JFZ 金斧子
WWW.JFZ.COM

自动化测试框架(数据驱动)

讲师 : 芳姐 | 时间 : 2018-9-6

11. 在pageObjects包中建一个文件名为CustomerPage.py的文件

在PageElementLocator.ini文件中添加内容：

```
[erp_Customer]
CustomerPage.fortuneCenter = xpath>//*[@name="财富中心客户"]
CustomerPage.AddLink = xpath>//*[text()='添加']
CustomerPage.frame = xpath>iframe10000014545120
CustomerPage.recordType = xpath>//*[@id="recordType"]/option[@value='1']
CustomerPage.username = xpath>//*[@id="name"]
CustomerPage.fromType = xpath>//*[@id="fromType"]/option[@value='6']
CustomerPage.sex = xpath>//*[@id="sex"]/option[@value='1']
CustomerPage.ctype = xpath>//*[@id="ctype"]/option[@value='2']
CustomerPage.phoneNumber = xpath>//*[@id="phoneNumber"]
CustomerPage.iframe = xpath>/html/body/div[3]/iframe
CustomerPage.yytime = xpath>//*[@id="yytime"]
CustomerPage.dpTodayInput = xpath>//*[@id="dpTodayInput"]
CustomerPage.saveCustomer = xpath>//*[@id="saveCustomer"]
```

新建审批流CustomerPage.py文件

```
class CustomerPage(object):
    def __init__(self, driver):
        self.driver = driver
        self.parseCF = ParseConfigFile()
        self.CustomerOptions = self.parseCF.getItemsSection("erp_Customer")

    # 进入到财富中心客户列表
    def fortuneCenter(self):
        try:
            # 从定位表达式配置文件中读取添加客户的定位表达式
            locateType, locatorExpression =
self.CustomerOptions["CustomerPage.fortuneCenter".lower()].split(">")
            # 获取添加客户资料页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e
```

```
# 点击添加按钮
    def addLink(self):
        try:
            # 从定位表达式配置文件中读取添加客户的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.AddLink".lower()].split(">")
            # 获取添加客户资料页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e

    # 切换到添加客户资料的fram里面
    def fram(self):
        try:
            # 从定位表达式配置文件中读取fram的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.frame".lower()].split(">")
            # 获取添加客户资料页面的fram，并返回给调用者
            self.driver.switch_to.frame(locatorExpression)
        except Exception as e:
            raise e
```

```
# 选择记录类型
    def recordType(self):
        try:
            # 从定位表达式配置文件中读取记录类型的定位表达式
            locateType, locatorExpression =
self.CustomerOptions["CustomerPage.recordType".lower()].split(">")
            # 获取添加客户资料页面的选择记录类型对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e

    # 输入客户姓名
    def userName(self):
        try:
            # 从定位表达式配置文件中读取客户姓名的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.username".lower()].split(">")
            # 获取添加客户资料页面的客户姓名类型对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e
```

```
# 选择获取方式
    def fromType(self):
        try:
            # 从定位表达式配置文件中读取获取方式的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.fromType".lower()].split(">")
            # 获取添加客户资料页面的获取方式对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e

    # 选择性别
    def sex(self):
        try:
            # 从定位表达式配置文件中读取性别的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.sex".lower()].split(">")
            # 获取添加客户资料页面的性别对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e
```

```
# 选择预约类型
    def ctype(self):
        try:
            # 从定位表达式配置文件中读取预约类型的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.ctype".lower()].split(">")
            # 获取添加客户资料页面的预约类型对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e

    # 输入手机号码
    def phoneNumber(self):
        try:
            # 从定位表达式配置文件中读取手机号码的定位表达式
            locateType, locatorExpression =
self.CustomerOptions["CustomerPage.phoneNumber".lower()].split(">")
            # 获取添加客户资料页面的手机号码对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e
```

```
# 选择预约时间
    def yytime(self):
        try:
            # 从定位表达式配置文件中读取预约时间的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.yytime".lower()].split(">")
            # 获取添加客户资料页面的预约时间对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e

    def ifram(self):
        try:
            # 从定位表达式配置文件中读取预约时间的定位表达式
            locateType, locatorExpression = self.CustomerOptions["CustomerPage.iframe".lower()].split(">")
            elementObj = getElement(self.driver, locateType, locatorExpression)
            # return elementObj
            self.driver.switch_to.frame(elementObj)
        except Exception as e:
            raise e
```

```
 # 选择预约时间为今天
    def dpTodayInput(self):
        try:
            # 从定位表达式配置文件中读取预约时间的定位表达式
            locateType, locatorExpression =
self.CustomerOptions["CustomerPage.dpTodayInput".lower()].split(">")
            # 获取添加客户资料页面的预约时间对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e

    # 切换出ifram
    def tofram(self):
        try:
            # 切换出当前的fram
            self.driver.switch_to.default_content()
        except Exception as e:
            raise e
```

