logger提供了应用程序可以直接使用的接口；

handler将(logger创建的)日志记录发送到合适的目的输出；

filter提供了细度设备来决定输出哪条日志记录；

formatter决定日志记录的最终输出格式

Logger

（RootLogger继承Logger，只是初始化属性name默认“root”，加传参初始化level值，之外没有新加和重写任何方法，root是RootLogger对象，level值为WARNING，结果本质上root是一个name=“root”的Logger对象）

logging.getLogger([name]):返回一个logger对象，name有入参，且已经存在，则返回同一个logger；如果没有，则一个新的logger；如果没有指定名字将返回root。  
 logging.debug()、logging.info()、logging.warning()、logging.error()、logging.critical()：是通过root来写日志，如果root还没有加handler，则调用logging.basicConfig()添加一个。  
 logging.basicConfig():

logging.basicConfig()的入参是字典，键值有："filename"、"filemode"（不填默认'a'）、"stream"、"format"（不填默认"%(levelname)s:%(name)s:%(message)s"）、"datefmt"（不填默认None）、"level"；

通过入参配置建立的是FileHandler(filename, mode),或StreamHandler(stream)的handeler，并加到root中；

如果root已经有handler，再运行logging.basicConfig()是不会使用的；

Logger对象的setLevel(lel):指定最低的日志级别，低于lel的级别将被忽略。

NOTSET（0） < DEBUG（10） < INFO （20）< WARNING （30）< ERROR （40）< CRITICAL（50），level值可以是int、long型，也可以是str。

Logger.addFilter(filt)、Logger.removeFilter(filt):添加或删除指定的filter  
 Logger.addHandler(hdlr)、Logger.removeHandler(hdlr)：增加或删除指定的handler  
 Logger对象的debug()、info()、warning()、error()、critical()：是用Logger对象来写相应lever级别的日志。

Handlers  
handler继承图解：

logging下：

class Filterer(object)

class Handler(Filterer)，

class StreamHandler(Handler)，

class FileHandler(StreamHandler)，

logging.handlers模块： class BaseRotatingHandler(logging.FileHandler)

class RotatingFileHandler(BaseRotatingHandler):

class TimedRotatingFileHandler(BaseRotatingHandler):

class WatchedFileHandler(logging.FileHandler):

class SocketHandler(logging.Handler):

class DatagramHandler(SocketHandler):

class SysLogHandler(logging.Handler):

class SMTPHandler(logging.Handler):

class NTEventLogHandler(logging.Handler):

class HTTPHandler(logging.Handler):

class BufferingHandler(logging.Handler):

class MemoryHandler(BufferingHandler):

handler对象负责发送相关的信息到指定目的地。Python的日志系统有多种Handler可以使用。有些Handler可以把信息输出到控制台，有些Logger可以把信息输出到文件，还有些 Handler可以把信息发送到网络上。如果觉得不够用，还可以编写自己的Handler。 Logger对象可以通过addHandler()方法添加多个多handler，removeHandler()删除。  
 Handler对象的setLevel(lel):指定被处理的信息级别，低于lel级别的信息将被忽略  
 Handler对象的setFormatter()：给这个handler选择一个格式  
 Handler对象的addFilter(filt)、Handler对象的removeFilter(filt)：新增或删除一个filter对象

Formatters

Formatter对象设置日志信息最后的规则、结构和内容，入参fmt（默认格式为"%(message)s"）, datefmt（默认格式为%Y-%m-%d %H:%M:%S+",%03d"的毫秒);

下面是fmt常用的一些信息设置方式：

|  |  |
| --- | --- |
| %(name)s | Logger的名字 |
| %(levelno)s | 数字形式的日志级别 |
| %(levelname)s | 文本形式的日志级别 |
| %(pathname)s | 调用日志输出函数的模块的完整路径名，可能没有 |
| %(filename)s | 调用日志输出函数的模块的文件名 |
| %(module)s | 调用日志输出函数的模块名 |
| %(funcName)s | 调用日志输出函数的函数名 |
| %(lineno)d | 调用日志输出函数的语句所在的代码行 |
| %(created)f | 当前时间，用UNIX标准的表示时间的浮 点数表示 |
| %(relativeCreated)d | 输出日志信息时的，自Logger创建以 来的毫秒数 |
| %(asctime)s | 字符串形式的当前时间。默认格式是 “2003-07-08 16:49:45,896”。逗号后面的是毫秒 |
| %(thread)d | 线程ID。可能没有 |
| %(threadName)s | 线程名。可能没有 |
| %(process)d | 进程ID。可能没有 |
| %(message)s | 用户输出的消息 |

