logging(日志)

日志记录是程序员工具箱中非常有用的工具。它可以帮助您更好地理解程序的流程，并发现您在开发过程中可能没有想到的场景。

日志为开发人员提供了额外的一组眼睛，这些眼睛不断关注应用程序正在经历的流程。它们可以存储信息，例如访问应用程序的用户或IP。如果发生错误，那么通过告诉您程序在到达发生错误的代码行之前的状态，它们可以提供比堆栈跟踪更多的见解。

通过从正确的位置记录有用的数据，您不仅可以轻松地调试错误，还可以使用数据分析应用程序的性能以规划扩展或查看使用模式以规划营销。

Python提供了一个日志系统作为其标准库的一部分，因此您可以快速将日志记录添加到您的应用程序中。在本文中，您将了解为什么使用此模块是向应用程序添加日志记录以及如何快速入门的最佳方式，您将了解一些可用的高级功能。

Logging模块

Python中的日志记录模块是一个随时可用且功能强大的模块，旨在满足初学者和企业团队的需求。它被大多数第三方Python库使用，因此您可以将日志消息与这些库中的日志消息集成，以为您的应用程序生成同类日志。

将记录添加到Python程序就像这样简单：

```
1 import logging
```

导入日志记录模块后，您可以使用称为“logger”的内容来记录您要查看的消息。默认情况下，有5个标准级别指示事件的严重性。每个都有一个相应的方法，可用于记录该严重级别的事件。按严重程度增加的顺序定义的级别如下：

- DEBUG
- 信息 info
- 警告 warning
- 错误 error
- 危急 critical

日志记录模块为您提供了一个默认记录器，使您无需进行太多配置即可开始使用。可以调用每个级别的相应方法，如以下示例所示：

```
1 import logging
2
3 logging.debug('这里是调试信息')
4 logging.info('这里是详情信息')
5 logging.warning('这里是警告信息')
6 logging.error('这里是错误信息')
7 logging.critical('这里是危机信息')
```

上述程序的输出如下所示：

```
1 WARNING:root:这里是警告信息
2 ERROR:root:这里是错误信息
3 CRITICAL:root:这里是危机信息
```

输出显示每条消息之前的严重性级别`root`，即日志记录模块为其默认记录器提供的名称。（记录器将在后面的章节中详细讨论。）此格式显示由冒号（`:`）分隔的级别，名称和消息，是默认的输出格式，可以配置为包括时间戳，行号和其他内容细节。

请注意，并没有记录`debug()`和`info()`消息。这是因为，默认情况下，日志记录模块会记录严重性级别为`WARNING`或更高的邮件。您可以通过将日志记录模块配置为根据需要记录所有级别的事件来更改它。您还可以通过更改配置来定义自己的严重性级别，但通常不建议这样做，因为它可能会导致您可能正在使用的某些第三方库的日志混淆。

基本配置

您可以使用该方法配置日志记录：`basicConfig(**kwargs)`

一些常用的参数`basicConfig()`如下：

- `level`：根记录器将设置为指定的严重性级别。
- `filename`：这指定文件。
- `filemode`：如果`filename`给定，则以此模式打开文件。默认值为`a`，表示追加。
- `format`：这是日志消息的格式。

通过使用该`level`参数，您可以设置要记录的日志消息级别。这可以通过传递类中可用的一个常量来完成，这将允许记录该级别或更高级别的所有日志记录调用。这是一个例子：

```
import logging

# 1.1 设置默认日志等级
logging.basicConfig(level=logging.DEBUG)
logging.debug('hello logger')
```

输出内容:

```
DEBUG:root:hello logger
```

`DEBUG`现在将记录`DEBUG`级别或更高级别的事件。

同样，对于记录到文件而不是控制台，`filename`并且`filemode`可以使用，您可以使用确定消息的格式`format`。以下示例显示了所有三个的用法：

```
import logging

# 1.2 配置写入到文件
logging.basicConfig(filename='app.log',
                    filemode='w',
                    format='%(name)s - %(levelname)s - %(message)s',
                    level=logging.DEBUG)
logging.warning('这条信息将被记录到文件')
```

