIAutomator2 继承了原 UI Automator 的优点，同时增加了许多新的 API 来简化测试过程。以下是一些常用的方法：

启动和管理会话：

UiDevice 类用于表示整个设备的状态和控制，可以通过 UiDevice.getInstance()获取实例。

```
UiDevice device =
UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());
```

查找 UI 元素：

使用 BySelector 构建器来定义选择条件，然后利用 findObject()方法来定位单个或多个 UI 元素。

```
BySelector selector = new UiSelector().text("Submit");
UiObject2 submitButton = device.findObject(selector);
```

操作 UI 元素：

对找到的 UI 元素执行各种动作，如点击、长按、输入文本等。

```
深色版本
submitButton.click();
```

滚动和滑动：

支持多种方式的滚动和滑动操作，包括垂直/水平滚动、页面内的滑动等。

```
device.swipe(500, 1500, 500, 500, 30); // 从屏幕底部向顶部滑动
```

等待条件：

提供了灵活的方式等待某个条件成立，比如等待特定 UI 元素出现或消失。

```
//
device.wait(Until.findObject(By.text("Loading")), 5000); // 等待 5 秒直
到文本为"Loading"的元素出现
处理 WebView：
//UIAutomator2 特别增强了对 WebView 的支持，允许直接与网页内容交互。
WebView webView =
device.findObject(By.clazz( WebView.class));
webView.webDriver().findElement(By.tagName("input")).sendKeys("test"
);
```

4.4 总结

UIAutomator 和 UIAutomator2 都是用于 Android 应用程序自动化测试的工具，但它们有一些关

键的区别，并且并不是完全由同一家公司开发和维护的。

UIAutomator

开发者：UIAutomator 是由 Google 开发并作为 Android SDK 的一部分提供的官方工具。

适用范围：主要用于白盒测试，适用于跨应用的 UI 自动化测试。

功能特点：

1. 支持基本的 UI 元素查找和操作。

2. 可以模拟用户交互，如点击、输入文本等。

3. 提供了有限的 API 来处理 WebView 中的内容。

局限性：

1. 对于复杂的应用场景支持不足，例如长按操作或更复杂的多点触控手势。

2. 不支持一些现代 Android 特性，比如浮动窗口。

UIAutomator2

开发者：UIAutomator2 最初是由 Appium 团队为了改进 Appium 的 Android 测试能力而开发的。虽然它不是直接由 Google 开发，但它依赖于 Google 提供的底层技术。

适用范围：UIAutomator2 继承并扩展了原始 UIAutomator 的功能，提供了更强大的特性和更好的性能。

功能特点：

1. 增强的功能集，包括对 WebView 的更好支持、更高效的元素查找算法以及更稳定的长按操作。

2. 支持更多的手势操作，如滑动、拖拽、多点触控等。

3. 更好的性能表现，特别是在处理复杂的 UI 结构时。

4. 持续更新和支持，确保与最新的 Android 版本兼容。

5. 社区与支持：由于它是 Appium 项目的一部分，因此拥有一个活跃的开发者社区，能够快速响应问题并分享最佳实践。

6. 维护情况

尽管 UIAutomator2 并不是由 Google 正式发布的工具，但它利用了 Google 提供的基础架构和技术。随着时间的发展，UIAutomator2 已经成为一个广泛接受和使用的工具，特别是在需要与现代 Android 应用和特性进行交互的情况下。与此同时，Google 自己也在不断改进其官方的测试工具链，包括 Espresso 和新的版本的 UIAutomator API。

