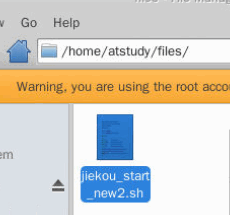
全程66期接口自动化 第三次

回顾：自动化代替人执行用例

1 pytest -----》 @pytest.mark.parametrize ---->单参数数据驱动

2 企业编程特点 ----》2.1 API表 2.2 工具类？？ **目标 降低 遗忘+没有学过成本**

****

### 内容1: logging灵魂三问？

**什么是logging？**

**python 自带日志库 产出日志信息~**

**print 也算日志**

**特点：简单 缺点 功能弱**

**为什么logging？**

**需求： 1 对日志信息级别分 （李四--测试--》看bug信息 王五--》开发---》调试 张三--》运维--》硬盘---》警告）**

1.1 安全 1.2 方便工作 精准定位问题

**2 对格式也是有要求 （时间？+程序？+级别？+行号？）**

**3 .可以在文件存放日志信息（实时日志+离线日志）**

**解决 logging 优点：强大 缺点： 学习成本---》遗忘成本**

**如何掌握logging？**

**1 掌握基本API 满足需求**

**2 自定义工具类 封装API+结合Pytest 进行使用**

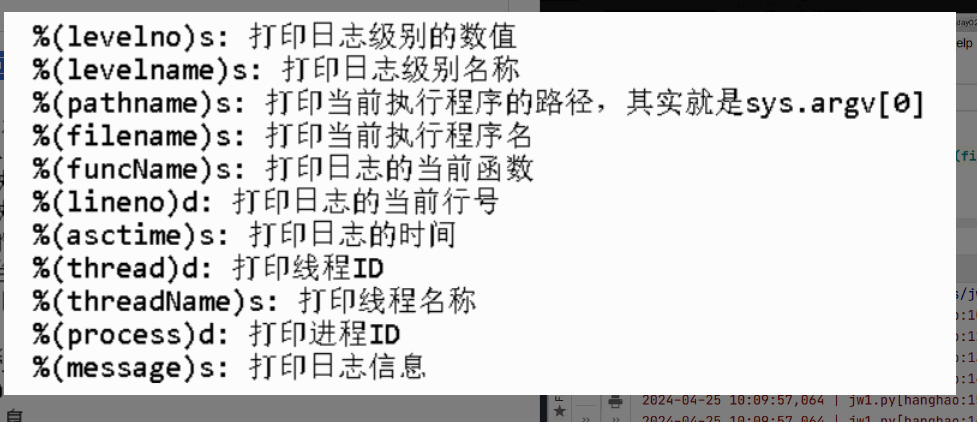
### 内容2：快速入门：

**例子： 实现对日志信息级别的划分？**

|  |
| --- |
| 默认情况下，会打印WARNING级别的日志   * DEBUG：详细信息，调试信息。 * INFO：确认一切按预期运行。 * 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。----默认级别 * 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。   注意：  日志等级 DEBUG详细<INFO<WARNING<ERROR<CRITICAL简略  只有级别高于或者等于日志级别的日志才会被输出，低于该等级的日志将会被丢弃 |

**格式**

****

****

**API:**

|  |  |
| --- | --- |
| **对象名字** | **对象API** |
| **logging** | **1 5个输出 日志API ==》小写~**  **2 设置级别 大写**  **basicConfig(level=大写，format=设置格式)** |
|  | **日志系统？（帮派一个道理）**  **1**  **老大logger=getLogger（“名字”）**  **设置总级别---安全**  **1.1 setLevel(ERROR)**    **2控制台小弟1StreamHandler()**  **2.1 setLevel(显示级别 子级别 ERROR CRITICAL)**  **子级别 不可以比总级别详细**  **2.2 setFormat(设置格式)**  **3 文件小弟1**  **文件模式 r w append 日志不断追加**  **fp1=FileHandler('文件位置11',mode=‘a’,encoding=‘utf-8’)**  **fp2=FileHandler('文件位置21',mode=‘a’,encoding=‘utf-8’)**  **。。。。**  **3.1 setLevel(显示级别 子级别)**  **3.2 setFormat （设置格式）**  **4 老大 派小弟干活**  **老大.addHandler(小弟)**  **5 对外 返回出对象 老大 （给 其他系统比如pytest 开发系统进行调用~）** |
|  |  |

