游戏抓包工具制作（二）—— 设计一个简单界面

原创 阿宽 游戏小测试 2024-04-08 19:07 广东

在上一篇文章，我们已经创建了一个简单的代理脚本，并且可以让客户端通过它来登录游戏了，但是目前来讲，它还是太简单了，并且它在执行过程中是无法进行干预的，如果我们以后有更多的需求，想要随时随地进行一些操作，通过之前的脚本是难以实现的，所以我们接下来要对它进行功能扩展，给它设计一个界面。

从推广的便利性来讲，使用flask搭一个web页面服务让大家通过网页来访问并使用工具，肯定是最优解，但是由于一些限制，flask在这里可能不太适用，做抓包工具，我目前使用的是PyQt5，所以我接下来主要讲解的是如何用PyQt5来绘制一个界面。

关于如何在pycharm里面配置QT Designer，网上有很多教程，对此熟悉的客官可以跳过😳

一、pycharm安装PyQt5插件

1.pip安装PyQt5

打开cmd，输入命令pip install PyQt5

2.pip安装PyQt5-tools

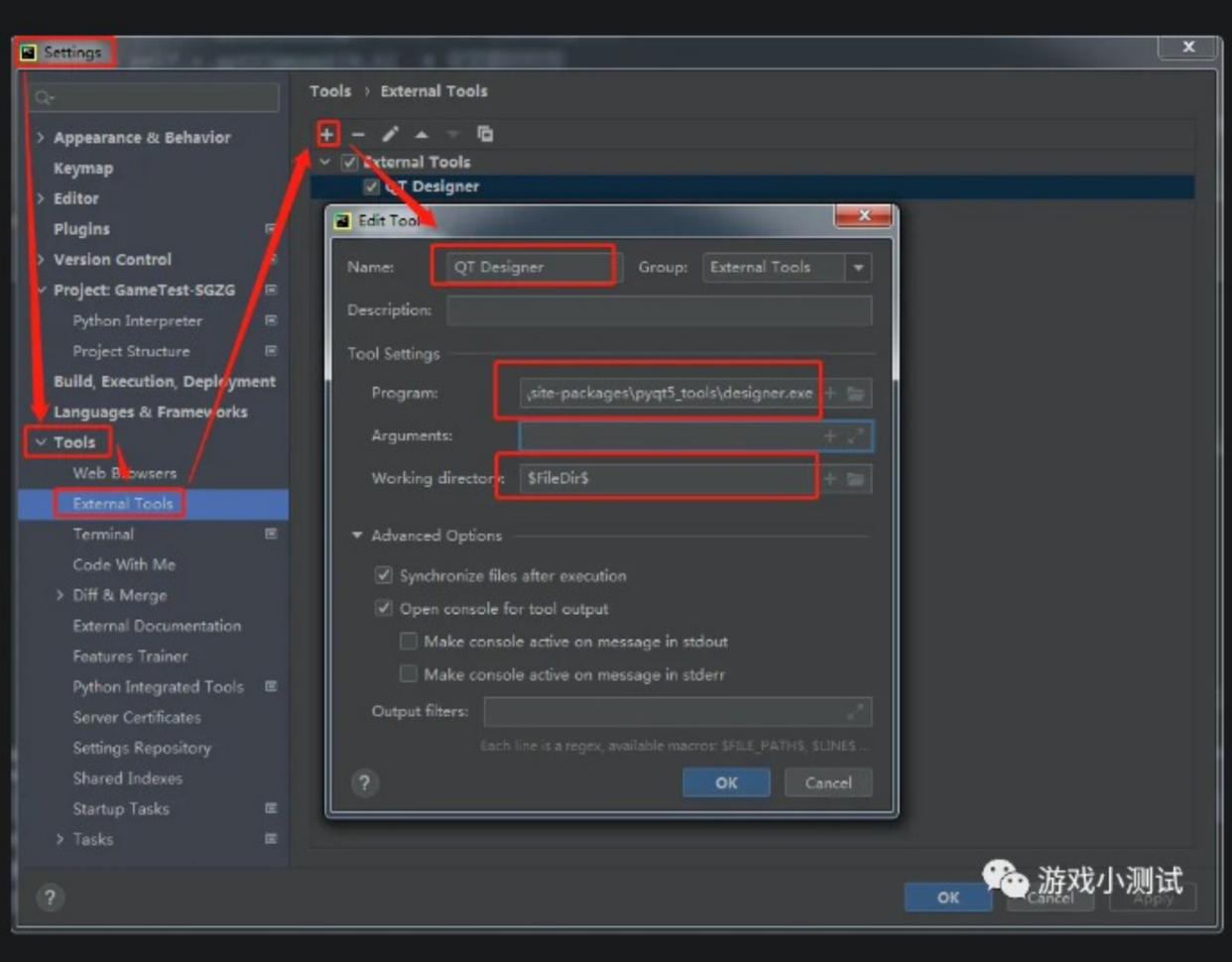
打开cmd，输入命令pip install PyQt5-tools

3.配置pycharm

1）配置QT Designer

