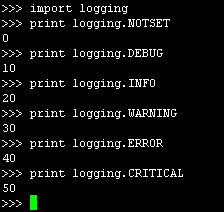
使用python的logging模块

一、从一个使用场景开始  
  
  
  
开发一个日志系统， 既要把日志输出到控制台， 还要写入日志文件  
  
  
  
Python代码

1. import logging
2. # 创建一个logger
3. logger = logging.getLogger('mylogger')
4. logger.setLevel(logging.DEBUG)
5. # 创建一个handler，用于写入日志文件
6. fh = logging.FileHandler('test.log')
7. fh.setLevel(logging.DEBUG)
8. # 再创建一个handler，用于输出到控制台
9. ch = logging.StreamHandler()
10. ch.setLevel(logging.DEBUG)
11. # 定义handler的输出格式
12. formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
13. fh.setFormatter(formatter)
14. ch.setFormatter(formatter)
15. # 给logger添加handler
16. logger.addHandler(fh)
17. logger.addHandler(ch)
18. # 记录一条日志
19. logger.info('foorbar')

*复制代码*

运行后， 在控制台和日志文件都有一条日志：  
  
  
  
Java代码    
1.2011-08-31 19:18:29,816 - mylogger - INFO - foorbar    
2011-08-31 19:18:29,816 - mylogger - INFO - foorbar   
  
  
  
二、logging模块的API  
  
  
  
结合上面的例子，我们说下几个最常使用的API  
  
logging.getLogger([name])  
返回一个logger实例，如果没有指定name，返回root logger。  
只要name相同，返回的logger实例都是同一个而且只有一个，即name和logger实例是  
一一对应的。这意味着，无需把logger实例在各个模块中传递。只要知道name，就能得到   
同一个logger实例  
  
  
Logger.setLevel(lvl)  
设置logger的level， level有以下几个级别：  
  
  
  
  
  
NOTSET < DEBUG < INFO < WARNING < ERROR < CRITICAL  
  
  
如果把looger的级别设置为INFO， 那么小于INFO级别的日志都不输出， 大于等于INFO级  
别的日志都输出  
Python代码

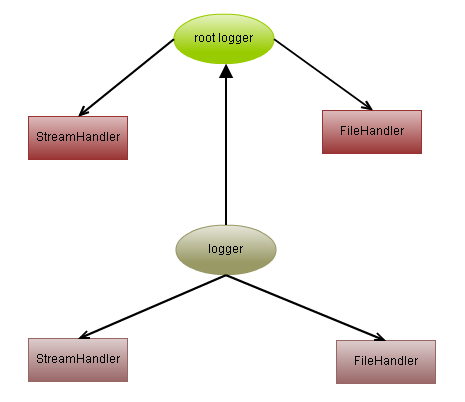
1. logger.debug("foobar")    # 不输出
2. logger.info("foobar")        # 输出
3. logger.warning("foobar") # 输出
4. logger.error("foobar")      # 输出
5. logger.critical("foobar")    # 输出

*复制代码*

Logger.addHandler(hdlr)  
logger可以雇佣handler来帮它处理日志， handler主要有以下几种：  
StreamHandler: 输出到控制台  
FileHandler:   输出到文件  
handler还可以设置自己的level以及输出格式。  
  
  
logging.basicConfig([\*\*kwargs])  
\* 这个函数用来配置root logger， 为root logger创建一个StreamHandler，   
   设置默认的格式。  
\* 这些函数： logging.debug()、logging.info()、logging.warning()、  
   logging.error()、logging.critical() 如果调用的时候发现root logger没有任何   
   handler， 会自动调用basicConfig添加一个handler  
\* 如果root logger已有handler， 这个函数不做任何事情  
  
  
使用basicConfig来配置root logger的输出格式和level：  
Python代码

1. import logging
2. logging.basicConfig(format='%(levelname)s:%(message)s', level=logging.DEBUG)
3. logging.debug('This message should appear on the console')

*复制代码*

三、关于root logger以及logger的父子关系  
  
  
前面多次提到root logger， 实际上logger实例之间还有父子关系， root logger就是处于  
最顶层的logger， 它是所有logger的祖先。如下图:   
  
root logger是默认的logger  
如果不创建logger实例， 直接调用logging.debug()、logging.info()logging.warning()、logging.error()、logging.critical()这些函数，  
那么使用的logger就是 root logger， 它可以自动创建，也是单实例的。  
  
  
如何得到root logger  
通过logging.getLogger()或者logging.getLogger("")得到root logger实例。  
  
  
默认的level  
root logger默认的level是logging.WARNING  
  
  
如何表示父子关系  
logger的name的命名方式可以表示logger之间的父子关系. 比如：  
parent\_logger = logging.getLogger('foo')  
child\_logger = logging.getLogger('foo.bar')  
  
  
什么是effective level  
logger有一个概念，叫effective level。 如果一个logger没有显示地设置level，那么它就  
用父亲的level。如果父亲也没有显示地设置level， 就用父亲的父亲的level，以此推....  
最后到达root logger，一定设置过level。默认为logging.WARNING  
child loggers得到消息后，既把消息分发给它的handler处理，也会传递给所有祖先logger处理，   
  
  
来看一个例子  
Python代码

1. import logging
2. # 设置root logger
3. r = logging.getLogger()
4. ch = logging.StreamHandler()
5. ch.setLevel(logging.DEBUG)
6. formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
7. ch.setFormatter(formatter)
8. r.addHandler(ch)
9. # 创建一个logger作为父亲
10. p = logging.getLogger('foo')
11. p.setLevel(logging.DEBUG)
12. ch = logging.StreamHandler()
13. ch.setLevel(logging.DEBUG)
14. formatter = logging.Formatter('%(asctime)s - %(message)s')
15. ch.setFormatter(formatter)
16. p.addHandler(ch)
17. # 创建一个孩子logger
18. c = logging.getLogger('foo.bar')
19. c.debug('foo')

*复制代码*

输出如下：  
  
  
Python代码

1. 2011-08-31 21:04:29,893 - foo
2. 2011-08-31 21:04:29,893 - DEBUG - foo

*复制代码*

可见， 孩子logger没有任何handler，所以对消息不做处理。但是它把消息转发给了它的父  
亲以及root logger。最后输出两条日志。