Python Logging模块的等级：

| 日志等级（level） | 描述 |
| --- | --- |
| 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 |
| 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 |

日记的内容：

一条日志信息对应的是一个事件的发生，而一个事件通常需要包括以下几个内容：

* 事件发生时间
* 事件发生位置
* 事件的严重程度--日志级别
* 事件内容

除此之外还可以包括一些其他信息，例如进程ID、进程名称、线程ID、线程名称, Http请求等。日志格式(format)就是用来定义一条日志记录中包含那些字段的，且日志格式通常都是可以自定义的。

使用Logging模块可以有两种风格：

第一种是使用logging提供的模块级别的函数，直接使用rootLogger, 通过basicConfig进行简单的配置使用。

例如：

import logging

LOG\_FORMAT = "%(asctime)s %(name)s %(levelname)s %(pathname)s %(message)s "#配置输出日志格式

DATE\_FORMAT = '%Y-%m-%d  %H:%M:%S %a ' #配置输出时间的格式

logging.basicConfig(level=logging.DEBUG, # 当等级为DEBUG或以上则输出

                    format=LOG\_FORMAT,

                    datefmt = DATE\_FORMAT ,

                    filename=r"d:\test\test.log" #指定filename参数，日志写入该文件而不是控制台中

                    )

logging.debug("msg1")

logging.info("msg2")

logging.warning("msg3")

P. S logging.debug(), logging.info()等方法的定义中，除了msg和args参数外，还有一个\*\*kwargs参数。它们支持3个关键字参数: exc\_info, stack\_info, extra

第二种是完全自定义的使用logging模块。

首先logging模块有四大组件：

| 组件名称 | 对应类名 | 功能描述 |
| --- | --- | --- |
| 日志器 | Logger | 提供了应用程序可一直使用的接口 |
| 处理器 | Handler | 将logger创建的日志记录发送到合适的目的输出 |
| 过滤器 | Filter | 提供了更细粒度的控制工具来决定输出哪条日志记录，丢弃哪条日志记录 |
| 格式器 | Formatter | 决定日志记录的最终输出格式 |

组件之间的关系描述：

日志器（logger）需要通过处理器（handler）将日志信息输出到目标位置，如：文件、sys.stdout、网络等；

不同的处理器（handler）可以将日志输出到不同的位置；

日志器（logger）可以设置多个处理器（handler）将同一条日志记录输出到不同的位置；

每个处理器（handler）都可以设置自己的过滤器（filter）实现日志过滤，从而只保留感兴趣的日志；

每个处理器（handler）都可以设置自己的格式器（formatter）实现同一条日志以不同的格式输出到不同的地方。

**Logger**类：

最常用的配置方法如下：

| 方法 | 描述 |
| --- | --- |
| Logger.setLevel() | 设置日志器将会处理的日志消息的最低严重级别 |
| Logger.addHandler() 和 Logger.removeHandler() | 为该logger对象添加 和 移除一个handler对象 |
| Logger.addFilter() 和 Logger.removeFilter() | 为该logger对象添加 和 移除一个filter对象 |

logger对象配置完成后，可以使用下面的方法来创建日志记录：

| 方法 | 描述 |
| --- | --- |
| Logger.debug(), Logger.info(), Logger.warning(), Logger.error(), Logger.critical() | 创建一个与它们的方法名对应等级的日志记录 |
| Logger.exception() | 创建一个类似于Logger.error()的日志消息 |
| Logger.log () | 需要获取一个明确的日志level参数来创建一个日志记录 |

获取logger对象：

logging.getLogger()用于返回一个日志对象，logging.getLogger()方法有一个可选参数name，该参数表示将要返回的日志器的名称标识，如果不提供该参数，则其值为'root'，也就是默认使用rootLogger。若以相同的name参数值多次调用getLogger()方法，将会返回指向同一个logger对象的引用。

**关于logger的层级结构与有效等级的说明：**

logger的名称是一个以'.'分割的层级结构，每个'.'后面的logger都是'.'前面的logger的children，例如，有一个名称为 foo 的logger，其它名称分别为 foo.bar, foo.bar.baz 和 foo.bam都是 foo 的后代。

logger有一个"有效等级（effective level）"的概念。如果一个logger上没有被明确设置一个level，那么该logger就是使用它parent的level;如果它的parent也没有明确设置level则继续向上查找parent的parent的有效level，依次类推，直到找到个一个明确设置了level的祖先为止。需要说明的是，root logger总是会有一个明确的level设置（默认为 WARNING）。当决定是否去处理一个已发生的事件时，logger的有效等级将会被用来决定是否将该事件传递给该logger的handlers进行处理。

child loggers在完成对日志消息的处理后，默认会将日志消息传递给与它们的祖先loggers相关的handlers。因此，我们不必为一个应用程序中所使用的所有loggers定义和配置handlers，只需要为一个顶层的logger配置handlers，然后按照需要创建child loggers就可足够了。我们也可以通过将一个logger的propagate属性设置为False来关闭这种传递机制。

**Handler类**

Handler对象的作用是（基于日志消息的level）将消息分发到handler指定的位置（文件、网络、邮件等）。Logger对象可以通过addHandler()方法为自己添加0个或者更多个handler对象

| Handler | 描述 |
| --- | --- |
| logging.StreamHandler | 将日志消息发送到输出到Stream，如std.out, std.err或任何file-like对象。 |
| logging.FileHandler | 将日志消息发送到磁盘文件，默认情况下文件大小会无限增长 |
| logging.handlers.RotatingFileHandler | 将日志消息发送到磁盘文件，并支持日志文件按大小切割 |
| logging.hanlders.TimedRotatingFileHandler | 将日志消息发送到磁盘文件，并支持日志文件按时间切割，可以按照时间间隔，凌晨划分文件 |
| logging.handlers.HTTPHandler | 将日志消息以GET或POST的方式发送给一个HTTP服务器 |
| logging.handlers.SMTPHandler | 将日志消息发送给一个指定的email地址 |
| logging.NullHandler | 该Handler实例会忽略error messages，通常被想使用logging的library开发者使用来避免'No handlers could be found for logger XXX'信息的出现。 |

Handler的方法：

Handler.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.setFormatter()：给这个handler选择一个格式  
 Handler.addFilter(filt)、Handler.removeFilter(filt)：新增或删除一个filter对象

* 我们注意到handler和logger都可以设置setLevel. 他们的关系是，logger是日志的入口，所以他的level是首先被判定的，如果不满足level 要求，那么日志信息便会被阻隔，不会传到handlers当中，即便handlers中的level要求很低。相反，如果满足logger的level要求，日志信息传到handlers中的时候，可以通过设置handlers的setlevel还设置是否处理这些日志。

**Formater类**

Formater对象用于配置日志信息的最终顺序、结构和内容。与logging.Handler基类不同的是，应用代码可以直接实例化Formatter类