QT Designer是一个界面编辑工具，通过QT Designer，我们可以设计一个自己想要的界面，具体使用方法后面讲。

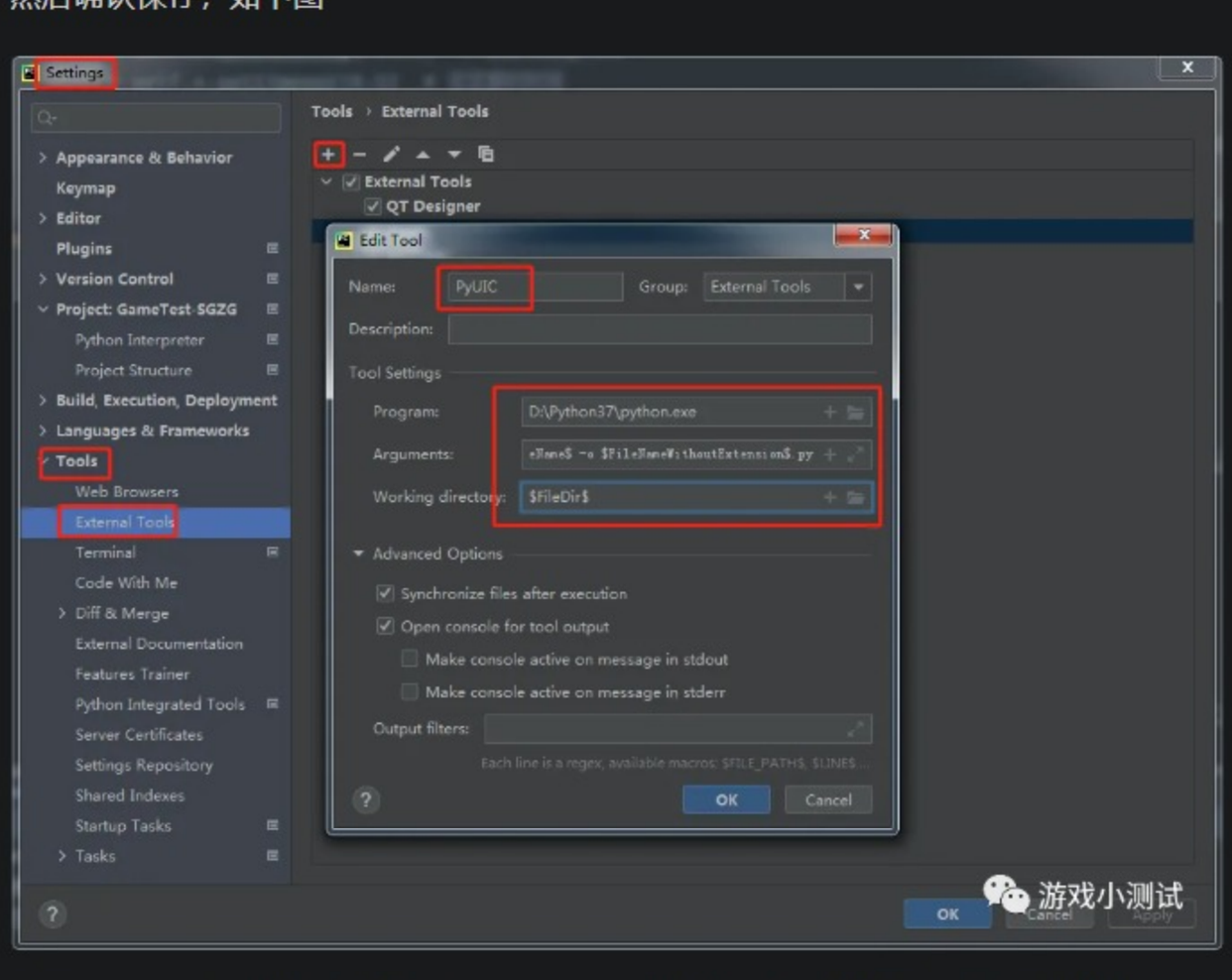
打开pycharm --> Settings --> Tools --> External Tools，然后点击加号（+），Name那里输入 QT Designer，Program那里输入designer.exe的路径（python目录下\Lib\site-packages\pyqt5_tools\designer.exe，新版本可能路径有区别，可以尝试搜索一下），然后在Working director那里输入$FileDir$，然后确认保存，如下图



2）配置PyUIC

PyUIC是一个界面转化工具，通过PyUIC，我们可以将QT Designer设计出来的ui文件，转化成py文件，这样我们的脚本在执行的时候，就会有一个界面了。

跟上一步骤一样，配置PyUIC，在Name那里输入PyUIC，在Program那里输入python.exe的路径，在Arguments那里输入-m PyQt5.uic.pyuic $FileName$ -o $FileNameWithoutExtension$.py，最后在Working director那里输入$FileDir$，然后确认保存，如下图

