|  |
| --- |
| 贵州易鲸捷信息技术有限公司 |
| OM\_AUTO\_TEST自动化代码说明文档 |

目录

[1 目录介绍 3](#_Toc156233669)

[2 Config文件夹 4](#_Toc156233670)

[2.1 sys\_config.ini 文件： 4](#_Toc156233671)

[2.2 email\_config文件 4](#_Toc156233672)

[3 Data文件夹 5](#_Toc156233673)

[3.1 Deploy 5](#_Toc156233674)

[3.1.1 deploy\_clusterlist.json 5](#_Toc156233675)

[3.2 web\_login 5](#_Toc156233676)

[3.2.1 login\_data.json 5](#_Toc156233677)

[3.3 data文件夹下,新增规则： 6](#_Toc156233678)

[4 Doc文件夹 6](#_Toc156233679)

[5 Html文件夹 6](#_Toc156233680)

[6 Img文件夹 6](#_Toc156233681)

[7 Lib文件夹 7](#_Toc156233682)

[7.1 base.py文件 7](#_Toc156233683)

[7.2 configInfo.py文件 8](#_Toc156233684)

[7.3 execCmd.py文件 9](#_Toc156233685)

[7.4 getDriver.py文件 10](#_Toc156233686)

[7.5 getElements.py文件 11](#_Toc156233687)

[7.6 JsonReader.py文件 12](#_Toc156233688)

[7.7 Logger.py文件 12](#_Toc156233689)

[7.8 sendMail.py文件 14](#_Toc156233690)

[7.9 sftpClient.py文件 16](#_Toc156233691)

[7.10 sshClient.py文件 19](#_Toc156233692)

[8 Log文件夹 20](#_Toc156233693)

[9 Page文件夹 20](#_Toc156233694)

[9.1 \_\_init\_\_.py文件 20](#_Toc156233695)

[9.2 page\_om\_login.py文件 22](#_Toc156233696)

[9.3 page\_dashboard\_services.py文件 23](#_Toc156233697)

[10 Server文件夹 27](#_Toc156233698)

[10.1 serverConfig.py 27](#_Toc156233699)

[10.2 serverOperations.py文件 27](#_Toc156233700)

[11 Shell文件夹 28](#_Toc156233701)

[12 Src文件夹 28](#_Toc156233702)

[12.1 test\_login.py文件 29](#_Toc156233703)

[12.2 test\_dashboard\_services.py文件 31](#_Toc156233704)

[13 Main.py 32](#_Toc156233705)

# 目录介绍

om\_auto\_test/

├── config: 存放配置文件

├── data: 存放json数据

├── doc: 存放测试用例的excel

├── html: 存放由模块htmltestreport 生成的测试报告

├── img: 存放截屏信息

├── lib: 存放基础类

├── log: 存放日志

├── main.py: 自动化主函数入口

├── page: 页面对象

├── server: server端操作

├── shell：存放shell脚本

└── src： 测试用例

# Config文件夹

## sys\_config.ini 文件：

存放自动化配置参数：

|  |
| --- |
| [omAutoTest]  om\_auto\_test\_path = /opt/om\_auto\_test  [sshConnection]  host\_list=10.14.40.184,10.14.40.185  port = 22  ssh\_name=root  ssh\_passwd=esg123.com  mail\_list=xin.zhao@esgyn.cn  [omclient]  web\_path = http://10.14.40.184:30005/  license = qian-x-0ChDkJ5vNcppAgacpHH3iuwIuENuGw9kGGAIoKA  #chrome or firefox  webdriver = chrome |

## email\_config文件

存放邮件发送时，所需要的环境描述信息：

|  |
| --- |
| &emsp;OM 自动化测试报告如下：<br><br>  <strong>1. 硬件环境：</strong><br>  &emsp;a) CPU型号: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz<br>  &emsp;b) vCPU：48<br>  &emsp;c) 内存: 256G<br>  &emsp;d) 网络: Speed: 10000Mb/s 万兆网卡<br>  &emsp;e) SATA HDD硬盘: 6\*1.8T sata机械盘\*raid0:/data <br>  &emsp;f) SATA SSD硬盘: 无<br>  &emsp;g) NVMe SSD硬盘: 无<br>  <strong>2. 软件环境：</strong><br>  &emsp;a) 操作系统：Kylin Linux Advanced Server release V10 (Tercel)<br>  &emsp;b) 测试产品版本：v3.4.3-RC18-0<br>  &emsp;c) 测试数据库版本：xTP v9.0.1-xTP<br> |

# Data文件夹

用于存放每个页面下，测试用例所需的测试数据和期望数据；

## Deploy

### deploy\_clusterlist.json

部署-集群页面的数据文件，以及格式；

|  |
| --- |
| {  "test\_deploy\_clusterlist\_01": {  "expected\_clust\_name": "zx"  },  "test\_deploy\_clusterlist\_02": {  "expected\_clust\_type": "XTP"  },  "test\_deploy\_clusterlist\_03": {  "expected\_clust\_status": "安装 完成"  }  } |

## web\_login

### login\_data.json

登录页面的数据文件，以及格式；

说明：例如test\_om\_login\_error\_user\_01为测试用例的函数名称，{}内是该函数需要的测试数据和期望数据；

|  |
| --- |
| {  "test\_om\_login\_error\_user\_01": {  "username": "qbadmin123",  "password": "qbadmin",  "expected\_error\_message": "用户名或者密码错误"  },  "test\_om\_login\_error\_pwd\_02": {  "username": "qbadmin",  "password": "qbadmin123",  "expected\_error\_message": "用户名或者密码错误"  },  "test\_om\_login\_success\_03": {  "username": "admin",  "password": "admin",  "expected\_welcome\_message": "部署"  }  } |

