LOG

Log简介

logging模块是Python内置的标准模块，主要用于输出运行日志，可以设置输出日志的等级、日志保存路径、日志文件回滚等；相比print，具备如下优点：

通过log的分析，可以方便用户了解系统或软件、应用的运行情况；如果你的应用log足够丰富，也可以分析以往用户的操作行为、类型喜好、地域分布或其他更多信息；如果一个应用的log同时也分了多个级别，那么可以很轻易地分析得到该应用的健康状况，及时发现问题并快速定位、解决问题，补救损失。

Log的用途

不管是使用何种编程语言，日志输出几乎无处不再。总结起来，日志大致有以下几种用途：

- 问题追踪：通过日志不仅仅包括我们程序的一些bug，也可以在安装配置时，通过日志可以发现问题。
- 状态监控：通过实时分析日志，可以监控系统的运行状态，做到早发现问题、早处理问题。
- 安全审计：审计主要体现在安全上，通过对日志进行分析，可以发现是否存在非授权的操作

Log等级

- DEBUG最详细的日志信息，典型应用场景是 问题诊断
- INFO信息详细程度仅次于DEBUG，通常只记录关键节点信息，用于确认一切都是按照我们预期的那样进行工作
- WARNING当某些不期望的事情发生时记录的信息（如，磁盘可用空间较低），但是此时应用程序还是正常运行的
- ERROR由于一个更严重的问题导致某些功能不能正常运行时记录的信息 如IO操作失败或者连接问题
- CRITICAL当发生严重错误，导致应用程序不能继续运行时记录的信息

Log模块的四大组件

- Loggers

  提供应用程序代码直接使用的接口

- Handlers

  用于将日志记录发送到指定的目的位置

```
  FileHandler：logging.FileHandler；日志输出到文件
  RotatingHandler：logging.handlers.RotatingHandler；日志回滚方式，支持日志文件最大数量和日志文件回滚
  SMTPHandler：logging.handlers.SMTPHandler；远程输出日志到邮件地址
  HTTPHandler：logging.handlers.HTTPHandler；通过"GET"或者"POST"远程输出到HTTP服务器
  。。。。。
```

- Filters

  提供更细粒度的日志过滤功能，用于决定哪些日志记录将会被输出（其它的日志记录将会被忽略）

- Formatters

  用于控制日志信息的最终输出格式

```
  %(levelno)s：打印日志级别的数值
  %(levelname)s：打印日志级别的名称
  %(pathname)s：打印当前执行程序的路径，其实就是sys.argv[0]
  %(filename)s：打印当前执行程序名
  %(funcName)s：打印日志的当前函数
  %(lineno)d：打印日志的当前行号
  %(asctime)s：打印日志的时间
  %(thread)d：打印线程ID
  %(threadName)s：打印线程名称
  %(process)d：打印进程ID
  %(message)s：打印日志信息
```

示例

```
  import logging
  logger = logging.getLogger(__name__)
  logger.setLevel(level = logging.INFO)
  handler = logging.FileHandler(“log.txt”)
  formatter = logging.Formatter('%(asctime)s - %(name)s - %
(levelname)s - %(message)s')
  handler.setFormatter(formatter)
  logger.addHandler(handler)

  logger.info("Start print log")
  logger.debug("Do something")
  logger.warning("Something maybe fail.")
  logger.info("Finish")
```

Django中的配置

```
  ADMINS = (
   ('tom','*******@163.com'),
  )
  #配置邮件
  EMAIL_BACKEND ='django.core.mail.backends.smtp.EmailBackend'
  SERVER_EMAIL=EMAIL_HOST_USER
  LOGGING = {
   'version': 1,
   'disable_existing_loggers': True,
   'formatters': {
   'standard': {
    'format': '%(asctime)s [%(threadName)s:%(thread)d] [%(name)s:%
(lineno)d] [%(module)s:%(funcName)s] [%(levelname)s]- %(message)s'}
   },
   'filters': { #过滤条件
   #要求debug是False才记录
   'require_debug_false': {
    '()': 'django.utils.log.RequireDebugFalse',
    }
   },
   'handlers': {
```

```
    'null': {
     'level': 'DEBUG',
     'class': 'logging.NullHandler',
    },
    'mail_admins': {#一旦线上代码报错 邮件提示
     'level': 'ERROR',
     'class': 'django.utils.log.AdminEmailHandler',
     'filters': ['require_debug_false'],
    },
    'debug': {
     'level': 'DEBUG',
             'class': 'logging.handlers.RotatingFileHandler',
             'filename': os.path.join(BASE_DIR, "log", 'debug.log'),
# 文件路径
             'maxBytes': 1024 * 1024 * 5, #5兆的数据
             'backupCount': 5, #允许有5这样的文件
             'formatter': 'standard', #格式
    },
    'console':{
     'level': 'DEBUG',
     'class': 'logging.StreamHandler',
     'formatter': 'standard',
    },
    },
    'loggers': {
    'django': {
     'handlers': ['console'],
     'level': 'DEBUG',
     'propagate': False
    },
    'django.request': {
     'handlers': ['debug','mail_admins'],
     'level': 'ERROR',
     'propagate': True,#是否继承父类的log信息
    },
    # 对于不在 ALLOWED_HOSTS 中的请求不发送报错邮件
    'django.security.DisallowedHost': {
     'handlers': ['null'],
     'propagate': False,
    },
    }
   }
```

使用：

```
import logging
logger = logging.getLogger("django") # 为loggers中定义的名称
logger.info("some info...")
```