1. 整体思路

1.1 目标：快速上传代码，简化操作，在本地 PC 完成上传以及版本 bin 文件的生成，最终 bin 文件回传到本地。

1.2 如何快速上传代码

经过多次尝试，采用多线程逐个子目录扫描上传，同时过滤到无用的子目录，嵌套子目录过多，线程数量开辟太多会增加耗时，多线程同时上传时有时会致使 sftp 上传出现异常（原因不清除），效果不佳。

后来发现 一次性解析代码目录结构，将所有需要上传的子目录都生成在一个列表里，然后利用 SSH 执行远程命令，逐个创建对应的子目录，再利用 sftp 上传每个子目录里的文件。此种方法，耗时很短，大概不到 20s。（注意：paramiko 里的 sftp 上传文件接口，每次只能上传或者下载一个文件（非目录））

上面分析的是上传整体代码的情形，还有一种考虑，如编译失败，本地修改相应错误后，只需要上传对应修改的文件即可，无需整个子目录上传，设计了检测 git 仓库修改，上传变动的文件，再继续编译，会更加快捷。

1.3 简化操作：

设计了一个简单的配置界面，使用者根据自己所需进行配置，且有对应的配置文件，记录实时的配置，下次启动时会载入上一次的配置。界面显示实际运行的状态信息，以及一些内部错误提示。

2. 详细代码说明

个人 python 代码编写没有注重很严格的规范，请见谅。严格的规范是：每个类，函数都要进行注释详细说明，类是做什么的，有什么成员，类函数是干什么的等等。后续可以利用 pydoc 将字符串注释导出一份使用参考。

后续理解，还是需要 python3（未涉及与 python2 较大差异的内容，了解 python2 亦可）及 Qt 相关的一些基础，paramiko 的基本使用（文件上传以及远程执行 linux 命令）。后面的说明并没有做到很细致。Python3 是未来，Python2 会被主键废弃掉，建议感兴趣的还是多了解下 python3。
涉及 python 的基础内容：os.path、json、traceback、sys.exec_info、subprocess.Popen 等的基本使用，可以参考 python 自带的 manuals。

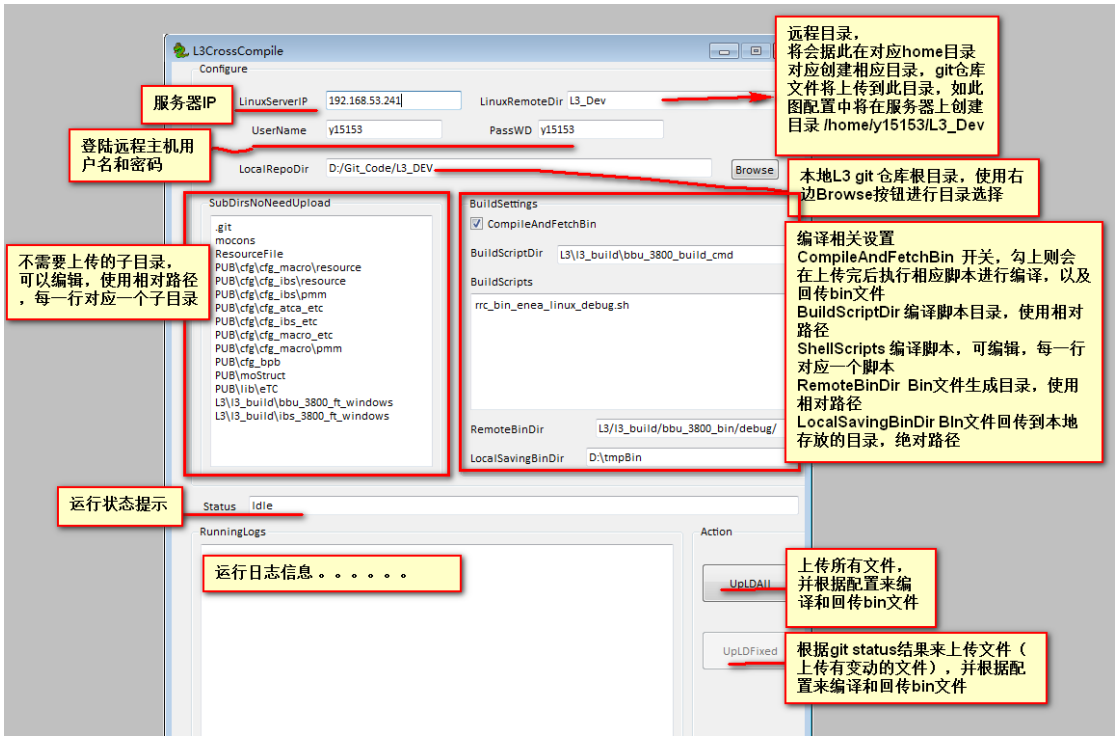
2.1 界面逻辑相关代码

```
class Ui_L3SSHCrossCompile(QtWidgets.QMainWindow):…
```