## data文件夹下,新增规则：

|  |
| --- |
| 1. 命名方式，请按照页面的英文名，命名； 2. Jason文件，请按照测试页面的英文名+.json |

# Doc文件夹

存放测试用例：OM v3.4.3-RC17-zyh测试用例.xlsx

# Html文件夹

存放测试报告：test\_report.html

# Img文件夹

存放截图

# Lib文件夹

## base.py文件

描述：该文件定义了Base类，用于对页面元素进行操作的基础方法封装，包括初始化方法、查找元素方法、点击方法、输入方法、获取文本方法、截图方法以及判断元素是否存在方法。

|  |
| --- |
| from selenium.webdriver.support.ui import WebDriverWait  from selenium.webdriver.support import expected\_conditions as EC  from lib.Logger import logger  import os  import time  class Base:  # 初始化方法  def \_\_init\_\_(self, driver):  """  初始化函数，接收 WebDriver 对象。  :param driver: WebDriver 对象  """  logger.info("正在初始化,driver对象:{}".format(driver))  self.driver = driver  # 查找元素方法  def base\_find(self, loc, timeout=15, poll=0.5):  logger.info("正调查找击元素：{}".format(loc))  return WebDriverWait(self.driver, timeout, poll).until(lambda x: x.find\_element(\*loc))  # 点击方法  def base\_click(self, loc):  logger.info("正调用点击元素：{}".format(loc))  element = self.base\_find(loc)  element.click()  # 输入方法  def base\_input(self, loc, value):  logger.info("正调用输入元素方法：{} 输入内容：{}".format(loc,value))  # 获取元素  el = self.base\_find(loc)  # 清空  logger.info("输入框清空前的值：{}".format(el.get\_attribute("value")))  # self.driver.execute\_script("arguments[0].value = '';", el) # 使用 JavaScript 清空输入框  time.sleep(1)  el.clear()  # 输入  WebDriverWait(self.driver, 10).until(EC.presence\_of\_element\_located(loc))  el.send\_keys(value)  # 获取文本方法  def base\_get\_text(self, loc):  logger.info("正调用获取元素信息方法：{}".format(loc))  return self.base\_find(loc).text  # 截图方法  def base\_get\_img(self):  logger.info("正调用截图方法")  img\_path = os.path.join(os.getcwd(), "img","{}.png".format(time.strftime("%Y-%m-%d\_%H-%M-%S")))  self.driver.get\_screenshot\_as\_file(img\_path)    # 判断元素是否存在  def base\_element\_is\_exist(self, loc, timeout=5):  try:  self.base\_find(loc, timeout=timeout)  logger.info("元素{}, 存在!".format(loc))  return True  except:  logger.warning("元素{}, 不存在!".format(loc))  return False |

## configInfo.py文件

描述：该文件定义了omInfo类，用于从配置文件中读取相关配置信息并进行初始化。包括对系统配置文件进行解析，以获取om\_auto\_test\_path、host\_list、port、ssh\_name、ssh\_passwd、mail\_list、web\_path、license、以及webdriver信息，并将其存储为类属性。

同时，文件最后实例化了omInfo类并将其存储为Userinof变量。

|  |
| --- |
| #!/usr/bin/env python  #-\*- coding:UTF-8 -\*-  import configparser  class omInfo:  om\_auto\_test\_path=None  host\_list=None  port = None  ssh\_name=None  ssh\_passwd=None  web\_path=None  license=None  mail\_list=None  def \_\_init\_\_(self):  sys\_config\_path = "config/sys\_config.ini"  config = configparser.RawConfigParser()  config.read(sys\_config\_path)    #omAutoTest  self.om\_auto\_test\_path = config.get('omAutoTest','om\_auto\_test\_path')  #sshConnection  self.host\_list = config.get('sshConnection','host\_list').split(',')  self.port = config.get('sshConnection','port')  self.ssh\_name = config.get('sshConnection','ssh\_name')  self.ssh\_passwd = config.get('sshConnection','ssh\_passwd')  self.mail\_list = config.get('sshConnection','mail\_list').split(',')  #omclient  self.web\_path = config.get('omclient','web\_path')  self.license = config.get('omclient','license')  self.webdriver = config.get('omclient','webdriver')    Userinof = omInfo() |

## execCmd.py文件

描述：该文件定义了ExecCmd类，用于执行远程命令。其中包括exec\_stmt方法，用于执行通过SSH连接的远程命令，记录相关日志及处理命令的输入和输出。

文件末尾的代码实例化了ExecCmd类并将其exec\_stmt方法存储为ssh\_exec变量。

|  |
| --- |
| from lib.Logger import logger  class ExecCmd:  def exec\_stmt(ssh, cmd):  logger.info("Preparing to execute command: " + cmd)  stdin, stdout, stderr = ssh.exec\_command(cmd)    # 处理错误输出  for line in stderr:  logger.error("Error while preparing to execute " + cmd + ": " + line.strip())  if "error" in line.strip().lower():  raise Exception("Error in command execution: " + line.strip())  output\_lines = []  # 处理正常输出  for line in stdout:  logger.info("Output from preparing to execute " + cmd + ": " + line.strip())  output\_lines.append(line.strip())  return "\n".join(output\_lines)  ssh\_exec = ExecCmd.exec\_stmt |

## getDriver.py文件

描述：该文件定义了GetDriver类，用于获取和管理WebDriver实例。其中包括get\_web\_driver方法用于获取特定类型的WebDriver实例并最大化窗口，quit\_web\_driver方法用于退出当前的WebDriver实例。

