logging日志库

2024年8月17日　　19:44

logging日志库

一、logging库日志级别

级别	级别	使用时机
NOTSET	0	使用自定义名称获取记录器时，未初始化日志级别，则默认为 NOTSET
DEBUG	10	详细信息，常用于调试
INFO	20	程序正常运行过程中产生的一些信息
WARNING（默认）	30	警告用户，虽然程序还在正常工作，但有可能发生错误
ERROR	40	由于更严重的问题，程序已经不能执行一些功能了
CRITICAL	50	严重错误，程序不能正常运行

二、基本编程方式

logging 标准库不是同步输出日志，为了控制一些大量的并发的日志输出，采用了多线程的机制进行输出日志

如果是同步输出，大量的日志会导致进程卡死。所以是异步的

1、基本函数

函数	说明
logging.debug(msg, *args, *kwargs)	创建一条严重级别为 DEBUG 的日志记录
logging.info(msg, *args, **kwargs)	创建一条严重级别为 INFO 的日志记录
logging.warning(msg, *args, **kwargs)	创建一条严重级别为 WARNING 的日志记录
logging.error(msg, *args, **kwargs)	创建一条严重级别为 ERROR 的日志记录
logging.critical(msg, *args, **kwargs)	创建一条严重级别为 CRITICAL 的日志记录
logging.log(level, *args, **kwargs)	创建一条严重级别为 level 的日志记录
logging.basicConfig(**kwargs)	对 root logger 进行一次性配置

2、基本输出

```
# 默认的输入级别为waring
# 所以 debug 和 info的日志 默认是不输出的
```

这样对于测试是非常方便的，debug是为了测试功能，在开发阶段使用，你上线之后的程序是不用输出debug日志的，使用logging之后，debug在运行时，默认是不输出的，就不用为了上线再去删除debug的输出了

```
#-*- encoding:utf-8 -*-
import logging
```

```
# 默认的输入级别为waring
# 所以 debug 和 info的日志 默认是不输出的

logging.debug("这是 debug 日志")
logging.info("这是 info 日志")
logging.warning("这是 warning 日志")
logging.error("这是 error 日志")
logging.critical("这是 critical 日志")
```

3、basicConfig()函数常见参数：

（1）设置日志输出级别 level

```
#-*- encoding:utf-8 -*-
import logging
# 修改日志的输出级别为DEBUG，注意要在日志输出前进行设置
logging.basicConfig(level=logging.DEBUG)
# 这样就会输出所有级别的日志
logging.debug("这是 debug 日志")
logging.info("这是 info 日志")
logging.warning("这是 warning 日志")
logging.error("这是 error 日志")
logging.critical("这是 critical 日志")
```

（2）设置日志输出到文件中 filename

```
# 设置日志输出到文件中
logging.basicConfig(filename='demo.log')
```

（3）设置写入格式 filemode

```
# 设置写入格式 w：每次清空重新写入；a：追加写入
logging.basicConfig(filemode='w') # 每次清空
logging.basicConfig(filemode='a') # 追加写入
```

（4）设置输出格式、添加一些公共信息

```
# 设置输出格式、添加一些公共信息
logging.basicConfig(
    #          时间          日志等级       文件名        行号        日志信息
    format='%(asctime)s|%(levelname)s|%(filename)s:%(lineno)s|%(message)s',
    datefmt='%Y-%m-%d %H:%M:%S', # 规定时间格式
    level=logging.DEBUG          # 设置输出等级
)
```

4、向日志输出变量

```
# 向日志输出变量
name = '张三'
age = 18
logging.basicConfig(level=logging.DEBUG)

logging.debug("姓名：%s，年龄：%d", name, age)

# 以 % 为运算符输出格式化字符串
logging.debug("姓名：%s，年龄：%d"%( name, age))

# 以format函数格式化字符串
logging.debug("姓名：{}，年龄：{}".format(name, age))
```

```
# python3 支持 f-string 格式化字符串
logging.debug(f"姓名：{name}，年龄：{age}")
```