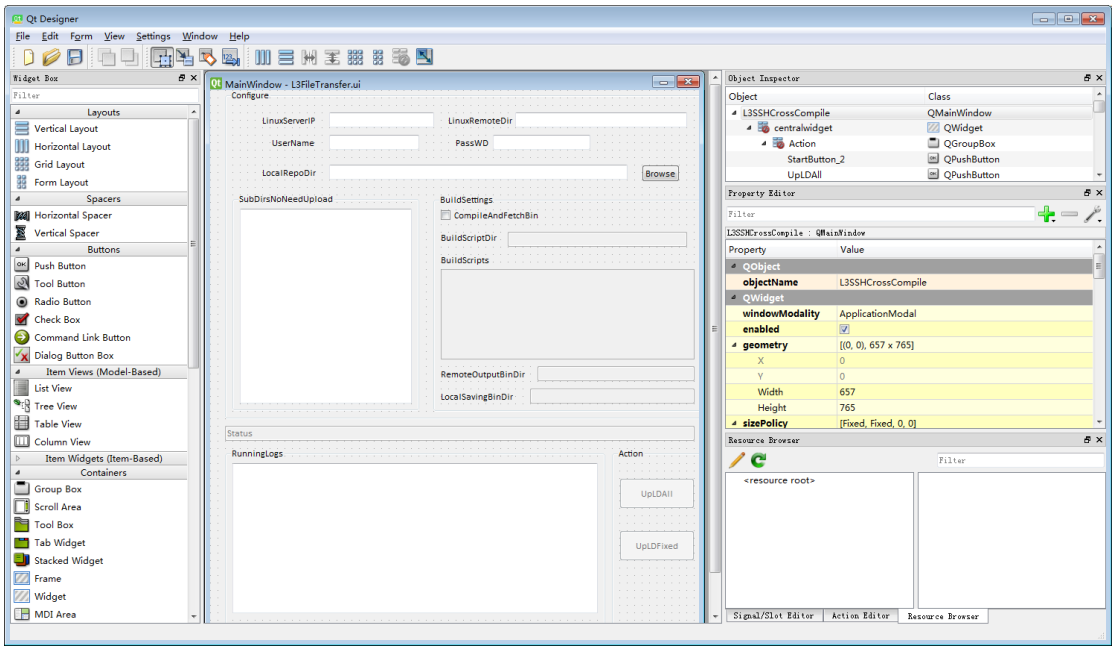
该类继承于 Qt 内部主窗口子类 QMainWindow，包含了基本的窗口界面结构，以及相关的窗口事件信号处理逻辑结构。

事先利用 Qt Designer 画出 界面 UI 文件 L3FileTransfer.ui，然后利用 pyuic 工具将该文件转换成 L3FileTranfer.py 文件，对其进行稍微改造利用，就是 Ui_L3SSHCrossCompile 中界面部分代码。

界面的基本说明



（Qt Designer PyUIC 工具相关见环境安装配置说明，Qt Designer 的使用可以找些简单的示例来练习，配合 PyUIC 生成 py 文件后，再对界面逻辑进行改造，熟悉基本部件的使用，可以参考 PyQt4.chm API 手册，注意 PyQt5 与 PyQt4 是有一些差异的，遇到困惑的地方可以上网查查）



