Flask-LogConfig Documentation

Release 0.4.2

Derrick Gilland

September 27, 2017

Contents

1 Requirements **3**
- 1.1 Compatibility . . . 3
- 1.2 Dependencies . . . 3

2 Installation **5**

3 Quickstart **7**

4 Configuration **9**
- 4.1 LOGCONFIG . . . 9
- 4.2 LOGCONFIG_QUEUE . . . 9
- 4.3 LOGCONFIG_REQUESTS_ENABLED . . . 10
- 4.4 LOGCONFIG_REQUESTS_LOGGER . . . 10
- 4.5 LOGCONFIG_REQUESTS_LEVEL . . . 10
- 4.6 LOGCONFIG_REQUESTS_MSG_FORMAT . . . 10

5 Log Record Request Context **13**

6 Guide **15**
- 6.1 Installation . . . 15

7 API Reference **17**
- 7.1 API Reference . . . 17

8 Project Info **19**
- 8.1 License . . . 19
- 8.2 Changelog . . . 19
- 8.3 Authors . . . 21

9 Indices and tables **23**

Python Module Index **25**

Flask extension for configuring Python logging module.

CHAPTER 1

Requirements

Compatibility

- Python 2.6
- Python 2.7
- Python 3.3
- Python 3.4

Dependencies

- Flask
- logconfig

CHAPTER 2

Installation

```
pip install Flask-LogConfig
```

CHAPTER 3

Quickstart

Use `Flask-LogConfig` to easily configure the Python logging module using your Flask app's `config` object:

```
import flask
from flask.ext.logconfig import LogConfig

class MyConfig(object):
    LOGCONFIG = {
        'version': 1,
        'disable_existing_loggers': False,
        'formatters': {
            'simple': {
                '()': 'myapp.logging.simple_formatter_factory'
            },
            'email': {
                '()': 'myapp.logging.email_formatter_factory'
            }
        },
        'filters': {
            'email': {
                '()': 'myapp.logging.RequestFilter'
            }
        },
        'handlers': {
            'smtp': {
                'class': 'logging.handlers.SMTPHandler',
                'level': 'ERROR',
                'formatter': 'email',
                'filters': ['email'],
                'mailhost': ('example.com', 587),
                'fromaddr': 'Mailer <mailer@example.com>',
                'toaddrs': ['admins@example.com'],
                'subject': 'Application Error',
                'credentials': ('mailer@example.com', 'password'),
                'secure': ()
            },
            'console': {
                'class': 'logging.StreamHandler',
                'level': 'DEBUG',
                'formatter': 'simple',
                'stream': 'ext://sys.stderr'
            }
        },
        'loggers': {
```

```
            'myapp': {
                'handlers': ['smtp', 'console'],
                'level': 'DEBUG'
            }
        }
    }

    LOGCONFIG_QUEUE = ['myapp']

app = flask.Flask(__name__)
app.config.from_object(MyConfig)
logcfg = LogConfig(app)

# or using lazy instantiation
logcfg = LogConfig()
logcfg.init_app(app)
```

CHAPTER 4

Configuration

Configuration of Python's logging module is specified using the standard `dictConfig` or `fileConfig` formats supported by logging.config. This allows Flask apps to be configured as one would in a Django app that uses logging.

LOGCONFIG

The main configuration option for `Flask-LogConfig` is `LOGCONFIG`. This option can either be a `dict` or a pathname to a configuration file. The format of the `dict` or config file must follow the format supported by `logging.config.dictConfig` or `loging.config.fileConfig`. See Logging Configuration for more details. If using a pathname, the supported file formats are `JSON`, `YAML`, and `ConfigParser`.

LOGCONFIG_QUEUE

The purpose of `LOGCONFIG_QUEUE` is to provide an easy way to utilize logging without blocking the main thread.

To set up a basic logging queue, specify the loggers you want to queuify by setting `LOGCONFIG_QUEUE` to a list of the logger names (as strings). These loggers will have their handlers moved to a queue which will then be managed by a queue handler and listener, one per logger.

Each logger's queue handler will be an instance of `flask_logconfig.FlaskQueueHandler` which is an extension of logging.handlers.QueueHandler (back ported to Python 2 via logutils). `FlaskQueueHandler` adds a copy of the current request context to the log record so that the queuified log handlers can access any Flask request globals outside of the normal request context (i.e. inside the listener thread) via `flask_logconfig.request_context_from_record`. The queue listener used is an instance of logconfig.QueueListener that extends logging.handlers.QueueListener with proper support for respecting a handler's log level (i.e. `logging.handlers.QueueListener` delegates all log records to a handler even if that handler's log level is set higher than the log record's while `logconfig.QueueListener` does not).