5、Formatters 格式

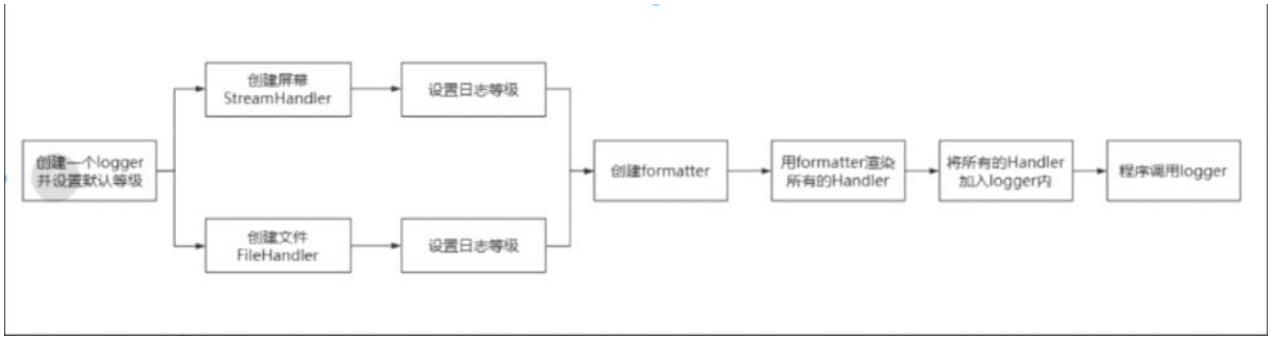
变量	格式	变量描述
asctime	%(asctime)s	将日志的时间构造成可读的形式，默认情况下是精确到毫秒，如 2018-10-13 23:24:57,832，可以额外指定 datefmt 参数来指定该变量的格式
name	%(name)	日志对象的名称
filename	%(filename)s	不包含路径的文件名
pathname	%(pathname)s	包含路径的文件名
funcName	%(funcName)s	日志记录所在的函数名
levelname	%(levelname)s	日志的级别名称
message	%(message)s	具体的日志信息
lineno	%(lineno)d	日志记录所在的行号
pathname	%(pathname)s	完整路径
process	%(process)d	当前进程ID
processName	%(processName)s	当前进程名称
thread	%(thread)d	当前线程ID
threadName	%threadName)s	当前线程名称

三、logging的高级应用

1、logging采用了模块化设计，主要包含四种组件：

名称	作用	比喻
Loggers	记录器，提供应用程序代码直接使用的接口	记录的笔
Handlers	处理器，将记录器产生的日志发送至目的地	记录的材质（纸张）
Filters	过滤器，提供更好的粒度控制，决定哪些日志会被输出	哪些需要写，哪些不需要写
Formatters	格式化器，设置日志内容的组成结构和消息字段	书写的格式

2、工作流程



（1）创建一个记录器，并设置默认等级
（2）创建一个输出的容器（你的日志是输出到屏幕还是输出到文件）
（3）设置输出日志的等级

（4）创建格式化器（设置日志格式）
（5）使用格式化器渲染处理器（对于不同的输出位置，也可以添加不同的格式化器，使它们的格式不同）
（6）将处理器添加到记录器中
（7）调用记录器，写日志

四、loggers记录器

1、提供应用程序的调用接口

logger = logging.getLogger(__name__)

logger 是单例的、只要后面的 __name__ 相同，拿到的是同一个记录器

如果不写 name 默认是 root的名字

name的作用是，可以在过滤器中，通过名字来过滤日志

2、决定日志的级别

logger.setLevel()

3、将日志内容传递到相关联的处理器 handlers中

```
logger.addHandler()    # 添加
logger.removeHandler()# 移除
```

```
logger = logging.getLogger("applog")
# 记录器默认等级为 0
# 因此要 设置等级
print(logger.level)
# 将 默认的 处理器等级 归零
logging.basicConfig()
# 设置记录器的等级
logger.setLevel(logging.DEBUG)
# 设置日志
logger.debug("这是 loger 的 debug 日志")
```

五、Handlers处理器

处理器会将日志分发到不同的目的地。可以是文件、标准输出、邮件、或者通过socket、HTTP等协议发送到任何地方

- StreamHandler
- FileHandler
- BaseRotatingHandler
- RotatingFileHandler
- TimedRotatingFileHandler

常用的几个处理器

- SocketHandler
- DatagramHandler
- SMTPHandler
- SysLogHandler
- NTEventLogHandler
- HTTPHandler
- WatchedFileHandler
- QueueHandler
- NullHandler

