## 问题思考

在自动化脚本运行过程中，IDE控制台一般都会输出运行日志。但是如果测试项目是在liunx服务器上面运行，没有IDE控制台输出log，那么我们该如何采集日志？

## 日志概述

#### 日志作用

不管是在项目开发还是测试过程中，项目运行一旦出现问题日志信息就非常重要了。日志是定位问题的重要手段，就像侦探人员要根据现场留下的线索来推断案情。

#### 日志级别

脚本运行会有很多的情况，比如调试信息、报错异常信息等。日志要根据这些不同的情况来继续分级管理，不然对于排查问题的筛选会有比较大的干扰。 。日志一般定位的级别如下：

| **级别** | **何时使用** |
| --- | --- |
| DEBUG | 调试信息，也是最详细的日志信息。 |
| INFO | 证明事情按预期工作。 |
| WARNING | 表明发生了一些意外，或者不久的将来会发生问题（如‘磁盘满了’）。软件还是在正常工作。 |
| ERROR | 由于更严重的问题，软件已不能执行一些功能了。 |
| CRITICAL | 严重错误，表明软件已不能继续运行了。 |

首先我们日志需要按照info、debug、error等级别来进行区分的。当然这个级别可以自己去设置。在一般的情况下我们普通的输出我们直接用info类型，调试的时候用debug类型，如果预计有错误时那么我们就需要用error类型的日志，一般情况去info级别最为合适。

#### 日志格式

日志格式化是为了提高日志的可阅读性，比如：**时间+模块+行数+日志具体信息** 的内容格式。如果日志信息杂乱无章的全部输出来，这样也不利于定位问题。如下所示就是日志格式化输出，非常便于阅读查看。

2018-01-10 18:02:35,633 backup.py[line:18] INFO ============test backup================

2018-01-10 18:02:39,253 backup.py[line:20] INFO click backup button

2018-01-10 18:02:54,025 backup.py[line:23] INFO click next button

2018-01-10 18:03:09,280 common\_fun.py[line:83] INFO Start send Email..

2018-01-10 18:03:11,840 common\_fun.py[line:91] INFO Send Email finish!

2018-01-10 18:03:13,305 common\_fun.py[line:168] INFO get backup screenshot

2018-01-10 19:36:00,238 backup.py[line:17] INFO ============test backup================

2018-01-10 19:36:04,530 backup.py[line:19] INFO click backup button

2018-01-10 19:37:20,107 backup.py[line:17] INFO ============test

#### 日志位置

一个项目中会有很多的日志采集点，而日志采集点必须结合业务属性来设置。比如在登录代码执行前可以插入“准备登录..”日志信息,如果登录完成之后，再设置登录的提示日志就会给人造成误解，无法判断到底是登录之前的问题还是登录之后的问题，因此日志采集点的位置很重要。

## logging模块

### 简介

Python的logging模块提供了通用的日志系统，这个模块提供不同的日志级别，并可以采用不同的方式记录日志，比如文件，HTTP GET/POST，SMTP，Socket等，甚至可以自己实现方式记录日志。

*#导入logging模块*

import logging

[logging模块官方文档](https://docs.python.org/3.5/library/logging.html?)

### logging构成

logging模块包括**logger，Handler，Filter，Formatter**四个部分。

* Logger 记录器，用于设置日志采集。
* Handler 处理器，将日志记录发送至合适的路径。
* Filter 过滤器，提供了更好的粒度控制，它可以决定输出哪些日志记录。
* Formatter 格式化器，指明了最终输出中日志的格式。

### Logger 记录器

Logger是一个树形层级结构，在使用接口debug，info，warn，error，critical；使用之前必须创建Logger实例，即创建一个记录器，如果没有显式的进行创建，则默认创建一个root logger，并应用默认的日志级别(WARN)，Handler和Formatter。