设置过滤器  
细心的朋友一定会发现前文调用logging.getLogger()时参数的格式类似于“A.B.C”。采取这样的格式其实就是为了可以配置过滤器。看一下这段代码：  
LOG=logging.getLogger(”chat.gui.statistic”)  
console = logging.StreamHandler()  
console.setLevel(logging.INFO)  
formatter = logging.Formatter(’%(asctime)s %(levelname)s %(message)s’)  
console.setFormatter(formatter)  
filter=logging.Filter(”chat.gui”)  
console.addFilter(filter)  
LOG.addHandler(console)  
和前面不同的是我们在Handler上添加了一个过滤器。现在我们输出日志信息的时候就会经过过滤器的处理。名为“A.B”的过滤器只让名字带有 “A.B”前缀的Logger输出信息。可以添加多个过滤器，只要有一个过滤器拒绝，日志信息就不会被输出。另外，在Logger中也可以添加过滤器。

每个Logger可以附加多个Handler。接下来我们就来介绍一些常用的Handler：  
1)    logging.StreamHandler  
使用这个Handler可以向类似与sys.stdout或者sys.stderr的任何文件对象(file object)输出信息。它的构造函数是：  
StreamHandler([strm])  
其中strm参数是一个文件对象。默认是sys.stderr  
2)   logging.FileHandler  
和StreamHandler类似，用于向一个文件输出日志信息。不过FileHandler会帮你打开这个文件。它的构造函数是：  
FileHandler(filename[,mode])  
filename是文件名，必须指定一个文件名。  
mode是文件的打开方式。参见Python内置函数open()的用法。默认是’a'，即添加到文件末尾。  
3)   logging.handlers.RotatingFileHandler  
这个Handler类似于上面的FileHandler，但是它可以管理文件大小。当文件达到一定大小之后，它会自动将当前日志文件改名，然后创建 一个新的同名日志文件继续输出。比如日志文件是chat.log。当chat.log达到指定的大小之后，RotatingFileHandler自动把 文件改名为chat.log.1。不过，如果chat.log.1已经存在，会先把chat.log.1重命名为chat.log.2。。。最后重新创建 chat.log，继续输出日志信息。它的构造函数是：  
RotatingFileHandler( filename[, mode[, maxBytes[, backupCount]]])  
其中filename和mode两个参数和FileHandler一样。  
maxBytes用于指定日志文件的最大文件大小。如果maxBytes为0，意味着日志文件可以无限大，这时上面描述的重命名过程就不会发生。  
backupCount用于指定保留的备份文件的个数。比如，如果指定为2，当上面描述的重命名过程发生时，原有的chat.log.2并不会被更名，而是被删除。  
4)   logging.handlers.TimedRotatingFileHandler  
这个Handler和RotatingFileHandler类似，不过，它没有通过判断文件大小来决定何时重新创建日志文件，而是间隔一定时间就 自动创建新的日志文件。重命名的过程与RotatingFileHandler类似，不过新的文件不是附加数字，而是当前时间。它的构造函数是：  
TimedRotatingFileHandler( filename [,when [,interval [,backupCount]]])  
其中filename参数和backupCount参数和RotatingFileHandler具有相同的意义。  
interval是时间间隔。  
when参数是一个字符串。表示时间间隔的单位，不区分大小写。它有以下取值：  
S 秒  
M 分  
H 小时  
D 天  
W 每星期（interval==0时代表星期一）  
midnight 每天凌晨  
5)   logging.handlers.SocketHandler  
6)   logging.handlers.DatagramHandler  
以上两个Handler类似，都是将日志信息发送到网络。不同的是前者使用TCP协议，后者使用UDP协议。它们的构造函数是：  
Handler(host, port)  
其中host是主机名，port是端口名  
7)  logging.handlers.SysLogHandler  
8)  logging.handlers.NTEventLogHandler  
9)  logging.handlers.SMTPHandler  
10) logging.handlers.MemoryHandler  
11) logging.handlers.HTTPHandler

**（源码还未跟踪）发现!：**

通过log1 = logging.getLogger(*name*)得到的log1的parent属性是<logging.RootLogger object at 0x0000000002B3E470>，另外通过给logging.root来addHandler也加到了log1上去了。虽然通过name可以get到不同的logger，但是root加的handler对所有的logger都加了。