文件引用了selenium中的webdriver模块

|  |
| --- |
| from selenium import webdriver  from lib.configInfo import \*  from lib.Logger import logger  class GetDriver:  \_\_web\_driver = None  @classmethod  def get\_web\_driver(cls, browser=Userinof.webdriver):  if cls.\_\_web\_driver is None :  try:  if browser.lower() == "chrome":  cls.\_\_web\_driver = webdriver.Chrome()  elif browser.lower() == "firefox":  cls.\_\_web\_driver = webdriver.Firefox()  else:  raise ValueError("Invalid browser name. Supported browsers: Chrome, Firefox")  cls.\_\_web\_driver.maximize\_window()  cls.\_\_web\_driver.get(Userinof.web\_path)  except Exception as e:  logger.error(f"Error creating WebDriver: {str(e)}")  return cls.\_\_web\_driver    @classmethod  def quit\_web\_driver(cls):  if cls.\_\_web\_driver:  try:  cls.\_\_web\_driver.quit()  except Exception as e:  logger.warning(f"Error quitting WebDriver: {str(e)}")  finally:  cls.\_\_web\_driver = None |

## getElements.py文件

描述：该文件定义了GetElements类，用于从指定URL获取页面元素。其中包括get\_elements方法，用于使用requests模块获取URL的页面内容，并使用BeautifulSoup库进行解析后返回BeautifulSoup对象。

文件引用了BeautifulSoup和requests模块。

|  |
| --- |
| from bs4 import BeautifulSoup  import requests  class GetElements:    def get\_elements(self,url):  response = requests.get(url)  content = response.text  # 创建BeautifulSoup对象并指定解析器为lxml  soup = BeautifulSoup(content, 'lxml')  return soup |

## JsonReader.py文件

描述：该文件定义了JsonReader类，用于读取和解析JSON文件数据。其中包括read\_json方法，用于从指定路径读取JSON文件并返回其内容，和get\_test\_data方法，用于获取特定测试方法名下的测试数据。

文件引用了json模块。

|  |
| --- |
| import json  class JsonReader:  def \_\_init\_\_(self, file\_path):  self.file\_path = file\_path  def read\_json(self):  with open(self.file\_path, "r", encoding="utf-8") as file:  json\_data = json.load(file)  # print(f"在JSON类中读取的结果: {self.file\_path}: {json\_data}")  return json\_data  def get\_test\_data(self, test\_method\_name):  test\_data = self.read\_json()  return test\_data.get(test\_method\_name, {}) |

## Logger.py文件

描述：该文件定义了Logger类，用于记录和输出日志信息。其中包括初始化方法\_\_init\_\_用于配置日志处理器，info、warning、error方法用于记录不同级别的日志，以及get\_file\_handler、get\_logger方法用于返回日志处理器和新的logger实例。

文件引用了logging和time模块，以及logging.handlers模块。

|  |
| --- |
| #!/usr/bin/env python  #-\*- coding:UTF-8 -\*-  import logging  import time  import logging.handlers  class Logger:  # out = '../log/'  out = 'log/'  # log\_f = '{:\*^120s}'  logger = None  # pauseHa = 900  # errPid = 999999999  def \_\_init\_\_(self):  data = time.strftime("%Y%m%d-%H%M", time.localtime())  self.out = self.out + data + '.out'  with open(self.out, 'w+', encoding='utf-8') as f:  f.write("")  formatString = logging.Formatter('%(asctime)s - %(levelname)s : %(message)s')  self.logger = logging.getLogger()  self.logger.setLevel(logging.INFO)  streamHandler = logging.StreamHandler()  streamHandler.setFormatter(formatString)  fileHandler = logging.FileHandler(self.out, 'a+', encoding='utf-8')  fileHandler.setFormatter(formatString)  self.logger.addHandler(streamHandler)  self.logger.addHandler(fileHandler)  self.logger.info("")  self.logger.info("")  def info(self, message='', s=1, \*\*kwargs):  message = str(message)  if s:  # string = self.log\_f.format(string)  message = message  else:  message = message  self.logger.info(message,\*\*kwargs)  def warning(self, string=''):  string = str(string)  string = string  self.logger.warning(string)  def error(self, string=''):  string = str(string)  # string = self.log\_f.format(string)  self.logger.error(string)  def get\_file\_handler(self):  """返回一个将日志记录到文件中的处理器(FileHandler)"""  file\_handler = logging.handlers.RotatingFileHandler(  self.out, mode='a', maxBytes=10\*1024\*1024, backupCount=10, encoding='utf-8')  formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')  file\_handler.setFormatter(formatter)  return file\_handler  def get\_logger(self, logger\_name):  """返回一个新的logger实例（加了FileHandler）"""  logger = logging.getLogger(logger\_name)  logger.setLevel(logging.INFO)  logger.addHandler(self.get\_file\_handler())  return logger  logger = Logger()  # logger = Logger.logger  date = Logger.out  # Example usage  logger.info("Starting OM testing...")  try:  # some code here  logger.info("Code executed successfully")  except Exception as e:  logger.error("An error occurred: {}".format(str(e))) |

## sendMail.py文件

描述：该文件定义了SendMail类，用于构建和发送邮件。包括read\_email\_config方法用于读取邮箱配置内容，make\_html\_content方法用于构建HTML格式的邮件内容，send方法用于发送邮件，其中包括连接到SMTP服务器，构建邮件内容并附件，并发送邮件。

