全程66期接口自动化 第八次：

回顾：

1上下游-------框架---难点：

|  |
| --- |
| 下游接口:  1 准备工具函数 1.1 find  1.2 relace （被替换数据 ${} 替换数据字典键）  # 准备工具代码  # 3.1.1判断data 是否有${}====》find  # 3.1.2 用getattr（Dynamic，${}的值，“默认值”）==》关联值  # 3.1.3 关联值 字符串替换${}===>relace替换  # 3.1.4 替换完值字符串转成字典方便给requests使用 ====>json.loads |

### 内容1： 实现下游接口函数：

接口定义函数参考：

|  |
| --- |
| # 3.1 def correlation(data):  def correlation(self,data):  # 准备工具代码  # 3.1.1判断data 是否有${}====》find  res\_data=Base.find(data)# 不确定的是列表中有几个值～  if res\_data: #['jwtoken']  for i in res\_data: # jwtoken 一一取出来～  # 3.1.2 用getattr（Dynamic，${}的值，“默认值”）==》关联值  linshi=getattr(DynamicParam,i,"没有取到临时值") # ezaasasa授权码～  # 3.1.3 关联值 字符串替换${}===>relace替换  # 被替换 data 数据库{"Authorization":"${jwtoken}"}  # 替换 linshi===>{"jwtoken": lingshizhi}  # 才能relace替换  relace\_dict={}  relace\_dict.update({i:linshi})# 替换数据构造完毕{"jwtoken": lingshizhi}  genghuan=Base.relace(data,relace\_dict)#"Authorization":"ezasa授权码～"  # 3.1.4 替换完值字符串转成字典方便给requests使用 ====>json.loads  data=json.loads(genghuan)  return data |

接口调用函数参考：

### descript

### 核心代码文件：

|  |
| --- |
| import json  import pytest  # 工具类：  from common import Base  from config.settings import DynamicParam  from utils.caseutil import RdTestCase  from utils.logutil import logger  from utils.requestutils import RequestSend  gongju2=RdTestCase() # 用例工具---》用工具1～  case\_list=gongju2.is\_run\_data('okr-api') # [{用例1}，{用例2}，{用例3}]  gongju4=RequestSend() # 发出接口工具  # 思想 跑用例： gongju2得到用例数据 交给 gongju4 使用  class TestApi:  @pytest.mark.parametrize('case',case\_list)  def test\_run(self,case): # 用例条数 决定了 test\_run 个数～  # 需求1：发接口时候 拼接接口服务器地址～  url=gongju2.loadConfKey('atstudy\_okr','url\_api')['value']+case['url']  method = case['method']  # 坑：来自于数据库 字符串---》字典才能给 工具4使用解决 eval（）  headers = eval(case['headers'])  #print("当前用例请求头类型",type(headers))  cookies = eval(case['cookies'])  data = eval(case['request\_body'])  relation =str( case['relation'])  case\_name=case['title']  try:  # 工具3  logger.info(f"主人，正在执行{case\_name}用例")  # 接口发送请求～  # 替换完成后才能交给 requests 发出接口请求～  print("志玲说：替换之前的数据",headers)  headers=self.correlation(headers)  print("志玲说：替换之后的数据", headers)  res\_data=gongju4.send(url,method,data=data,headers=headers,cookies=cookies)  if res\_data:# 接口发送成功 再调用～  if relation!=None: # 只有上游接口需要设置关联值  self.set\_relation(relation,res\_data)  #guanlian=res\_data['headers'].get("Authorization",'无临时值～')  logger.info(f"实际结果：{res\_data}")  #发送失败 断言失败 发送成功 不能说明断言成功 进一步比对预期和实际 ==》测试结果  except:  logger.error("用例执行失败，请进一步排查")  assert False  return res\_data  # 1def set\_relation(形参relation，实际结果res\_data):  def set\_relation(self,relation,res\_data):  # 健壮性 1验证 2 try方便定位报错代码～  try:  # 1.1判断relation 是否有值=非空=》上游用例  if relation:  # 1.2 切割 relation  jwlist=relation.split("=") #[jwtoken,Authorization]  # 1.2.1 =左边 ====》动态属性名字  zuobian=jwlist[0]  # 1.2.2 根据=右边到实际结果res\_data中查找===》动态属性值  youbian=jwlist[1]  guanlian=res\_data.get("headers").get(youbian,"无关联值")  # 1.3 setattr(类，动态属性名字，动态属性值)==》存放结束  setattr(DynamicParam,zuobian,guanlian)  #验证：  print("志玲验证是否存放成功:",getattr(DynamicParam,zuobian,"存放临时值失败～"))  except:  print("47~57行写的可能有问题～～")  # 3.1 def correlation(data):  def correlation(self,data):  # 准备工具代码  # 3.1.1判断data 是否有${}====》find  res\_data=Base.find(data)# 不确定的是列表中有几个值～  if res\_data: #['jwtoken']  for i in res\_data: # jwtoken 一一取出来～  # 3.1.2 用getattr（Dynamic，${}的值，“默认值”）==》关联值  linshi=getattr(DynamicParam,i,"没有取到临时值") # ezaasasa授权码～  # 3.1.3 关联值 字符串替换${}===>relace替换  # 被替换 data 数据库{"Authorization":"${jwtoken}"}  # 替换 linshi===>{"jwtoken": lingshizhi}  # 才能relace替换  relace\_dict={}  relace\_dict.update({i:linshi})# 替换数据构造完毕{"jwtoken": lingshizhi}  genghuan=Base.relace(data,relace\_dict)#"Authorization":"ezasa授权码～"  # 3.1.4 替换完值字符串转成字典方便给requests使用 ====>json.loads  data=json.loads(genghuan)  return data  if \_\_name\_\_ == '\_\_main\_\_':  pytest.main(["-s","-v","test\_run.py"]) |