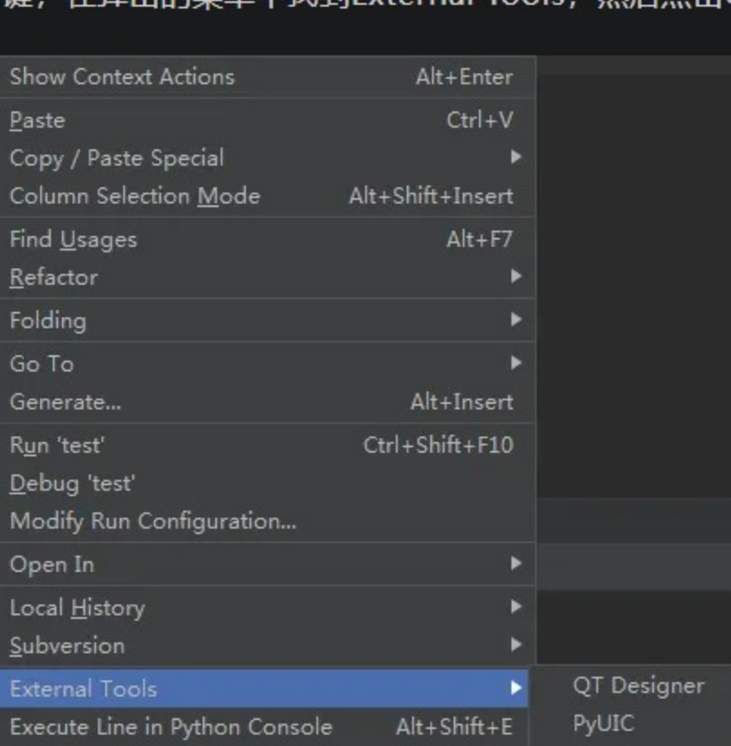


PS：还有一些其他插件，比如Pyrcc，可以将资源文件转化成py文件，这样UI上可以显示美术资源，但是在我们的工具以实用为主，不追求商业化，所以不赘述。

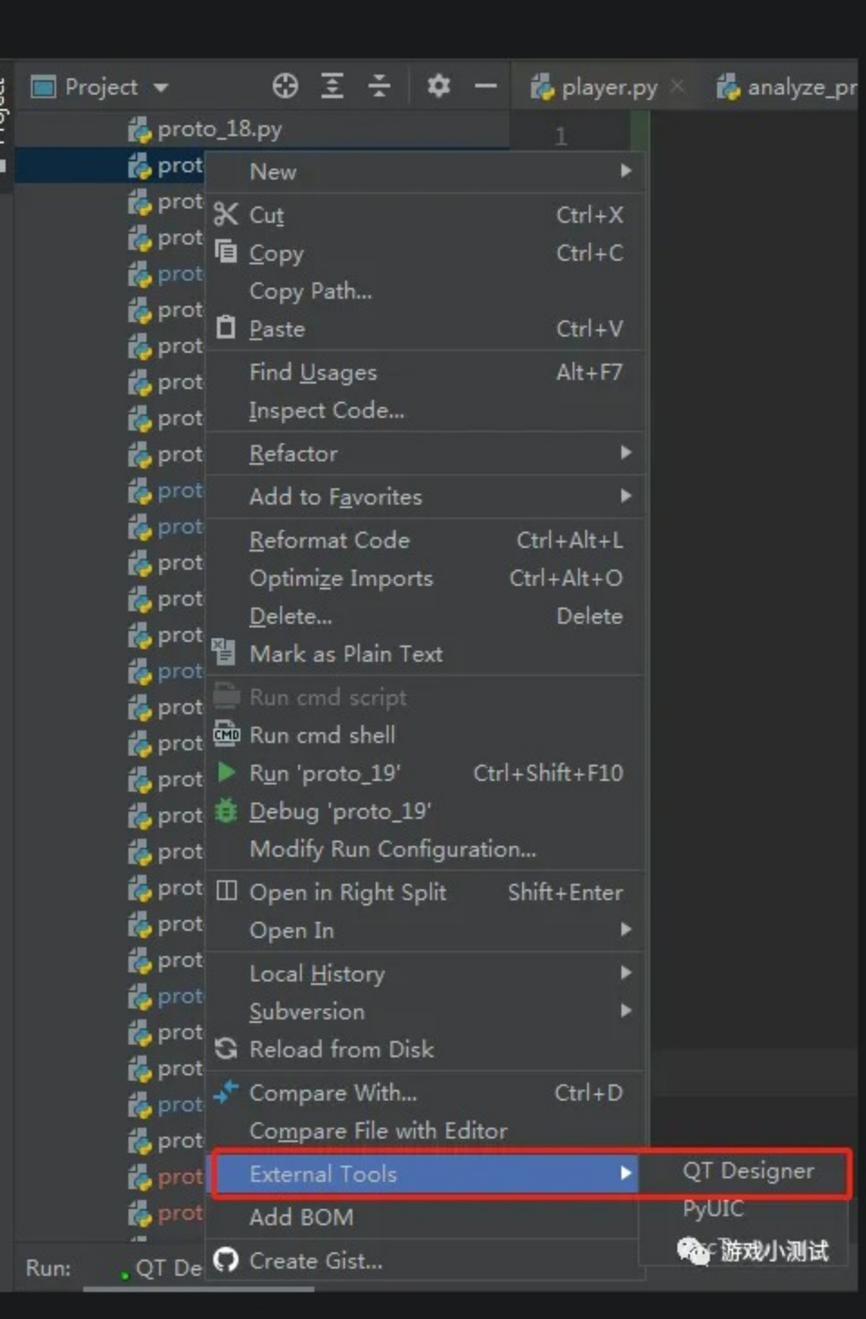
二、QT Designer编辑界面

1.在pycharm中打开QT Designer的三种方式：

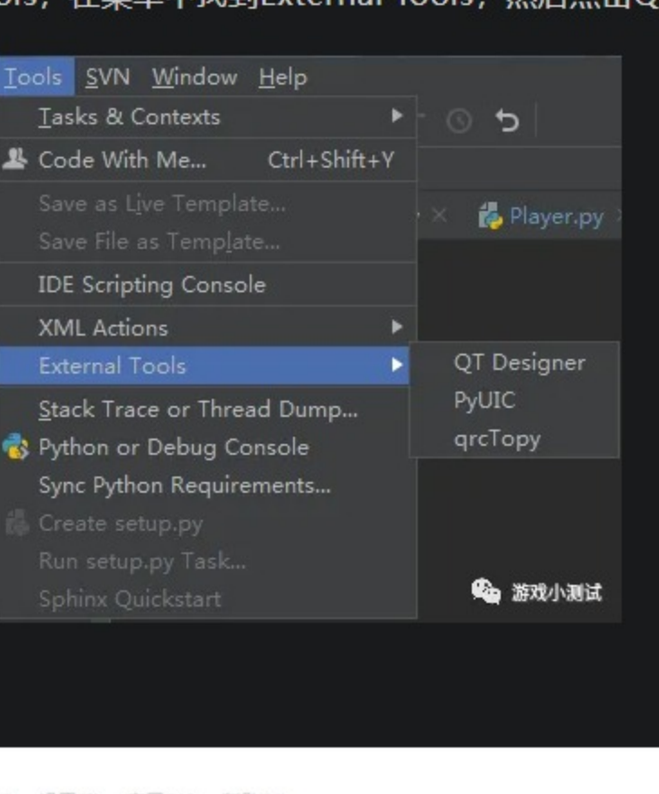
1）脚本编辑栏右键，在弹出的菜单中找到External Tools，然后点击QT Designer