输出内容:

```
root - WARNING - 这条信息将被记录到文件
```

该消息将如下所示，但将写入名为`app.log`而不是控制台的文件。filemode设置为`w`，这意味着每次`basicConfig()`调用日志文件都以“写入模式”打开，程序的每次运行都将重写该文件。filemode的默认配置`a`是append。

您可以使用更多参数进一步自定义根记录器`basicConfig()`，可在此处找到。

应该注意，`basicConfig()`只有在之前没有配置根记录器的情况下，调用配置根记录器才有效。**基本上，这个函数只能调用一次。**

`debug()`，`info()`，`warning()`，`error()`，和`critical()`也称`basicConfig()`自动无参数，如果它以前没有叫。这意味着在第一次调用上述函数之一后，您无法再配置根记录器，因为它们会在`basicConfig()`内部调用该函数。

默认设置`basicConfig()`是将记录器设置为以下列格式写入控制台：

```
ERROR:root:This is an error message
```

格式化输出

虽然您可以将任何可以表示为字符串的变量作为消息传递给您的日志，但是有一些基本元素已经成为其中的一部分，`LogRecord`并且可以轻松添加到输出格式中。如果要将进程ID与级别和消息一起记录，可以执行以下操作：

```
import logging

logging.basicConfig(format='%(process)d-%(levelname)s-%(message)s')
logging.warning('This is a Warning')
```

```
18472-WARNING-This is a Warning
```

`format`可以使用`LogRecord`您喜欢的任何排列中的属性字符串。可以在此处找到可用属性的完整列表。

这是另一个可以添加日期和时间信息的示例：

```
import logging

logging.basicConfig(format='%(asctime)s - %(message)s', level=logging.INFO)
logging.info('Admin logged in')
```

```
2018-07-11 20:12:06,288 - Admin logged in
```

`%(asctime)s`增加了创造时间`LogRecord`。可以使用`datefmt`属性更改格式，该属性使用与datetime模块中的格式化函数相同的格式化语言，例如`time.strftime()`：

```
import logging

logging.basicConfig(format='%(asctime)s - %(message)s', datefmt='%d-%b-%y
%H:%M:%S')
logging.warning('Admin logged out')
```

```
12-Jul-18 20:53:19 - Admin logged out
```

你可以在这里找到指南。

记录变量数据

在大多数情况下，您可能希望在日志中包含应用程序中的动态信息。您已经看到日志记录方法将字符串作为参数，并且在单独的行中使用可变数据格式化字符串并将其传递给log方法似乎很自然。但实际上，这可以通过使用消息的格式字符串并将变量数据作为参数附加来直接完成。这是一个例子：

```
import logging

name = 'John'

logging.error('%s raised an error', name)
```

类和函数

到目前为止，我们已经看到了一个名为默认记录器`root`，用于通过日志模块，只要其功能被直接称为是这样的：`logging.debug()`。您可以（并且应该）通过创建`Logger`类的对象来定义自己的记录器，尤其是在应用程序具有多个模块的情况下。我们来看看模块中的一些类和函数。

日志记录模块中定义的最常用类如下：

- Logger：这是一个类，其对象将直接在应用程序代码中用于调用函数。
- LogRecord：记录器自动创建`LogRecord`具有与记录事件相关的所有信息的对象，例如记录器的名称，功能，行号，消息等。
- Handler：处理程序将`LogRecord`控制器或文件发送到所需的输出目标。`Handler`对于像的子类的碱`StreamHandler`，`FileHandler`，`SMTPHandler`，`HTTPHandler`，等等。这些子类将日志记录输出发送到相应的目标，如`sys.stdout`磁盘文件。
- Formatter：您可以通过指定列出输出应包含的属性的字符串格式来指定输出的格式。

