**flask框架实战—优雅的使用log模块**

<https://www.jianshu.com/p/daf5c9e57c65>

## 前言

**本文转载自：**[**https://www.flyml.net**](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.flyml.net)  
最近在做一个测试工具平台，选择使用flask框架。过程中遇到一些 bug需要定位，之前一直使用的print调试。但是感觉这样太麻烦，浪费了很多时间。于是使用log来进行定位。  
于是上网查询了一些文档。发现网上很杂乱，初学者看了很可能一脸懵逼。没有一篇很完整的使用说明。我这里尝试归纳下，并总结自己的用法。如果你有什么疑问或者更好的建议，欢迎一起交流学习哦～本人QQ：995774387

## 使用current\_app

### from flask import current\_app 是什么？

简单来讲，current\_app表示当前运行程序文件的程序实例，属于应用上下文。如有兴趣深入了解可以查看文章[《flask开发之--请求/应用上下文》](https://links.jianshu.com/go?to=https%3A%2F%2Fysbolg.com)   
所以，在程序实例化的过程中，设置好log的路径，参数等信息。在views.py(视图函数)中，使用current\_app.logger.info("这是条测试日志")就可以实现日志打印啦！  
备注：pycharm等IDE工具有时并不能自动匹配到current\_app.logger，放心调用即可。

## flask--log使用

**会从以下几个问题切入讨论使用：**

* 日志在Flask之中的基础使用方法
* 如何在Flask之中配置日志的格式、文件存储地址、自动切分日志
* 在Blueprint之中如何使用日志
* 通过邮件或者Http接口输出错误日志
* 多机环境下， 如何使用日志进行定位的思路(比如需要增加hostname 定位具体在哪个docker环境)

### 日志在Flask之中的基础使用方法

首先我们从最简单的Flask程序开始。 从官网复制一个最小的能运行的Flask程序， 如下：

# -\*- coding:utf-8 -\*-

# Copied From :http://flask.pocoo.org/docs/1.0/quickstart/#a-minimal-application

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello, World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.debug = True

app.run()

毫无疑问， 访问http://127.0.0.1:5000 就能看到Hello World的输出。 接下来， 我们开始设置日志。 具体参考下面的代码：

# -\*- coding:utf-8 -\*-

from flask import Flask

import logging

from logging import FileHandler

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

app.logger.info("Info message")

app.logger.warning("Warning msg")

app.logger.error("Error msg!!!")

return 'Hello, World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.debug = True

handler = logging.FileHandler('flask.log')

app.logger.addHandler(handler)

app.run()

我们在main 函数之中， 设定：

* 我们会将日志写到flask.log之中
* Flask自带的app.logger 使用我们设定好的handler  
  再次访问http://127.0.0.1:5000 ，在Console看到如下信息：

[2018-12-11 20:41:07,344] INFO in main: Info message

[2018-12-11 20:41:07,344] WARNING in main: Warning msg

[2018-12-11 20:41:07,345] ERROR in main: Error msg!!!

127.0.0.1 - - [11/Dec/2018 20:41:07] "GET / HTTP/1.1" 200 -

可以看到， 我们在hello\_world 函数之中期望打印出来的日志都正常输出了。 同时， 你也应该能看到一个flask.log文件。 不过里面的内容就不如Console的log那样， 包含了时间来源等等的信息了。 内容如下：

Info message

Warning msg

Error msg!!!

### 日志配置

虽然是在Flask之中调用了自带的app.logger， 但是毕竟还是使用了公共库的logging。 配置方面应该能找到很多很多资料。 再次演示一下自认为比较好用的一个配置。

如果一个日志不停的增长下去， 显然不是什么好事。 因此日志必须要进行切分。 常见的两种方式：

* 按照大小
* 按照时间

**按照日志大小切分**  
如果是按照大小进行切分， 引入RotatingFileHandler 即可。 举例：

from logging.handlers import RotatingFileHandler

handler = RotatingFileHandler("flask.log", maxBytes=1024000, backupCount=10)

简单解释一下：

* “flask.log” 就是日志的文件名
* maxBytes 就是 日志大小
* backupCount 就是保留的日志个数。 比如flask.log 写满了， 就会被重命名成flask.log.1, 程序继续向flask.log写入。

