logging模块简介

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：

- 可以通过设置不同的日志等级，在release版本中只输出重要信息，而不必显示大量的调试信息；
- print将所有信息都输出到标准输出中，严重影响开发者从标准输出中查看其它数据；logging则可以由开发者决定将信息输出到什么地方，以及怎么输出；

logging模块使用

基本使用

- logging 日志基本格式化

```
logging.basicConfig(format='%(asctime)s - %(pathname)s[line:%(lineno)d] - %
(levelname)s: %(message)s',
                    level=logging.DEBUG)
```

- logging 日志基本格式

```
logging.basicConfig(level=logging.DEBUG,#控制台打印的日志级别
                    filename='new.log',
                    filemode='a',##模式，有w和a，w就是写模式，每次都会重新写日志，覆盖
之前的日志
                    #a是追加模式，默认如果不写的话，就是追加模式
                    format=
                    '%(asctime)s - %(pathname)s[line:%(lineno)d] - %(levelname)s:
%(message)s'
                    #日志格式
                    )
```

- 基本方式

```
import logging
logging.basicConfig(level = logging.INFO,format = '%(asctime)s - %(name)s - %
(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

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

- 打印日志
  logging中可以选择很多消息级别，如debug、info、warning、error以及critical。通过赋予logger或者handler不同的级别，开发者就可以只输出错误信息到特定的记录文件，或者在调试时只记录调试信息。

```
logging.debug('debug 信息')
logging.info('info 信息')
logging.warning('warning 信息')
logging.error('error 信息')
logging.critical('critial 信息')
```

> 这种打印日志需要将日志格式化放在首次调用的位置，一旦格式化执行执行后，其余格式化将不再执行 参考

> 如需要多个日志输出则需要使用动态输出日志 参考

基本参数

logging.basicConfig函数各参数：

参数：	作用
filename：	指定日志文件名
filemode：	和file函数意义相同，指定日志文件的打开模式，'w'或者'a'
format：	指定输出的格式和内容，format可以输出很多有用的信息
-	-
%(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

参数：	作用
%(message)s：	打印日志信息
-	-
datefmt：	指定时间格式，同time.strftime()
level：	设置日志级别，默认为logging.WARNNING
stream：	指定将日志的输出流，可以指定输出到sys.stderrsys.stdout或者文件，默认输出到sys.stderr，当stream和filename同时指定时，stream被忽略

logging.logger & logging.handler

logger & handler 基本使用

典型的日志记录步骤：

- 创建 logger
- 创建 handler
- 定义 formatter
- 给 handler 添加 formatter
- 给 logger 添加 handler

代码：

```
import logging

# 1、创建一个logger
logger = logging.getLogger('mylogger')
logger.setLevel(logging.DEBUG)

# 2、创建一个handler，用于写入日志文件
fh = logging.FileHandler('test.log')
fh.setLevel(logging.DEBUG)

# 再创建一个handler，用于输出到控制台
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)

# 3、定义handler的输出格式（formatter）
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %
(message)s')

# 4、给handler添加formatter
fh.setFormatter(formatter)
ch.setFormatter(formatter)

# 5、给logger添加handler
```

```
logger.addHandler(fh)
logger.addHandler(ch)
```

参考

将日志写入文件

设置logging，创建一个FileHandler，并对输出消息的格式进行设置，将其添加到logger，然后将日志写入到指定的文件中

```
import logging
logger = logging.getLogger(__name__)
logger.setLevel(level = logging.INFO)
handler = logging.FileHandler("log.txt")
handler.setLevel(logging.INFO)
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")
```

将日志同时输出到屏幕和日志文件

logger中添加StreamHandler，可以将日志输出到屏幕上

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

console = logging.StreamHandler()
console.setLevel(logging.INFO)

logger.addHandler(handler)
logger.addHandler(console)

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

可以发现，logging有一个日志处理的主对象，其他处理方式都是通过addHandler添加进去，logging中包含的handler主要有如下几种:

handler名称	位置	作用
StreamHandler	logging.StreamHandler	日志输出到流，可以是sys.stderr, sys.stdout或者文件
FileHandler	logging.FileHandler	日志输出到文件
BaseRotatingHandler	logging.handlers.BaseRotatingHandler	基本的日志回滚方式
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
TimeRotatingHandler	logging.handlers.TimeRotatingHandler	日志回滚方式，在一定时间区域内回滚日志文件
SocketHandler	logging.handlers.SocketHandler	远程输出日志到TCP/IP sockets
DatagramHandler	logging.handlers.DatagramHandler	远程输出日志到UDP sockets
SMTPHandler	logging.handlers.SMTPHandler	远程输出日志到邮件地址
SysLogHandler	logging.handlers.SysLogHandler	日志输出到syslog
NTEventLogHandler	logging.handlers.NTEventLogHandler	远程输出日志到Windows NT/2000/XP的事件日志
MemoryHandler	logging.handlers.MemoryHandler	日志输出到内存中的指定buffer
HTTPHandler	logging.handlers.HTTPHandler	通过"GET"或者"POST"远程输出到HTTP服务器

日志回滚

使用RotatingFileHandler，可以实现日志回滚

```
import logging
from logging.handlers import RotatingFileHandler

logger = logging.getLogger(__name__)
logger.setLevel(level = logging.INFO)
# 定义一个RotatingFileHandler，最多备份3个日志文件，每个日志文件最大1K
rHandler = RotatingFileHandler("log.txt",maxBytes = 1*1024,backupCount = 3)
rHandler.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %
(message)s')
rHandler.setFormatter(formatter)

console = logging.StreamHandler()
console.setLevel(logging.INFO)
console.setFormatter(formatter)
```

```
logger.addHandler(rHandler)
logger.addHandler(console)

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