恭喜 临时存储和使用 上下接口处理结束~

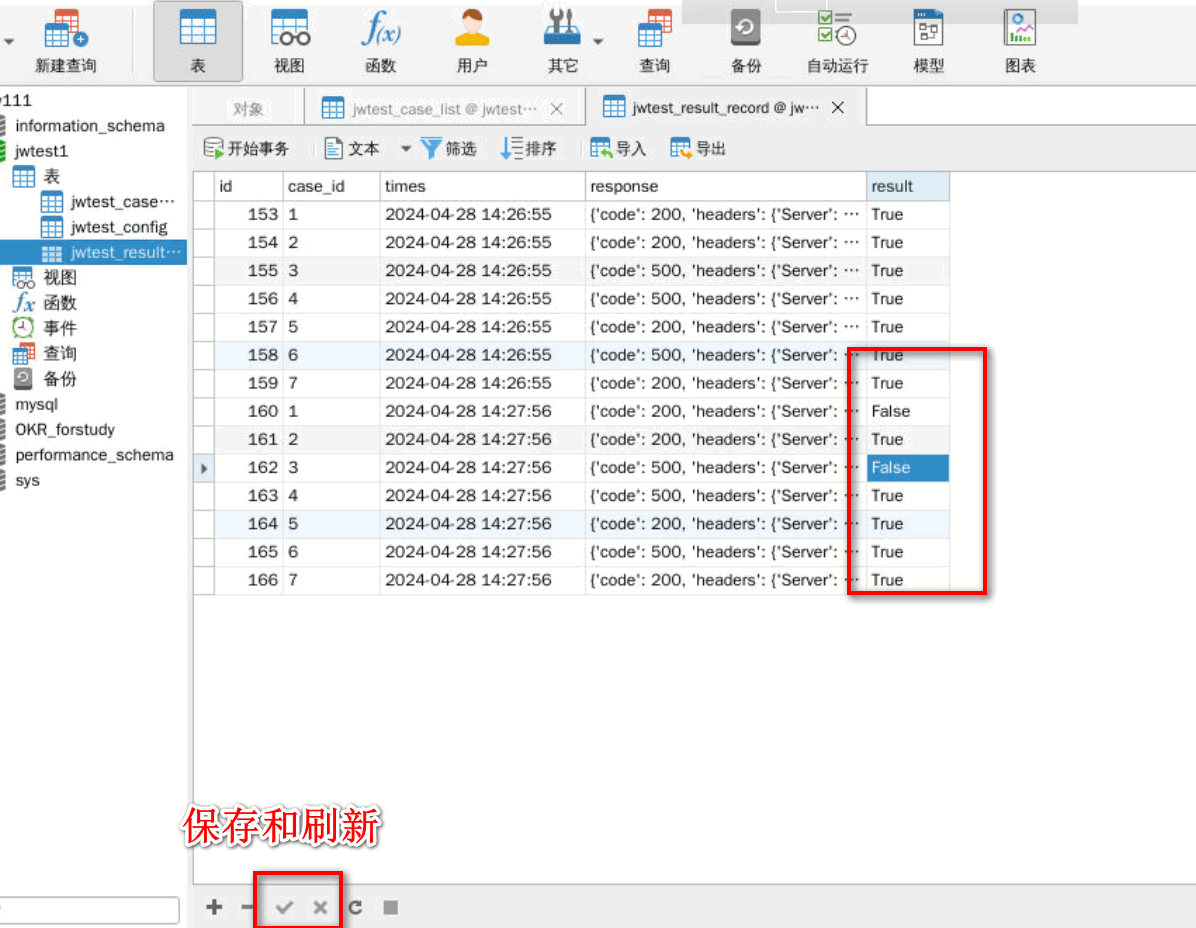
### 内容2： 断言+结果回写

|  |
| --- |
| 思路：  def jwassert\_response(case用例，res\_data实际):  1 比对预期结果 和实际结果 是否一致===》测试结果==》是否通过is\_pass+assert  2 调用工具2 updateResults(实际，测试结果，case['id']) |