文件引用了smtplib、os、datetime、email.mime中的相关模块，同时引用了Logger类和configInfo模块

|  |
| --- |
| import smtplib  import os  from datetime import datetime  from email.mime.multipart import MIMEMultipart  from email.mime.application import MIMEApplication  from email.mime.text import MIMEText  from lib.Logger import logger  from lib.configInfo import \*  class SendMail:  smtpsrvr = "mail.esgyn.cn" # SMTP server address  smtpport = 587 # SMTP server port  username = "publicuser@esgyn.cn" # SMTP username  password = "D85vR42tt2" # SMTP password  sender = "publicuser@esgyn.cn" # Sender email address  def read\_email\_config(cls):  email\_config = os.path.join(os.getcwd(),'config','email\_config')  with open(email\_config,'r', encoding='utf-8') as f:  content = f.read()    return content    # 构建邮件内容  def make\_html\_content(cls):  content = cls.read\_email\_config()  html\_content = f"""  <html>  <body>  Hi all:<br>  {content}  <strong>3. 测试结果及报告:</strong><br>  &emsp;&emsp;请查看附件: test\_report.html <br>  Best Regards,<br>  xin<br>  </body>  </html>  """  return html\_content  def send(cls,filename):  message=cls.make\_html\_content()  recipient\_list=Userinof.mail\_list  today = datetime.now().strftime('%Y%m%d')  subject = f"[{today}] OM Auto Test "  try:  server = smtplib.SMTP(cls.smtpsrvr, cls.smtpport) # Connect to the SMTP server  server.starttls() # Start TLS encryption  server.login(cls.username, cls.password) # Login to the SMTP server  msg = MIMEMultipart() # Create a new email message  msg["From"] = cls.sender # Set the sender of the email  msg["To"] = ", ".join(recipient\_list) # Set the recipients of the email  msg["Subject"] = subject # Set the subject of the email  msg.attach(MIMEText(message, "html")) # Add the plain text message to the email  # 添加附件  with open(filename, 'rb') as f:  attachment = MIMEApplication(f.read(), 'html')  attachment.add\_header('Content-Disposition', 'attachment', filename=os.path.basename(filename))  msg.attach(attachment)  server.send\_message(msg) # Send the email  server.quit() # Disconnect from the SMTP server  logger.info("Email sent successfully!")  return True  except smtplib.SMTPException as e:  logger.error(e)  return False    mail=SendMail() |

## sftpClient.py文件

描述：该文件定义了SFTPClient类，用于与SFTP服务器进行文件传输操作。包括初始化方法\_\_init\_\_用于连接SFTP服务器，uploadDir方法用于上传本地目录到远程服务器，uploadFile方法用于上传本地文件到远程服务器，upload方法根据文件类型进行上传操作，close方法用于关闭SFTP连接。

文件引用了os、sys、paramiko模块，并引用了Logger模块。

|  |
| --- |
| import os  import sys  import paramiko  from lib.Logger import logger  \_XFER\_FILE = 1  \_XFER\_DIR = 2  class SFTPClient:  def \_\_init\_\_(self, remote\_path, host, user, password, port=22):  """  初始化SFTP客户端。  param remote\_path: 将文件上传到的远程路径。  param host: SFTP服务器主机。  param user: SFTP用户名。  param password: SFTP密码。  param port: SFTP服务器端口(默认为22)。  """  self.host = host  self.remote\_path = remote\_path  self.transport = paramiko.Transport((self.host, port))  self.transport.connect(username=user, password=password)  self.sftp = paramiko.SFTPClient.from\_transport(self.transport)  def uploadDir(self, local\_dir='./', remote\_dir='./'):  """  将本地目录上传到远程SFTP服务器。  param local\_dir: 要上传的本地目录路径。  param remote\_dir: 要上传到的远程目录路径。  """  if not os.path.isdir(local\_dir):  return  for file in os.listdir(local\_dir):  src = os.path.join(local\_dir, file)  if os.path.isfile(src):  self.uploadFile(src, file, os.path.join(self.remote\_path, remote\_dir))  elif os.path.isdir(src):  try:  logger.info('子路径 ==', file)  self.sftp.mkdir(os.path.join(self.remote\_path, remote\_dir, file))  except Exception as e:  sys.stderr.write(f'创建目录时出错：{e}')  self.uploadDir(src, os.path.join(remote\_dir, file, '/'))  def uploadFile(self, local\_path, remote\_filename, remote\_path):  """  将本地文件上传到远程SFTP服务器。  param local\_path: 要上传的本地文件路径。  param remote\_filename: 远程文件名。  param remote\_path: 远程目录路径。  """  if not os.path.isfile(local\_path):  return  file\_path = os.path.join(os.path.dirname(os.path.realpath(\_\_file\_\_)) + '/../', local\_path)  logger.info('+++ 上传 {} 到 {}:{}'.format(file\_path, self.host, os.path.join(remote\_path, remote\_filename)))  try:  self.sftp.put(file\_path, os.path.join(remote\_path, remote\_filename))  except Exception as e:  sys.stderr.write(f'上传文件时出错：{e}')  def \_\_filetype(self, src):  """  确定源文件类型（文件或目录）。  param src: 源文件路径。  return: 包含文件类型（文件或目录）和文件名的元组。  """  if os.path.isfile(src):  index = src.rfind('\\') if '\\' in src else src.rfind('/')  return \_XFER\_FILE, src[index + 1:]  elif os.path.isdir(src):  return \_XFER\_DIR, ''  def upload(self, src):  """  将文件或目录上传到远程SFTP服务器。  param src: 源文件或目录路径。  """  filetype, filename = self.\_\_filetype(src)  logger.info("文件类型 =={},文件名 =={}" .format(filetype,filename))  if filetype == \_XFER\_DIR:  self.srcDir = src  self.uploadDir(self.srcDir)  elif filetype == \_XFER\_FILE:  self.uploadFile(src, filename, self.remote\_path)  self.close()  def close(self):  """关闭SFTP连接。"""  self.sftp.close()  self.transport.close()  # sftp\_client = SFTPClient('/opt/om\_auto\_test', '10.14.40.184', 'root', 'esg123.com')  # sftp\_client.upload('../shell') |

## sshClient.py文件

描述：该文件定义了sshClient类，用于通过SSH协议与远程服务器建立连接。包括初始化方法\_\_init\_\_用于设置SSH连接的基本信息，conn方法用于建立SSH连接，close方法用于关闭SSH连接。

