**Python TimedRotatingFileHandler 多进程环境下的问题和解决方法**

**问题**

Python 自带了一个 handler 可以实现每天自动切割日志文件的功能（其实支持各种按时间切割的方法，不过按日期切割是最常用的一种）。

**切割** 这件事的触发和执行逻辑可以从 BaseRotatingHandler(logging.FileHandler) 里看到：

def emit(self, record):

"""

Emit a record.

Output the record to the file, catering for rollover as described

in doRollover().

"""

try:

if self.shouldRollover(record):

self.doRollover()

logging.FileHandler.emit(self, record)

except (KeyboardInterrupt, SystemExit):

raise

except:

self.handleError(record)

而 TimedRotatingFileHandler 只不过是依自己的逻辑实现了 self.shouldRollover 和 self.doRollover 方法。

共所周知的，这个 handler 在多进程环境下会有问题，比如丢失日志。这对于 web server 来说简直是个灾难。而这个问题的原因来自于它切割文件的方式，即 doRollover 方法。简化掉多余的代码后它看起来是这样的：

def doRollover(self):

if self.stream:

self.stream.close()

self.stream = None

# get the time that this sequence started at and make it a TimeTuple

dfn = self.baseFilename + "." + time.strftime(self.suffix, timeTuple)

if os.path.exists(dfn):

os.remove(dfn)

if os.path.exists(self.baseFilename):

os.rename(self.baseFilename, dfn)

self.rolloverAt = newRolloverAt

假设配置的文件名是 error.log，日志正准备从 11.11 切换到 11.12。那么 self.baseFilename 就是 error.log，dfn 就是 error.log.2016-11-11。即执行流程为：

1. 判断准备切入的文件是否存在，如果是，删掉之
2. 然后把 error.log 重命名为 error.log.2016-11-11
3. doRollover 结束，FileHandler.emit 将消息写入 error.log (新建)

单进程环境中这个流程没问题，但是多进程下因为每个进程都会调用一次 doRollover，就可能会发生像一个进程已经 rollover 完成，但是下一个进程把之前的 error.log 又给删掉了之类的问题。

**解决方法**

最简单直接的一种方法其实是：把多进程的 log handler 配置改为一个 TimedRotatingFileHandler + N 个 FileHandler，这样就绕过了 **多进程** 这个环境，其余的进程只负责向 error.log 文件写。

这需要实现进程间的不同配置，可以通过一个锁来实现，第一个成功 acquire 到这个锁的进程进行 rollover。

然而这存在一个切割不精确的问题，即在切割进程成功进行 rollover 之前，其他进程会把新日志写进旧文件。因此更好的办法是去修改 TimedRotatingFileHandler。改这个类的方法多种多样，我的做法是抛弃文件重命名这个操作，将 rollover 的实现变为进程无冲的。方法为：写文件时始终向带日期后缀的文件写，然后做一个 error.log 的软链接，指向最新的一条日志。

这样就抛弃了重命名的过程，行为和 FileHandler 更加类似。软连接的删除和新建也不存在进程间冲突。

code(可以直接用，但没有处理 utc):

class MultiProcessSafeDailyRotatingFileHandler(BaseRotatingHandler):

"""Similar with `logging.TimedRotatingFileHandler`, while this one is

- Multi process safe

- Rotate at midnight only

- Utc not supported

"""

def \_\_init\_\_(self, filename, encoding=None, delay=False, utc=False, \*\*kwargs):

self.utc = utc

self.suffix = "%Y-%m-%d"

self.baseFilename = filename

self.currentFileName = self.\_compute\_fn()

BaseRotatingHandler.\_\_init\_\_(self, filename, 'a', encoding, delay)

def shouldRollover(self, record):

if self.currentFileName != self.\_compute\_fn():

return True

return False

def doRollover(self):

if self.stream:

self.stream.close()

self.stream = None

self.currentFileName = self.\_compute\_fn()

def \_compute\_fn(self):

return self.baseFilename + "." + time.strftime(self.suffix, time.localtime())

def \_open(self):

if self.encoding is None:

stream = open(self.currentFileName, self.mode)

else:

stream = codecs.open(self.currentFileName, self.mode, self.encoding)