##### 方法：

basicConfig(\*\*kwargs) 为日志记录系统做基本配置。

##### 部分参数

**filename** 指定日志文件名称

**filemode** 指定打开文件的模式，如果指定了filename（如果文件模式未指定，则默认为'a）

Tips：文件读写模式

* w 以写方式打开，
* W 文件若存在，首先要清空，然后（重新）创建
* a 以追加模式打开 (从 EOF 开始, 必要时创建新文件)
* r+ 以读写模式打开
* w+ 以读写模式打开 (参见 w )
* a+ 以读写模式打开 (参见 a )

**format** 为处理程序使用指定的格式字符串。

**datefmt** 使用指定的日期/时间格式。样式如果指定了格式字符串，则使用它来指定 格式字符串的类型.

**level** 将根记录器级别设置为指定级别。

logging\_test.py

**import** **logging**

# **logging**.basicConfig(level=logging.DEBUG)

**logging**.basicConfig(level=logging.INFO)

**logging**.debug('debug info')

**logging**.info('hello 51zxw ！')

**logging**.warning('warning info')

**logging**.error('error info')

**logging**.critical('critical info')

### Handler 处理器

Handler 处理器，将日志记录发送至合适的路径,Handler处理器类型有很多种，比较常用的有三个：

##### 1.StreamHandler

将日志记录输出发送到诸如sys.stdout，sys.stderr或任何类似文件流的对象。上面例子就是输出到控制台

##### 2.FileHandler

将日志记录输出发送到磁盘文件。 它继承了StreamHandler的输出功能。

logging.basicConfig(filename='runlog.log',level=logging.DEBUG)

##### 3.NullHandler

不做任何格式化或输出。 它本质上是一个开发人员使用的“无操作”处理程序。

### Filter 过滤器

Handlers和Loggers可以使用Filters来完成比级别更复杂的过滤。

### Formatter

使用Formatter对象设置日志信息最后的规则、结构和内容，默认的时间格式为%Y-%m-%d %H:%M:%S。

| **格式** | **描述** |
| --- | --- |
| %(levelno)s | 打印日志级别的数值 |
| %(levelname)s | 打印日志级别名称 |
| %(pathname)s | 打印当前执行程序的路径 |
| %(filename)s | 打印当前执行程序名称 |
| %(funcName)s | 打印日志的当前函数 |
| %(lineno)d | 打印日志的当前行号 |
| %(asctime)s | 打印日志的时间 |
| %(thread)d | 打印线程id |
| %(threadName)s | 打印线程名称 |
| %(process)d | 打印进程ID |
| %(message)s | 打印日志信息 |

使用方法：

logging.basicConfig(filename='runlog.log',level=logging.DEBUG,

format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s')

输出结果：

2018**-04-19** 17:25:04,679 **logging\_test**.py[line:12] **DEBUG** **debug** **info**

2018**-04-19** 17:25:04,680 **logging\_test**.py[line:13] **INFO** **hello** 51**zxw** 2020

2018**-04-19** 17:25:04,680 **logging\_test**.py[line:14] **WARNING** **waning** **info**

2018**-04-19** 17:25:04,680 **logging\_test**.py[line:15] **ERROR** **error** **info**

2018**-04-19** 17:25:04,680 **logging\_test**.py[line:16] **CRITICAL** **critical** **info**

## Logging实战操作

#### 测试场景

将前面所学的启动考研帮App的脚本增加log采集功能，设置指定的日志格式输出，并将日志保存到指定文件。

#### 代码实现

kyb\_logger.py

from appium **import** webdriver

**import** yaml

from selenium.common.exceptions **import** NoSuchElementException

**import** logging

file=open('../yaml/desired\_caps.yaml','r')

**data**=yaml.load(file)

logging.basicConfig(level=logging.INFO,filename='runlog.log',

format='%(asctime)s %(filename)s[line:%(lineno)d]%(levelname)s%(message)s')

desired\_caps={}

desired\_caps['platformName']=**data**['platformName']

desired\_caps['platformVersion']=**data**['platformVersion']

desired\_caps['deviceName']=**data**['deviceName']

desired\_caps['app']=**data**['app']

desired\_caps['appPackage']=**data**['appPackage']

desired\_caps['appActivity']=**data**['appActivity']

desired\_caps['noReset']=**data**['noReset']

logging.info('start app...')

