20211008-Python logging模块调研

一、什么是日志?

日志是一种可以追踪某些软件运行时所发生事件的方法。软件开发人员可以向他们的代码中调用日志记录相关的方法来表明发生了某些事情。一个事件可以用一个可包含可选变量数据的消息来描述。此外，事件也有重要性的概念，这个重要性也可以被称为严重性级别（level）。

1. 日志的作用

我们通过记录和分析日志可以了解一个系统或软件程序运行情况是否正常，也可以在应用程序出现故障时快速定位问题。日志的作用可以简单总结为以下3点：

- 程序调试
- 了解软件程序运行情况，是否正常
- 软件程序运行故障分析与问题定位

2. 日志的等级

两个重要问题：

- 作为开发人员，在开发一个应用程序时需要什么日志信息？在应用程序正式上线后需要什么日志信息？
- 作为应用运维人员，在部署开发环境时需要什么日志信息？在部署生产环境时需要什么日志信息？

在软件开发阶段或部署开发环境时，为了尽可能详细的查看应用程序的运行状态来保证上线后的稳定性，我们可能需要把该应用程序所有的运行日志全部记录下来进行分析，这是非常耗费机器性能的。当应用程序正式发布或在生产环境部署应用程序时，我们通常只需要记录应用程序的异常信息、错误信息等，这样既可以减小服务器的I/O压力，也可以避免我们在排查故障时被淹没在日志的海洋里。所以日志天然是分级别的，这就是日志等级的作用。日志详细点分为以下几个等级：

- DEBUG
- INFO
- NOTICE
- WARNING
- ERROR
- CRITICAL
- ALERT
- EMERGENCY

3. 日志的格式应该是怎么样的

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（即日志级别）
- 事件内容

当然还可以包括一些其他信息，如进程ID、进程名称、线程ID、线程名称等。日志格式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。输出一条日志时，日志内容和日志级别是需要开发人员明确指定的。对于而其它字段信息，只需要是否显示在日志中就可以了。

4. 日志功能的实现

几乎所有开发语言都会内置日志相关功能，或者会有比较优秀的第三方库来提供日志操作功能，比如：log4j，log4php等。它们功能强大、使用简单。Python自身也提供了一个用于记录日志的标准库模块--logging。

二、logging模块简介

logging模块是Python的一个标准库模块，由标准库模块提供日志记录API的关键好处是所有Python模块都可以使用这个日志记录功能。所以，你的应用日志可以将你自己的日志信息与来自第三方模块的信息整合起来。

1. logging模块的日志级别

logging模块默认定义了以下几个日志级别（也可以定义其他级别，但是最好不要这么做，这样会导致别人使用你的库时候日志混乱）：

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

开发应用程序或部署开发环境时，可以使用DEBUG或INFO级别的日志获取尽可能详细的日志信息来进行开发或部署调试；应用上线或部署生产环境时，应该使用WARNING或ERROR或CRITICAL级别的日志来降低机器的I/O压力和提高获取错误日志信息的效率。日志级别的指定通常都是在应用程序的配置文件中进行指定的：

- 上面列表中的日志等级是从上到下依次升高的，即：DEBUG < INFO < WARNING < ERROR < CRITICAL，而日志的信息量是依次减少的；
- 当为某个应用程序指定一个日志级别后，应用程序会记录所有日志级别大于或等于指定日志级别的日志信息，而不是仅仅记录指定级别的日志信息。

2. logging模块使用方式介绍

logging模块提供两种记录日志的方式：

- 第一种方式是使用logging提供的模块级别的函数。
- 第二种方式是使用Logging日志系统的四大组件。

其中，logging模块的模块级别常用函数：

函数	说明
logging.debug(msg, *args, **kwargs)	创建一条严重级别为DEBUG的日志记录
logging.info(msg, *args, **kwargs)	创建一条严重级别为INFO的日志记录
logging.warning(msg, *args, **kwargs)	创建一条严重级别为WARNING的日志记录
logging.error(msg, *args, **kwargs)	创建一条严重级别为ERROR的日志记录
logging.critical(msg, *args, **kwargs)	创建一条严重级别为CRITICAL的日志记录
logging.log(level, *args, **kwargs)	创建一条严重级别为level的日志记录
logging.basicConfig(**kwargs)	对root logger进行一次性配置

其中 `logging.basicConfig(**kwargs)` 函数主要用于指定“要记录的日志级别”、“日志格式”、“日志输出位置”、“日志文件打开模式”等信息，其他几个都用于记录各个级别日志的函数。

其中，logging模块四大组件：

组件	说明
loggers	提供应用程序代码直接使用的接口
handlers	用于将日志记录发送到指定的目的位置
filters	提供更细粒度的日志过滤功能，用于决定哪些日志记录将会被输出（其它的日志记录将会被忽略）
formatters	用于控制日志信息的最终输出格式

logging模块提供的模块级别的那些函数实际上也是通过这几个组件的相关实现类来记录日志的，只是在创建这些类的实例时设置了一些默认值。

三、使用logging提供的模块级别的函数记录日志

上面提到：

- 可以通过logging模块定义的模块级别的方法去完成简单的日志记录
- 只有级别大于或等于日志记录器指定级别的日志记录才会被输出，小于该级别的日志记录将会被丢弃。