文件引用了paramiko模块和configInfo模块中的Userinof类。

|  |
| --- |
| #!/usr/bin/env python  # -\*- coding:UTF-8 -\*-  import paramiko  from lib.configInfo import \*  class sshClient:  def \_\_init\_\_(self):  self.ssh = paramiko.SSHClient()  self.username = Userinof.ssh\_name  self.password = Userinof.ssh\_passwd  self.port = Userinof.port  def conn(self,ip):  try:  self.ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())  self.ssh.connect(hostname=ip, port=self.port, username=self.username, password=self.password)  return self.ssh  except:  print("ssh链接失败!!!")  return False  def close(self):  if self.ssh :  self.ssh.close()  sshclient = sshClient() |

# Log文件夹

用于存放产生的日志

# Page文件夹

用于存放OM每一个页面对象；

## \_\_init\_\_.py文件

描述：该文件包含了用于标识网页元素的选择器常量，按页面和全局元素类别进行组织。每个元素都使用了By模块提供的CSS\_SELECTOR和XPATH等属性进行标识。通过组织这些元素选择器，可以方便地在页面对象模型(Page Object Model)中使用和维护这些元素定位信息。

|  |
| --- |
| from selenium.webdriver.common.by import By  """  page\_om\_login页面  """  #用户名  web\_login\_username = By.CSS\_SELECTOR,".ant-input#userName"  #密码  web\_login\_pwd = By.CSS\_SELECTOR,".ant-input#password"  #登录  web\_login\_btn = By.CSS\_SELECTOR, ".ant-btn.antd-pro-pages-user-login-components-login-index-submit.ant-btn-primary.ant-btn-lg.ant-btn-two-chinese-chars"  #登录成功信息  web\_success\_login = By.CSS\_SELECTOR,"#root > div > section > aside > div > ul > li.ant-menu-submenu.ant-menu-submenu-inline.ant-menu-submenu-selected > div > span > span:nth-child(2)"  #异常信息  web\_login\_error = By.CSS\_SELECTOR, "#root > div > div.antd-pro-layouts-user-layout-content > div.antd-pro-pages-user-login-style-main > div > form > div.ant-alert.ant-alert-error > span.ant-alert-message"  """  global  """  components\_global\_header = By.CSS\_SELECTOR, '.antd-pro-components-global-header-index-action .antd-pro-components-global-header-index-badge'  components\_global\_header\_option = By.CSS\_SELECTOR, 'div.ant-modal-content'  """  page\_deploy\_clustlist页面  """  deploy\_clustlist= By.CSS\_SELECTOR , "div.ant-table-content"  """  page\_dashboard\_services页面  """  #监控  dashboard\_page = By.XPATH, '//\*[@id="root"]/div/section/aside/div/ul/li[2]/div/span/span[2]'  #服务  dashboard\_services\_page = By.XPATH, '//a[contains(@href, "/xtp/dashboard/services")]'  #刷新节点数  dashboard\_services\_Refresh\_node\_count\_button = By.XPATH, '//\*[@id="root"]/div/section/section/div[1]/main/div/div/div/div[2]/div/div/div[1]/div[1]/div/div[2]/div/div[1]/button/span'  #服务管理  dashboard\_services\_management\_button = By.XPATH, '//\*[@id="root"]/div/section/section/div[1]/main/div/div/div/div[2]/div/div/div[1]/div[1]/div/div[2]/div/div[2]/button/span'  #主机  dashboard\_services\_host\_button = By.XPATH, "//button[contains(@class, 'ant-btn-default') and contains(@class, 'ant-btn-two-chinese-chars')]"  #database stop  dashboard\_service\_database\_stop\_button = By.XPATH, "//button[@class='ant-btn ant-btn-sm' and contains(span/@class, 'antd-pro-pages-v2-services-index-stop-disabled')]"  #database start  dashboard\_service\_database\_start\_button = By.XPATH, "//button[@class='ant-btn ant-btn-sm' and contains(span/@class, 'antd-pro-pages-v2-services-index-start-icon')]"  #database restart  dashboard\_service\_database\_restart\_button = By.XPATH, "//button[@class='ant-btn ant-btn-sm' and contains(span/@class, 'antd-pro-pages-v2-services-index-restart-icon')]"  #确认是否重启：  dashboard\_service\_database\_yes = By.XPATH,"//div[@class='ant-modal-confirm-btns']//button[@class='ant-btn ant-btn-primary']/span[text()='是']"  # 主机数  dashboard\_service\_hosts = By.XPATH, "//div[@class='antd-pro-pages-v2-services-index-cluster-item-label' and contains(text(), '主机数')]"  #获取实例数  dashboard\_service\_instances = By.XPATH, "//div[@class='antd-pro-pages-v2-services-index-cluster-item-label' and contains(text(), '实例数')]"  #角色表  dashboard\_service\_role = By.XPATH, '//\*[@id="ant-design-pro-table"]/div/div/div[2]/div/div/div/div/div/table/tbody'  dashboard\_service\_role\_thead = By.XPATH,'//\*[@id="ant-design-pro-table"]/div[1]/div/div[2]/div/div/div/div/div/table/thead/tr'  dashboard\_service\_role\_tbody = By.XPATH, '//\*[@id="ant-design-pro-table"]/div/div/div[2]/div/div/div/div/div/table/tbody/tr/td'  #选中的操作（需要选择角色，才能操作，按钮默认是disable）  dashboard\_service\_role\_dropdown = By.CSS\_SELECTOR,"#ant-design-pro-table > div.ant-card > div > div.ant-pro-table-toolbar > div.ant-pro-table-toolbar-title" |

## page\_om\_login.py文件

描述：该代码段是一个基于Selenium和页面对象模型（Page Object Model）的自动化测试脚本。它实现了一个名为PageOmLogin的页面操作类，继承自Base类。在该类中，包含了一系列页面操作方法，如输入用户名、密码，点击登录按钮，获取信息，点击异常信息，截屏等。在方法定义中使用了页面元素的定位器来定位元素并执行相应的操作。此外，还提供了一个组合操作方法page\_login，用于模拟用户登录流程。