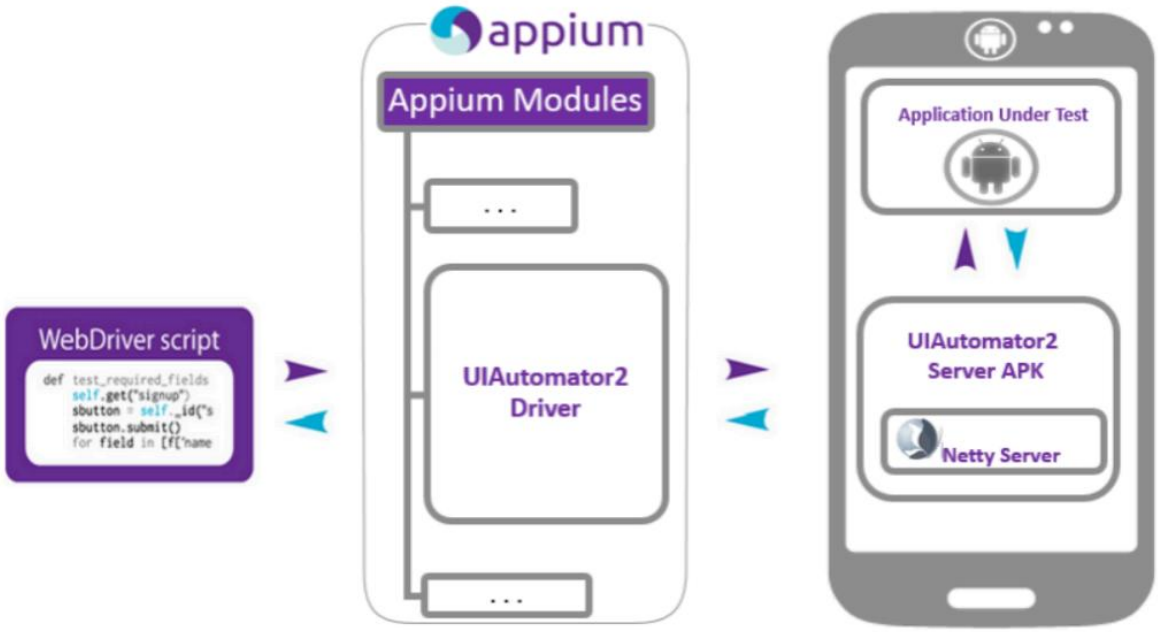
总结来说，虽然两者都服务于相似的目的，但它们在功能、维护和支持方面存在差异。

UIAutomator2 在很多方面是对原始 UIAutomator 的补充和增强，特别适合那些寻求更多功能和更好性能的开发者。

五、最佳实践

其实在日常测试中，单纯的某一个框架很难满足我们的测试需求，这个时候就需要我们把这些工具组合起来使用，博采众长，才能更好的满足我们的需求。一般我们都是

采用 Appium + UIAutomator2 来搭建整体测试框架。



WebDriver Script:

自动化测试脚本，通常使用 Selenium WebDriver API 编写。

示例代码:

```
def test_required_fields(self):
 self.get("signup")
 sbutton = self._id('s')
 sbutton.submit()
 for field in ['f', 'name']:
 ...
```

功能：这些脚本通过 HTTP 请求与 Appium Server 通信，发送命令来控制测试设备上的应用程序。

Appium Modules:

Appium Server 的核心模块，负责接收和处理来自 WebDriver 脚本的命令。

UIAutomator2 Driver:

这是 Appium 用于 Android 平台的驱动程序，它通过 UIAutomator2 与设备上的应用程序进行交互。

其他模块: Appium Server 还包括其他模块，如用于处理不同平台（如 iOS）的驱动程序。

UIAutomator2 Server APK:

这是一个安装在测试设备上的 Android 应用，用于执行自动化测试命令。

功能: UIAutomator2 Server APK 接收来自 Appium Server 的命令，并在设备上执行相应的操作，如点击、输入文本等。

Netty Server:

Netty 是一个高性能的异步事件驱动的网络应用框架，用于简化网络应用的开发。

功能：在 Appium 中，Netty Server 用于处理与 WebDriver 脚本的 HTTP 通信，接收命令并将其转发给 UIAutomator2 Server APK。

Aplication Under Test (AUT):

需要测试的 Android 应用程序。

功能: AUT 接收来自 UIAutomator2 Server APK 的命令，并执行相应的操作。

交互流程

1. WebDriver Script:测试脚本通过 HTTP 请求发送命令到 Appium Server。

2. Appium Server 接收命令，并通过 UIAutomator2 Driver 将命令转发给 UIAutomator2 ServerAPK。

3. UIAutomator2 Server APK 接收命令，并在设备上执行相应的操作。

4. 操作的结果通过 Netty Server 返回给 Appium Server。

5. Netty Server 处理 HTTP 通信，将结果返回给 WebDriver 脚本。

通过这种方式，WebDriver 脚本可以控制设备上的应用程序，实现自动化测试。

-------- END ---------