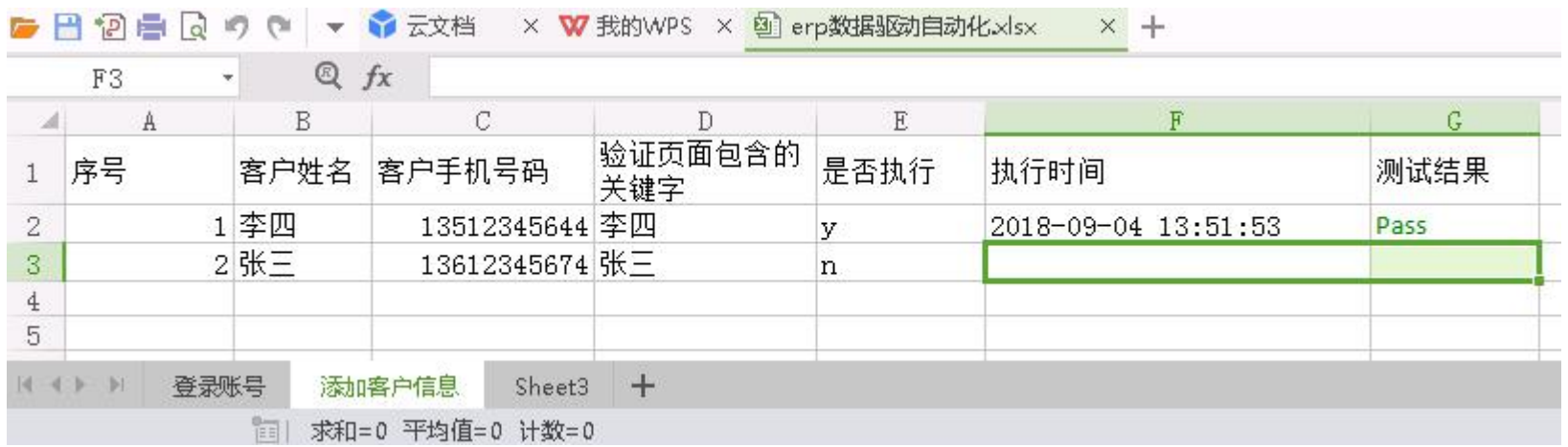
```
# 点击保存按钮
    def saveCustomer(self):
        try:
            # 从定位表达式配置文件中读取保存按钮的定位表达式
            locateType, locatorExpression =
self.CustomerOptions["CustomerPage.saveCustomer".lower()].split(">")
            # 获取添加客户资料页面的保存对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e
```

12、在DataDrivenFrameWork新建一个名叫testData文件夹

在DataDrivenFrameWork新建一个名叫testData文件夹，并在改目录下新建一个名叫“erp数据驱动自动化.xlsx”的excel文件，并在excel文件中创建两个工作表，分别叫“登录账号”“添加客户信息”的工作表，其内容如下：

序号	用户名	密码	数据表	是否执行	测试结果
1	defang1	123	添加客户信息	y	Pass
2	defang2	123	添加客户信息	n	

登录账号 | 添加客户信息 | Sheet3



序号	客户姓名	客户手机号码	验证页面包含的关键字	是否执行	执行时间	测试结果
1	李四	13512345644	李四	y	2018-09-04 13:51:53	Pass
2	张三	13612345674	张三	n		