`Ui_L3SSHCrossCompile` 类中 由 `QtWidgets` 生成的都是基本界面部件（如按钮，标签，输入栏，编辑栏等）。

Qt 界面设计里一个重要的机制是信号-槽，简要说明当我们在界面窗口移动鼠标，或点击某个按钮，编辑某一输入栏等等操作，就会产生一个相关信号（也可能是多个），可以定义一个槽函数，绑定相应信号（也称事件），当信号产生时，相应的槽函数就会被调用处理。事件信号可以是 Qt 内部窗体已内置定义的，也可自定义。信号与槽函数可以是多对多的关系，槽函数中也可发射信号事件，界面逻辑处理的设计是很灵活方便的。

```
self.LinuxServerIP.textChanged.connect(self.configureEdited)
self.LinuxRemoteDir.textChanged.connect(self.configureEdited)
self.PassWD.textChanged.connect(self.configureEdited)
self.UserName.textChanged.connect(self.configureEdited)
self.LocalRepoDir.textChanged.connect(self.configureEdited)
```

当用户编辑相关配置时，发送对应部件的 `textChanged` 信号，上面这些信号都连接到同一个槽函数 `self.configureEdited`

```
def configureEdited(self):…
```

这个槽函数的作用是，对基础配置做简单的检测（是否有为空的），当配置都存在时，使能界面的 UpLDAll UpLDFIxed 按钮，并记录的相关配置参数，否则置灰按钮

```
self.BrowseFileButton.clicked.connect(self.fileBrowsePressed)

def fileBrowsePressed(self):…
```

这对信号-槽的作用是 当用户点击 Browse 按钮会弹出选择浏览目录窗口，方便用户配置本地代码仓库目录。

```
self.CompileAndFetchBin.stateChanged.connect(self.CompileAndFetchBinSwitchChanged)
def CompileAndFetchBinSwitchChanged(self, state):…
```

这对信号-槽是检测是否开启了编译 bin 选项，若未开启，相关的配置参数均不可编辑。

```
internalExceptionSignal = QtCore.pyqtSignal()
self.internalExceptionSignal.connect(self.internalExceptionProcess)
def internalExceptionProcess(self):…
```

自定义的内部处理异常信号，绑定内部异常处理函数（只是简单设计内部运行异常后，界面能执行的操作，更改配置，执行 UpLDALL）

```
self.UpLDAll.clicked.connect(self.UploadAllSrcFiles)

self.UpLDFixed.clicked.connect(self.UploadFixedFiles)

def UploadAllSrcFiles(self):…

def UploadFixedFiles(self):…
```

按钮点击后，会创建处理逻辑线程，用于文件上传，远程编译，回传等处理。后续会详细说明。

```
def UploadAllSrcFilesPrepareProc(self):…

class OpenSSHThread(QtCore.QThread):…
```

这个 `UploadAllSrcFilesPrepareProc` 是因为初始时，或者检测到配置有变，需要创建新的 SSH 连接，而 SSH 连接建立有一定的耗时，所以利用了 OpenSSHThread 线程来创建 SSH 连接。此逻辑与界面显示逻辑相分开，不会出现界面卡顿或者延迟。（解决初版里第一次启动时，点击 UpLoadAll 界面卡顿问题）

```
self.UploadFileThread.threadOverSignal.connect(self.uploadThreadOverProcess)
self.UploadFileThread.logSignal.connect(self.LogParsingShow)

self.UploadFileThread.threadOverSignal.connect(self.uploadThreadOverProcess)
self.UploadFileThread.logSignal.connect(self.LogParsingShow)

def LogParsingShow(self, log_type, log_info):…
def uploadThreadOverProcess(self):…
```

绑定 上传编译处理的线程 内部自定义的两个信号 `threadOverSignal` `logSignal` 到对应处理函数 线程结束处理函数 `uploadThreadOverProcess`，线程运行日志实时解析并打印打印函数 `LogParsingShow`。

```
def internalLogPrint(self, text):
```

日志显示函数

```
def loadCfg(self):…

def saveCfg(self):…

def updateRelatedCfg(self):…
```

loadCfg 配置载入函数，启动时使用；saveCfg 配置变更时更新保存函数；

updateRelatedCfg 更新线程使用的相关配置（一部分配置参数是 上传编译处理线程需要使用的参数，每次点击上传按钮时会检测相关配置是否有变化，进行更新）。

2.2 上传编译及回传处理的相关代码

```
class UploadFilesThread(QtCore.QThread):…
```

该类继承于 Qt 类置的线程类 QThread，用于上传代码，编译，回传 bin 文件处理。

```
def get_lowest_subdir(self, rootDir):…
def check_if_dir_need_upload(self, dir):…
```