**示例：**

|  |
| --- |
| **import logging**  **logging.basicConfig(level=logging.INFO,**  **format='%(filename)s | %(asctime)s | [hanghao:%(lineno)d] - %(levelname)s: %(message)s')**  **if \_\_name\_\_ == '\_\_main\_\_':**  **# 日志系统 默认的级别 是WARNING 级别**  **logging.debug("我是代码调试信息11～")**  **logging.info("我是系统正常输出信息11")**  **logging.warning("----我是警告信息11，比如内存不够～-----")**  **logging.debug("我是代码调试信息22～")**  **logging.error("-----我是一个小bug11！！------")**  **logging.info("我是系统正常输出信息22")**  **logging.warning("----我是警告信息22，比如内存不够～-----")**  **logging.error("-----我是一个小bug22！！------")**  **logging.warning("----我是警告信息33，比如内存不够～-----")**  **logging.critical("----我是bug 严重事故～导致系统崩溃-----")** |

**小结： 1级别 2格式**

### 内容3： 日志多终端分发（日志系统）：

**需求**： **终端--最终和交互的端 比如 命令行 文件**

**----张三 ---运维 ---》命令行终端1（日志）+级别 +格式**

**----李四 ----测试 ---》 保存文件终端 （日志）+级别+格式**

**----XXXX人 ----》文件终端2 3 4 （日志）+级别+格式**

**解决 ：API =======》当然是记不住 ======》工具类 直接复用~10点46分继续**

|  |
| --- |
| **# 日志系统？（帮派一个道理）**  **# 类---\_\_init\_\_---》对象**  **import logging**  **class logutil:**  **def \_\_init\_\_(self):# 想表达 使用工具（对象）帮派 已经产生**  **# 1# 老大logger = getLogger（“名字”）**  **self.logger=logging.getLogger("坤坤")**  **# 1.1# setLevel(ERROR)# 设置总级别 - --安全**  **self.logger.setLevel(logging.INFO)**  **# 2# 控制台小弟1StreamHandler()**  **sh=logging.StreamHandler()**  **# 2.1# setLevel(显示级别# 子级别)# 不可以比总级别详细**  **sh.setLevel(logging.INFO)**  **# 2.2# setFormat(设置格式)**  **sh.setFormatter(logging.Formatter('%(filename)s | %(asctime)s | [hanghao:%(lineno)d] - %(levelname)s: %(message)s'))**  **## 3**  **# 文件小弟1# 文件模式 # append 日志不断追加**  **# 相对或者绝对路径都可以**  **fh=logging.FileHandler('jw.log',mode='a',encoding='utf-8')**  **# 3.1# setLevel(显示级别# 子级别)**  **# 3.2# setFormat （设置格式）**  **fh.setLevel(logging.WARNING)**  **# 2.2# setFormat(设置格式)**  **fh.setFormatter(**  **logging.Formatter(' %(asctime)s - %(filename)s - [line:%(lineno)d] - %(levelname)s: %(message)s'))**  **fh.close()**  **# 4# 老大**  **# 派小弟干活**  **# 老大.addHandler(小弟)**  **self.logger.addHandler(sh)**  **self.logger.addHandler(fh)**  **#**  **# 5**  **# 对外# 返回出对象# 老大 （给# 其他系统比如pytest# 开发系统进行调用# ~）**  **def log(self):**  **return self.logger**  **# 获取老大**  **# 注释：**  **# 1 日志系统 代替print 完成日志终端分发**  **# 2 日志级别划分**  **# 3 日志支持 格式设置**  **logger=logutil().log()**  **if \_\_name\_\_ == '\_\_main\_\_':**  **# 自测**  **# 日志系统 默认的级别 是WARNING 级别**  **logger.debug("我是代码调试信息11～")**  **logger.info("我是系统正常输出信息11")**  **logger.warning("----我是警告信息11，比如内存不够～-----")**  **logger.debug("我是代码调试信息22～")**  **logger.error("-----我是一个小bug11！！------")**  **logger.info("我是系统正常输出信息22")**  **logger.warning("----我是警告信息22，比如内存不够～-----")**  **logger.error("-----我是一个小bug22！！------")**  **logger.warning("----我是警告信息33，比如内存不够～-----")**  **logger.critical("----我是bug 严重事故～导致系统崩溃-----")** |

