Daiquiri

Mar 09, 2020

Contents

The daiquiri library provides an easy way to configure logging. It also provides some custom formatters and handlers.

- Free software: Apache license
- Source: https://github.com/jd/daiquiri

CHAPTER 1

Installation

pip install daiquiri

If you want to enable systemd support, you must install the *systemd* flavor:

```
pip install daiquiri[systemd]
```

CHAPTER 2

Usage

The basic usage of daiquiri is to call the *daiquiri.setup* function that will setup logging with the options passed as keyword arguments. If no arguments are passed, the default will log to *stderr*. If *stderr* is a terminal, the output will use colors.

```
import logging

import daiquiri

daiquiri.setup(level=logging.INFO)

logger = daiquiri.getLogger(__name__)
logger.info("It works and log to stderr by default with color!")
```

You can specify different outputs with different formatters. The *daiquiri.output* module provides a collection of *Output* classes that you can use to your liking to configure the logging output. Any number of output can be configured.

```
import logging
import sys

import daiquiri

# Log both to stdout and as JSON in a file called /dev/null. (Requires
# `python-json-logger`)
daiquiri.setup(level=logging.INFO, outputs=(
    daiquiri.output.Stream(sys.stdout),
    daiquiri.output.File("/dev/null",
                         formatter=daiquiri.formatter.JSON_FORMATTER),
    ))

logger = daiquiri.getLogger(__name__, subsystem="example")
logger.info("It works and log to stdout and /dev/null with JSON")
```

If the default output configurations suit your needs, then for convenience you may pass the name of an output as a string rather than needing to import the class and produce an instance.

```
import logging

import daiquiri

daiquiri.setup(level=logging.INFO, outputs=('stdout', 'stderr'))

logger = daiquiri.getLogger(__name__)
logger.info("It works and logs to both stdout and stderr!")
```

At the moment the names *'stderr'*, *'stdout'*, *'syslog'*, and *'journal'* are available, assuming the underlying handler is available.

2.1 Picking format

You can configure the format of any output by passing a formatter as the *formatter* argument to the contructor. Two default formatters are available: *daiquiri.formatter.TEXT_FORMATTER* which prints log messages as text, and the *daiquiri.formatter.JSON_FORMATTER* which prints log messages as parsable JSON (requires *python-json-logger*).

You can provide any class of type *logging.Formatter* as a formatter.

```
import logging

import daiquiri
import daiquiri.formatter

daiquiri.setup(level=logging.INFO, outputs=(
    daiquiri.output.Stream(formatter=daiquiri.formatter.ColorFormatter(
        fmt="%(asctime)s [PID %(process)d] [%(levelname)s] "
        "%(name)s -> %(message)s")),
    ))

logger = daiquiri.getLogger(__name__)
logger.info("It works with a custom format!")
```

2.2 Python warning support

The Python *warnings* module is sometimes used by applications and libraries to emit warnings. By default, they are printed on *stderr*. Daiquiri overrides this by default and log warnings to the *py.warnings* logger.

This can be disabled by passing the *capture_warnings=False* argument to *daiquiri.setup*.

2.3 Extra usage

While it's not mandatory to use *daiquiri.getLogger* to get a logger instead of *logging.getLogger*, it is recommended as daiquiri provides an enhanced version of the logger object. It allows any keyword argument to be passed to the logging method and that will be available as part of the record.

```
import logging

import daiquiri
```

(continues on next page)

(continued from previous page)

```
import daiquiri.formatter

daiquiri.setup(level=logging.INFO, outputs=(
    daiquiri.output.Stream(formatter=daiquiri.formatter.ColorFormatter(
        fmt=(daiquiri.formatter.DEFAULT_FORMAT +
             " [%(subsystem)s is %(mood)s]"))),
    ))

logger = daiquiri.getLogger(__name__, subsystem="example")
logger.info("It works and log to stderr by default with color!",
            mood="happy")
```

