# 8. 文件，日志，异常

## 日志

### 日志是什么

日志是十分有用的工具，来帮助我们理解程序的运行状态，在前面学习过程中，大部分内容是通过print在控制台上，但线上真正的运行环境是不能把日志输出到控制台的，不方便后期进行程序问题查找与排查。

在日志中，可以定义数据的产生的时间，发生的程序位置，提供错误的堆栈信息，帮助我们快速定位到问题所在。

日中存在多种级别，分别为CRITICAL,ERROR,WARNING,INFO,DEBUG,NOTSET6个级别。不同的日志级别对应的消息重要性不同。 \* CRITICAL: 当发生严重错误，导致程序都不能正常运行所记录的信息。 \* ERROR: 由于某个严重的问题，导致程序某个功能不能正常运行。 \* WARNING: 比错误等级低一个级别，可能出现的问题，但不影响程序执行，需要监控人员注意 \* INFO: 用来记录关键点信息，检查程序是否按照正常流程执行，一般用在流程信息输出。 \* DEBUG:日志信息更详细的输出，用来帮助解决程序问题。

在开发阶段一般我们将日志模式设置为DEBUG，帮助我们梳理程序执行，并检查意外情况发生状态，在线上阶段应该使用WARNING以上级别日志，减少磁盘IO输出，并且能提高获取错误日志信息，但在新系统刚上线，一般还会先采用一段时间INFO帮助检查程序运行的状态。

### 日志应该怎么使用

在Python中，已经集成logging日志模块，直接可以引用即可。

import logging  
  
logging.debug("test debug")  
logging.info("test info")  
logging.warning("test warning")  
logging.error("test error")  
logging.critical("test critical")

执行该文件，输出内容如下所示。

ruiqi@ruiqi:~/Desktop/code/code/study$ python log.py   
WARNING:root:test warning  
ERROR:root:test error  
CRITICAL:root:test critical

有的同学看到内容就会显得疑问，我们上面明明设置了五种不同类型内容，为什么会只输出三种呢？每个消息体前面的大写字母代表什么？root代表什么呢？

1. 每个消息体前面的大写字母代表什么？

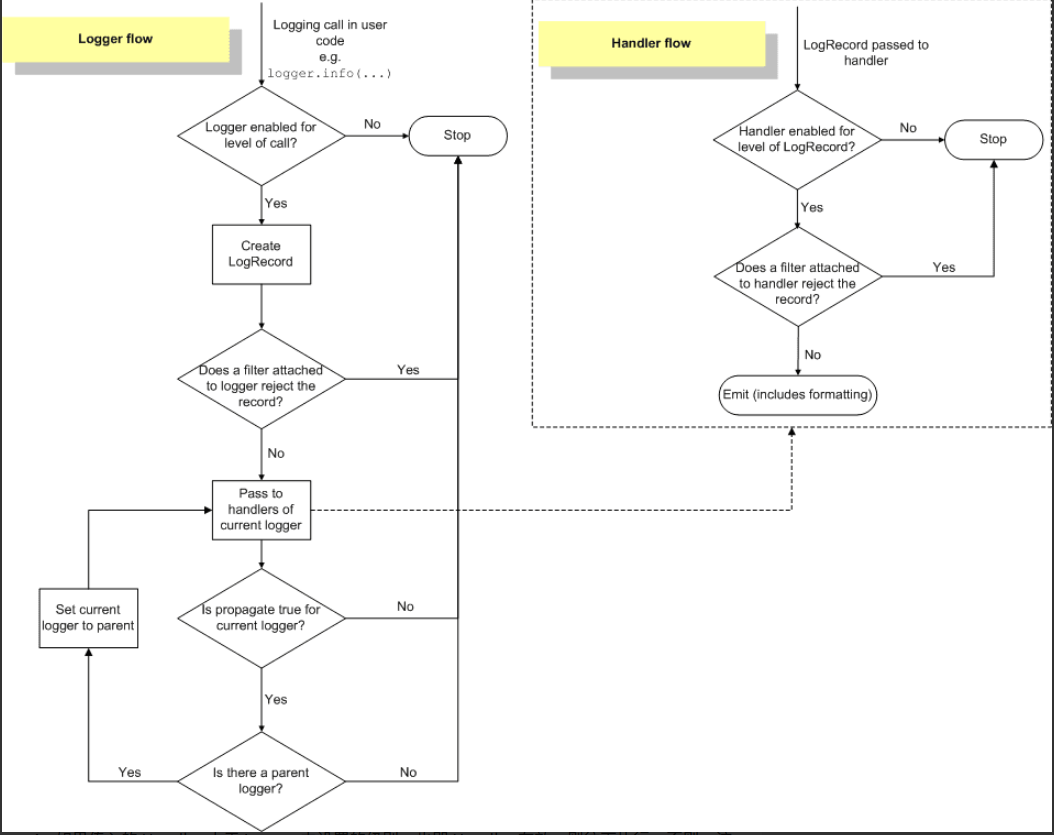
在每行日志内容前面展示的是日志等级，该形式是日志模式中默认的输出格式。在后面的案例中，我们会对其案例进行修改，提高可辨识度。 2. 为什么上面只输出三种内容？