**工作场景：**

**开发系统 （测试pytest框架）+整合日志系统 （以上代码）====》运行开发系统结果 ===》产生日志文件**

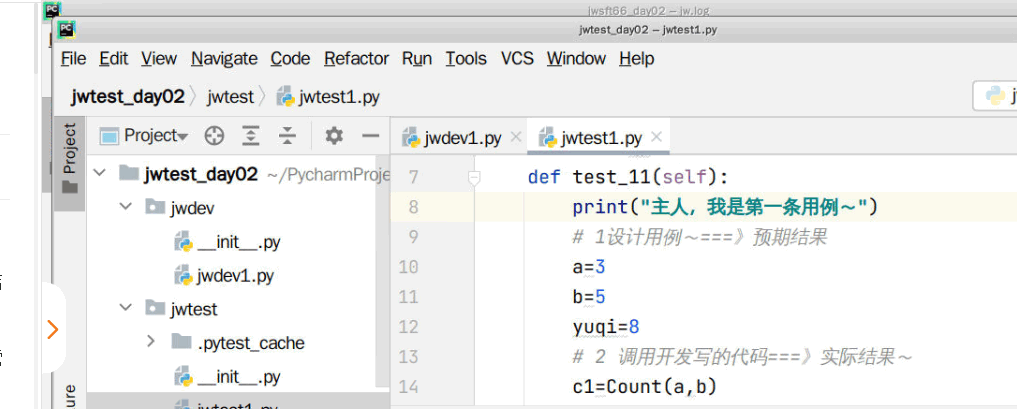
**文件1 《===运维**

**文件2 《===开发**

**文件3 《====测试**

**体现不同的人看不同的级别**

**需求： XXXX小白 写Pytest 测试框架| 开发系统 用到 日志系统,发现 日志系统遗忘+企业不给你时间？**



实现：

被测

|  |
| --- |
| class Count:  def \_\_init\_\_(self,a,b):  self.a=int(a)  self.b=int(b)  def add(self):  return self.a+self.b  def sub(self):  return self.a-self.b |

测试

|  |
| --- |
| import pytest  from jwdev.jwdev1 import Count  from utils.jw2 import logger  class TestAdd():  def test\_11(self):  logger.info("主人，我是第一条用例～")  # 1设计用例～===》预期结果  try:  a = 3  b = 5  yuqi = 81  # 2 调用开发写的代码===》实际结果～  c1 = Count(a, b)  shiji = c1.add()  assert yuqi == shiji  except: # 断言失败 也会except 分支~  logger.error("主人，我是第一条用例有bug～～")  def test\_22(self):  logger.info("主人，我是第二条用例～")  assert 200==Count(100,100).add()  def test\_33(self):  logger.info("主人，我是第三条用例～")  assert 202==Count(101,101).add()  if \_\_name\_\_ == '\_\_main\_\_':  pytest.main(['-v', '-s', 'jwtest1.py']) # 不要忘了改成你的文件名~ |

工具代码:---参考上述logutil

小结：

1工具类价值 ： 不熟悉 源码编写+ 遗忘库场景下 方便通过注释 直接复用！！

2 工具类从哪里来？

**2.1 企业提供 （包） | 老程序 类库整理工具类 | 网络 GPT（版本）**

3 如何使用工具类？

静态： 注释明确职责

动态： 利用自测代码运行工具类 满足注释

1. 库还要不要学？

答: 不学 能 工作

学了 更好 1 才可以自定义 方便下次使用 2 修改工具类 3 自信

预告： 1 增加工具类~ 2pymysql~