2.4 Advanced Extra usage

The enhanced logger object provided by *daiquiri.getLogger* is also capable of supporting keyword arguments to the logging method without the logger itself having been configured to expect those specific keywords. This requires the use of the ExtrasFormatter or the ColorExtrasFormatter classes. The documentation for the ExtrasFormatter specifies the various options you can configure on it.

```
import logging

import daiquiri
import daiquiri.formatter

daiquiri.setup(level=logging.INFO, outputs=[
    daiquiri.output.Stream(formatter=daiquiri.formatter.ColorExtrasFormatter(
        fmt=(daiquiri.formatter.DEFAULT_FORMAT +
             " [%(subsystem)s is %(mood)s]" +
             "%(extras)s"),
        keywords=['mood', 'subsystem'],
    ))])

logger = daiquiri.getLogger(__name__, subsystem="example")
logger.info("It works and log to stderr by default with color!",
            mood="happy",
            arbitrary_context="included"
            )
```

2.5 Syslog support

The *daiquiri.output.Syslog* output provides syslog output.

2.6 Systemd journal support

The *daiquiri.output.Journal* output provides systemd journal support. All the extra arguments passed to the logger will be shipped as extra keys to the journal.

2.7 File support

The *daiquiri.output.File* output class provides support to log into a file.

daiquiri.output.RotatingFile class logs to a file that rotates when a maximum file size has been reached.

daiquiri.output.TimedRotatingFile will rotate the log file on a fixed interval.

```
import datetime
import logging

import daiquiri

daiquiri.setup(
    level=logging.DEBUG,
    outputs=(
        daiquiri.output.File('errors.log', level=logging.ERROR),
        daiquiri.output.TimedRotatingFile(
            'everything.log',
            level=logging.DEBUG,
            interval=datetime.timedelta(hours=1))
    )
)

logger = daiquiri.getLogger(__name__)

logger.info('only to rotating file logger')
logger.error('both log files, including errors only')
```

2.8 Excepthook Integration

The *daiquiri.setup* method accepts an optional *set_excepthook* keyword argument (defaults to *True*) which controls whether or not Daiquiri will override the global *sys.excepthook*. Disabling this can be useful when using Daiquiri alongside another library which requires setting the excepthook, e.g. an error reporting library.

```
import daiquiri

daiquiri.setup(set_excepthook=False)
logger = daiquiri.getLogger(__name__)

# This exception will not pass through Daiquiri:
raise Exception("Something went wrong")
```

CHAPTER 3

API

class daiquiri.**KeywordArgumentAdapter**(*logger*, *extra*)

Logger adapter to add keyword arguments to log record's extra data

Keywords passed to the log call are added to the "extra" dictionary passed to the underlying logger so they are emitted with the log message and available to the format string.

Special keywords:

extra An existing dictionary of extra values to be passed to the logger. If present, the dictionary is copied and extended.

process(*msg*, *kwargs*)

Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword args or both. Return the message and kwargs modified (or not) to suit your needs.

Normally, you'll only need to override this one method in a LoggerAdapter subclass for your specific needs.

setLevel(*level*)

Set the specified level on the underlying logger.

daiquiri.**getLogger**(*name=None*, ***kwargs*)

Build a logger with the given name.

Parameters **name** (*string*) – The name for the logger. This is usually the module name, `__name__`.

daiquiri.**parse_and_set_default_log_levels**(*default_log_levels*, *separator='='*)

Set default log levels for some loggers.

Parameters **default_log_levels** – List of strings with format

<logger_name><separator><log_level>

daiquiri.**set_default_log_levels**(*loggers_and_log_levels*)

Set default log levels for some loggers.

Parameters **loggers_and_log_levels** – List of tuple (logger name, level).

daiquiri.**setup**(*level=30, outputs=[<daiquiri.output.Stream object>], program_name=None, capture_warnings=True, set_excepthook=True*)

Setup Python logging.

This will setup basic handlers for Python logging.

Parameters

- **level** – Root log level.
- **outputs** – Iterable of outputs to log to.
- **program_name** – The name of the program. Auto-detected if not set.
- **capture_warnings** – Capture warnings from the 'warnings' module.