更详细的解释可以看看官网说明: [https://docs.python.org/2/library/logging.handlers.html#rotatingfilehandler](https://links.jianshu.com/go?to=https%3A%2F%2Fdocs.python.org%2F2%2Flibrary%2Flogging.handlers.html%23rotatingfilehandler)  
**按照日期进行切分**  
个人比较习惯这种方式。 在logging这个库之中， 还支持按照分钟、小时、天等级别进行切分。 根据我们业务的大小， 我一般选择按照“天” 进行切分。 可以参考下面的配置：

from logging.handlers import TimedRotatingFileHandler

handler = TimedRotatingFileHandler(

"flask.log", when="D", interval=1, backupCount=15,

encoding="UTF-8", delay=False, utc=True)

* when=D： 表示按天进行切分
* interval=1： 每天都切分。 比如interval=2就表示两天切分一下。
* backupCount=15: 保留15天的日志
* encoding=UTF-8: 使用UTF-8的编码来写日志
* utc=True: 使用UTC+0的时间来记录 （一般docker镜像默认也是UTC+0）

**配置日志格式**  
前面我们也看到， 在日志文件之中， 除了记录下来的消息， 其他辅助信息完全没有。 以下是我自己的配置以及相应的输出

# My Config

# [%(asctime)s][%(filename)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s

# Output

[2018-12-11 21:13:23,315][main.py:10][INFO][24012] - Info message

[2018-12-11 21:13:23,315][main.py:11][WARNING][24012] - Warning msg

[2018-12-11 21:13:23,316][main.py:12][ERROR][24012] - Error msg!!!

注意： 设置%(thread)d 并不是必须的。 但是如果你在一个多线程、或者多个docker环境的时候， 加上这个thread， 有助于你把同一个会话进程抽取出来。 因为多进程、多线程的时候， 日志的顺序可能会被打乱。

### Blueprint 之中使用日志

当你的Flask项目膨胀到一定规模的时候， 全部都写到主入口之中。 一定需要按照模块进行拆分。 Blueprint(蓝图)就是这个时候需要使用的东西。 那么在Blueprint之中， 如何使用日志呢？

我们先基于前面的程序搭好框架：  
**主入口 main.py**

# -\*- coding:utf-8 -\*-

from flask import Flask

import logging

from logging.handlers import TimedRotatingFileHandler

from views.simple\_page import simple\_page

app = Flask(\_\_name\_\_)

app.register\_blueprint(simple\_page, url\_prefix="/simple\_page")

@app.route('/')

def hello\_world():

app.logger.info("Info message")

app.logger.warning("Warning msg")

app.logger.error("Error msg!!!")

return 'Hello, World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.debug = True

formatter = logging.Formatter(

"[%(asctime)s][%(filename)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s")

handler = TimedRotatingFileHandler(

"flask.log", when="D", interval=1, backupCount=15,

encoding="UTF-8", delay=False, utc=True)

app.logger.addHandler(handler)

handler.setFormatter(formatter)

app.run()

**注意：** 已经注册好了蓝图simple\_page, 并且设置了url\_prefix=simple\_page  
**蓝图 views.simple\_page**

from flask import Blueprint

simple\_page = Blueprint('simple\_page', \_\_name\_\_)

@simple\_page.route('/')

def show():

return "simple page"

文件目录长下面这样：

flask-blueprint-file-structure.jpg

其实在blueprint之中， 使用日志的方式也很简单。 参考下面simple\_page.py之中增加日志调用之后的代码：

from flask import Blueprint

from flask import current\_app

simple\_page = Blueprint('simple\_page', \_\_name\_\_)

@simple\_page.route('/')

def show():

current\_app.logger.info("simple page info...")

current\_app.logger.warning("warning msg!")

current\_app.logger.error("ERROR!!!!!")

return "simple page"

关键就是from flask import current\_app就可以获取当前的flask的app了。 我们来看看在日志文件之中的日志的样子：

[2018-12-12 08:39:13,431][simple\_page.py:9][INFO][22908] - simple page info...

[2018-12-12 08:39:13,433][simple\_page.py:10][WARNING][22908] - warning msg!

[2018-12-12 08:39:13,433][simple\_page.py:11][ERROR][22908] - ERROR!!!!!

看起来一切正常。 Console 之中的日志：

[2018-12-12 08:39:13,431] INFO in simple\_page: simple page info...

[2018-12-12 08:39:13,433] WARNING in simple\_page: warning msg!

[2018-12-12 08:39:13,433] ERROR in simple\_page: ERROR!!!!!