核心文件：

|  |
| --- |
| import json  import pytest  # 工具类：  from common import Base  from config.settings import DynamicParam  from utils.caseutil import RdTestCase  from utils.logutil import logger  from utils.requestutils import RequestSend  gongju2=RdTestCase() # 用例工具---》用工具1～  case\_list=gongju2.is\_run\_data('okr-api') # [{用例1}，{用例2}，{用例3}]  gongju4=RequestSend() # 发出接口工具  # 思想 跑用例： gongju2得到用例数据 交给 gongju4 使用  class TestApi:  @pytest.mark.parametrize('case',case\_list)  def test\_run(self,case): # 用例条数 决定了 test\_run 个数～  # 需求1：发接口时候 拼接接口服务器地址～  url=gongju2.loadConfKey('atstudy\_okr','url\_api')['value']+case['url']  method = case['method']  # 坑：来自于数据库 字符串---》字典才能给 工具4使用解决 eval（）  headers = eval(case['headers'])  #print("当前用例请求头类型",type(headers))  cookies = eval(case['cookies'])  data = eval(case['request\_body'])  relation =str( case['relation'])  case\_name=case['title']  try:  # 工具3  logger.info(f"主人，正在执行{case\_name}用例")  # 接口发送请求～  # 替换完成后才能交给 requests 发出接口请求～  print("志玲说：替换之前的数据",headers)  headers=self.correlation(headers)  print("志玲说：替换之后的数据", headers)  res\_data=gongju4.send(url,method,data=data,headers=headers,cookies=cookies)  if res\_data:# 接口发送成功 再调用～  if relation!=None: # 只有上游接口需要设置关联值  self.set\_relation(relation,res\_data)  #guanlian=res\_data['headers'].get("Authorization",'无临时值～')  logger.info(f"实际结果：{res\_data}")  #发送失败 断言失败 发送成功 不能说明断言成功 进一步比对预期和实际 ==》测试结果  except:  logger.error("用例执行失败，请进一步排查")  assert False  #用例 实际结果  self.jwassert\_response(case,res\_data)  return res\_data  # 1def set\_relation(形参relation，实际结果res\_data):  def set\_relation(self,relation,res\_data):  # 健壮性 1验证 2 try方便定位报错代码～  try:  # 1.1判断relation 是否有值=非空=》上游用例  if relation:  # 1.2 切割 relation  jwlist=relation.split("=") #[jwtoken,Authorization]  # 1.2.1 =左边 ====》动态属性名字  zuobian=jwlist[0]  # 1.2.2 根据=右边到实际结果res\_data中查找===》动态属性值  youbian=jwlist[1]  guanlian=res\_data.get("headers").get(youbian,"无关联值")  # 1.3 setattr(类，动态属性名字，动态属性值)==》存放结束  setattr(DynamicParam,zuobian,guanlian)  #验证：  print("志玲验证是否存放成功:",getattr(DynamicParam,zuobian,"存放临时值失败～"))  except:  print("47~57行写的可能有问题～～")  # 3.1 def correlation(data):  def correlation(self,data):  # 准备工具代码  # 3.1.1判断data 是否有${}====》find  res\_data=Base.find(data)# 不确定的是列表中有几个值～  if res\_data: #['jwtoken']  for i in res\_data: # jwtoken 一一取出来～  # 3.1.2 用getattr（Dynamic，${}的值，“默认值”）==》关联值  linshi=getattr(DynamicParam,i,"没有取到临时值") # ezaasasa授权码～  # 3.1.3 关联值 字符串替换${}===>relace替换  # 被替换 data 数据库{"Authorization":"${jwtoken}"}  # 替换 linshi===>{"jwtoken": lingshizhi}  # 才能relace替换  relace\_dict={}  relace\_dict.update({i:linshi})# 替换数据构造完毕{"jwtoken": lingshizhi}  genghuan=Base.relace(data,relace\_dict)#"Authorization":"ezasa授权码～"  # 3.1.4 替换完值字符串转成字典方便给requests使用 ====>json.loads  data=json.loads(genghuan)  return data  # 结果验证方法  #用例 实际结果  def jwassert\_response(self, case, res\_data):  # 测试结论 默认是False  is\_pass = False  # 异常处理，捕获assert抛出的异常，不直接抛出  try:  ceshijieguo = str(case['expected\_code']) in \  [str(res\_data.get('body').get('msg', "没有msg键")),  str(res\_data.get('body').get('message', "没有message键")),  str(res\_data.get('body').get('success', "没有success键")),  str(res\_data.get('code'))]  assert ceshijieguo #不是为了报告体现～ 断言失败 执行 except分支～  # 打印信息  logger.info("用例断言成功")  # 设置变量为True  is\_pass = True  # 捕获异常  except:  # 设置变量为False  is\_pass = False  # 打印日志  logger.info("用例断言失败")  # 无论是否出现异常，都执行下面内容代码  finally:  # 把结果更新到数据库  gongju2.updateResults(res\_data, is\_pass, str(case['id']))  # 根据变量结果是True/False，进行断言验证，成功则通过，失败则未通过  assert is\_pass #报告中体现～  if \_\_name\_\_ == '\_\_main\_\_':  pytest.main(["-s","-v","test\_run.py"]) |