在Python日志模块中默认设置的日志等级是WARNING,在不修改日志等级的基础上，日志只输出高于其等级的日志内容。 3. root代表什么内容呢？

root是默认提供的Logger实例名称，在不同的项目中，可以设置属于自己的实例，Python包之间关系还可以实现继承。

### 日志的基本配置

我们在使用日志模块主要是使用Logger,Handler,Formatter,Filter,LogRecord该五种类型。 \* Logger: 日志记录器，用来记录日志信息，不能直接使用，需要通过logging.getLogger(name),来创建实例。 \* Handler: 日志处理器，将Logger产生的数据发送到指定位置，例如文件，控制台，或者日志服务器等。 \* Formatter: 格式化器，用来格式化日志，提供统一化的日志输出格式。 \* Filter:过滤器，提供更细微力度日志信息输出，决定哪些日志内容会被输出。 \* LogRecord: logger中的实现，每次输出日志会创建，来输出日志内容。

为了方便了解整个的日志工作流程，可参考流程图进行理解，流程图如下：  1. 进入日志处理流程，如果日志输出的日志等级低于设置的日志等级，不可用，流程结束；相反如果高于或者等于设置的日志等级，流程可用往下执行。 2. 创建LogRecord,并且根据注册的Filter对象进行过滤，返回内容为True，该日志流程继续，将该日志记录交给日志器上添加的处理器进行处理；返回False则流程结束。 3. LogRecord对象将Handlers 传输给该日志对象， 4. 检查该日志对象是否还有父日志对象，有需要将日志对象设置成父日志对象，直到满足是root 日志对象为止。 5. 进入日志处理器处理流程，判断该条日志级别是否满足日志处理器的日志级别，不满足则直接终止，满足要求则进入日志处理器的过滤器中，过滤后满足条件则按照自身定义的格式器进行日志信息的格式化，最后将内容输出到指定的位置。，过滤后不满足条件则终止。**满足的条件是日志级别大于等于处理器设置的日志级别** 6. 如果第三步设置多个日志处理器，那么第五步会执行多次。

那问题来了，我们怎么使用这些内容输出想要的日志内容呢？答案就是直接使用baseConfig()方法即可解决我们日常的配置工作。 1. baseConfig()

我们使用baseConfig()方法配置基本的参数，level设置日志等级信息为Debug,并且指定日志格式化内容format，format配置内容如下表所示： | Attribyte name(参数名称) | Format | Description(参数描述) | |:—-|:—-:|:—-|:—-:|:—-| | asctime | %(asctime)s | 帮助把日志时间设置成可读的形式，默认是2019-01-01 12:12:12,324 该参数可以指定datafmt来进行格式化， datafmt= ‘%Y-%m-%d %H:%M:%S’ | | created | %(created)f | 创建日志的记录的时间 | | filename | %(filename)s | 当前文件的名字，不包含路径信息 | | funcName | %(funcName)s | 函数名字 | | levelname | %(levelname)s | 设置日志等级信息(‘DEBUG’, ‘INFO’, ‘WARNING’, ‘ERROR’, ‘CRITICAL’). | | levelno | %(levelno)s | 设置日志数字形式等级信息 (DEBUG, INFO, WARNING, ERROR, CRITICAL). | | lineno | %(lineno)d | 输出日志所在行号 | | message | %(message)s | 输出的自定义消息内容 | | module | %(module)s | 模块名称 | | name | %(name)s | 日志实例对象的名称 | | pathname | %(pathname)s | python全文件路径名称 | | process | %(process)d | 当前进程id | | processName | %(processName)s | 当前进程名称 | | relativeCreated | %(relativeCreated)d | 创建日志记录的时间 | | thread | %(thread)d | 当前线程ID | | threadName | %(threadName)s | 当前线程名称 |