After the log handlers are queuified, their listener thread will be started automatically unless you specify otherwise. You can access the listeners via the `LogConfig` instance:

```
logcfg = LogConfig()

# start_listeners=True by default
logcfg.init_app(app, start_listeners=False)

assert isinstance(logcfg, list)

# start listeners manually
```

```
logcfg.start_listeners(app)

# stop listeners
logcfg.stop_listeners(app)
```

See the Log Record Request Context section for details on accessing an application's request context from within a queue.

LOGCONFIG_REQUESTS_ENABLED

When set to `True`, `LOGCONFIG_REQUESTS_ENABLED` turns on logging for all requests. Defaults to `False`.

Requests will be logged at the end of the request via the `app.after_request` hook. In addition to providing a custom log msg, additional `extra` arguments will be passed to the logging call:

- `response`
- `request`

These can later be accessed from the log record via `record.response` and `record.request`. This provides a convenient way for the log filters, handlers, and formatters to access request/response specific data.

LOGCONFIG_REQUESTS_LOGGER

The logger name to use when logging all requests. Defaults to `None` which uses `app.logger`.

LOGCONFIG_REQUESTS_LEVEL

The log level at which to log all requests. Defaults to `logging.DEBUG`.

LOGCONFIG_REQUESTS_MSG_FORMAT

The message format used to generate the `msg` argument to `log()` when logging all requests. Defaults to `'{method} {path} - {status_code}'`.

When generating the message, `LOGCONFIG_REQUESTS_MSG_FORMAT.format(**kargs)` will be called with the following keyword arguments:

From request.environ

- `SERVER_PORT`
- `SERVER_PROTOCOL`
- `SCRIPT_NAME`
- `REQUEST_METHOD`
- `HTTP_HOST`
- `PATH_INFO`

- QUERY_STRING
- CONTENT_LENGTH
- SERVER_NAME
- CONTENT_TYPE

NOTE: Additional data may be available depending on the WSGI environment provided.

From request

- method
- path
- base_url
- url
- remote_addr
- user_agent

From response

- status_code
- status

From flask

- session

NOTE: The session argument is a computed as follows:

```
from collections import defaultdict
from flask import session

session_data = defaultdict(lambda: None)
session_data.update(dict(session)
```

This means that you can safely access session values even if they aren't explictly set. When they are missing, None will be returned instead.

From computed

- execution_time (in milliseconds) **NOTE:** This is the time between the start of the request and then end.

CHAPTER 5

Log Record Request Context

When using `LOGCONFIG_QUEUE`, accessing Flask's request globals from within a log handler requires using the request context that is attached to the emitted log record.

Below is an example that uses a logging `Filter` to attach the request environment to the log record using `flask_logconfig.request_context_from_record`:

```
import logging
from pprint import pformat
from flask import request

from flask_logconfig import request_context_from_record

class RequestFilter(logging.Filter):
    """Impart contextual information related to Flask HTTP request."""
    def filter(self, record):
        """Attach request contextual information to log record."""
        with request_context_from_record(record):
            record.environ_info = request.environ.copy()
            record.environ_text = pformat(record.environ_info)
        return True
```

It's also safe to use `request_context_from_record` from directly inside Flask's request context:

```
with request_context_from_record():
    # do something using Flask request globals
    pass
```

If no request context exists (either on the log record provided or inside the actual Flask request context), then a `flask_logconfig.FlaskLogConfigException` will be thrown.

CHAPTER 6

Guide

Installation

Flask-LogConfig requires Python >= 2.6 or >= 3.3.

To install from PyPi:

```
pip install Flask-LogConfig
```

CHAPTER 7

API Reference

Includes links to source code.

API Reference

Flask-LogConfig module.

class flask_logconfig.**LogConfig**(*app=None*, *start_listeners=True*, *queue_class=None*, *handler_class=None*, *listener_class=None*)

Flask extension for configuring Python's logging module from application's config object.

add_listener(*app*, *name*, *listener*)

Add *listener* indexed by *name* to application.

after_request(*response*)

Log request.

before_request()

Store information related to start of request.

default_handler_class

alias of FlaskQueueHandler

default_listener_class

alias of QueueListener

default_queue_class

alias of Queue

get_app(*app=None*)

Look up and return application.

get_execution_time()

Get response time for request in milliseconds.

get_listeners(*app=None*)

Return listeners associated with application.

get_request_message_data(*response*)

Return data for use in request message format string.

get_requests_logger()

Get designated logger for requests.

get_state(*app=None*)
Return stored state for application.

init_app(*app*, *start_listeners=True*, *queue_class=None*, *handler_class=None*, *listener_class=None*)
Initialize extension on Flask application.