driver=webdriver.Remote('http://'+str(**data**['ip'])+':'+str(**data**['port'])+'/wd/hub',desired\_caps)

def check\_cancelBtn():

logging.info('check cancelBtn')

try:

cancelBtn = driver.find\_element\_by\_id('android:id/button2')

except NoSuchElementException:

logging.info('no cancelBtn')

**else**:

cancelBtn.click()

def check\_skipBtn():

logging.info('check skipBtn')

try:

skipBtn = driver.find\_element\_by\_id('com.tal.kaoyan:id/tv\_skip')

except NoSuchElementException:

logging.info('no skipBtn')

**else**:

skipBtn.click()

check\_cancelBtn()

check\_skipBtn()

### 问题思考

前面我们已经实现了在代码中增添log，log也按照预期的采集到了，看似一切完美无瑕。但是该log配置的作用域也只是控制当前的脚本 。 然而一个自动化项目中通常有很多模块脚本，难道我们需要每一个脚本都这样配置吗？

### 解决思路

将这些日志配置的参数抽离出来，各个模块需要使用则直接引用即可。

### 日志格式配置

将log输出格式，输出路径等参数抽离出来作为一个配置表，如下所示：

**log.conf**

**[loggers]**

keys=root,infoLogger

**[logger\_root]**

level=DEBUG

handlers=consoleHandler,fileHandler

**[logger\_infoLogger]**

handlers=consoleHandler,fileHandler

qualname=infoLogger

propagate=0

**[handlers]**

keys=consoleHandler,fileHandler

**[handler\_consoleHandler]**

class=StreamHandler

level=INFO

formatter=form02

args=(sys.stdout,)

**[handler\_fileHandler]**

class=FileHandler

level=INFO

formatter=form01

args=('runlog.log', 'a')

**[formatters]**

keys=form01,form02

**[formatter\_form01]**

format=%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s

**[formatter\_form02]**

format=%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s

在需要调用的模块增加如下代码：

import logging

import logging.config

CON\_LOG='log.conf'

logging.config.fileConfig(CON\_LOG)

logging=logging.getLogger()

方法：

fileConfig(fname, defaults=None, disable\_existing\_loggers=True)

该放在作用是从ConfigParser格式的文件中读取日志配置，同时如果当前脚本有配置log参数，则覆盖当前log配置选项。

#### 代码实现

**kyb\_logconf.py**

from appium **import** webdriver

**import** yaml

from selenium.common.exceptions **import** NoSuchElementException

**import** logging

**import** logging.config

stream=open('../yaml/desired\_caps.yaml','r')

**data**=yaml.load(stream)

CON\_LOG='log.conf'

logging.config.fileConfig(CON\_LOG)

logging=logging.getLogger()

desired\_caps={}

desired\_caps['platformName']=**data**['platformName']

desired\_caps['platformVersion']=**data**['platformVersion']

desired\_caps['deviceName']=**data**['deviceName']

desired\_caps['app']=**data**['app']

desired\_caps['noReset']=**data**['noReset']

desired\_caps['appPackage']=**data**['appPackage']

desired\_caps['appActivity']=**data**['appActivity']

driver = webdriver.Remote('http://'+str(**data**['ip'])+':'+str(**data**['port'])+'/wd/hub', desired\_caps)

def check\_updateBtn():

logging.info("check\_pdateBtn")

try:

element = driver.find\_element\_by\_id('android:id/button2')

except NoSuchElementException:

logging.info('update element is not found!')

**else**:

element.click()

def check\_skipBtn():

logging.info("check\_skipBtn")

try:

element = driver.find\_element\_by\_id('com.tal.kaoyan:id/tv\_skip')

except NoSuchElementException:

logging.info('skipBtn element is not found!')

**else**:

element.click()

check\_updateBtn()

check\_skipBtn()

#### 参考资料

<https://www.jianshu.com/p/feb86c06c4f4>