这段代码是用于自动化测试的关键部分，它封装了页面操作的逻辑，并通过调用Base类中的方法来实现对页面元素的操作，实现了测试脚本的模块化和可维护性。

|  |
| --- |
| from lib.base import Base  import page  class PageOmLogin(Base):  #输入用户名  def page\_input\_username(self,username):  self.base\_input(page.web\_login\_username,username)  #输入密码  def page\_input\_password(self,password):  self.base\_input(page.web\_login\_pwd,password)  #点击登录按钮  def page\_click\_login\_btn(self):  self.base\_click(page.web\_login\_btn)    #获取信息  def page\_get\_text(self):  welcome\_info = self.base\_get\_text(page.web\_success\_login)  return welcome\_info  #点击异常信息  def page\_get\_error\_info(self):  return self.base\_get\_text(page.web\_login\_error)  #截屏  def page\_get\_img(self):  self.base\_get\_img()    #组合操作  def page\_login(self,username="admin", pwd="admin"):  self.page\_input\_username(username)  self.page\_input\_password(pwd)  self.page\_click\_login\_btn() |

## page\_dashboard\_services.py文件

描述：该文件定义了DashboardServices类，该类继承自PageOmLogin类，包含了各种对仪表盘服务页面进行操作的方法，例如点击监控、点击服务、刷新节点数、服务管理、主机操作、DB server的停止、启动、重启和确认操作等。这些方法基于Selenium的WebDriver进行页面元素的定位和操作，通过调用Base类中定义的方法来实现对元素的操作。

|  |
| --- |
| import page  from page.page\_om\_login import PageOmLogin  from selenium.webdriver.common.by import By  from selenium.webdriver.support.ui import WebDriverWait  from selenium.webdriver.support import expected\_conditions as EC  from lib.Logger import logger  class DashboardServices(PageOmLogin):  #点击监控  def dashboard\_page(self):  self.base\_click(page.dashboard\_page)    #点击服务  def dashboard\_service\_page(self):  WebDriverWait(self.driver, 10).until(  EC.element\_to\_be\_clickable(page.dashboard\_services\_page)  )  self.base\_click(page.dashboard\_services\_page)    #刷新节点数  def dashboard\_service\_Refresh\_node\_count(self):  self.base\_click(page.dashboard\_services\_Refresh\_node\_count\_button)  #服务管理  def dashboard\_service\_management(self):  self.base\_click(page.dashboard\_services\_management\_button)  #主机  def dashboard\_service\_host(self):  self.base\_click(page.dashboard\_services\_host\_button)  # DB server stop  def dashboard\_service\_database\_stop(self):  WebDriverWait(self.driver, 10).until(  EC.element\_to\_be\_clickable(page.dashboard\_service\_database\_stop\_button)  )  button\_element = self.base\_find(page.dashboard\_service\_database\_stop\_button)  if button\_element and not button\_element.is\_enabled():  logger.info("DB server stop button is disabled.")  return False # 返回 False 表示按钮不可点击  self.base\_click(page.dashboard\_service\_database\_stop\_button)  # DB server start  def dashboard\_service\_database\_start(self):  WebDriverWait(self.driver, 10).until(  EC.element\_to\_be\_clickable(page.dashboard\_service\_database\_start\_button)  )  button\_element = self.base\_find(page.dashboard\_service\_database\_start\_button)  if button\_element and not button\_element.is\_enabled():  logger.info("DB server start button is disabled.")  return False # 返回 False 表示按钮不可点击  self.base\_click(page.dashboard\_service\_database\_start\_button)  # DB server restart  def dashboard\_service\_database\_restart(self):  WebDriverWait(self.driver, 10).until(  EC.element\_to\_be\_clickable(page.dashboard\_service\_database\_restart\_button)  )  button\_element = self.base\_find(page.dashboard\_service\_database\_restart\_button)  if button\_element and not button\_element.is\_enabled():  logger.info("DB server restart button is disabled.")  return False # 返回 False 表示按钮不可点击  self.base\_click(page.dashboard\_service\_database\_restart\_button)  #操作数据库点击确定  def dashboard\_service\_database\_yes(self):  WebDriverWait(self.driver, 10).until(  EC.element\_to\_be\_clickable(page.dashboard\_service\_database\_yes)  )  self.base\_click(page.dashboard\_service\_database\_yes)  #获取主机数  def dashboard\_service\_hosts\_num(self):  return self.base\_get\_text(page.dashboard\_service\_hosts)  #获取实例数  def dashboard\_service\_instances\_num(self):  return self.base\_get\_text(page.dashboard\_service\_instances)  #角色信息  def dashboard\_service\_relo(self):  # 获取特定tr元素内的td元素  td\_elements = WebDriverWait(self.driver, 10).until(  EC.visibility\_of\_all\_elements\_located(page.dashboard\_service\_role\_tbody)  )  role\_info\_list = [] # 用于存储角色信息的列表  current\_row\_data = []  for td in td\_elements:  row\_data = td.text  if row\_data:  current\_row\_data.append(row\_data)  else:  # 当 row\_data 为空时，将 current\_row\_data 添加到 role\_info\_list，并重置 current\_row\_data  if current\_row\_data:  role\_info\_list.append(current\_row\_data)  current\_row\_data = []  logger.info(role\_info\_list)  return role\_info\_list    #角色表名  def dashboard\_service\_relo\_name(self):  # 获取特定tr元素  td\_elements = WebDriverWait(self.driver, 10).until(  EC.visibility\_of\_all\_elements\_located(page.dashboard\_service\_role\_thead)  )  role\_name\_list = [] # 用于存储角色信息的列表  # print(td\_elements)  for td in td\_elements:  role\_name\_list += td.text.split('\n')  logger.info(role\_name\_list)  return role\_name\_list    #点击最近的命令  def dashboard\_service\_global\_recent\_commands(self):  WebDriverWait(self.driver, 20).until(  EC.element\_to\_be\_clickable(page.components\_global\_header)  )  self.base\_click(page.components\_global\_header)  # 获取正在运行的命令  def dashboard\_service\_global\_option(self):  table\_element = WebDriverWait(self.driver, 20).until(  EC.visibility\_of\_all\_elements\_located(page.components\_global\_header\_option)  )  commands\_info = []  for td in table\_element:  commands\_info += td.text.split('\n')  # logger.info(commands\_info)  if commands\_info:  latest\_command\_info = commands\_info[9:13]  logger.info("最新执行的命令：{}".format(latest\_command\_info))  return latest\_command\_info  else:  logger.error("没有找到任何命令信息")  return None  #选择操作（需要选择角色，才能操作，按钮默认是disable）  def dashboard\_service\_relo\_operation(self):  pass  #组合  def dashboard\_service\_all(self):  self.dashboard\_page()  self.dashboard\_service\_page() |

