django使用python内建的logging模块去建造自己的系统日志的，如果你想详细了解这个模块的话，请自己去看python的说明文档，这里仅仅介绍django中的日志系统

日志配置包括四个部分：记录器，处理器，过滤器和格式器，下面我们来一一讲解

**记录器**

一个记录器是日志系统的一个实体，每一个记录器是一个已经命名好的可以将消息为进程写入的“桶”。

每一个记录器都会有一个日志等级，每个等级描述了记录器即将处理的信息的严重性，python定义了以下五个等级：

debug：出于调试目的的低层次系统信息

* info：普通的系统信息
* warning：描述已经发生的小问题
* error：描述已经发生的主要问题
* critical：描述已经发生的严重问题

每一条被写入记录器的信息成为一条日志记录，每条日志记录都有一个表明该记录严重性的日志等级，每条日志信息也会包含一些有用的元信息表明已经被记录的事件，比如栈追溯和错误代码。

当一条信息被发往记录器的时候，消息的记录等级会和记录器的等级相比较，如果符合甚至超越当前等级，则被发往处理器处理，否则会被忽略掉。

**处理器**

处理器是决定日志记录器中对应的实体消息发生了什么的引擎，描述了一个具体的日志行为，比如输出到屏幕，或者一个文件，抑或一个网络socket。

和记录器一样，没有到达相应等级的消息会被忽略。

一个记录器可以有多个处理器，一个处理器可以有不同的日志等级，因此你可以根据消息的重要性而提供不同的提示。

**过滤器**

过滤器是用来提供额外的控制，控制哪些日志记录可以被传给处理器处理。

默认情况下，只要日志消息符合相应的等级要求就会传给对应的处理器处理，然而，通过安装过滤器，你可以在日志记录过程中设置额外的内容，例如，你可以安装一个过滤器使得某个源只有error级别的消息才会被发送。你也可以使用过滤器修改之前会被发送的消息，例如，你可以写一个过滤器使得符合某些条件的error等级的消息降级为warning等级。

过滤器可以给处理器和记录器使用，多个过滤器可以级联使用。

**格式器**

控制日志输出的格式，格式使用python的字符串控制格式

LOGGING **= {  
    'version':**1,  
    **'disable\_existing\_loggers':**True,  
#     If the disable\_existing\_loggers key in the LOGGING dictConfig is set to True (which is the default) then  
# all loggers from the default configuration will be disabled. Disabled loggers are not the same as removed; the logger  
# will still exist, but will silently discard anything logged to it, not even propagating entries to a parent logger. Thus  
# you should be very careful using ’disable\_existing\_loggers’: True; it’s probably not what you want.  
# Instead, you can set disable\_existing\_loggers to False and redefine some or all of the default loggers; or  
# you can set LOGGING\_CONFIG to None and handle logging config yourself.  
    **'formatters': {  
        'standard': {  
            'format': '%(asctime)s [%(name)s:%(lineno)d] [%(module)s:%(funcName)s] [%(levelname)s]- %(message)s'}  
    }**,  
    **'handlers': {  
        'file': {  
            'level': 'DEBUG'**,  
            **'class': 'logging.FileHandler'**,  
            **'filename': 'log/debug.log'**,  
            **'formatter': 'standard'  
        }**,  
        **'console': {  
            'level': 'DEBUG'**,  
            **'class': 'logging.StreamHandler'**,  
            **'formatter': 'standard'  
        }**,  
    **}**,  
    **'loggers': {  
        'stu': {  
            'handlers': ['file']**, # 可以设置多个handlers  
            **'level': 'DEBUG'**,  
            **'propagate':**False,  
        **}**,  
        **'stu.views': {  
            'handlers': ['console']**,  
            **'level': 'ERROR'**,  
            # propagate 如果设置为 True 默认会向上一个级别找相应logger来继续处理，例如这里会找一个stu的logger来处理  
            # （一般我们设置为False不向上找）  
            **'propagate':**False,  
        **}**,  
    **}**,  
**}**

**import**logging  
error\_logger **=**logging.getLogger**(**\_\_name\_\_**)  
try:** open**('r3'**,**'r')  
except**Exception **as**e**:**

error\_logger.error**(**str**(**e**))**

**备注：**

**1.如果消息被第一个接收到的logger过滤掉了则不会继续传递父logger了**

**2.** logging.getLogger**(**\_\_name\_\_**) 这样写会直接找到模块名字，可以定义logger的时候直接用模块名字，从而实现每个模块都有自己的logger，个人感觉要写一个通用的logger（命名common）就行了**