1. 最简单的日志输出

输出不同级别的日志记录：

```
import logging

logging.debug("This is a debug log.")
logging.info("This is a info log.")
logging.warning("This is a warning log.")
logging.error("This is a error log.")
logging.critical("This is a critical log.")
```

也可以这么写：

```
import logging

logging.log(logging.DEBUG, "This is a debug log.")
logging.log(logging.INFO, "This is a info log.")
logging.log(logging.WARNING, "This is a warning log.")
logging.log(logging.ERROR, "This is a error log.")
logging.log(logging.CRITICAL, "This is a critical log.")
```

输出：

```
WARNING:root:This is a warning log.
ERROR:root:This is a error log.
CRITICAL:root:This is a critical log.
```

问题1：为什么前两条日志没有被打印出来？

因为logging模块提供的日志记录函数所使用的日志器设置的日志级别为 `WARNING`，因此只有 `WARNING` 级别以及大于 `WARNING`级别的日志记录被输出，小于的被丢弃掉了。

问题2：打印出来的日志信息中各字段表示什么意思？为什么会这样输出？

日志输出的格式为：

```
日志级别：日志器名称：日志内容
```

之所以会这样输出，是因为logging模块提供的日志记录函数所使用的日志器设置的日志格式默认是 `BASIC_FORMAT`，其值为：

```
"%(levelname)s:%(name)s:%(message)s"
```

问题3：为什么日志记录不输出到文件中，而是打印到控制台？

因为在logging模块提供的日志记录函数所使用的日志器设置的处理器所指定的日志输出位置默认为：`sys.stderr`

问题4：我是怎么知道这些的？

查看日志记录函数的实现代码，就可以发现：没有任何配置信息的时候，函数会默认调用 `logging.basicConfig(**kwargs)` 方法，且不会传任何参数。

问题5：怎么修改这些默认设置呢？

修改 `logging.basicConfig(**kwargs)` 传入的参数即可。

2. logging.basicConfig()函数说明

该方法用于为logging日志系统做一些基本配置，方法定义如下：

```
logging.basicConfig(**kwargs)
```

该函数可接受的关键字如下：

参数名称	描述
filename	指定日志输出目标文件的文件名，指定该设置项后日志信心就不会被输出到控制台了
filemode	指定日志文件的打开模式，默认为'a'。需要注意的是，该选项要在filename指定时才有效
format	指定日志格式字符串，即指定日志输出时所包含的字段信息以及它们的顺序。logging模块定义的格式字段下面会列出。
datefmt	指定日期/时间格式。需要注意的是，该选项要在format中包含时间字段%(asctime)s时才有效
level	指定日志器的日志级别
stream	指定日志输出目标stream，如sys.stdout、sys.stderr以及网络stream。需要说明的是，stream和filename不能同时提供，否则会引发ValueError异常
style	Python 3.2中新添加的配置项。指定format格式字符串的风格，可取值为'%'、'{'和'$'，默认为'%'
handlers	Python 3.3中新添加的配置项。该选项如果被指定，它应该是一个创建了多个Handler的可迭代对象，这些handler将会被添加到root logger。需要说明的是：filename、stream和handlers这三个配置项只能有一个存在，不能同时出现2个或3个，否则会引发ValueError异常。

3. logging模块定义的格式字符串

logging模块中定义好的可以用于format格式字符串中字段有以下这些：

字段/属性名称	使用格式	描述
asctime	%(asctime)s	日志事件发生的时间--人类可读时间，如：2003-07-08 16:49:45,896
created	%(created)f	日志事件发生的时间--时间戳，就是当时调用time.time()函数返回的值
relativeCreated	%(relativeCreated)d	日志事件发生的时间相对于logging模块加载时间的相对毫秒数
msecs	%(msecs)d	日志事件发生事件的毫秒部分
levelname	%(levelname)s	该日志记录的文字形式的日志级别（'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'）

字段/属性名称	使用格式	描述
levelno	%(levelno)s	该日志记录的数字形式的日志级别（10, 20, 30, 40, 50）
name	%(name)s	所使用的日志器名称，默认是'root'，因为默认使用的是 rootLogger
message	%(message)s	日志记录的文本内容，通过 msg % args计算得到的
pathname	%(pathname)s	调用日志记录函数的源码文件的全路径
filename	%(filename)s	pathname的文件名部分，包含文件后缀
module	%(module)s	filename的名称部分，不包含后缀
lineno	%(lineno)d	调用日志记录函数的源代码所在的行号
funcName	%(funcName)s	调用日志记录函数的函数名
process	%(process)d	进程ID
processName	%(processName)s	进程名称，Python 3.1新增
thread	%(thread)d	线程ID
threadName	%(thread)s	线程名称