当然在baseConfig中不止这两个配置参数，我们来看下还有哪些配置参数： | Attribyte name(参数名称) | Description(参数描述) | |:—-|:—-| | filename | 日志所输出的文件名称,**w是清除写入，a是追加写入，制定该参数代表使用的是fileHandler.** | | filemode | 文件模式，可读，可写 | | datefat | 设置日期时间的格式 | | level | 设置日志输出的级别 | | handles | 定义处理器，创建Handler对象，该参数不能跟下面的stream共同使用，会出现ValueError异常信息 | | stream | 定义输出流，用来将日志内容输出，可以输出到文件，控制台，日志系统等，默认是StreamHandler ，该参数是不能跟filename联合使用，会出现ValueError异常信息 |

配置案例如下：

import logging  
  
# 设置basicConfig  
logging.basicConfig(level=logging.DEBUG, filename="test\_base.log", datefmt="%Y-%m-%d %H:%M:%S",  
 format="%(asctime)s %(pathname)s:%(lineno)d %(levelname)-5s Thread - %(thread)d - %(message)s")  
  
  
logger.debug("test debug")  
logger.info("test info")  
logger.warning("test warning")  
logger.error("test error")  
logger.critical("test critical")  
  
在test\_base.log文件中输出结果如下：  
2019-08-18 23:20:03 log.py:34 DEBUG Thread - 140199123691328 - test debug  
2019-08-18 23:20:03 log.py:35 INFO Thread - 140199123691328 - test info  
2019-08-18 23:20:03 log.py:36 WARNING Thread - 140199123691328 - test warning  
2019-08-18 23:20:03 log.py:37 ERROR Thread - 140199123691328 - test error  
2019-08-18 23:20:03 log.py:38 CRITICAL Thread - 140199123691328 - test critical

1. Handler

Handler可以帮助程序进行指定输出，我们可以指定如下的Handler:

* StreamHandler:logging.StreamHandler;日志输出流，可以输出到控制台。
* FileHandler: 输出日志到文件中。
* HttpHandler:通过HTTP的方式将日志输出到HTTP服务器。
* RotaingHandler和TimeRotatingHandler:日志回滚的方式。
* SocketHandler:远程输出日志通过TCP/IP Socker的方式。
* DatagramHandler:输出日志通过UDP的方式。

当然还有更多的方式去将日志输出到我们想要输出的地方，下面介绍几个Handler的具体实现

1. FileHandler

import logging  
  
logger = logging.getLogger("study")  
# 设置日志的等级,控制日志的输出  
logger.setLevel(level=logging.DEBUG)  
  
# 设置handler  
logger\_file = "test.log"  
file\_handler = logging.FileHandler(logger\_file, mode='w')  
file\_handler.setLevel(level=logging.DEBUG)  
  
# 设置输出的格式  
formatter = logging.Formatter(  
 fmt="[%(asctime)s] [%(pathname)s:%(lineno)d] [%(levelname)-5s] [Thread - %(thread)d] - %(message)s",  
 datefmt="%Y-%m-%d %H:%M:%S",  
)  
file\_handler.setFormatter(formatter)  
logger.addHandler(file\_handler)  
  
logger.debug("test debug")  
logger.info("test info")  
logger.warning("test warning")  
logger.error("test error")  
logger.critical("test critical")  
  
在test.log文件输出结果如下：  
[2019-08-18 23:09:42] [log.py:20] [DEBUG] [Thread - 139890638694208] - test debug  
[2019-08-18 23:09:42] [log.py:21] [INFO ] [Thread - 139890638694208] - test info  
[2019-08-18 23:09:42] [log.py:22] [WARNING] [Thread - 139890638694208] - test warning  
[2019-08-18 23:09:42] [log.py:23] [ERROR] [Thread - 139890638694208] - test error  
[2019-08-18 23:09:42] [log.py:24] [CRITICAL] [Thread - 139890638694208] - test critical  
由于没有设置在控制台输出，则该内容并没有在控制台输出只在文件中输出。

1. StreamHandler

在控制台进行输出  
import logging  
import sys  
  
logger = logging.getLogger("study")  
# 设置日志的等级,控制日志的输出  
logger.setLevel(level=logging.DEBUG)  
  
# 设置handler  
logger\_file = "test.log"  
file\_handler = logging.FileHandler(logger\_file, mode='w')  
file\_handler.setLevel(level=logging.DEBUG)  
  