登录账号 | 添加客户信息 | Sheet3

求和=0 平均值=0 计数=0

13、在util包中新建一个名叫ParseExcel.py的文件

```
class ParseExcel(object):
    def __init__(self):
        self.workbook = None
        self.excelFile = None
        self.font = Font(color=None)  # 设置字体颜色
        # 颜色对于的RGB值
        self.RGBDict = {'red': 'FFFF3030', 'green': 'FF008B00'}

    def loadWorkBook(self, excelPathAndName):
        # 将excel文件加载到内存，并获获取其workbook对象
        try:
            self.workbook = openpyxl.load_workbook(excelPathAndName)
        except Exception as e:
            raise e
        self.excelFile = excelPathAndName
        return self.workbook
```

```
def getSheetByName(self, sheetName):
        # 根据sheet名获取该sheet对象
        try:
            sheet = self.workbook.get_sheet_by_name(sheetName)
            return sheet
        except Exception as e:
            raise e

    def getSheetByindex(self, sheetIndex):
        # 根据sheet的索引号获取sheet对象
        try:
            sheetname = self.workbook.get_sheet_by_names()[sheetIndex]
        except Exception as e:

            raise e
        sheet = self.workbook.get_sheet_by_name(sheetname)
        return sheet

    def getRowsNumber(self, sheet):
        # 根据sheet中有数据区域的结束行号

        return sheet.max_row
```

```
def getColsNumber(self, sheet):
        # 获取sheet中有数据区域的结束列号
        return sheet.max_column

    def getStartRowNumber(self, sheet):
        # 获取sheet中有数据区域的开始的行号
        return sheet.min_row

    def getStartColNumber(self, sheet):
        return sheet.min_column

    def getRow(self, sheet, rowNo):
        # 获取sheet中某一行，返回的是这一行所有的数据内容组成的tuple
        # 下标从1开始，sheet.rows[1]表示第一行
        try:
            return sheet.rows[rowNo - 1]
        except Exception as e:
            raise e
```

```
def getColumn(self, sheet, colNo):
    # 获取sheet中某一列，返回的是这一列所有的数据内容组成的tuple，
    # 下标从1开始，sheet.columns[1]表示第一列
    try:
        return sheet.columns[colNo - 1]
    except Exception as e:
        raise e
def getCellOfValue(self, sheet, coordinate=None, rowNo=None, colsNo=None):
    # 根据单元格所在的位置索引该单元格中的值，下标从1开始
    # sheet.cell（row=1，column=1）。value，表示excel中第一行第一列的值
    # 如getCellObject（sheet，rowNo，colsNo=2)
    if coordinate is not None:
        try:
            return sheet.cell(coordinate=coordinate).value
        except Exception as e:
            raise e
    elif coordinate is None and rowNo is not None and colsNo is not None:
        try:
            return sheet.cell(row=rowNo, column=colsNo).value
        except Exception as e:
            raise e
    else:
        raise Exception("Insufficient Coordinates of cell!")
```

```
def getCellOfObject(self, sheet, coordinate=None, rowNo=None, colsNo=None):
    # 获取某个单元格的对象，可以根据单元格所在位置的数字索引，
    # 也可以直接根据excel中单元格的编码及坐标
    # 如getCellObject（sheet，rowNo = 1，colsNo = 2）
    if coordinate is not None:
        try:
            return sheet.cell(coordinate=coordinate)
        except Exception as e:
            raise e
    elif coordinate is None and rowNo is not None and colsNo is not None:
        try:
            return sheet.cell(row=rowNo, column=colsNo)
        except Exception as e:
            raise e
    else:
        raise Exception("Insufficient Coordinates of cell!")
```

```
def writeCell(self, sheet, content, coordinate=None, rowNo=None, colsNo=None, style=None):
    # 根据单元格在excel中的编码坐标或者数字索引坐标向单元格中写入数据
    # 下标从1开始，参数style表示字体颜色的名字，比如red，green
    if coordinate is not None:
        try:
            sheet.cell(coordinate=coordinate).value = content
            if style is not None:
                sheet.cell(coordinate=coordinate).font = Font(color=self.RGBDict[style])
            self.workbook.save(self.excelFile)
        except Exception as e:
            raise e
    elif coordinate is None and rowNo is not None and colsNo is not None:
        try:
            sheet.cell(row=rowNo, column=colsNo).value = content
            if style:
                sheet.cell(row=rowNo, column=colsNo).font = Font(color=self.RGBDict[style])
            self.workbook.save(self.excelFile)
        except Exception as e:
            raise e
    else:
        raise Exception("Insufficient Coordinates of cell!")
```

```
def writeCurrentTime(self, sheet, coordinate=None, rowNo=None, colsNo=None):
    # 写入当前的时间，下标从1开始
    now = int(time.time())  # 显示时间戳
    timeArray = time.localtime(now)
    currentTime = time.strftime("%Y-%m-%d %H:%M:%S", timeArray)
    if coordinate is not None:
        try:
            sheet.cell(coordinate=coordinate).value = currentTime
            self.workbook.save(self.excelFile)
        except Exception as e:
            raise e
    elif coordinate is None and rowNo is not None and colsNo is not None:
        try:
            sheet.cell(row=rowNo, column=colsNo).value = currentTime
            self.workbook.save(self.excelFile)
        except Exception as e:
            raise e
    else:
        raise Exception("Insufficient Coordinates of cell!")
```