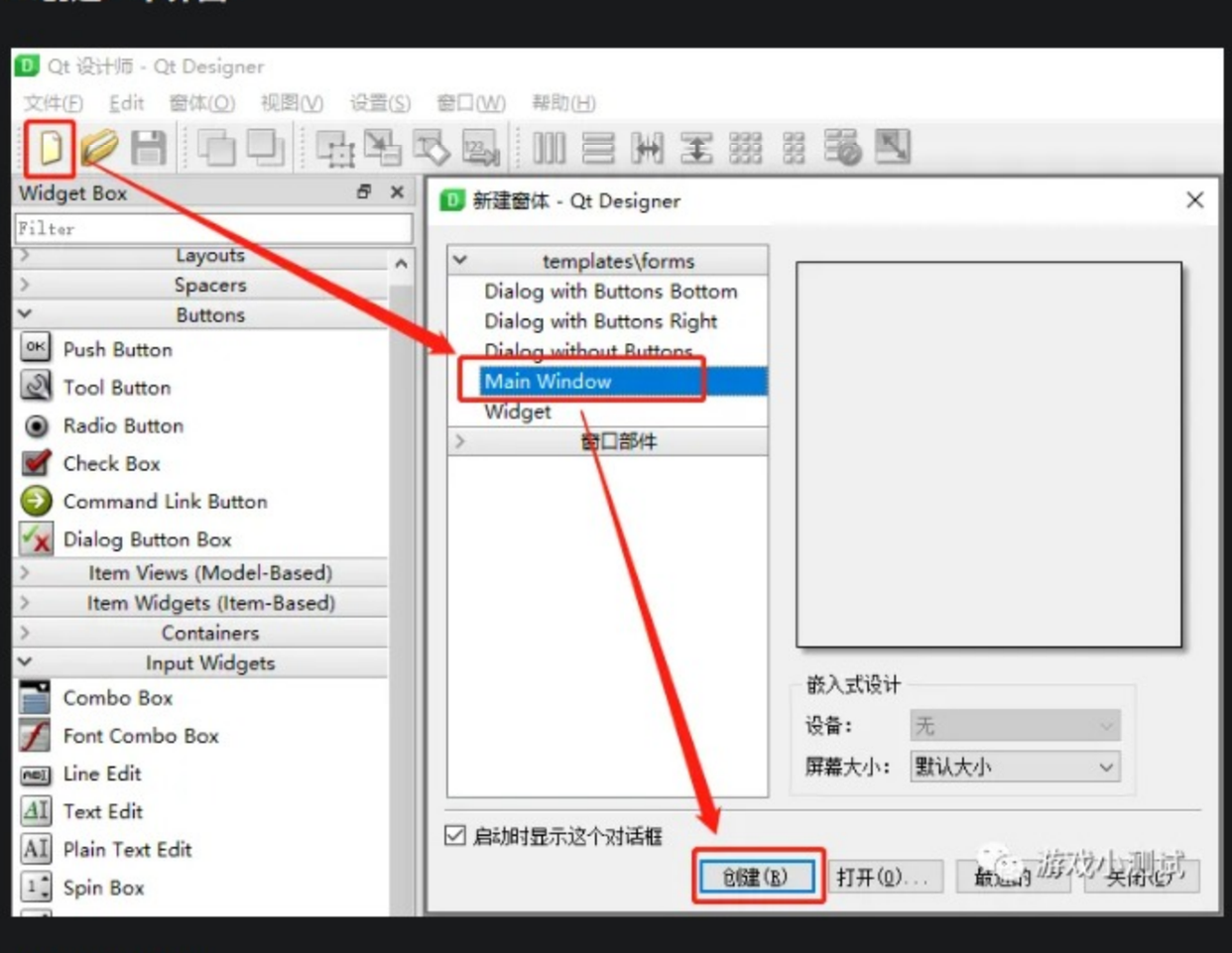
2）右键点击项目的脚本树，在弹出的菜单中找到External Tools，然后点击QT Designer



