Logger Helper Documentation

Release 0.1.0

vimist

May 12, 2022

Contents

CHAPTER 1

Getting Started

Please read the *Installation* section before going any further.

1.1 Importing and Setup

Import `logging` and the *`logger_helper.LoggerHelper`* class to get started:

```
>>> import logging
>>> import logger_helper
```

Here, as an example, we're going to set up and use a simple file handler to capture our logs, but you can (and should) configure the logging module as you would normally (see the Python documentation on logging configuration) if you're going to use it in your own application:

```
>>> logger = logging.getLogger(__name__)
>>> handler = logging.FileHandler('/tmp/my_log.log', 'w')
>>> logger.addHandler(handler)
>>> logger.setLevel(logging.DEBUG)
```

Now we've got a basic logger configured we can create a new instance of the *`logger_helper.LoggerHelper`* class. Pass it the logger we want it to write to and the level at which it should write to it with:

```
>>> log = logger_helper.LoggerHelper(
...     logging.getLogger(__name__), logging.DEBUG)
```

That's it, you're done! (almost). The only thing that's left to do from here is to choose the modules, classes, methods and functions to wrap!

1.2 Usage

Once you have the logger configured, classes, methods and functions can all be wrapped very simply, just use the class instance as a decorator:

```
>>> @log
>>> class MyClass:
>>>     def method_one(self, param):
>>>         print('Method one:', param)
>>>         return 123
>>>
>>> @log
>>> def my_function(param_1, param_2):
>>>     print('Doing something with {} and {}'.format(param_1, param_2))
>>>     raise Exception('Something didn\'t work out...')
```

After you've wrapped your classes and functions, you can use them just as you would normally:

```
>>> my_class = MyClass()
>>> my_class.method_one('Hi')
Method one: Hi
>>>
>>> try:
...     my_function('Blue', 'Green')
... except Exception as ex:
...     print('Caught:', ex)
Doing something with Blue and Green
Caught: Something didn't work out...
```

Now, lets take a look at `/tmp/my_log.log` (the handler we set up at the beginning of this tutorial:

```
cat /tmp/my_log.log
```

```
Calling __main__.MyClass.method_one(param = 'Hi')
Returned 123 from __main__.MyClass.method_one
Calling __main__.my_function(param_1 = 'Blue', param_2 = 'Green')
Exception Exception occurred in __main__.my_function, "Something didn't work out..."
```

Done! (for real this time). That's how simple it is to get started with Logger Helper!

1.3 The `mod` Method

There is one other useful feature that Logger Helper provides, the *`logger_helper.LoggerHelper.mod()`* method. When it's passed a module, it will wrap all functions and classes within it.

Assuming that we've set up our `logger` and `LoggerHelper` (as described above) and you have `import my_module` in your file, we can then do the following:

```
>>> log.mod(my_module) # Remember that `log` is an instance of `logger_helper.
↪LoggerHelper`
```

You can also pass a list of symbols (classes/functions) that you want to wrap within your module to limit what gets wrapped:

```
>>> log.mod(my_module, ['ClassOne', 'some_function'])
```

1.4 Further Reading

Take a look at the *logger_helper.LoggerHelper.mod()*, *logger_helper.LoggerHelper.cls()*, *logger_helper.LoggerHelper.meth()* and *logger_helper.LoggerHelper.func()* docstrings for more information.

CHAPTER 2

Logger Helper

Logger Helper main classes and utility functions.

class logger_helper.**LoggerHelper**(*logger*, *log_level*)

Log calls to class methods and functions.

Create a new helper that writes to a specific logger.

Parameters

- **logger** (*logging.Logger*) – The logger to write to.
- **log_level** (*int*) – The log level to log at. This should be one of the `logging.XXXXX` constants, for example `logging.DEBUG`.

call_log_format

str – The format string to use when formatting a call to a callable. The available tokens are:

- `callable` - The name of the callable.
- `args` - The arguments passed to the callable. The format of the arguments is defined in the `argument_format` property.

argument_format

str – The format of each argument. The available tokens are:

- `name` - The name of the argument.
- `value` - The value of the argument.

argument_separator

str – The separator to join the arguments together with.

return_log_format

str – The format to log the return with. The available tokens are:

- `callable` - The name of the callable.
- `return_value` - The return value (after it's been run through `repr`).

exception_log_format
str – The format to log exceptions with. The available tokens are:

- `callable` - The name of the callable.
- `name` - The name of the exception.
- `message` - The exception message.

__call__(*obj*)
Wrap the class methods or functions in our decorator.

Raises `TypeError` – When the object isn't a class or function.

Returns The fully wrapped class or function.

cls(*cls*)
Wrap a classes methods (that don't start and end with __).

Parameters **cls** – The class to wrap.

Returns A *copy* of the given class with all of it's methods wrapped.

Return type `cls`

func(*function*)
Wrap a function.

Parameters **function** – The function to wrap.

Returns A wrapped *copy* of the given function.

Return type `function`

meth(*method*)
Wrap a method belonging to a class.

Returns A wrapped *copy* of the fiven method.

Return type `method`

mod(*mod*, *symbols=None*)
Wrap an entire module's classes and functions.

Parameters

- **mod** (*module*) – The module to wrap.
- **symbols** (*list*) – If this is specified, only the symbols (classes/functions) listed will be wrapped. Each item in the list should be a string.

Returns Nothing is returned, the class is wrapped in place.

Return type None

logger_helper.**get_callable_name**(*clbl*)
Get the fully qualified name of a callable.

Parameters **clbl** – The callable to get the name for.

Returns A string representing the full path to the given callable.

Return type str

CHAPTER 3

Installation

Installing Logger Helper is super simple, you can either use `pip`:

```
pip install logger-helper
```

or you can install it manually after having cloned or downloaded this repository:

```
python3 setup.py install
```

You're ready to go! Read the *Getting Started* guide for information on what to do next!

Python Module Index

l

Index

Symbols

A

C

E

F

G

L

M

R