3.1 Output

class daiquiri.output.**Datadog**(*hostname='127.0.0.1', port=10518, formatter=<daiquiri.formatter.DatadogFormatter object>, level=None*)

class daiquiri.output.**File**(*filename=None, directory=None, suffix='.log', program_name=None, formatter=<daiquiri.formatter.ColorExtrasFormatter object>, level=None*)

Ouput to a file.

class daiquiri.output.**Journal**(*program_name=None, formatter=<daiquiri.formatter.ColorExtrasFormatter object>, level=None*)

class daiquiri.output.**Output**(*handler, formatter=<daiquiri.formatter.ColorExtrasFormatter object>, level=None*)

Generic log output.

add_to_logger(*logger*)

Add this output to a logger.

class daiquiri.output.**RotatingFile**(*filename=None, directory=None, suffix='.log', program_name=None, formatter=<daiquiri.formatter.ColorExtrasFormatter object>, level=None, max_size_bytes=0, backup_count=0*)

Output to a file, rotating after a certain size.

do_rollover()

Manually forces a log file rotation.

class daiquiri.output.**Stream**(*stream=<open file '<stderr>', mode 'w'>, formatter=<daiquiri.formatter.ColorExtrasFormatter object>, level=None*)

Generic stream output.

class daiquiri.output.**Syslog**(*program_name=None, facility='user', formatter=<daiquiri.formatter.ColorExtrasFormatter object>, level=None*)

class daiquiri.output.**TimedRotatingFile**(*filename=None, directory=None, suffix='.log', program_name=None, formatter=<daiquiri.formatter.ColorExtrasFormatter object>, level=None, interval=datetime.timedelta(1), backup_count=0*)

Rotating log file output, triggered by a fixed interval.

do_rollover()
Manually forces a log file rotation.

3.2 Handlers

class daiquiri.handlers.**JournalHandler**(*program_name*)
Journald based handler. Only available on platforms using systemd.

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

class daiquiri.handlers.**PlainTextSocketHandler**(*hostname*, *port*, *encoding='utf-8'*)
Socket handler that uses format and encode the record.

makePickle(*record*)
Pickles the record in binary format with a length prefix, and returns it ready for transmission across the socket.

class daiquiri.handlers.**SyslogHandler**(*program_name*, *facility=None*)
Syslog based handler. Only available on UNIX-like platforms.

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

class daiquiri.handlers.**TTYDetectorStreamHandler**(*stream=None*)
Stream handler that adds a hint in the record if the stream is a TTY.

format(*record*)
Format the specified record.

If a formatter is set, use it. Otherwise, use the default formatter for the module.

3.3 Formatter

class daiquiri.formatter.**ColorExtrasFormatter**(*keywords=None*, *extras_template='[{0}: {1}]'*, *extras_separator=' '*, *extras_prefix=' '*, *extras_suffix=''*, **args*, ***kwargs*)
Combines functionality of ColorFormatter and ExtrasFormatter.

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class daiquiri.formatter.**ColorFormatter**(*fmt=None*, *datefmt=None*)
Colorizes log output

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class daiquiri.formatter.**DatadogFormatter**

add_fields(*log_record*, *record*, *message_dict*)
Override this method to implement custom logic for adding fields.

class daiquiri.formatter.**ExtrasFormatter**(*keywords=None*, *extras_template='[{0}: {1}]'*, *extras_separator=' '*, *extras_prefix=' '*, *extras_suffix=''*, **args*, ***kwargs*)
Formats extra keywords into %(extras)s placeholder.

Any keywords passed to a logging call will be formatted into a "extras" string and included in a logging message. Example:

logger.info('my message', extra='keyword')

will cause an "extras" string of: [extra: keyword]

to be inserted into the format in place of %(extras)s.

The optional *keywords* argument must be passed into the init function to enable this functionality. Without it normal daiquiri formatting will be applied. Any keywords included in the *keywords* parameter will not be included in the "extras" string.

Special keywords:

keywords A set of strings containing keywords to filter out of the "extras" string.

extras_template A format string to use instead of '[{0}: {1}]'

extras_separator What string to "join" multiple "extras" with.

extras_prefix and extras_suffix Strings which will be prepended and appended to the "extras" string respectively. These will only be prepended if the "extras" string is not empty.

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

Python Module Index

d

Index

T