递归解析本地仓库目录，过滤无关子目录，生成需要上传的子目录列表

```
def upLoadfiles(self):…
```

该函数用于上传所有子目录文件。因为 paramiko sftp 接口内次调用只能上传或者下载一个文件，所以需要先在远程 linux 主机创建好所有对应子目录，然后在逐个子目录上传其包含的文件。

```
def compile(self):…
```

执行远程编译脚本。

```
def fetchBinFiles(self):…
```

将生成的 bin 文件拷贝到本地。

```
def run(self):
```

线程执行函数。

分两种情形：

（1） UpLDAll 首先清除远程 Linux 目录，解析代码 windows 子目录，创建对应 Linux 子目

录，上传文件，执行编译脚本，回传 bin 文件到本地。

（2） UpLDFixed 这种只能用于当执行 UpLDAll 出现编译失败后，根据编译错误信息修改本地代码文件，然后执行 UpLDFixed。

切换到代码仓库目录，执行 git add .命令（为方便后续解析），然后执行 git status，解析仓库变动结果，分为三种文件 ------ 新增文件，修改文件，删除文件。

对于删除文件，则清除远程 Linux 目录下对应文件；对于修改文件，则先删除远程 Linux 文件，然后再上传此文件；对于新增文件，则直接上传。（注意没有考虑新增子目录的情形，此种情况较少，若有，请使用 UpLDAll）

然后则是继续编译，回传 bin 文件。

线程运行过程中，实时发送日志输出信号，并且当出现执行远程命令出错或者编译错误时，反馈出错信息，并发送日志信号。日志信号对应的窗口槽函数，会解析日志并进行实时显示。

```
except Exception:
    os.chdir(pwd)
    internalException = extractExceptionInfo()
    self.logSignal.emit(LogInfoSignal, internalException)
    self.logSignal.emit(StatusSignal, "Unexpected Internal Exception.\n")
    self.threadExceptionSignal.emit()
    self.threadOverSignal.emit()
```

执行过程出现内部异常，则发送内部异常信号 threadExceptionSignal，同时提取内部异常退出信息，发送日志信号。

线程正常结束或者异常结束都会发送线程结束信号 threadOverSignal。

2.3 Main 部分代码说明

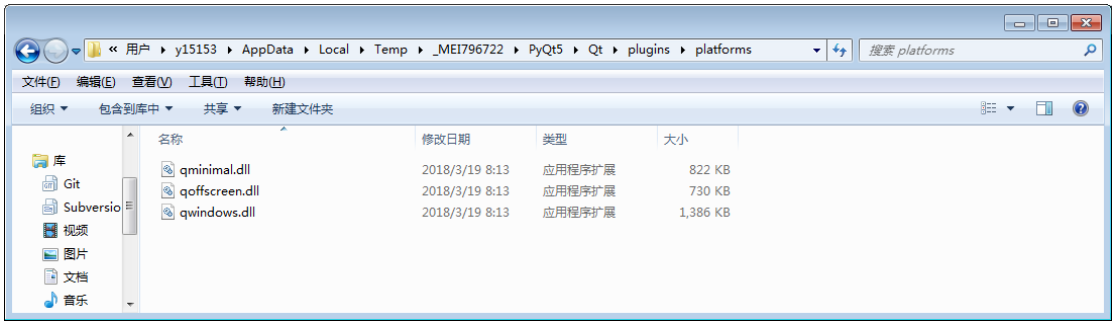
```
if __name__ == '__main__':
    env = os.environ
    old_qt_path = env["QT_QPA_PLATFORM_PLUGIN_PATH"]
    try:
        if "QT_QPA_PLATFORM_PLUGIN_PATH" in env.keys():
            if hasattr(sys, '_MEIPASS'):
                new_qt_path = sys._MEIPASS + '/PyQt5/Qt/plugins'
                env["QT_QPA_PLATFORM_PLUGIN_PATH"] = new_qt_path
        app = QtWidgets.QApplication(sys.argv)
        MyUI = Ui_L3SSHCrossCompile()
        MyUI.setFixedSize(MyUI.width(), MyUI.height())
        MyUI.show()
        sys.exit(app.exec_())
    finally:
        env["QT_QPA_PLATFORM_PLUGIN_PATH"] = old_qt_path
```