14、在appModules包中新建一个名叫CustomerAction.py文件

```
class CustomerAction(object):
    def __init__(self):
        print("添加客户信息")
    @staticmethod
    def customer(driver, username, yytime, phone):
        try:
            # 新建添加客户资料的对象
            cp = CustomerPage(driver)
            # 点击财富中心客户菜单
            cp.fortuneCenter().click()
            time.sleep(5)
            # 切换入添加客户资料页面的fram
            cp.fram()
            # 点击添加按钮
            cp.addLink().click()
            time.sleep(3)
            # 输入客户姓名
            cp.userName().send_keys(customerUserName)
            # 选择记录类型
            cp.recordType().click()
            # 选择获取方式
            cp.fromType().click()
```

```
            # 选择性别
            cp.sex().click()
            # 选择预约类型
            cp.ctype().click()
            # 输入手机号码
            cp.phoneNumber().send_keys(customerPhoneNumber)
            # 输入预约时间
            cp.yytime().click()
            time.sleep(3)
            # 切换出当前fram，进入到日期选择fram
            cp.tofram()
            # 进入到日期的ifram里面
            cp.ifram()
            # 选择今天
            cp.dpTodayInput().click()
           # 切换出日期的ifram
            cp.tofram()
            time.sleep(3)
            # 进入到新建客户的ifram里面
            cp.fram()
            # 点击保存按钮
            cp.saveCustomer().click()
    except Exception as e:
            raise e
```

15、修改config包下的VarConfig.py文件

```
# 获取数据文件存放绝对路径
dataFilePath = parentDirPath + "\\testdata\\erp数据驱动自动化.xlsx"

# 登录账号工作表中，每列对应的数字序号
account_username = 2
account_password = 3
account_dataBook = 4
account_isExecute = 5
account_testResult = 6

# 添加客户信息工作表中，每列对应的数字序号
customer_customerUserName = 2
customer_customerPhoneNumber = 3
customer_contactKeyWords = 4
customer_isExecute = 5
customer_runTime = 6
customer_testResult = 7
```