1、StreamHandler

标准输出（stdout）分发器，如显示器

```
sh = logging.StreamHandler(stream=None)
```

2、FileHandler

将日志保存到磁盘文件的处理器
只能往一个文件中写

```
# FileHandler
fh = logging.FileHandler(filename='filename',mode='w',encoding='utf-8',delay=False)
```

3、RotatingFileHandler

多日志多文件的输出，可以按照一定的方式来生成多个文件，比如按照文件大小来输出多个文件。可以使我们的程序更健壮

4、TimedRotatingFileHandler

多日志多文件的输出，可以按照时间来生成多个日志文件，比如每天生成一个单独的日志文件

常用的处理器方法

（1）setFormatter()：给处理器设置日志格式

```
# setFormtter() 给处理器设置格式
sh = logging.StreamHandler()
# formatter格式
formatter = logging.Formatter("%(asctime)s|%(levelname)s|%(filename)s:%(lineno)s|%(message)s")
sh.setFormatter(formatter)
```

（2）setLevel()：设置日志级别
处理器的默认级别为 WARNING 级别，实际的级别 取 记录器 和 处理器两者更严格的
sh.setLevel(logging.DEBUG)

六、过滤器 Filter

用来过滤 记录器和处理器，通过名字进行过滤

```
logger = logging.getLogger('dq.zz.applog')
sh = logging.StreamHandler()
# 定义一个过滤器
flt = logging.Filter('dq.zz')
# 关联过滤器到记录器上
logger.addFilter(flt)
# 关联过滤器到处理器上
sh.addFilter(flt)
```

七、高级编程方式

```
#-*- encoding:utf-8 -*-
import logging
# 高级用法
# 1.记录器
logger = logging.getLogger('dp.zz.applog')
logger.setLevel(logging.INFO)
# 2.处理器
consoleHandler = logging.StreamHandler()
consoleHandler.setLevel(logging.WARNING)
# 没有给 fileHandler设置级别，使用logger的级别
fileHandler = logging.FileHandler(filename='appdemo.log')
# formtter格式
formatter = logging.Formatter("%(asctime)s|%(levelname)s|%(filename)s:%(lineno)s|%(message)s")
# 给处理器设置格式，可以给不同的处理器设置不同的格式
consoleHandler.setFormatter(formatter)
fileHandler.setFormatter(formatter)
# 给记录器设置处理器
logger.addHandler(consoleHandler)
logger.addHandler(fileHandler)
# 定义一个过滤器
flt = logging.Filter("dp.zz")
# 关联过滤器
logger.addFilter(flt)
# 打印日志
logger.debug("这是 debug 日志")
logger.info("这是 info 日志")
logger.warning("这是 warning 日志")
logger.error("这是 error 日志")
```

八、配置文件方式

配置文件

```
[loggers] 定义记录器（root必须写）
keys=root,applog

[handlers] 定义处理器
keys=fileHandler,consoleHandler

[formatters] 格式
keys=simpleFormatter

[logger_root] 定义root记录器
level=DEBUG  日志等级
handlers=consoleHandler  绑定处理器

[logger_applog] 定义applog记录器
level=DEBUG  日志等级
handlers=fileHandler  绑定处理器
qualname=applog   别名
propagate=0  继承，0为无继承
```

```
[handler_consoleHandler] 定义处理器
class=StreamHandler
args=(sys.stdout,)  默认为标准输出
level=DEBUG
formatter=simpleFormatter 设置格式

[handler_fileHandler]
class=handlers.TimedRotatingFileHandler 以时间规划的滚动文件
args=('applog.log','midnight',0,7)   midnight表示在午夜12点进行备份分割，0表示向后拖延0秒（立即分割），7表示日志保留7天
level=DEBUG
formatter=simpleFormatter

[formatter_simpleFormatter] 格式
format=%(asctime)s|%(levelname)s|%(filename)s:%(lineno)s|%(message)s
datefmt=%Y-%m-%d %H:%M:%S
```

```
#-*- encoding:utf-8 -*-
import logging
import logging.config
# 配置文件方式
logging.config.fileConfig('logging.conf')
rootLogger = logging.getLogger()
rootLogger.debug("这是 root Logger 的 debug 日志")
logger = logging.getLogger('applog')
logger.warning("这是 applog Logger 的 warning 日志")
```