其中，我们主要处理`Logger`类的对象，这些对象使用模块级函数进行实例化`logging.getLogger(name)`。`getLogger()`使用相同的多次调用`name`将返回对同一`Logger`对象的引用，这使我们无法将记录器对象传递到需要它的每个部分。这是一个例子：

```
import logging

logger = logging.getLogger('example_logger')
logger.warning('This is a warning')
```

输出内容:

```
This is a warning
```

同样，与根记录器不同，无法使用自定义记录器进行配置`basicConfig()`。您必须使用处理程序和格式化程序对其进行配置：

使用 Handler

当您想要配置自己的记录器并在生成日志时将日志发送到多个位置时，处理程序就会出现。处理程序将日志消息发送到已配置的目标（如标准输出流或文件或HTTP）或通过SMTP发送到您的电子邮件。

您创建的记录器可以有多个处理程序，这意味着您可以将其设置为保存到日志文件并通过电子邮件发送。

与记录器一样，您也可以在处理程序中设置严重性级别。如果要为同一记录器设置多个处理程序但希望每个处理程序具有不同的严重性级别，这将非常有用。例如，您可能希望将具有级别`WARNING`及更高级别的日志记录到控制台，但是具有级别`ERROR`及更高级别的所有内容也应保存到文件中。这是一个执行该操作的程序：

```python
# logging_example.py

import logging

# Create a custom logger
logger = logging.getLogger(__name__)

# Create handlers
c_handler = logging.StreamHandler()
f_handler = logging.FileHandler('file.log')
c_handler.setLevel(logging.WARNING)
f_handler.setLevel(logging.ERROR)

# Create formatters and add it to handlers
c_format = logging.Formatter('%(name)s - %(levelname)s - %(message)s')
f_format = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
c_handler.setFormatter(c_format)
f_handler.setFormatter(f_format)

# Add handlers to the logger
logger.addHandler(c_handler)
logger.addHandler(f_handler)

logger.warning('This is a warning')
logger.error('This is an error')

```

输出内容:

```
__main__ - WARNING - This is a warning
__main__ - ERROR - This is an error
```

在这里，`logger.warning()` 创建一个 `LogRecord` 包含事件的所有信息并将其传递给它拥有的所有处理程序：`c_handler` 和 `f_handler`。

`c_handler` 是一个 `StreamHandler` 带有级别 `WARNING` 并从中获取信息 `LogRecord` 以生成指定格式的输出并将其打印到控制台。`f_handler` 是一个 `FileHandler` 有级别的 `ERROR`，它忽略 `LogRecord` 了它的级别 `WARNING`。

当 `logger.error()` 调用时，`c_handler` 行为与以前完全相同，并 `f_handler` 获得a `LogRecord` 级别 `ERROR`，因此它继续生成输出 `c_handler`，但不是将其打印到控制台，而是以这种格式将其写入指定的文件：

输出内容:

```
2018-08-03 16:12:21,723 - __main__ - ERROR - This is an error
```

将与 `__name__` 变量对应的记录器的名称记录为 `__main__`，这是 Python 分配给执行开始的模块的名称。如果此文件由某个其他模块导入，则该 `__name__` 变量将对应于其名称 `logging_example` 。这是它的样子：

```
# run.py

import logging_example
```

输出内容:

```
logging_example - WARNING - This is a warning
logging_example - ERROR - This is an error
```

保持冷静并阅读日志

记录模块被认为非常灵活。它的设计非常实用，应该适合您的开箱即用。您可以将基本日志记录添加到一个小项目中，或者如果您正在处理一个大项目，则可以创建自己的自定义日志级别，处理程序类等。

如果您还没有在应用程序中使用日志记录，那么现在是开始的好时机。完成后，日志记录肯定会消除开发过程中的大量摩擦，并帮助您找到将应用程序提升到新的水平的机会。