# simulate file name structure of `logging.TimedRotatingFileHandler`

if os.path.exists(self.baseFilename):

try:

os.remove(self.baseFilename)

except OSError:

pass

try:

os.symlink(self.currentFileName, self.baseFilename)

except OSError:

pass

return stream

# python多进程打日志的问题

# 出现的问题：

* 明明日志定义的是5个G，备份1个，结果却是文件和备份文件都只有10多M。
* 经常性的磁盘空间不足，实际占用并没有占那么多，而是程序占用文件的句柄，需要重启释放句柄才可以。

# 猜测问题和解决办法：

我们之前打日志用的RotatingFileHandler，这个是线程安全的，但是并不是进程安全的，我们使用gunicorn多进程的方式启动服务，所以猜测是单个文件被好几个进程写入，所以它到底该听谁的？是吧。

# 解决方案：

#### 使用ConcurrentRotatingFileHandler

该ConcurrentRotatingFileHandler类是一个下拉更换为Python的标准日志处理RotatingFileHandler。此模块使用文件锁定，以便多个进程可以同时记录到单个文件，而不会丢弃或破坏日志事件。此模块提供与RotatingFileHanler类似的文件循环方案。特别注意确保在旋转过程开始之前可以安全地旋转日志。（此模块解决Windows 上的RotatingFileHandler文件重命名问题，其中旋转失败意味着删除所有后续日志事件）。

此模块尝试不惜一切代价保留日志记录。这意味着日志文件将大于指定的最大（旋转）大小。因此，如果磁盘空间紧张，您可能希望坚持使用RotatingFileHandler，它将严格遵守最大文件大小。  
如果您有多个脚本（或多个脚本）的实例同时运行并写入同一个日志文件，那么所有脚本都应该使用ConcurrentRotatingFileHandler。您不应尝试混合和匹配RotatingFileHandler和ConcurrentRotatingFileHandler。  
这个包捆绑了[portalocker](https://links.jianshu.com/go?to=http%3A%2F%2Fcode.activestate.com%2Frecipes%2F65203%2F)来处理文件锁定。请注意，portalocker目前仅支持Unix（posix）和NT平台，因此该软件包也仅支持这些平台。  
使用以下命令安装此程序包：

pip install ConcurrentLogHandler

### 一个简单的事例：

from logging import getLogger, INFO

from cloghandler import ConcurrentRotatingFileHandler

import os

log = getLogger()

# Use an absolute path to prevent file rotation trouble.

logfile = os.path.abspath("mylogfile.log")

# Rotate log after reaching 512K, keep 5 old copies.

rotateHandler = ConcurrentRotatingFileHandler(logfile, "a", 512\*1024, 5)

log.addHandler(rotateHandler)

log.setLevel(INFO)

log.info("Here is a very exciting log message, just for you")

如果您要分发代码并且不确定ConcurrentLogHandler包是否已安装在代码运行的任何地方，那么Python可以轻松地优雅地回退到内置的RotatingFileHandler，这是一个示例：

try:

from cloghandler import ConcurrentRotatingFileHandler as RFHandler

except ImportError:

# Next 2 lines are optional: issue a warning to the user

from warnings import warn

warn("ConcurrentLogHandler package not installed. Using builtin log handler")

from logging.handlers import RotatingFileHandler as RFHandler

log = getLogger()

rotateHandler = RFHandler("/path/to/mylogfile.log", "a", 1048576, 15)

log.addHandler(rotateHandler)

### 配置文件示例

此示例显示如何将此日志处理程序与日志记录配置文件解析器一起使用。这允许您将日志记录配置代码与应用程序代码分开。

配置文件示例： logging.ini:

[loggers]

keys=root

[handlers]

keys=hand01

[formatters]

keys=form01

[logger\_root]

level=NOTSET

handlers=hand01

[handler\_hand01]

class=handlers.ConcurrentRotatingFileHandler

level=NOTSET

formatter=form01

args=("rotating.log", "a", 512\*1024, 5)

[formatter\_form01]

format=%(asctime)s %(levelname)s %(message)s

示例Python代码：app.py：

import logging, logging.config

import cloghandler

logging.config.fileConfig("logging.ini")

log = logging.getLogger()

log.info("Here is a very exciting log message, just for you")