# 设置输出的格式  
formatter = logging.Formatter(  
 fmt="[%(asctime)s] [%(pathname)s:%(lineno)d] [%(levelname)-5s] [Thread - %(thread)d] - %(message)s",  
 datefmt="%Y-%m-%d %H:%M:%S",  
)  
file\_handler.setFormatter(formatter)  
logger.addHandler(file\_handler)  
  
# 设置控制台输出  
stream\_handler = logging.StreamHandler(sys.stdout)  
stream\_handler.setFormatter(formatter)  
stream\_handler.setLevel(logging.INFO)  
logger.addHandler(stream\_handler)  
  
执行目标文件在test.log文件中输出结果如下：  
[2019-08-18 23:15:11] [log.py:27] [DEBUG] [Thread - 139976458704704] - test debug  
[2019-08-18 23:15:11] [log.py:28] [INFO ] [Thread - 139976458704704] - test info  
[2019-08-18 23:15:11] [log.py:29] [WARNING] [Thread - 139976458704704] - test warning  
[2019-08-18 23:15:11] [log.py:30] [ERROR] [Thread - 139976458704704] - test error  
[2019-08-18 23:15:11] [log.py:31] [CRITICAL] [Thread - 139976458704704] - test critical  
  
控制台输出如下：  
[2019-08-18 23:15:11] [log.py:28] [INFO ] [Thread - 139976458704704] - test info  
[2019-08-18 23:15:11] [log.py:29] [WARNING] [Thread - 139976458704704] - test warning  
[2019-08-18 23:15:11] [log.py:30] [ERROR] [Thread - 139976458704704] - test error  
[2019-08-18 23:15:11] [log.py:31] [CRITICAL] [Thread - 139976458704704] - test critical

从上面可以发现，我们设置了多个Handler，但是并不影响他们各自的输出结果，设置不同日志等级，并且具体输出在不同的目标下。 在这样handler共存的方式下，我们可以设置多个日志输出的目标。

### 自定义日志的使用

每个系统中都会有自定义的日志形式，在大型系统中进行快速的定位疑难杂症，因此还需要自定义Logger.在Python的日志系统中，logging的配置模块提供了父子模块共享配置的机制，还记得我们在前面使用的getLogger(**name**)吗？\_\_name\_\_提供当前模块的名称,当然也可以选择不使用\_\_name\_\_方式，采用“study.original”包之间使用.作为分隔符的命名空间间隔。 比如有如下不同的包空间,可以引入不同的包空间内容，并且在输出时可以指定输出的名称，帮助输出日志信息。

在这里采用全局的方式输出日志baseConfig进行设置。  
# 日志配置文件 log.py baseconfig  
logger\_file = "test.log"  
file\_handler = logging.FileHandler(logger\_file, mode='a')  
file\_handler.setLevel(level=logging.DEBUG)  
  
# 设置输出的格式  
formatter = logging.Formatter(  
 fmt="[%(asctime)s] %(name)s [%(pathname)s:%(lineno)d] [%(levelname)-5s] [Thread - %(thread)d] - %(message)s",  
 datefmt="%Y-%m-%d %H:%M:%S",  
)  
file\_handler.setFormatter(formatter)  
  
handlers =[]  
handlers.append(file\_handler)  
stream\_handler = logging.StreamHandler(sys.stdout)  
stream\_handler.setFormatter(formatter)  
stream\_handler.setLevel(logging.INFO)  
handlers.append(stream\_handler)  
logging.basicConfig(level=logging.INFO, datefmt="%Y-%m-%d %H:%M:%S",  
 format="%(asctime)s %(name) %(pathname)s:%(lineno)d %(levelname)-5s Thread - %(thread)d - %(message)s",  
 handlers=handlers)  
# 调用输出日志的方法  
import original.test\_original as test\_original  
# 注意 一定要引入该文件，不然log文件相当于没有初始化，在调用日志的时候还会采用默认文件  
import log   
test\_original.run()  
# 需要输出日志的original/test\_original.py  
  
import logging  
# 这里使用\_\_name\_\_获取命名空间，  
logger = logging.getLogger(\_\_name\_\_)  
def run():  
 logger.error("print test\_original info logger")  
  
# 输出内容  
[2019-08-19 23:54:16] original.test\_original [/home/ruiqi/Desktop/code/code/study/original/test\_original.py:6] [ERROR] [Thread - 140152527845184] - print test\_original info logger

**original.test\_original**就是\_\_name\_\_获取的结果。

如果没有中间调用方法地方，直接在original/test\_original.py中执行。**需要在该文件中引入log文件进行日志的初始化**，在大型项目中，进行创建项目时，会在第一时刻初始化好。