**9.1日志处理·日志处理**

Log日志都是一个系统不可缺少的一个重要组成部分，也是一个正规系统都应该有的功能，否则当运行良好的系统突然无法正常运转的时候，你就会束手无策，而有了日志，你就可以通过日志查看其中缘由，从而解决问题。

# 日志级别

## ERROR

发生错误时，如IO操作失败或者连接问题

## WARNING

发生很重要的事件，但是并不是错误时，如用户登录密码错误

## INFO

处理请求或者状态变化等日常事务

## DEBUG

调试过程中使用DEBUG等级，如算法中每个循环的中间状态

# Flask的打印日志

## 常规打印

app.logger.debug('A value for debugging')  
app.logger.warning('A warning occurred (%d apples)', 42)  
app.logger.error('An error occurred')

## 蓝图中打印

from flask import current\_app  
  
current\_app.logger.debug('A value for debugging')  
current\_app.logger.warning('A warning occurred (%d apples)', 42)  
current\_app.logger.error('An error occurred')

# 新建mlogging目录

mkdir mlogging

# 新建mlogging/\_\_init\_\_py文件

这里只简单的配置了三种日志处理方式：

* 控制台输出StreamHandler
* 按照日志大小进行切分RotatingFileHandler
* 按照日期进行切分TimedRotatingFileHandler

并去掉了flask默认的处理器default\_handler

import logging  
import os  
from logging.handlers import RotatingFileHandler, TimedRotatingFileHandler  
  
from flask.logging import default\_handler  
  
  
class LoggingConfig:  
 """  
 日志配置  
 """  
 # 日志输出格式配置  
 formatter = "[%(asctime)s][%(filename)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s"  
 # 日志等级  
 log\_level = "DEBUG"  
 # 文件日志默认存放在项目根目录下的logs目录下  
 log\_path = None  
 # 日志处理器配置StreamHandler/TimedRotatingFileHandler/RotatingFileHandler  
 log\_handler = "StreamHandler"  
 # 日志文件最大值默认10M  
 max\_bytes = 1024 \* 1024 \* 10  
 # 保留日志文件数  
 backup\_count = 30  
  
 def \_\_init\_\_(self, app=None):  
 self.app = app  
 if app:  
 self.init\_app(app)  
  
 def init\_app(self, app):  
 self.app = app  
 if app.config.get("LOG\_FORMATTER"):  
 self.formatter = app.config.get("LOG\_FORMATTER")  
 if app.config.get("LOG\_LEVEL"):  
 self.log\_level = app.config.get("LOG\_LEVEL")  
 self.log\_path = os.path.join(app.root\_path, 'logs')  
 if app.config.get("LOG\_PATH"):  
 self.log\_path = app.config.get("LOG\_PATH")  
 if app.config.get("LOG\_HANDLER"):  
 self.log\_handler = app.config.get("LOG\_HANDLER")  
 if app.config.get("LOG\_MAX\_BYTES"):  
 self.max\_bytes = app.config.get("LOG\_MAX\_BYTES")  
 if app.config.get("LOG\_BACKUP\_COUNT"):  
 self.backup\_count = app.config.get("LOG\_BACKUP\_COUNT")  
 # 设置flask-logger的日志级别  
 self.app.logger.setLevel(self.get\_level())  
 # 删除默认的日志处理器-控制台输出  
 self.app.logger.removeHandler(default\_handler)  
 if self.log\_handler.lower() == "TimedRotatingFileHandler".lower():  
 self.app.logger.addHandler(self.build\_timed\_rotating\_file\_handler())  
 elif self.log\_handler.lower() == "RotatingFileHandler".lower():  
 self.app.logger.addHandler(self.build\_rotating\_file\_handler())  
 else:  
 handler = logging.StreamHandler()  
 handler.setFormatter(logging.Formatter(self.formatter))  
 self.app.logger.addHandler(handler)  
  
 def build\_rotating\_file\_handler(self):  
 """  
 按照日志大小进行切分  
 :return:  
 """  
 if not os.path.exists(self.log\_path):  
 os.makedirs(self.log\_path)  
 handler = RotatingFileHandler(os.path.join(self.log\_path, 'flask.log'), maxBytes=self.max\_bytes,  
 backupCount=self.backup\_count)  
 handler.setLevel(self.get\_level())  
 handler.setFormatter(logging.Formatter(self.formatter))  
 return handler  
  