16、修改testScripts包中的TestAddCustomer.py文件

```
# 创建解析excel对象
excelObj = ParseExcel()
# 将excel数据文件加载到内存
excelObj.loadWorkBook(dataFilePath)


def LaunchBrowser():
    # 创建Chrome浏览器的一个Options实例对象
    chrome_options = Options()
    # 向Options实例中添加禁用扩展插件的设置参数项
    chrome_options.add_experimental_option("excludeSwitches", ["--disable-extensions"])
    # 添加浏览器最大化的设置参数项，已启动就是最大化
    chrome_options.add_argument('--start-maximized')
    # 启动带有自定义设置的chrome浏览器
    driver = webdriver.Chrome(executable_path="E:\\Python36\\chromedriver",
chrome_options=chrome_options)
    # 访问erp首页
    driver.get("https://qa1-erp.jfz.com")
    time.sleep(3)
    return driver
```

```
def TestAddCustomer():
    try:
        # 根据excel文件中sheet名称获取此sheet对象
        userSheet = excelObj.getSheetByName("登录账号")
        # 获取登录账号sheet中是否执行行列
        isExecuteUser = excelObj.getColumn(userSheet, account_isExecute)
        # 获取登录账号在sheet中的数据表列
        dataBookColumn = excelObj.getColumn(userSheet, account_dataBook)
        print("测试理财师添加客户信息执行开始")
        for idx, i in enumerate(isExecuteUser[1:]):
            # 循环遍历登录账号的登录名，为需要执行的账号添加客户信息
            if i.value == "y":  # 表示要执行
                # 获取第i行的数据
                userRow = excelObj.getRow(userSheet, idx + 2)
                # 获取第i行中的用户名
                username = userRow[account_username - 1].value

                # 获取第i行中的密码
                password = str(userRow[account_password - 1].value)

                print(username, password)
```

```
# 创建浏览器实例对象
        driver = LaunchBrowser()

        # 登录erp系统
        LoginAction.login(driver, username, password)
        # 登录3秒，让浏览器启动完成，以便正常进行后续操作
        time.sleep(3)
        # 获取为第i行中用户添加的客户信息数据表sheet名
        dataBookName = dataBookColumn[idx + 1].value
        # 获取对应的数据表对象
        dataSheet = excelObj.getSheetByName(dataBookName)
        # 获取了联系人数据表中是否执行行列对象
        isExecuteData = excelObj.getColumn(dataSheet, customer_isExecute)
        contactNum = 0  # 记录添加成功客户的个数
        isExecuteNum = 0  # 记录需要执行客户的个数
        for id, data in enumerate(isExecuteData[1:]):
            # 循环遍历是否执行添加联系人列
            # 如果被设置为添加，则进行联系人添加操作
            if data.value == "y":
                # 如果第id行的联系人被设置为执行，则isExecuteNum自增
                isExecuteNum += 1
                # 获取联系人表第id+2行对象
                rowContent = excelObj.getRow(dataSheet, id + 2)
                # 获取客户姓名
                customerUserName = rowContent[customer_customerUserName - 1].value
```

```
# 获取联系人手机号
                customerPhoneNumber = rowContent[customer_customerPhoneNumber - 1].value
                # 添加客户信息成功之后，获取断言信息
                assertKeyWord = str(rowContent[customer_contactKeyWords - 1].value)
                print(customerUserName, customerPhoneNumber, assertKeyWord)
                # 执行新建客户操作
                AddCustomer.customer(driver, customerUserName, customerPhoneNumber)
                time.sleep(2)
                # 在添加客户工作表中写入添加客户执行时间
                excelObj.writeCurrentTime(dataSheet, rowNo=id + 2, colsNo=customer_runTime)
                try:
                    # 断言给定的关键字是否出现在页面中
                    assert assertKeyWord in driver.page_source
                except AssertionError as e:
                    # 断言失败，在添加客户工作表中写入添加客户信息测试失败信息
                    excelObj.writeCell(dataSheet, "Faild", rowNo=id + 2, colsNo=customer_testResult,
style="red")
                else:
                    # 断言成功，写入添加客户信息成功信息
                    excelObj.writeCell(dataSheet, "Pass", rowNo=id + 2, colsNo=customer_testResult,
                                       style="green")
                    contactNum += 1
            print("contactNum = %s,isExecuteNum = %s" % (contactNum, isExecuteNum))
```

```
 if contactNum == isExecuteNum:
                    # 如果成功添加的客户数与需要添加的联系人数相等
                    # 说明给第i个用户添加客户信息测试用例执行成功
                    # 在登录账号工作表中写入成功信息，否则写入失败信息
                    excelObj.writeCell(userSheet, "Pass", rowNo=idx + 2,
colsNo=account_testResult, style="green")
                    print("为用户%s添加%d个客户，测试通过！" % (username, contactNum))
                else:
                    excelObj.writeCell(userSheet, "Fail", rowNo=idx + 2,
colsNo=account_testResult, style="red")
        else:
            print("用户%s被设置为忽略执行！" % excelObj.getCellOfValue(userSheet,
rowNo=idx + 2, colsNo=account_username))
        driver.quit()
    except Exception as e:
        print("数据驱动框架主程序发生异常，异常信息为：")
        # 打印异常堆栈信息
        print(traceback.print_exc())
```

17、在DataDrivenFramework过程根目录下创建一个名为RunTest.py文件

```
from testScripts.TestAddCustomer import TestAddCustomer

if __name__ == '__main__':
    TestAddCustomer()
```