# Server文件夹

该文件夹用存放数据库服务的操作类

## serverConfig.py

描述：该文件包含了一个名为sh\_check\_env\_xtp的变量，用于存储“check\_env\_xtp.sh”脚本的文件名。此脚本用于检查XTP数据库的健康状况。

添加方式：将脚本的名称放置到此文件中，然后通过：

from server.serverConfig import \* 调用

|  |
| --- |
| #!/usr/bin/env python  # -\*- coding:UTF-8 -\*-  # 检查XTP数据库健康状况脚本  sh\_check\_env\_xtp = "check\_env\_xtp.sh" |

## serverOperations.py文件

描述：ServerOptions类包含了一个用于服务器操作的方法check\_database\_xtp。在该方法中，首先使用SSH连接到服务器，并执行名为sh\_check\_env\_xtp的shell脚本，用于检查XTP数据库的健康状况。随后根据命令执行结果进行相应的日志记录和返回操作状态。该类还包含了一个\_\_init\_\_方法，用于初始化属性case\_path。在代码的最后创建了一个ServerOptions的实例，并将其赋值给了SPtions变量。

|  |
| --- |
| from lib.Logger import logger  from lib.sshClient import \*  from lib.execCmd import \*  from lib.configInfo import \*  from server.serverConfig import \*  class ServerOptions:  def \_\_init\_\_(self):  self.case\_path = Userinof.om\_auto\_test\_path + '/shell/'  def check\_database\_xtp(self):  logger.info("check\_database\_xtp...")  cmd = "sh " + self.case\_path + sh\_check\_env\_xtp  try:  result = ssh\_exec(sshclient.conn(Userinof.host\_list[0]), cmd)  # 处理命令执行结果  if "Success" in result:  logger.info("Database health check: Success!")  return True  else:  logger.error("Database health check: False!" )  return False  finally:  sshclient.close()  SPtions = ServerOptions() |

# Shell文件夹

用于存放数据库服务端的shell脚本

# Src文件夹

用于存放OM测试用例

目录结构：

└── src

├── dashboard

│   └── test\_dashboard\_services.py

├── deploy

│   └── test\_deploy\_clusterlist.py

└── web\_login

├── \_\_pycache\_\_

└── test\_login.py

每个子目录用于存放，每个OM模块的测试用例，分离开主要考虑是每个人独立负责测试模块，也方便分配；

## test\_login.py文件

描述：这是一个基于unittest框架的自动化测试脚本。该脚本包括了对OM系统登录页面的多个测试用例，分别为输入错误的用户名、输入错误的密码以及登录成功。其中，使用了Page Object模式对页面元素和操作进行了封装。

在每个测试用例中，首先从JSON文件获取测试数据，然后调用PageOmLogin类的页面操作方法进行操作，并获取相应的结果进行断言验证。测试执行之前，会在setUpClass中进行浏览器驱动的获取，在tearDownClass中进行驱动的关闭。

test\_om\_login\_success\_03用例中，还包含了获取页面文本和日志的操作。

|  |
| --- |
| import os  import sys  sys.path.append('./')  import unittest  from lib.getDriver import GetDriver  from page.page\_om\_login import PageOmLogin  from lib.JsonReader import JsonReader  from lib.Logger import logger  class TestLogin(unittest.TestCase):  driver = None  path\_date = os.path.join(os.getcwd(), "data", "web\_login","login\_data.json")  json\_reader = JsonReader(path\_date)  @classmethod  def setUpClass(cls):  #获取driver  cls.driver = GetDriver().get\_web\_driver()  cls.driver.delete\_all\_cookies()  cls.login = PageOmLogin(cls.driver)  @classmethod  def tearDownClass(cls):  GetDriver().quit\_web\_driver()    def setUp(self):  self.driver.refresh()  def test\_om\_login\_error\_user\_01(self):  # 输入错误的用户名  test\_data01 = self.json\_reader.get\_test\_data("test\_om\_login\_error\_user\_01")  username01 = test\_data01.get("username")  password01 = test\_data01.get("password")  expected\_error\_message = test\_data01.get("expected\_error\_message")  self.login.page\_login(username01, password01)  error\_info = self.login.page\_get\_error\_info()  self.assertIn(expected\_error\_message, error\_info)  def test\_om\_login\_error\_pwd\_02(self):  # 输入错误的密码  test\_data02 = self.json\_reader.get\_test\_data("test\_om\_login\_error\_pwd\_02")  username02 = test\_data02.get("username")  password02 = test\_data02.get("password")  expected\_error\_message = test\_data02.get("expected\_error\_message")  self.login.page\_login(username02, password02)  error\_info = self.login.page\_get\_error\_info()  self.assertIn(expected\_error\_message, error\_info)  def test\_om\_login\_success\_03(self):  #登录成功  test\_data03 = self.json\_reader.get\_test\_data("test\_om\_login\_success\_03")  username = test\_data03.get("username")  password = test\_data03.get("password")  expected\_welcome\_message = test\_data03.get("expected\_welcome\_message")  self.login.page\_login(username, password)  welcome\_info = self.login.page\_get\_text()  # self.login.page\_get\_img() #注：截屏占60KB  logger.info("查找到的元素是：{}".format(welcome\_info))  self.assertIn(expected\_welcome\_message, welcome\_info)  #调试方式  # if \_\_name\_\_ == '\_\_main\_\_':  # unittest.main(exit=False) |