QT_QPA_PLATFORM_PLUGIN_PATH 需要设置该系统环境变量为 exe 执行时临时解包出来的文件中的 Qt/plugins，防止个人（配置过 Qt 开发环境的用户）有设置该环境变量，导致安装配置的差异出现 使用本地安装的不同版本的插件而出现 exe 运行失败。

```
if hasattr(sys, '_MEIPASS'):
    new_qt_path = sys._MEIPASS + '/PyQt5/Qt/plugins'
    env["QT_QPA_PLATFORM_PLUGIN_PATH"] = new_qt_path
```

这个判断是因为当生成单一 exe 执行时，会生成临时的_MEIPASS（随机值）,并在用户 appdata 中创建以此值命名的临时的文件夹。在个人开发环境中，运行 py 文件不会有此种问题。

如下运行 exe 时会创建的临时文件夹，该文件夹会在程序正常关闭退出时被自动清理掉，程序异常结束（运行崩溃，手动后台 kill 等）则不会被清理掉。



2.4 其他代码说明

```
def extractExceptionInfo():
```

该函数用于提取代码执行错误退出信息。

```
def open_ssh(configure):
```

该函数用使用 paramiko 创建 SSH 连接。

```
def ssh_exec_cmd(ssh, cmd):
```

该函数用于 ssh 远程执行 linux 命令。

```
def ssh_exec_sudo_cmd(ssh, cmd, passwd):
```

这函数用于 ssh 远程执行 sudo 命令，sudo 命令需要密码交互验证，调用 paramiko exec_command 接口时，需调用时设置参数 get_pty 为 True。此接口只用在了清除远程目录，没有用在构建脚本调用，所以不能用来执行 Linux FT 运行。

```
def git_cmd_exec(git_cmd):
    startupinfo = STARTUPINFO()
    startupinfo.dwFlags |= STARTF_USESHOWWINDOW
    startupinfo.wShowWindow = SW_HIDE
    p = Popen("cmd", stdin=PIPE, stdout=PIPE, stderr=PIPE, startupinfo=startupinfo)
    cmd_outs, cmd_errs = p.communicate(git_cmd)
```

```
    p.terminate()
    return cmd_outs
```

该函数利用 subprocess 模块的 Popen 类，创建新的进程 cmd，执行 git 命令，返回命令结果。

STARTUPINFO 模块设子 windows 启动的属性，这里面启动 windows cmd，但是设置为隐藏 SW_HIDE，因为生成最终界面 exe，里使用了 –w 选项，不启用 cmd window。

```
def git_status_parsing():
```

该函数 执行 git status 命令，并解析命令返回的结果。

3. 将 Py 文件打包成单一 EXE 文件

利用 pyinstaller 将 py 文件打包成 exe 文件。可以上网搜索相关使用。

打开 cmd，在源文件 src 目录，执行命令：
c:\Python36\Scripts\pyinstaller.exe -w -F --icon=python.ico L3CrossCompile.py

4. 踩坑记录

（1） paramiko sftp 接口每次只能上传下载一个文件，不能用目录。

（2） Qt 中信号与槽的处理。 槽函数绑定的是窗体部件产生的信号时，若槽函数中多次对其他单一部件进行输出显示（setText），此时只会显示最后一次。
若想做到窗口部件实时状态变更显示，则必须利用多线程 QThread，另一个线程发射信号，主界面相关槽函数绑定该信号，进行状态更新显示。
建议 Qt 界面程序中使用 Qt 内置封装的多线程 API，不要使用其他模块里的线程库函数。

（3） 启用新进程调用 cmd，执行 git 命令。 最开始使用 os.popen 来执行 git 命令获取结果，运行 py 是可以的，因为默认启用 cmd 窗口，但是在生成单一 exe（生成时设置 -w 选项）后，就会出现调用失败。
后来上网找到解决办法，利用 subprocess 模块的 Popen，设置隐藏窗口模式调用 cmd 执行命令。

（4） 出现读取 git 输出结果解码错误，调试发现 Popen 执行 cmd 命令的输出包含中英文字符的，中文与英文的编码格式不一样，不能采用单一格式对结果进行解码。
因为只关心英文内容，设置 errors 为 ignore，忽略中间部分错误解码以解决。

```
outs.decode(encoding='utf-8', errors='ignore')
```