logging库自学报告

徐浩博 2020010108 xuhb20@mails.tsinghua.edu.cn

概述

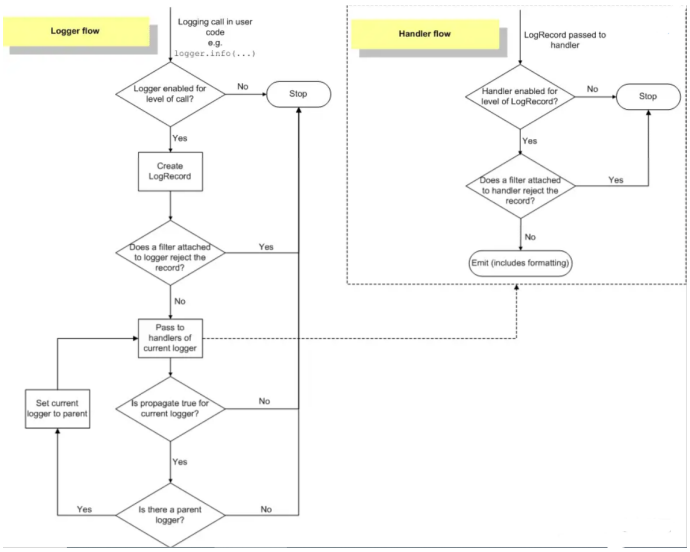
logging库是python标准库之一，作为应用程序构建块(Application Building Blocks)的重要组成部分，主要用于记录日志。通过logging库API，任何消息都可以作为日志记录和输出，既包括自己设定的日志内容，也包括第三方库的日志消息。日志级别有六种：NOTSET（0）、DEBUG（10）、INFO（20）、WARNING（30）、ERROR（40）、CRITICAL（50），括号内数字越大代表重要性越高，logging执行时会执行大于等于指定日志级别的内容。

logging库的基础类主要有四种：

- logger记录器，应用程序可以直接使用接口实现日志记录等功能
- handler处理器，将记录的日志发送到输出
- filter过滤器，对日志输出进行过滤控制
- formatter格式化器，定义日志记录输出的布局和样式

logging的主要工作流程如下：

1. logging被调用，记录的日志先由logger判断，若logger设置的日志级别高于记录的级别（见logger节），则终止，否则进入2
2. 判断filter设置的筛选条件是否将日志过滤，如果被过滤掉则终止，否则进入3
3. handler对象传入logger对象，记录日志的级别再由handler判断，若handler设置的日志级别高于记录的级别（见handler节），则终止，否则进入4
4. 判断当前logger是否有父对象，如果没有或者logger设置的propagate参数为False（见logger节），则终止；否则对于父对象logger重复3、4步，直到程序终止。注意，所有非根logger均有根logger作为父亲（见logger节），而根节点不存在父对象，因此此循环一定可以终止。



logger

logger默认的日志输出格式为`日志级别:Logger实例名:日志内容`，一个例子为`INFO:root:message`，并且默认的日志级别为WARNING。调用logging.basicConfig()方法可更改输出设置，常见可选参数有：

- filename 日志输出到文件的文件名，与stream参数互斥

- filemode 文件的格式，默认值为'a'，可选项有r(+) w(+) a(+)
- stream 日志输出的流，与filename互斥
- format 日志输出格式字符串，默认格式为%-formatting，该格式也是最常用的格式，一般采用%d表示signed integer，%d表示float，%s表示字符串(str()转换得)，-转换值将靠左对齐，例如`'%(asctime)s %(levelname)-8s %(message)s'`。更多信息请见Python文档
- style format的格式字符串风格（如果设置了format），默认值为%，可选值为`'%'`，`'{'`或`'$'`分别对应于 printf 风格，`str.format()`或`string.Template`，一般常用%风格
- level 日志输出级别，默认值为WARNING
- datefmt 日期显示格式，例如`'%Y-%m-%d %H:%M:%S'`，格式与time.strftime()相同

一个系统只有一个logger根对象，且不能被实例化。除此之外，自定义的logger对象输出日志都依靠根对象。查找并返回logger对象的方法为logging.getLogger(name=None)，当name为None时返回层级根目录，name一般为句点分割的层级值，如`foo.bar`是`foo`的子级。logger有以下常用方法：

- propagate：如果propagate为True，则该记录器事件除了会发送到该记录器所有处理程序之外，还会传递给祖先logger，如果为False，则不会传递。构造器默认为True。
- setLevel(level)：给记录器设置阈值为level，小于此阈值的消息将会被忽略。以下是一个实际应用的例子：

```
import logging

myLogger = logging.getLogger()
myLogger.setLevel(logging.INFO)
myLogger.setLevel(20)
```