看起来也不错。 完美！

### 错误日志发送邮件或者调用HTTP接口

当系统上线之后， 多多少少程序会因为各种各样的问题产生Error级别的日志。 但是我们又不能一直盯着线上日志。 一个简单的办法， 当出现错误日志的时候， 主动通知。 比如发邮件或者调用Http Webhook 接口。

在标准日志库logging之中，就有SMTPHandler跟HTTPHandler可以实现这个功能。 下面以发邮件为例子。

我们接着上面Blueprint的代码接着写。  
**主入口main.py**

# -\*- coding:utf-8 -\*-

from flask import Flask

import logging

from logging.handlers import TimedRotatingFileHandler

from logging.handlers import SMTPHandler

from views.simple\_page import simple\_page

app = Flask(\_\_name\_\_)

app.register\_blueprint(simple\_page, url\_prefix="/simple\_page")

@app.route('/')

def hello\_world():

app.logger.info("Info message")

app.logger.warning("Warning msg")

app.logger.error("Error msg----1")

app.logger.error("Error msg----2")

app.logger.error("Error msg----3")

return 'Hello, World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.debug = True

# File and Console handler &amp; formtter

formatter = logging.Formatter(

"[%(asctime)s][%(module)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s")

handler = TimedRotatingFileHandler(

"flask.log", when="D", interval=1, backupCount=15,

encoding="UTF-8", delay=False, utc=True)

app.logger.addHandler(handler)

handler.setFormatter(formatter)

# Email Handler

mail\_handler = SMTPHandler(

mailhost='10.64.1.85',

fromaddr='flask-admin@trendmicro.com',

toaddrs=['wenjun\_yang@trendmicro.com'],

subject='Flask Application Error'

)

mail\_handler.setLevel(logging.ERROR)

mail\_handler.setFormatter(logging.Formatter(

"[%(asctime)s][%(module)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s"

))

app.logger.addHandler(mail\_handler)

app.run()

关键部分：

# 引入类库

from logging.handlers import SMTPHandler

# 配置handler&amp;formatter

mail\_handler = SMTPHandler(

mailhost='10.64.xxx,yyy',

fromaddr='flask-admin@abc.com',

toaddrs=['superman@abc.com'],

subject='Flask Application Error'

)

mail\_handler.setLevel(logging.ERROR)

mail\_handler.setFormatter(logging.Formatter(

"[%(asctime)s][%(module)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s"

))

# app.logger

app.logger.addHandler(mail\_handler)

注意： 我们在函数之中， 连续输入了3条Error信息， EmailHandler并不会帮助我们合并这三条信息， 而是会分别发送邮件过来。

**错误!未指定文件名。**

image

### 在Log之中增加其他的辅助信息

增加辅助信息有两种比较优雅的方式：

* 通过LogFilter
* 通过自定义的Formatter  
  假设我们在Log之中需要增加当前的环境的hostname，我们新的formatter长下面这样

# 原来的formatter

[%(asctime)s][%(filename)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s

# 新的Formatter

(%(hostname)s)[%(asctime)s][%(filename)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s

**使用LogFilter的方式**

# 首先自定义一个LogFilter

class ContextFilter(logging.Filter):

'''Enhances log messages with contextual information'''

def filter(self, record):

record.hostname = "my-windows-10"

return True

# 在main函数之中， 增加加载这个filter即可

handler.addFilter(ContextFilter())

### 多机环境下的错误定位思路

比如我们使用前后端的架构， 会在多台机器甚至分布在不同数据中心的机器上面部署相同的程序， 相互构成一个集群。

这种场景下面， 出现错误的时候， 其实定位问题是非常麻烦的。 所以， 首先我们要定位在哪个环境上面出的问题。同时， 我们还需要借助其他的手段。 下面说一下思路：

增加hostname 字段  
flask的日志默认是不带hostname字段的， 增加的方法就需要用到上一节提到的方法。

通过ELK等方式， 将日志统一收集到一个集中的地方进行处理

如果使用ELK， 可以直接在Kibana上面查找Error级别的日志， 并且通过thread / hostname 等过滤出某一个环境上面的日志进行查看。

* 增加hostname 字段
* flask的日志默认是不带hostname字段的， 增加的方法就需要用到上一节提到的方法。
* 通过ELK等方式， 将日志统一收集到一个集中的地方进行处理
* 如果使用ELK， 可以直接在Kibana上面查找Error级别的日志， 并且通过thread / hostname 等过滤出某一个环境上面的日志进行查看。

2人点赞

[Code-flask框架](/nb/39705169)

更多精彩内容，就在简书APP

作者：张雨生在路上  
链接：https://www.jianshu.com/p/daf5c9e57c65  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。