# 昨日回顾

V6--PO模式深入封装

1. **抽取PO模式对象库层定位方法为find\_element(location)**
2. **抽取PO模式操作层输入操作为input\_text(element,text)**

数据驱动--概念

1. 了解一下数据驱动的概念
2. **认识JSON文件语法和JSON数据的操作**
3. **使用parameterized实现数据驱动(步骤)**

# 今日内容

日志基本介绍

1. 知道日志的概念和作用
2. **知道日志有哪些级别**

日志基本使用

1. **logging记录日志的基本用法**

日志组件使用

1. 知道logging日志模块的四大组件
2. **把日志信息输出到控制台和文件多个位置**
3. **把日志内容按每日拆分日志文件**

**对比手功测试流程--记住自动化测试的流程**

# 一. 日志相关概念

## 1. 概念和日志级别

1. **日志就是用于记录系统运行时的信息log**
2. 作用:
   1. 调试程序--记录输入输出
   2. 运行状态--记录异常状态
   3. 问题定位--记录崩溃位置
   4. 分析统计--记录用户信息
3. **日志级别**

|  |  |  |
| --- | --- | --- |
| * **级别** | * **作用** |  |
| * **DEBUG** | * **调试级别--详细日志--代码调试** | * 10 |
| * **INFO** | * **信息级别--运行过程** | * 20 |
| * **WARNING** | * **警告级别--潜在错误** | * **30** |
| * **ERROR** | * **错误级别--异常错误** | * 40 |
| * **CRITICAL** | * **严重级别--严重错误** | * 50 |

# 二. 日志的基本用法

## 1. 输出日志

import logging  
  
# 通过logging模块输出日志  
logging.debug("这是一条调试信息")  
logging.info("这是一条普通信息")  
logging.warning("这是一条警告信息")  
logging.error("这是一条错误信息")  
logging.critical("这是一条严重错误信息")

**注意三点**(默认)

1. 日志仅仅显示 warning,error,critical级别日志
2. 日志格式显示 "日志级别:日志名称:日志信息" 内容
3. 日志内容显示输出在终端中

**想要自定义以上三点需要用到logging.basicConfig()方法进行设置**

## 2. 设置日志输出级别

1. 设置日志输出级别 -- logging.basicConfig(level=logging.DEBUG)
2. 输出日志 -- logging.debug("这是一条调试日志")

import loggin  
  
# 设置日志输出级别  
logging.basicConfig(level=logging.DEBUG)  
  
# 通过logging模块输出日志  
logging.debug("这是一条调试日志")  
logging.info("这是一条普通日志)  
logging.warning("这是一条警告日志")  
logging.error("这是一条错误日志")  
logging.critical("这是一条严重错误日志")

## 3. 设置日志输出格式

1. 构造日志格式模板 -- fmt = "%(levelname)s:%(name)s:%(message)s"
2. 设置日志输出格式 -- logging.basicConfig(format=fmt)
3. 输出日志 -- logging.debug("这是一条调试日志")

import loggin  
  
# 构造日志格式模板  
fmt = "%(asctime)s %(levelname)s [%(name)s] [%(filename)s(%(funcName)s:%(lineno)d)] - %(message)s"  
  
# 设置日志输出格式  
logging.basicConfig(format=fmt)  
  
# 通过logging模块输出日志  
logging.debug("这是一条调试日志")  
logging.info("这是一条普通日志)  
logging.warning("这是一条警告日志")  
logging.error("这是一条错误日志")  
logging.critical("这是一条严重错误日志")

**补充-格式占位符**

1. **%(asctime)s -- 当前时间字符串**
2. **%(levelname)s -- 日志级别**
3. %(name)s -- 日志器名
4. **%(filename)s -- 输出位置--文件名**
5. %(funcName)s -- 输出位置--方法名
6. **%(lineno)d -- 输出位置--代码行号**
7. **%(message)s-- 输出信息**

## 4. 设置日志输出位置

1. 设置日志输出文件 -- logging.basicConfig(filename="a.log")
2. 输出日志 -- logging.debug("这是一条调试日志")

import logging  
  
# 设置日志输出位置--文件  
logging.basicConfig(filename="a.log")  
  
# 通过logging模块输出日志  
logging.debug("这是一条调试日志")  
logging.info("这是一条普通日志)  
logging.warning("这是一条警告日志")  
logging.error("这是一条错误日志")  
logging.critical("这是一条严重错误日志")

import loggin  
  
# 构造日志格式模板  
fmt = "%(asctime)s %(levelname)s [%(name)s] [%(filename)s(%(funcName)s:%(lineno)d)] - %(message)s"  
  
# 设置日志 -- 输出文件-日志格式-日志级别  
logging.basicConfig(filename="a.log", format=fmt, level=logging.DEBUG)  
  
# 通过logging模块输出日志  
logging.debug("这是一条调试日志")  
logging.info("这是一条普通日志)  
logging.warning("这是一条警告日志")  
logging.error("这是一条错误日志")  
logging.critical("这是一条严重错误日志")