## test\_dashboard\_services.py文件

描述：该文件包括了对仪表盘服务页面的数据库重启功能的测试用例。在测试用例中，首先使用Page Object模式进行页面操作，依次进行了仪表盘服务页面的重启数据库、确认并获取全局最近命令等操作，并获取重启数据库的命令信息。同时进行页面端和后端的断言验证，确保数据库启动命令状态为成功。在测试执行前，在setUpClass中进行了浏览器驱动的获取，在tearDownClass中进行了驱动的关闭。

|  |
| --- |
| #!/usr/bin/env python  #-\*- coding:UTF-8 -\*-  import os  import sys  import time  sys.path.append('./')  from lib.Logger import logger  from lib.JsonReader import JsonReader  from lib.configInfo import \*  import unittest  from lib.getDriver import GetDriver  from page.page\_dashboard\_services import DashboardServices  from server.serverOperations import \*  class TestDashboardServices(unittest.TestCase):  driver = None  path\_date = os.path.join(os.getcwd(), "data", "web\_login","login\_data.json")  json\_reader = JsonReader(path\_date)  @classmethod  def setUpClass(cls):  if not cls.driver:  cls.driver = GetDriver().get\_web\_driver()  cls.driver.delete\_all\_cookies()  cls.dashboardservices = DashboardServices(cls.driver)  cls.dashboardservices.page\_login()  @classmethod  def tearDownClass(cls):  GetDriver().quit\_web\_driver()    def test\_dashboard\_service\_01(self):  # 重启数据库  self.dashboardservices.dashboard\_service\_all()  time.sleep(1)  self.dashboardservices.dashboard\_service\_database\_restart()  time.sleep(1)  self.dashboardservices.dashboard\_service\_database\_yes()  time.sleep(3)  self.dashboardservices.dashboard\_service\_global\_recent\_commands()  time.sleep(3)  # 获取重启数据库的命令信息  command = None  max\_attempts = 10 # 最多重试10次，每次等待3秒  for \_ in range(max\_attempts):  command = self.dashboardservices.dashboard\_service\_global\_option()  logger.info(command)  if command and command[2] == "成功":  break  time.sleep(10)    # 页面端断言是否成功  self.assertIsNotNone(command, "未能获取到数据库启动命令信息")  self.assertEqual(command[2], "成功", "数据库启动命令状态不为成功")  # 后端断言是否成功  result\_database\_healthy = SPtions.check\_database\_xtp()  self.assertTrue(result\_database\_healthy, "测试用例失败！")  # if \_\_name\_\_ == '\_\_main\_\_':  # unittest.main(exit=False) |

# Main.py

类描述：OM\_test是一个用于测试和上传文件的类。

属性：

report\_path: 存储测试报告的路径

方法：

run\_tests(test\_suite): 该方法创建测试套件并运行测试用例，并生成HTML测试报告。

transport\_test\_files(): 该方法用于上传后端执行代码到所有节点。首先创建远程路径并清空目录，然后使用SFTPClient将代码上传到各个节点。

\_\_main\_\_: 在主方法中，OM\_test类实例化后，通过用户输入选择要测试的模块，并发送测试报告的邮件。

|  |
| --- |
| import sys  import unittest  from src.web\_login.test\_login import TestLogin  from src.dashboard.test\_dashboard\_services import TestDashboardServices  from htmltestreport import HTMLTestReport  from lib.sendMail import \*  from lib.sftpClient import SFTPClient  from lib.configInfo import \*  from lib.sshClient import \*  from lib.execCmd import \*  class OM\_test:    report\_path = "html/test\_report.html"    def run\_tests(self,test\_suite):  # 创建测试套件  test\_suites = unittest.TestLoader().loadTestsFromTestCase(test\_suite)  # 运行测试用例  result = HTMLTestReport(self.report\_path).run(test\_suites)  return result  #上传后端执行代码到所有节点  def transport\_test\_files(self):  remote\_path = Userinof.om\_auto\_test\_path + '/shell/'  logger.info(remote\_path)  #记录一下：这里需要写一个创建目录，如果目录存在，需要清空目录  cmd = 'rm -rf {} && mkdir -p {}'.format(remote\_path,remote\_path)  for ip in Userinof.host\_list:  ssh\_exec(sshclient.conn(ip),cmd)  sftp\_client = SFTPClient(remote\_path, ip, Userinof.ssh\_name, Userinof.ssh\_passwd)  sftp\_client.upload('shell/')    if \_\_name\_\_ == "\_\_main\_\_":  om\_test = OM\_test()  om\_test.transport\_test\_files()  module\_choice = input("请选择要测试的模块: web\_login|dashboard|all: ").strip().lower()  if module\_choice == 'dashboard':  om\_test.run\_tests(TestDashboardServices)  elif module\_choice == 'web\_login':  om\_test.run\_tests(TestLogin)  elif module\_choice == 'all':  om\_test.run\_tests(TestLogin)  om\_test.run\_tests(TestDashboardServices)  else:  logger.error("无效的选择！")  sys.exit()  mail.send(om\_test.report\_path) |