效果： 数据库中回写结果：



恭喜 代码结束：

小结：

教学代码： 第一行---》最后一行 行行在剧情~

企业代码区别：

不断迭代 不断 调试过程

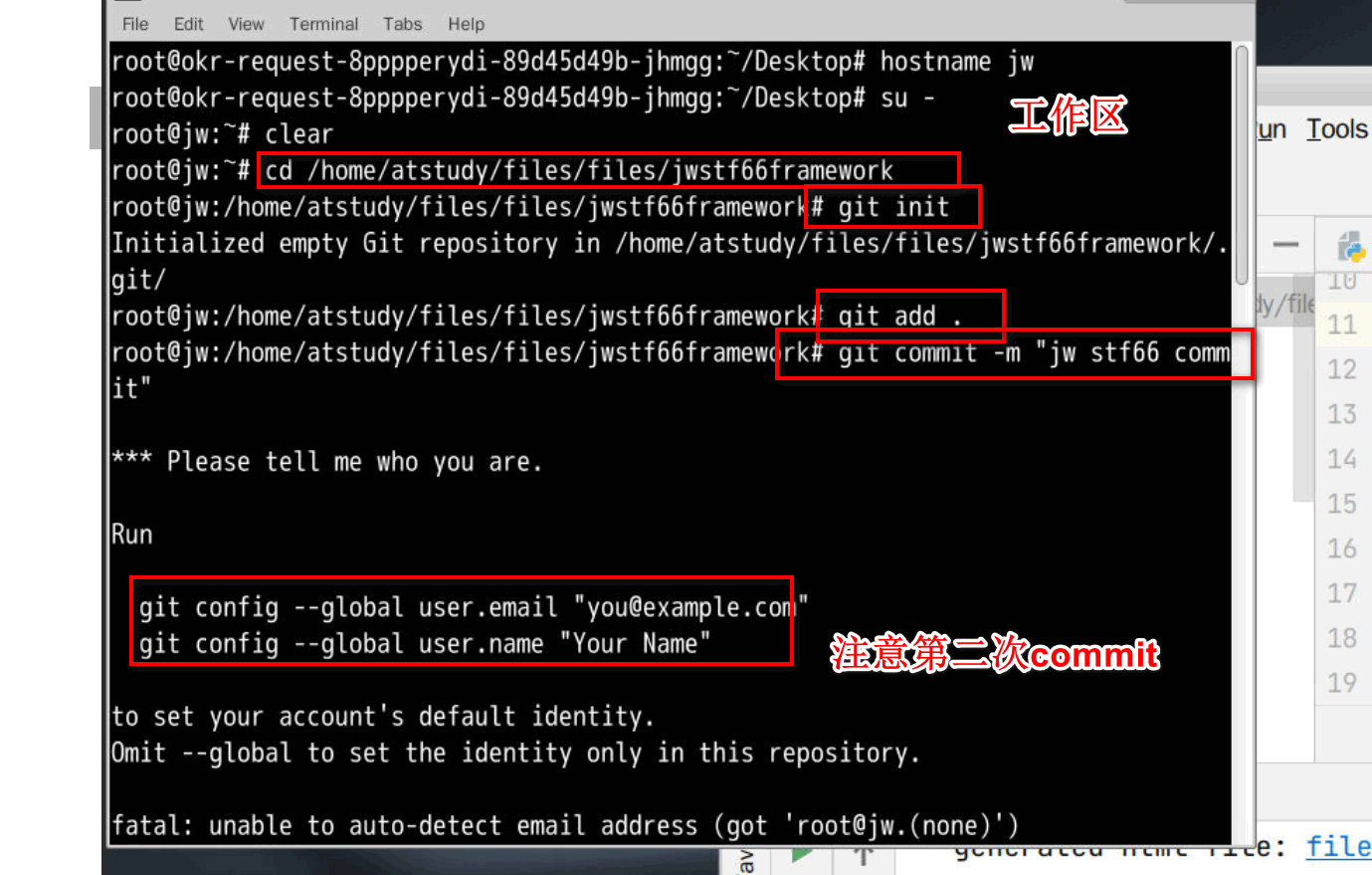
V1（熟悉工具类）《----》V2-配置<----V3（独立）---V4(上下游)----》V5===》最新版本代码执行测试用例

### 内容3: git+jenkins+allure （无人值守用例执行） ---2点51分~

剧情：

1第一步代码 托管 gitee

1.1 代码新版本---》本地仓库



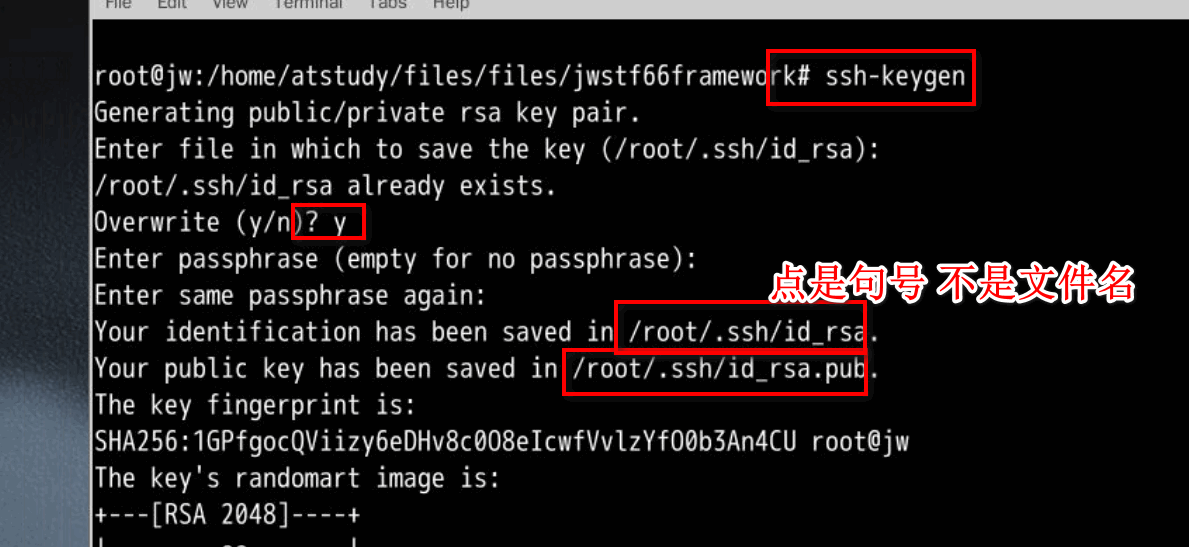
1.2 新建私有远程仓库 ---》共享



1.3 本地仓库 -----SSH授权---》远程仓库

ssh-keygen 1.3.1 私钥文件 ---》钥匙 id\_rsa （有钥匙机器开锁） 1.3.2 公钥文件 id\_rsa.pub 锁(安装码云)