  最后两句话是等效的。
- debug(msg,*args,**kwargs)，在此记录器上记录DEBUG级别的message，msg为字符串，args为字符串格式化的参数，kwargs会检查四个关键字参数。kwargs四个参数中较为常用的参数是exc_info，当记录异常消息时，需要设定exc_info为True（采用exception()除外）。
- info(msg,*args,**kwargs)，在此记录器上记录INFO级别的message，参数含义同debug()。
- warning(msg,*args,**kwargs)，在此记录器上记录WARNING级别的message，参数含义同debug()。
- error(msg,*args,**kwargs)，在此记录器上记录ERROR级别的message，参数含义同debug()。
- critical(msg,*args,**kwargs)，在此记录器上记录CRITICAL级别的message，参数含义同debug()。
- exception(msg,*args,**kwargs)，在此记录器上记录CRITICAL级别的message，但与critical()不同的的是，该方法仅用作记录异常消息，一般消息不应该调用此方法。
- log(level,msg,*args,**kwargs)，在此记录器上记录日志等级为level级别的消息，其余参数同debug()。下面给出一个应用的例子：

```
...
try:
    ans = x / y
except Exception as e:
    logging.error("Error occurred", exc_info = True)
    logging.exception("Error occurred")
    logging.error(level = logging.ERROR, "Error occurred", exc_info = True)
    logging.error(level = 40, "Error occurred", exc_info = True)
```

以上这四种log的方式可以互换。

- addHandler(hdlr) / removeHandler(hdlr) 将指定的handler处理器hdlr添加到此logger上/从logger中删除处理器hdlr
- addFilter(filter) / removeHandler(filter) 将指定的过滤器filter添加到此logger上/从logger中删除过滤器filter

handler

处理器handler也不可以被直接实例化，一般采用派生的方法。注意子类__init__()方法需要调用Handler.__init__()。handler有以下常用方法：

- __init__(level = NOTSET)，初始化时需要设置handler的日志处理级别，低于此级别的日志将被忽略。
- setLevel(level)，同上，设置handler的日志处理级别。
- setFormatter(fmt)，见logger节对于logging.basicConfig()的format参数介绍，可以将处理器的formatter设置为fmt。
- addFilter(filter) / removeHandler(filter) 将指定的过滤器filter添加到此logger上/从logger中删除过滤器filter

formatter

logging.Formatter(fmt=None, datefmt=None, style='%', validate=True, defaults=None)返回Formatter的一个实例对象，其中的主要参数介绍如下：

- style，可以是 '%', '{' 或 '$' 之一，默认为'%'，详见logger节logging.basicConfig()的style参数介绍
- fmt，日志输出格式字符串，详见logger节logging.basicConfig()的format参数介绍
- datefmt，日期显示格式，详见logger节logging.basicConfig()的datefmt参数介绍

formatter也可以通过重写内部方法达到指定格式的目的，但以上参数能够满足绝大多数情况下的需求，因此并不常用，不再赘述。

filter

logging.filter(name='')返回filter的一个实例对象，name对应logger的name，表明该logger及其子logger将通过此logger进行过滤。

filter只有一个常用方法：

- filter(record)，返回0表示此record被过滤掉不进行记录，非0表示保留该记录。在此方法中也可以对record就地进行修改。

模块级函数

除已介绍的logging.getLogger()外，logging还有一些常用函数：

- logging.debug(msg, *args, **kwargs)，与logger的debug方法类似，但此函数默认在根logger记录日志。
- logging.info、logging.warning、logging.error、logging.critical、logging.log与logging.debug类似，与logger的方法一一对应。
- logging.addLevelName(level, levelName)，自定义level名称，定义的名称可用logging.getLevelName(level)查询，后者也可以查询到预定义的CRITICAL、ERROR、WARNING、INFO、DEBUG级别

- logging.basicConfig(**kwargs)，此函数已在logger部分介绍过，需要强调的是，该函数通过使用默认的Formatter创建一个StreamHandler并将其加入根logger来执行。如果没有为根logger定义handler则debug()等函数将自动调用basicConfig的配置，否则在force参数不为True的条件下采用handler的配置。
    - 若basicConfig设置filename参数，则采用指定的文件名创建FileHandler
    - 若basicConfig没有设置参数，则采用指定的stream初始化StreamHandler
    - FileHandler和StreamHandler可以理解为一种特殊的handler，详见python官方文档

logging.config简介

- 字典中获取配置：logging.config.dictConfig(config)可以从字典中获取Logger、Handler等配置信息，其中config为字典
- 从.ini文件中获取配置：

```
logging.config.fileConfig(fname='xxx.ini', disable_existing_loggers=False)
logger = logging.getLogger("sampleLogger")
```

- 从.yaml文件中获取配置：采用yaml库将yaml文件内容转换成字典，之后采用第一种方法

综合实例

```
#创建FileHandler挂在根logger上，以文件形式输出
logging.basicConfig(
        format='%(asctime)s %(levelname)-8s %(message)s',
        level=logging.INFO,
        datefmt='%Y-%m-%d %H:%M:%S',
        filename='xxx.txt',
        filemode='w'
    )
#创建StreamHandler挂在根logger上，以stream（控制台）形式输出
console = logging.StreamHandler()
console.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s %(levelname)-8s %(message)s')
console.setFormatter(formatter)
logging.getLogger('').addHandler(console)
```

如果执行上面一段代码，则调用logging.info时，日志将以固定格式同时在控制台输出并打印在指定文件之中。

参考文献

1. logging — Logging facility for Python — Python 3.10.6 documentation
2. logging.config — Logging configuration — Python 3.10.6 documentation
3. logging.handlers — Logging handlers — Python 3.10.6 documentation
4. Python日志库logging总结 - 掘金 (juejin.cn)