在config包中的VarConfig.py文件中定义了多个常量，在测试脚本文件TestAddCustomer.py中多行代码调用了这些常量，实现了测试数据在测试方法中的重复使用，如果需要修改数据，只需要修改VarConfig.py文件中的常量值就可以实现在全部测试过程生效，减少了代码的维护成本，同时也增加了测试代码的可读性。

在TestAddCustomer.py文件中改为从excel数据文件中读取测试数据，作为数据驱动框架测试过程中的数据来源，执行完某条测试用例后，则会在excel数据文件最后两列分别写入“测试执行时间”和“测试结果”

18、通过logging模块，为数据驱动框架加入打印日志功能

在config包中新建一个名叫Logger.conf的文件，用于配置日志基本信息，具体内容如下：

```
#################################################
[loggers]
keys = root,example01,example02
[logger_root]
level = DEBUG
handlers = hand01,hand02

[logger_example01]
handlers = hand01,hand02
qualname = example01
propagate = 0

[logger_example02]
handlers = hand01,hand03
qualname = example02
propagate = 0
```

```
#########################################################
[handlers]
keys = hand01,hand02,hand03

[handler_hand01]
class = StreamHandler
level = INFO
formatter = form01
args = (sys.stderr,)

[handler_hand02]
class = FileHandler
level = DEBUG
formatter = form01
args = ('log\\DataDrivenFrameWork.log','a')

[handler_hand03]
class = handlers.RotatingFileHandler
level = INFO
formatter = form01
args = ('log\\DataDrivenFrameWork.log','a',10*1024*1024,5)
```

```
#############################################################
[formatters]
keys = form01,form02

[formatter_form01]
format = %(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s%(message)s
datefmt=%Y-%m-%d%H:%M:%S

[formatter_form02]
format = %(name)-12s: %(levelname)-8s%(message)s
datefmt=%Y-%m-%d%H:%M:%S
```

19、在util包中新建一个名叫Log.py的文件

```
import logging
from config.VarConfig import parentDirPath
logging.config.fileConfig(parentDirPath + "\config\Loger.conf")

# 选择一个日志格式
logger = logging.getLogger("example02")

def debug(message):
    # 定义debug级别日志打印方法
    logger.debug(message)

def info(message):
    # 定义info级别日志打印方法
    logger.info(message)

def warning(message):
    # 定义warning级别日志打印方法
    logger.warning(message)
```

18、通过logging模块，为数据驱动框架加入打印日志功能

在config包中新建一个名叫Logger.conf的文件，用于配置日志基本信息，具体内容如下：

```
#################################################
[loggers]
keys = root,example01,example02
[logger_root]
level = DEBUG
handlers = hand01,hand02

[logger_example01]
handlers = hand01,hand02
qualname = example01
propagate = 0

[logger_example02]
handlers = hand01,hand03
qualname = example02
propagate = 0
```

在DataDrivenFramework过程根目录下创建一个名为log的文件夹

修改testScripts包中的TestAddCustomer.py文件

logging.info("测试理财师添加客户信息执行开始")

logging.info("添加客户 %s 成功" % customerPhoneNumber)

logging.info("断言关键字 %s 失败" % assertKeyWord)
logging.info("为用户%s添加%d个客户，%d个成功！" % (username, isExecuteNum, contactNum))

```
E:\Python36\python.exe "E:/python/ERP_selenium UI自动化(Python)/第十二课/DataDrivenFrameWork/RunTest.py"
2018-09-04 13:51:10 TestAddCustomer.py[line:45] INFO 测试理财师添加客户信息执行开始
2018-09-04 13:51:53 TestAddCustomer.py[line:94] INFO 添加客户 13512345644 成功
2018-09-04 13:51:53 TestAddCustomer.py[line:110] INFO 断言关键字 李四 成功
2018-09-04 13:51:53 TestAddCustomer.py[line:113] INFO 客户 张三 被忽略执行
2018-09-04 13:51:53 TestAddCustomer.py[line:114] INFO contactNum = 1,isExecuteNum = 1
2018-09-04 13:51:53 TestAddCustomer.py[line:122] INFO 为用户defang1添加1个客户，1个成功!
2018-09-04 13:51:56 TestAddCustomer.py[line:126] INFO 用户defang2被设置为忽略执行!
```

作业：

1、完成上面的数据驱动框架代码，并思考

Thanks!

科　技　提　升　投　资　品　质

金斧子
WWW.JFZ.COM