1 ssh-keygen 生成公私钥



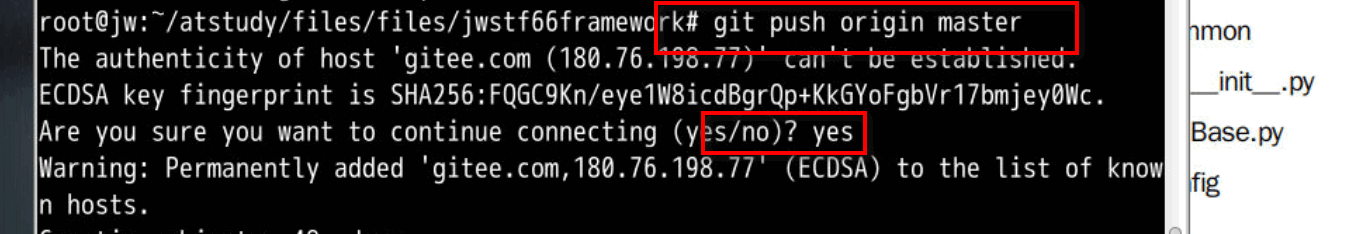
2公钥 ---》码云

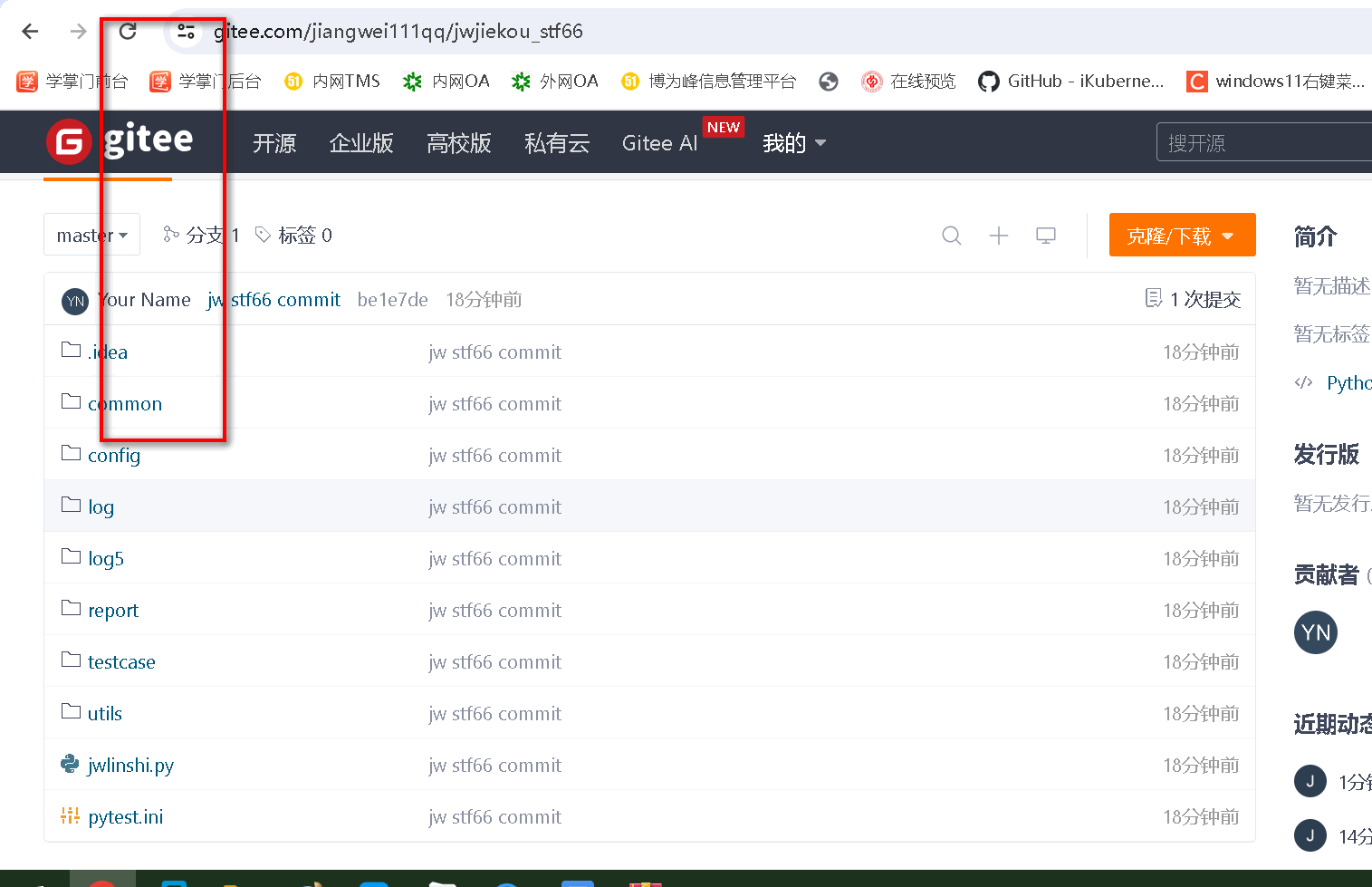
效果 有私钥机器 上传文件 给 安装了公钥机器~



3例子 本地仓库代码上传远程仓库~