3）点击菜单栏的Tools，在菜单中找到External Tools，然后点击QT Designer



2.创建一个界面



常用组件说明：

1）Push Button

我们最常用的按钮组件，可以给按钮绑定点击事件，在点击按钮之后执行对应的事件

2)Line Edit

单行文本编辑框，经常用来做一些输入操作，如账号密码输入等

3）Label

标签，用来做一些提示性的文本内容，如账号密码输入框前面的"账号""密码"等文字

4）Text Browser

文本框，不可编辑，用来做一些超长文本信息展示

5）Combo Box

下拉框，用来做一些下拉选择

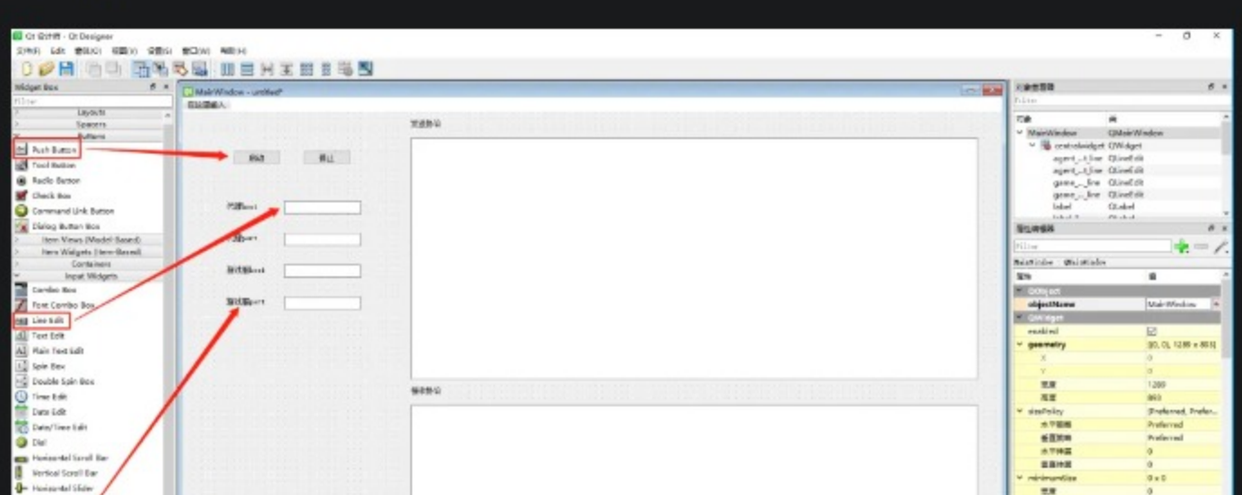
6）Tab Widget

标签分页组件，用来分页展示不同的功能

7）Layout

布局组件，有了布局组件，在缩放工具窗口的时候，会随缩放动态调整和匹配各个组件的显示区域

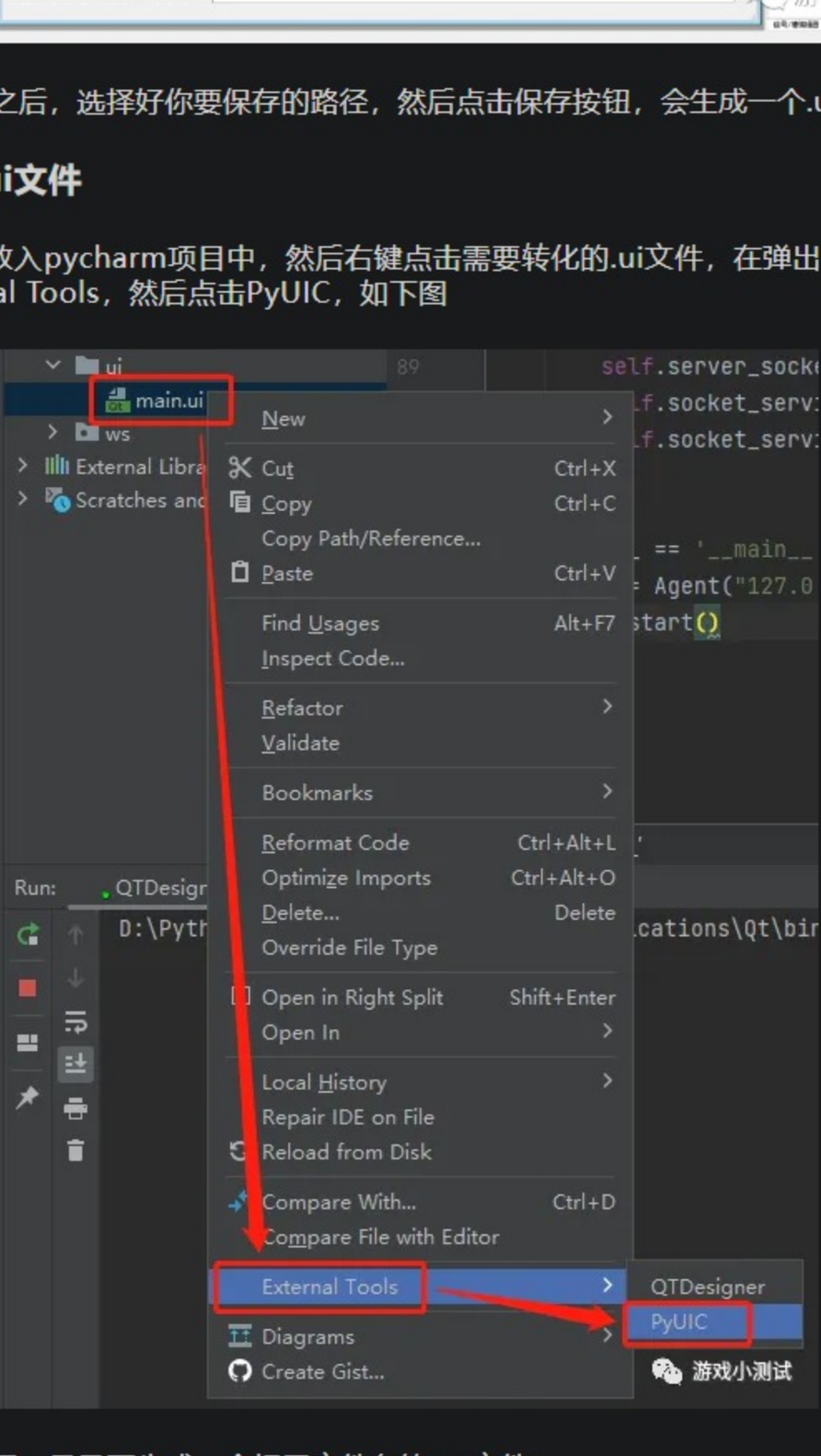
在我们写工具的时候，比较常用的就上面这些，虽然PyQt5还有其他很多好用的功能，但我们比较少用到，所以不再详细说明，如有需要的可以自行研究。接下来我们先画一个简单的界面，来实现开启代理，并显示收发协议的简单功能（后面慢慢再进行功能扩展）