# 三. 日志的高级用法

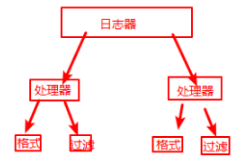
## 1. 组件介绍

关于日志的一些高级操作,需要借助日志的组件实现

|  |  |  |
| --- | --- | --- |
| 组件 | 类 | 作用 |
| 日志器 | Logger | 日志入口 |
| 处理器 | Handler | 输出处理 |
| 格式化器 | Formatter | 日志格式 |
| 过滤器 | Filter | 过滤日志 |
|  |  |  |

**注意它们的关系**

1. 日志器通过处理指定日志输出位置 -- 日志器可以有多个处理器
2. 处理器的输出格式通过格式化器指定 -- 处理器有自己的格式器
3. 处理器的输出内容通过过滤器过滤 -- 处理器有自己的过滤器

* 

## 2. 日志器Logger

日志器的主要所用和和操作: 输出日志,设置级别,指定处理器,过滤日志

|  |  |  |
| --- | --- | --- |
| 操作 | 使用 | 注意 |
| 获取日志器对象 | logger = logging.getLogger()<br />logger = logging.getLogger('日志器名字') | 不指定日志器名<br />返回root日志器对象 |
| 打印日志 | logger.debug("日志内容")<br />.......<br />logger.critical("日志内容") | 通过日志器输出日志 |
| 指定处理器 | logger.addHandler(处理器对象) | 需要实例化处理器对象 |
| 日志级别 | logger.setLevel(logging.XXX) | logging.XXX日志级别 |
|  |  |  |

## 3. 处理器Handler

日志需要输出到不同的位置时需要借助于处理器实现

|  |  |  |
| --- | --- | --- |
| 操作 | 使用 | 注意 |
| 获取处理器对象 | **hdl = Logging.StreamHandler()**<br />hdl = Logging.FileHandler()<br />**hdl = Logging.handlers.TimedRotatingFileHandler()** | 终端<br />文件<br />时分文件 |
| 日志格式 | hdl.setFormatter(格式化器对象) | 需要实例化格式化器对象 |
| 日志级别 | hdl.setLevel(logging.XXX) | logging.XXX日志级别 |
|  |  |  |
|  |  |  |

## 4. 格式化器Formatter

日志处理器定义日志格式时需要使用的对象

|  |  |  |
| --- | --- | --- |
| 操作 | 使用 | 注意 |
| 获取格式化器对象 | fmtr = loggind.Formatter(fmt = None) | fmt是格式化字符串 |
|  |  |  |
|  |  |  |

import logging.handlers  
  
# 获取日志器  
logger = logging.getLogger("test")  
  
# 获取格式化器  
fmtr = logging.Formatter(  
 fmt="%(asctime)s %(levelname)s [%(name)s] [%(filename)s(%(funcName)s:%(lineno)d)] - %(message)s")  
  
# 获取处理器 终端--文件--时间划分文件  
shdl = logging.StreamHandler()  
# 文件 -- filename -- encoding  
fhdl = logging.FileHandler(filename="./log/test.log", encoding="utf-8")  
# 时间划分文件 -- filename -- when -- interval -- backupCount -- encoding  
tfhdl = logging.handlers.TimedRotatingFileHandler(filename='./log/testtime.log', when='M', interval=1, backupCount=2, encoding='utf8')  
  
# 处理器添加格式化器  
shdl.setFormatter(fmtr)  
fhdl.setFormatter(fmtr)  
tfhdl.setFormatter(fmtr)  
  
# 日志器添加处理器  
logger.addHandler(shdl)  
logger.addHandler(fhdl)  
logger.addHandler(tfhdl)  
  
# 设置日志输出级别  
logger.setLevel(logging.DEBUG)  
  
# 输出日志  
logger.debug("debugtest")  
logger.info("infotest")  
logger.warning("warningtest")  
logger.error("errortest")  
logger.critical("criticaltest")

# 四. 项目实战介绍

## 1. web项目自动化测试流程

1. 需求分析
2. 挑选合适的功能展开自动化
3. 设计测试用例
4. 搭建自动化测试环境
5. 设计自动化测试项目框架
6. 编写用例代码
7. 执行用例脚本
8. 生成测试报告分析结果

# 今日内容

日志基本介绍

1. 知道日志的概念和作用
2. **知道日志有哪些级别**

日志基本使用

1. **logging记录日志的基本用法**

日志组件使用

1. 知道logging日志模块的四大组件
2. **把日志信息输出到控制台和文件多个位置**
3. **把日志内容按每日拆分日志文件**

**对比手功测试流程--记住自动化测试的流程**