**小结 1自己写代码--commit-》本地----》授权远程----推送push-》远程**

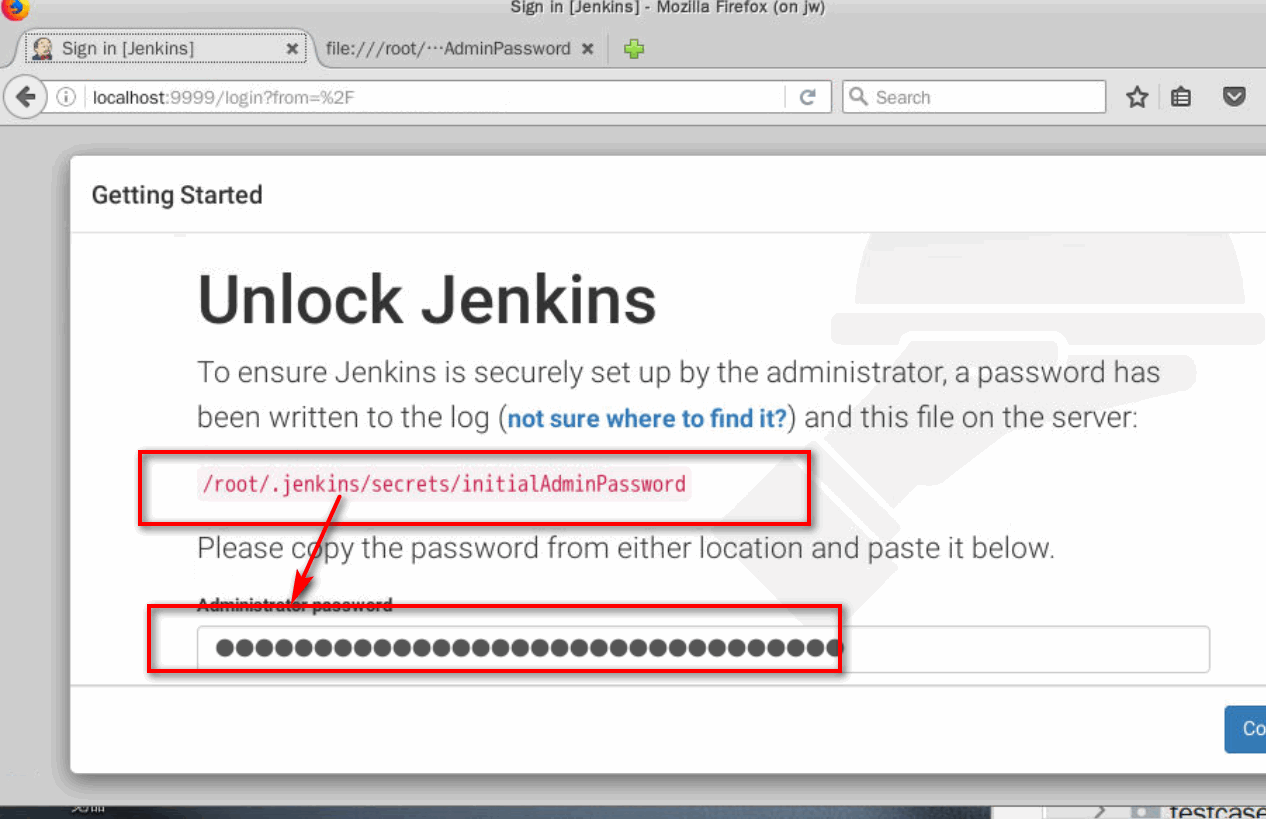
**2自动化jenkins 持续 一点点 集成 用例+框架 +前端页面+后端接口+jira禅道需求+运维镜像 产出满足需求产品**