界面编辑好之后，选择好你要保存的路径，然后点击保存按钮，会生成一个.ui文件

三、转化ui文件

1.将ui文件放入pycharm项目中，然后右键点击需要转化的.ui文件，在弹出的菜单中，找到External Tools，然后点击PyUIC，如下图



点击后会在同一目录下生成一个相同文件名的.py文件

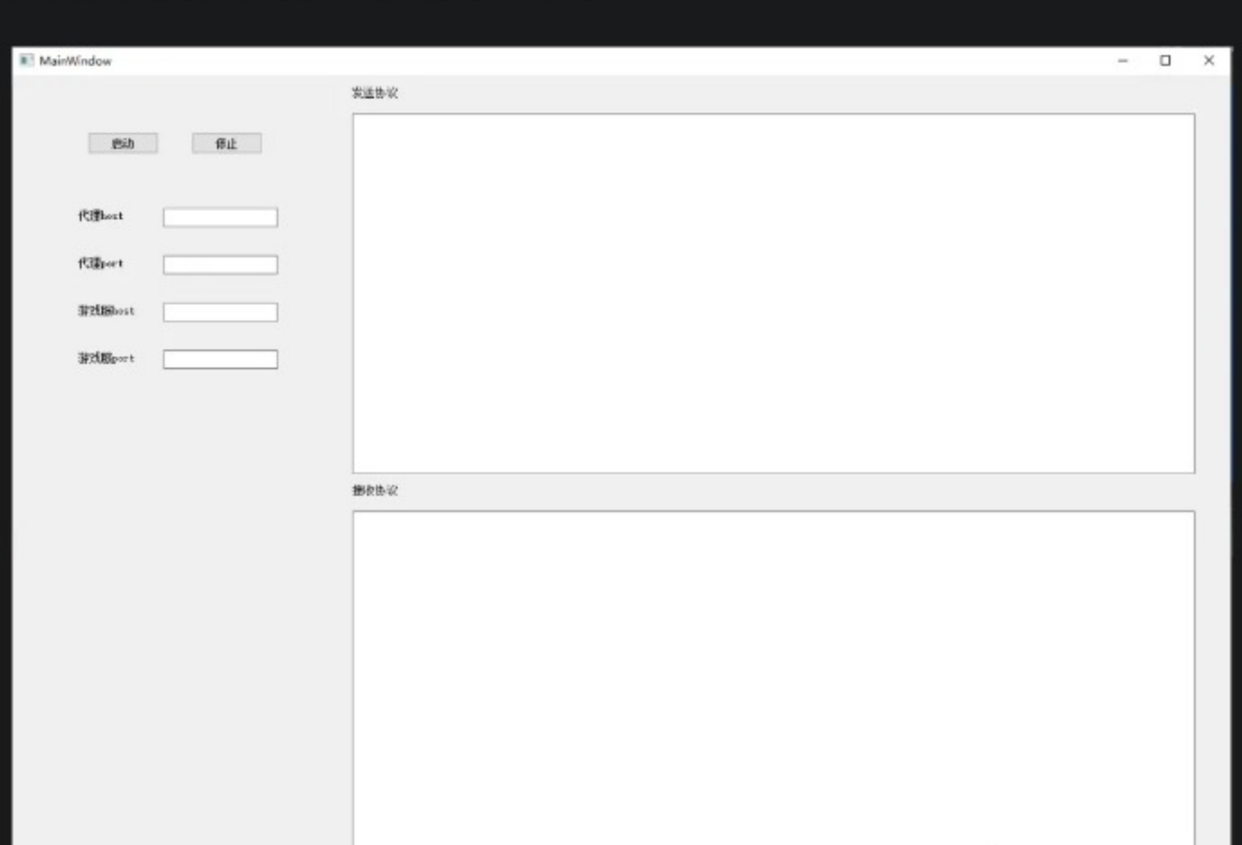
这样我们的简单界面就做好了。

四、让界面运行起来

新建一个py脚本（如wpe_tools.py），在脚本里创建一个工具类WpeTools，继承自之前我们画的那个UI类（可在pyuic转化后的py文件中查看类名），以及QMainWindow类

```python
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication
from ui.main import Ui_MainWindow

class WpeTools(Ui_MainWindow, QMainWindow):
    def __init__(self, parent=None):
        super(WpeTools, self).__init__(parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    main = WpeTools()
    main.show()
    sys.exit(app.exec_())
```

然后点击运行，就可以看到我们的工具了。



罢特！现在这个工具并没有添加任何的方法，点击启动和停止按钮是没有反应的，我们接下来就让界面上的组件与我们之前写的那个脚本进行关联。

需要注意的是，PyQt运行时，由于界面是实时绘制的，所以主线程是不允许阻塞的，一些可能会影响到主界面绘制的方法，我们需要通过线程的方式来启动，PyQt自带有QThread类，我们需要写一个类来执行启动方法。