 def build\_timed\_rotating\_file\_handler(self):  
 """  
 按照日期进行切分  
 :return:  
 """  
 if not os.path.exists(self.log\_path):  
 os.makedirs(self.log\_path)  
 handler = TimedRotatingFileHandler(os.path.join(self.log\_path, 'flask.log'), when="D", interval=1,  
 backupCount=self.backup\_count, encoding="UTF-8",  
 delay=False, utc=True)  
 handler.setLevel(self.get\_level())  
 handler.setFormatter(logging.Formatter(self.formatter))  
 return handler  
  
 def get\_level(self):  
 """  
 获取日志输出级别  
 :return:  
 """  
 if self.log\_level.lower() == "INFO".lower():  
 return logging.INFO  
 elif self.log\_level.lower() == "WARN":  
 return logging.WARN  
 elif self.log\_level.lower() == "ERROR":  
 return logging.ERROR  
 elif self.log\_level.lower() == "FATAL":  
 return logging.FATAL  
 else:  
 return logging.DEBUG

# 修改config.py

增加日志相关配置

一些配置说明：

* LOG\_FORMATTER日志：输出格式配置
* LOG\_LEVEL：日志等级
* LOG\_PATH：文件日志默认存放在项目根目录下的logs目录下
* LOG\_HANDLER：日志处理器配置StreamHandler/TimedRotatingFileHandler/RotatingFileHandler
* LOG\_MAX\_BYTES：日志文件最大值默认10M
* LOG\_BACKUP\_COUNT：保留日志文件数

代码片段：

class BaseConfig(ｏｂｊｅｃｔ):  
 """  
 基础配置  
 """  
 #### 省略其他配置……  
 # ------日志相关配置------  
 # 日志格式输出配置  
 LOG\_FORMATTER = os.getenv("LOG\_FORMATTER", "[%(asctime)s][%(filename)s:%(lineno)d][%(levelname)s][%(thread)d] - %(message)s")  
 # 日志等级DEBUG/INFO/WARN/ERROR  
 LOG\_LEVEL = os.getenv("LOG\_LEVEL", "DEBUG")  
 # 日志存放目录-文件日志默认存放在项目根目录下的logs目录下  
 LOG\_PATH = os.getenv("LOG\_PATH", None)  
 # 日志处理器StreamHandler/TimedRotatingFileHandler/RotatingFileHandler  
 LOG\_HANDLER = os.getenv("LOG\_HANDLER", "StreamHandler")  
 # 日志文件最大值默认10M  
 LOG\_MAX\_BYTES = int(os.getenv("LOG\_MAX\_BYTES", 1024 \* 1024 \* 10))  
 # # 保留日志文件数  
 LOG\_BACKUP\_COUNT = int(os.getenv("LOG\_BACKUP\_COUNT", 30))  
 # ------日志相关配置------

# 修改app.py

初始化日志配置

代码片段：

from mlogging import LoggingConfig  
  
# 配置日志  
LoggingConfig(app)

# 修改controllers/user\_controller.py

登录接口增加日志输出

代码片段：

@user.route("/login", methods=['POST'])  
def user\_login():  
 """  
 登录  
 :return:  
 """  
 form = LoginForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 res = user\_service.login(form.userName.data, form.password.data)  
 current\_app.logger.info("[{id}][{userName}][登录系统]".format(\*\*res))  
 token\_strategy = TokenStrategyFactory.get\_instance()  
 return R.data(token\_strategy.set({  
 "userId": res.get("id"),  
 "userName": res.get("userName")  
 }))

# 运行Flask服务

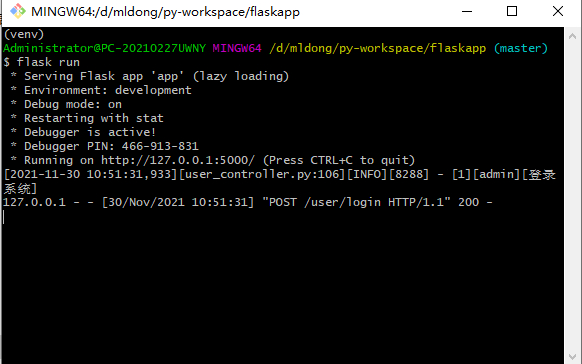
默认为控制台输出

flask run

# 使用Postman接口测试工具访问

## 访问/user/login

查看控制台日志



# 配置环境变量重新启动Flask服务

export LOG\_HANDLER=RotatingFileHandler  
  
flask run

# 再次使用Postman接口测试工具访问

## 访问/user/login

日志会输出到文件