make_request_message(*data*)
Return string formatted message for request log message.

setup_logging(*app*)
Setup logging configuration for application.

setup_queue(*app*, *start_listeners*, *queue_class*, *listener_class*, *handler_class*)
Setup unified logging queue for application.

start_listeners(*app=None*)
Start all queue listeners for application.

stop_listeners(*app=None*)
Stop all queue listeners for application.

class flask_logconfig.**FlaskQueueHandler**(*queue*)
Extend QueueHandler to attach Flask request context to record since request context won't be available inside listener thread.

prepare(*record*)
Return a prepared log record. Attach a copy of the current Flask request context for use inside threaded handlers.

exception flask_logconfig.**FlaskLogConfigException**
Base exception class for Flask-LogConfig.

flask_logconfig.**request_context_from_record**(**args*, ***kwds*)
Context manager for Flask request context attached to log record or if one doesn't exist, then from top of request context stack.

Raises FlaskLogConfigException – If no request context exists on *record* or stack.

CHAPTER 8

Project Info

License

The MIT License (MIT)

Copyright (c) 2014 Derrick Gilland

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Changelog

v0.4.2 (2015-07-29)

- Fix bug in application lookup where `None` would be returned instead of the application instance when extension was previously initialized with an application instance passed in.

v0.4.1 (2015-05-20)

- Fix issue where an extra request context stack was created during `reqest_context_from_record()` call.

v0.4.0 (2015-02-03)

- Add `execution_time` to log's extra data and request message data.

- Rename `FlaskLogConfig.after_request_handler` to `FlaskLogConfig.after_request`. **(possible breaking change)**
- Rename `FlaskLogConfig.get_request_msg_data` to `FlaskLogConfig.get_request_message_data`. **(possible breaking change)**
- Rename `FlaskLogConfig.make_request_msg` to `FlaskLogConfig.make_request_message`. **(possible breaking change)**

v0.3.1 (2015-01-26)

- Add metadata to main module:
  - `__title__`
  - `__summary__`
  - `__url__`
  - `__version__`
  - `__author__`
  - `__email__`
  - `__license__`

v0.3.0 (2015-01-25)

- Add support for logging all requests.
- Don't store any application specific state on `LogConfig` class. Move `LogConfig.listeners` access to `LogConfig.get_listeners`. **(breaking change)**
- Make `LogConfig.__init__` and `LogConfig.init_app` accept custom queue class via `queue_class` argument.
- Make `LogConfig.start_listeners()` and `LogConfig.stop_listeners()` accept optional `app` argument to access listeners associated with that app. If no `app` passed in, then `flask.current_app` will be accessed.
- Rename supported configuration keys from `LOGGING` and `LOGGING_QUEUE``to ``LOGCONFIG` and `LOGCONFIG_QUEUE` respectively. **(breaking change)**

v0.2.0 (2015-01-22)

- Make `LogConfig.__init__` and `LogConfig.init_app` accept custom handler and listener classes via `handler_class` and `listener_class` arguments.

v0.1.0 (2014-12-24)

- First release.

Authors

Lead

- Derrick Gilland, dgilland@gmail.com, dgilland@github

Contributors

None

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*

f

flask_logconfig, 17

A

add_listener() (flask_logconfig.LogConfig method), 17
after_request() (flask_logconfig.LogConfig method), 17

B

before_request() (flask_logconfig.LogConfig method), 17

D

default_handler_class (flask_logconfig.LogConfig attribute), 17
default_listener_class (flask_logconfig.LogConfig attribute), 17
default_queue_class (flask_logconfig.LogConfig attribute), 17

F

flask_logconfig (module), 17
FlaskLogConfigException, 18
FlaskQueueHandler (class in flask_logconfig), 18

G

get_app() (flask_logconfig.LogConfig method), 17
get_execution_time() (flask_logconfig.LogConfig method), 17
get_listeners() (flask_logconfig.LogConfig method), 17
get_request_message_data() (flask_logconfig.LogConfig method), 17
get_requests_logger() (flask_logconfig.LogConfig method), 17
get_state() (flask_logconfig.LogConfig method), 17

I

init_app() (flask_logconfig.LogConfig method), 18

L

LogConfig (class in flask_logconfig), 17

M

make_request_message() (flask_logconfig.LogConfig method), 18

P

prepare() (flask_logconfig.FlaskQueueHandler method), 18

R

request_context_from_record() (in module flask_logconfig), 18

S

setup_logging() (flask_logconfig.LogConfig method), 18
setup_queue() (flask_logconfig.LogConfig method), 18
start_listeners() (flask_logconfig.LogConfig method), 18
stop_listeners() (flask_logconfig.LogConfig method), 18