在此之前，我们需要修改一下之前写的代理脚本，以便在它可以跟界面进行绑定

第一个修改的地方是Agent初始化方法，我们需要传一个Q线程实例进来（现在只有1个Q线程，所以这样比较好理解，后面有很多Q线程的时候，会将传入的参数换成工具UI界面，方便对多个Q线程的信号进行调用）

```python
class Agent(object):
  def __init__(self, agent_host, agent_port, server_host, server_port, ui_thre
    """
    代理初始化一个实例
    """
    self.ui_thread = ui_thread
    # 这里省略之前的初始化代码……
```

接下来新建一个py文件，用于专门存放Q线程（以后功能多了可能会有很多Q线程）

然后新建一个BeginAgentService类，这个类继承自QThread类

```python
class BeginAgentService(QThread):
  """
  启动转发代理的thread类
  """
  # 定义两个信号，信号参数类型为str，在信号触发的时候，会将str参数传给绑定的ui方法
  send_signal = pyqtSignal(str)
  recv_signal = pyqtSignal(str)

  def __init__(self, agent_host, agent_port, game_host, game_port, parent=Nor
    super(BeginAgentService, self).__init__(parent)
    self.ui = ui
    self.agent_host = agent_host
    self.agent_port = agent_port
    self.game_host = game_host
    self.game_port = game_port

  def run(self):
    self.ui.agent = Agent(self.agent_host, self.agent_port, self.game_host, s
    self.ui.agent.start()
```

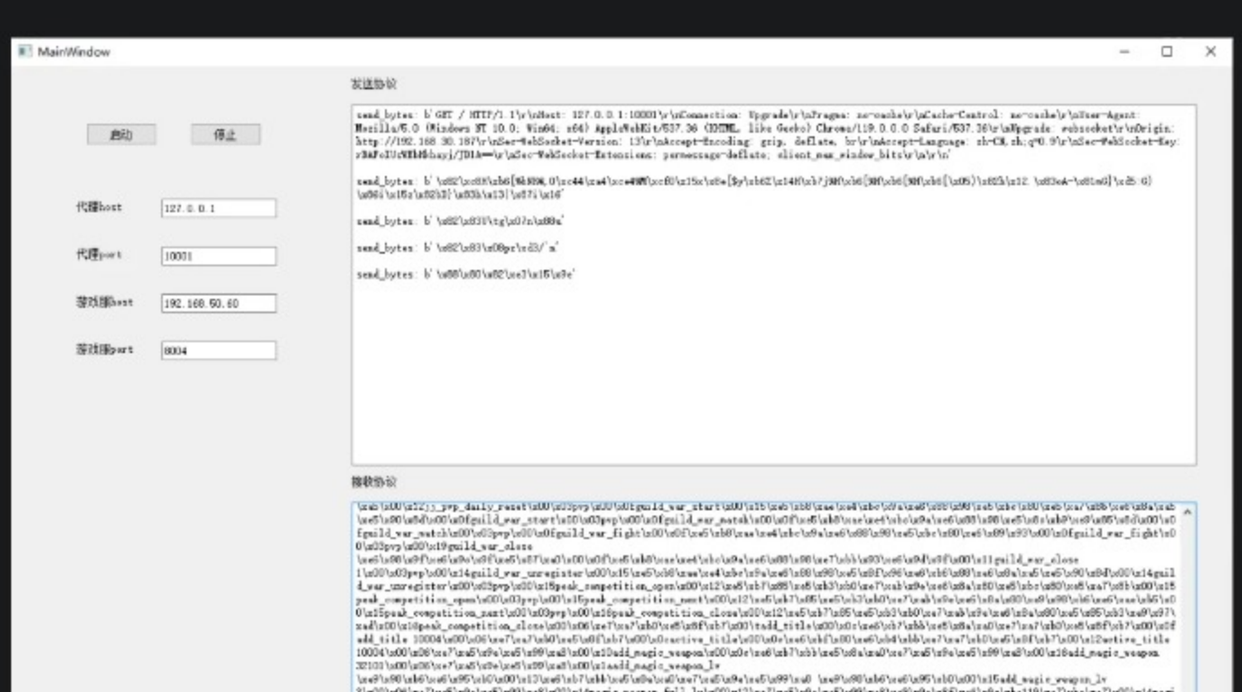
然后修改之前编写的那个Agent代理脚本的协议转发方法，我们之前在收到协议之后，是进行了打印，现在我们将它和界面进行关联，这两句代码的意思是，将我们收发的字节流转换成字符串，并通过ui_thread的信号发送出去。

```python
# client_to_server(self): 中的print方法我们修改如下:
self.ui_thread.send_signal.emit(f'send_bytes: {data}\n')

# server_to_client(self): 中的print方法我们修改如下:
self.ui_thread.recv_signal.emit(f'recv_bytes: {data}\n')
```

最后回到我们的工具类，将按钮点击事件，以及Q线程的信号与界面方法进行绑定（在触发对应的事件的时候，会调用绑定的方法），修改之后的工具类脚本如下：

```python
class WpeTools(Ui_MainWindow, QMainWindow):
  def __init__(self, parent=None):
    super(WpeTools, self).__init__(parent)
    self.ui = Ui_MainWindow()
    self.ui.setupUi(self)
    self.agent = None # 先给它个空对象，占个属性名
    # 绑定按钮方法
    self.ui.start_btn.clicked.connect(self.start)
    self.ui.stop_btn.clicked.connect(self.stop)

  def start(self):
    """
    代理启动方法
    :return:
    """
    # 获取文本框中的ip和端口等信息
    agent_host = self.ui.agent_host_line.text()
    agent_port = int(self.ui.agent_port_line.text())
    game_host = self.ui.game_host_line.text()
    game_port = int(self.ui.game_port_line.text())
    # 生成Q线程
    self.agent_thread = BeginAgentService(agent_host, agent_port, game_host,
    # 绑定线程信号
    self.agent_thread.send_signal.connect(self.send_proto_view)
    self.agent_thread.recv_signal.connect(self.recv_proto_view)
    # 启动Q线程
    self.agent_thread.start()

  def stop(self):
    """
    代理停止方法
    :return:
    """
    self.agent.stop()

  def send_proto_view(self, send_proto):
    """
    将发送协议的内容展示到文本框中
    :param send_proto::return:
    """
    self.ui.send_browser.append(send_proto)

  def recv_proto_view(self, recv_proto):
    """
    将发送协议的内容展示到文本框中
    :param recv_proto::return:
    """
    self.ui.recv_browser.append(recv_proto)
```