4. 关于logging.basicConfig()的一些说明

`logging.basicConfig()` 有以下的一系列特性：

- `logging.basicConfig()` 函数是一个一次性的简单配置工具使，也就是说只有在第一次调用该函数时会起作用，后续再次调用该函数时完全不会产生任何操作的，多次调用的设置并不是累加操作。
- 日志器（Logger）是有层级关系的，上面调用的 `logging` 模块级别的函数所使用的日志器是RootLogger类的实例，其名称为'root'，它是处于日志器层级关系最顶层的日志器，且该实例是以单例模式存在的。
- 如果要记录的日志中包含变量数据，可使用一个格式字符串作为这个事件的描述消息（logging.debug、logging.info等函数的第一个参数），然后将变量数据作为第二个参数*args的值进行传递如：`logging.warning('%s is %d years old.', 'Tom', 10)`
- 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`，这三个关键参数含义为：
  - **exc_info**：其值为布尔值，如果该参数的值设置为True，则会将异常异常信息添加到日志消息中。如果没有异常信息则添加None到日志信息中。
  - **stack_info**：其值也为布尔值，默认值为False。如果该参数的值设置为True，栈信息将会被添加到日志信息中。
  - **extra**：这是一个字典（dict）参数，它可以用来自定义消息格式中所包含的字段，但是它的key不能与logging模块定义的字段冲突。

举例说明：在日志消息中添加exc_info和stack_info信息，并添加两个自定义的字端 ip和user：

```
import logging

LOG_FORMAT = "%(asctime)s - %(levelname)s - %(user)s[%(ip)s] - %(message)s"
DATE_FORMAT = "%m/%d/%Y %H:%M:%S %p"
```

```
logging.basicConfig(format=LOG_FORMAT, datefmt=DATE_FORMAT)
logging.warning("Some one delete the log file.", exc_info=True, stack_info=True,
extra={'user': 'Tom', 'ip':'47.98.53.222'})
```

输出日志为：

```
05/08/2017 16:35:00 PM - WARNING - Tom[47.98.53.222] - Some one delete the log
file.
NoneType
Stack (most recent call last):
  File "C:/Users/wader/PycharmProjects/LearnPython/day06/log.py", line 45, in
<module>
    logging.warning("Some one delete the log file.", exc_info=True,
stack_info=True, extra={'user': 'Tom', 'ip':'47.98.53.222'})
```

四、logging模块日志流处理流程

1. logging日志模块的四大组件

logging模块四大组件：

组件	说明
loggers	提供应用程序代码直接使用的接口
handlers	用于将日志记录发送到指定的目的位置
filters	提供更细粒度的日志过滤功能，用于决定哪些日志记录将会被输出（其它的日志记录将会被忽略）
formatters	用于控制日志信息的最终输出格式

logging模块就是通过这些组件来完成日志处理的，上面所使用的logging模块级别的函数也是通过这些组件对应的类来实现的。组件之间的关系：

- 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。

简单点说就是：日志器（logger）是入口，真正干活儿的是处理器（handler），处理器（handler）还可以通过过滤器（filter）和格式器（formatter）对要输出的日志内容做过滤和格式化等处理操作。

2. logging日志模块相关类及其常用方法介绍

2.1 Logger类

Logger对象有3个任务要做：

- （1）向应用程序代码暴露几个方法，使应用程序可以在运行时记录日志消息；
- （2）基于日志严重等级（默认的过滤设施）或filter对象来决定要对哪些日志进行后续处理；
- （3）将日志消息传送给所有感兴趣的日志handlers。

Logger对象最常用的方法分为两类：**配置方法**和**消息发送方法**，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

内建等级中，级别最低的是DEBUG，级别最高的是CRITICAL。例如setLevel(logging.INFO)，此时函数参数为INFO，那么该logger将只会处理INFO、WARNING、ERROR和CRITICAL级别的日志，而DEBUG级别的消息将会被忽略/丢弃。

logger对象配置完成后，可以采用以下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

注意：

- Logger.exception()与Logger.error()的区别在于：Logger.exception()将会输出堆栈追踪信息，另外通常只是在一个exception handler中调用该方法。
- Logger.log()与Logger.debug()、Logger.info()等方法相比，虽然需要多传一个level参数，显得不是那么方便，但是当需要记录自定义level的日志时还是需要该方法来完成。

如何得到一个Logger对象？

- 通过Logger类的实例化方法创建一个Logger类的实例
- 更常用第二种：logging.getLogger()方法。

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'。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。

对于logger的层级结构和有效等级：

- 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。

2.2 Handler类

References

[1] https://www.cnblogs.com/chengd/articles/7225626.html

[2] https://fangpenlin.com/posts/2012/08/26/good-logging-practice-in-python/

[3] https://pythonguidecn.readthedocs.io/zh/latest/writing/logging.html

[4] https://www.jianshu.com/p/feb86c06c4f4

[5] https://www.cnblogs.com/yyds/p/6901864.html

[6] https://www.cnblogs.com/xybaby/p/7954610.html