**jenkins 大量插件 支持持续集成~**

**完成jenkins 初始工作---目的1 使用配置个用户名 admin 密码 123456**

**目的2 让插件起效果 +卸载有bug插件 让jenkins正常运行**

**访问：** <http://localhost:9999>

****

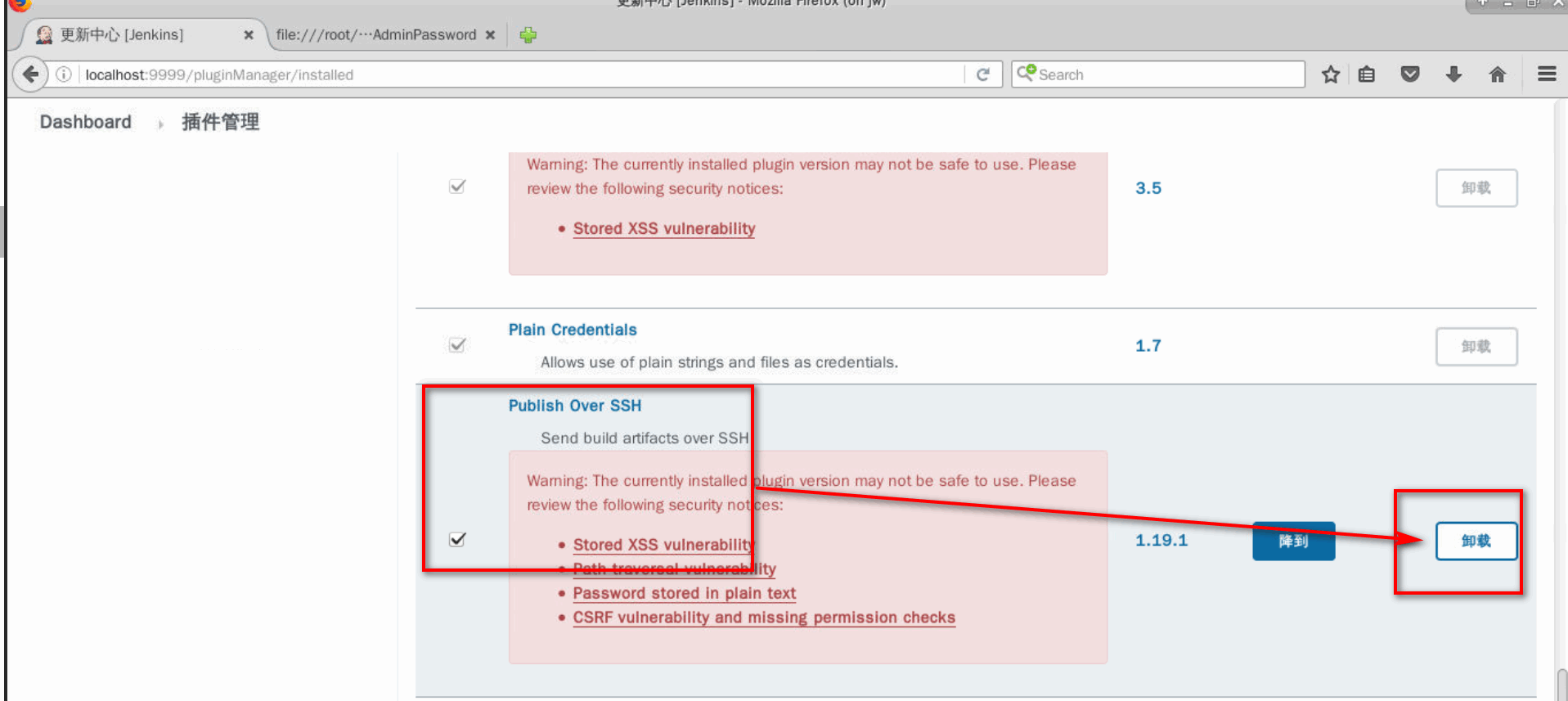
****

****

**包含 有问题插件 jenkins 开源~~吸引全世界来开发 团队 支持不同的插件~**

****

****

****

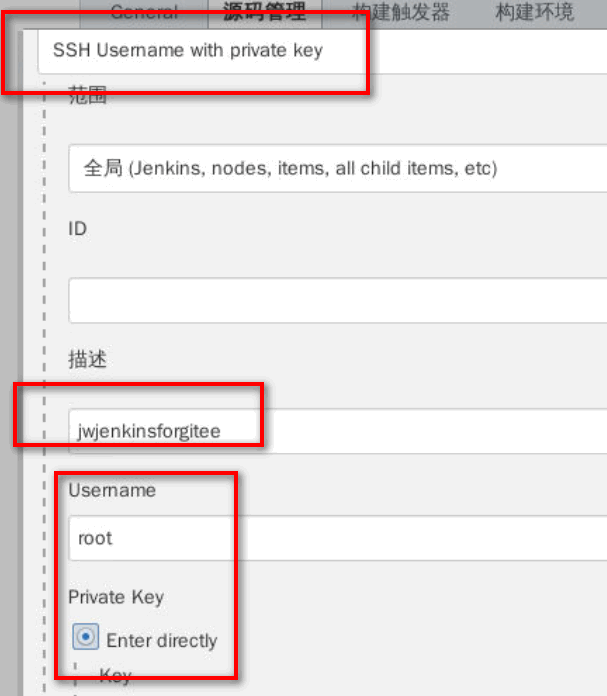
****

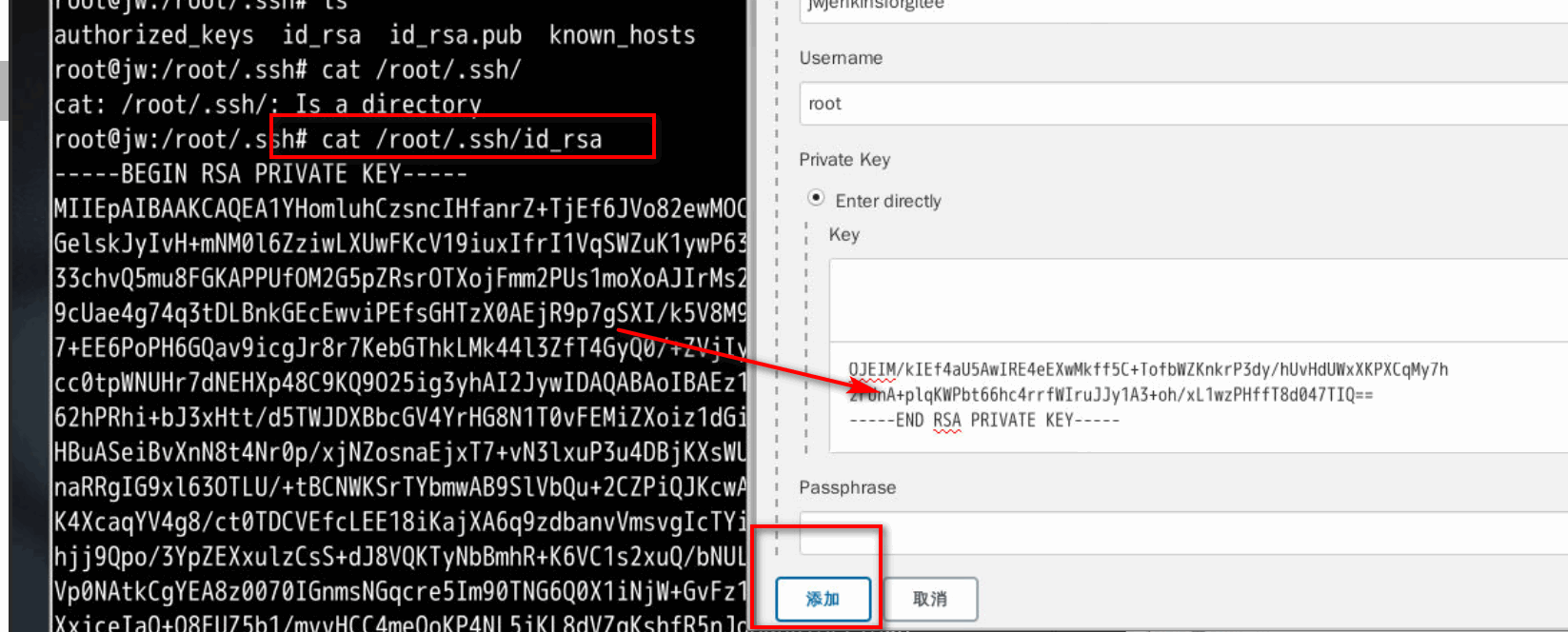