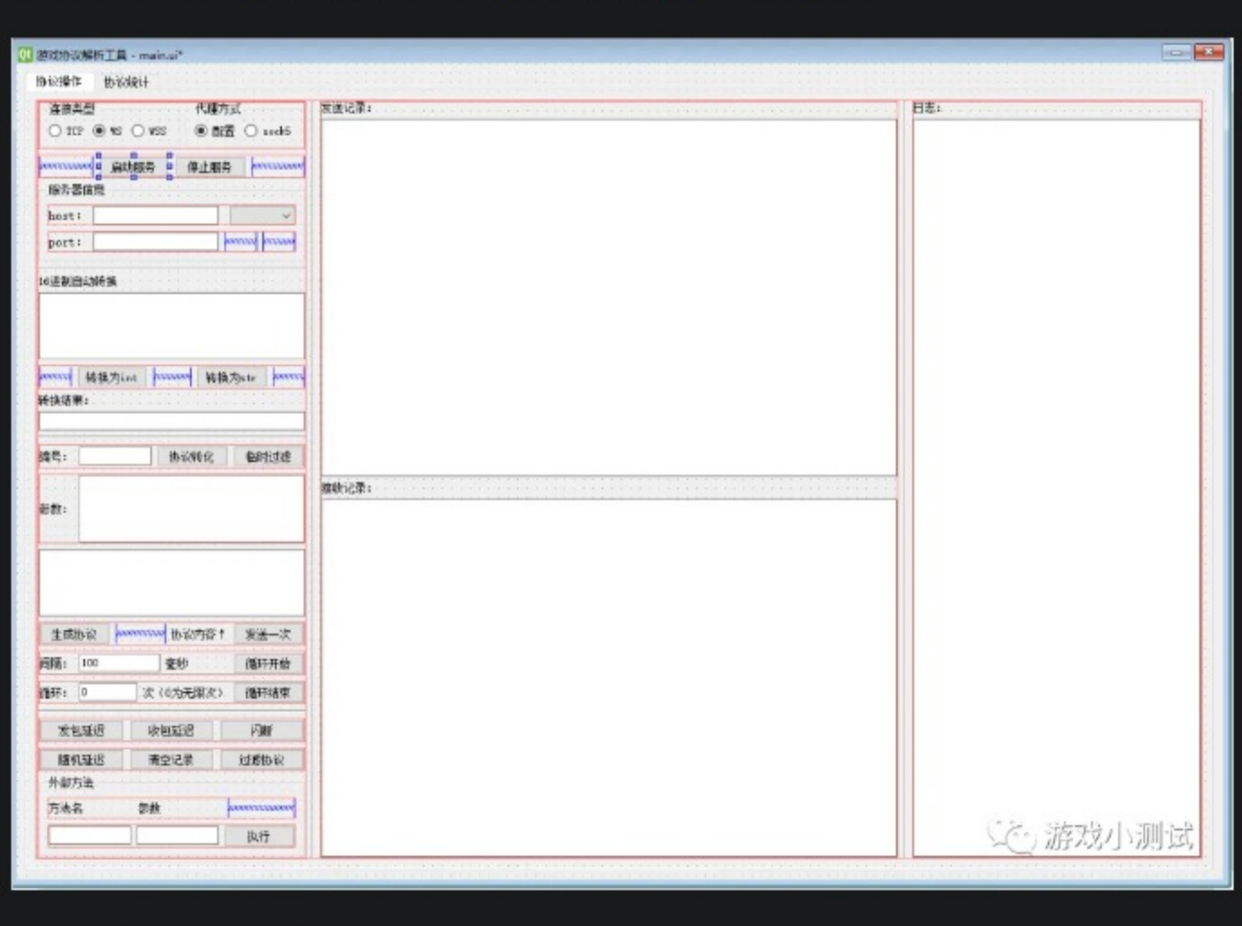
最后我们运行看看效果，像之前一样输入代理和游戏服的ip和端口号，区别在于这次我们在工具栏中输入，输入之后点击运行，然后再打开游戏通过之前配置的代理服进行登录，然后发现协议已经可以显示到工具上了，对比之前纯脚本，我们又迈出了一步~



现在我们的工具已经有了雏形，后面慢慢的来丰富它的功能吧。

最后，在这个脚本里有个坑，那就是点击停止按钮的时候，界面会卡死崩溃，原因我上面有讲到过，有兴趣的同学可以自己试试如何让它在点击停止按钮的时候不会崩溃吧~🐱

最后的最后，上面的界面只是一个很简单很基础的界面，对这个工具有兴趣的同学可以提前准备，编辑一个包含你想要的功能的工具界面，比如我的工具界面，它大概长下面这个样纸，后面我就直接照这个上面的功能来进行分享了。



#抓包工具 15 #测试工具 10

测试工具 · 目录

＜上一篇 游戏抓包工具制作（一）—— 最简单转发代理的实现

下一篇＞ 游戏抓包工具制作（三）—— 协议解析方法

个人观点，仅供参考