****

**jenkins 初始化 工作完毕 ---3点50分继续~**

2.1 自动拉取代码









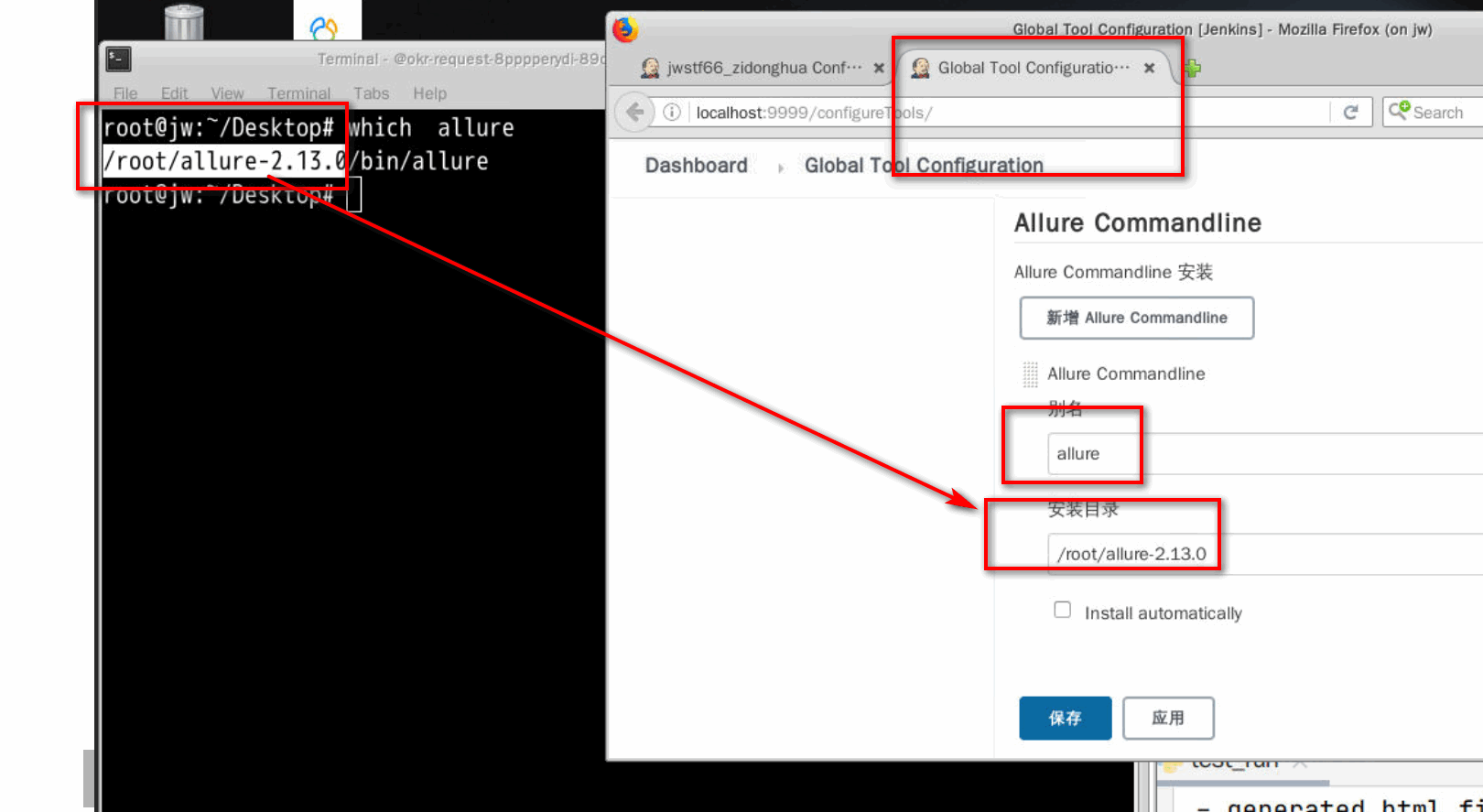




2.2 自动执行pytest代码

|  |
| --- |
| echo ${WORKSPACE}  cd ${WORKSPACE}/testcase  pytest --alluredir="${WORKSPACE}/allure-results" test\_run.py |





2.3 自动值守用例产出allure报告~

### descript

**总结：**

**1每一个模块 实现思路**

**2架构图+ 准备难点（上下游接口）**

**3 参考用例+框架代码 复用即可~**

### 