flask-mongo-profiler Documentation

Release 0.5.0a3post0

Peergrade

Mar 28, 2019

Contents

```
$ pip install --user flask-mongo-profiler
```

CHAPTER 1

Example

```
# inside of a virtualenv
$ pip install flask-mongo-profiler

$ pip install -r requirements/contrib.txt
$ cd examples/flask_todo

# With debugger
$ env FLASK_ENV=develoment flask run

# Without
$ flask run

# Load pages, go to http://localhost:5000/admin/
```

CHAPTER 2

Project details

Python support	2.7, >= 3.5, pypy, pypy3
DB support	pymongo 3.7+, MongoEngine 0.15+ (optional)
Web support	Werkzeug 0.14+, Flask 1.0+ (optional)
Source	https://github.com/peergradeio/flask-mongo-profiler
Docs	https://flask-mongo-profiler.readthedocs.io
API	https://flask-mongo-profiler.readthedocs.io/en/latest/api.html
Changelog	https://flask-mongo-profiler.readthedocs.io/en/latest/history.html
Issues	https://github.com/peergradeio/flask-mongo-profiler/issues
Travis	http://travis-ci.org/peergradeio/flask-mongo-profiler
Test Coverage	https://codecov.io/gh/peergradeio/flask-mongo-profiler
pypi	https://pypi.python.org/pypi/flask-mongo-profiler
Open Hub	https://www.openhub.net/p/flask-mongo-profiler
License	MIT.
git repo	`$ git clone https://github.com/peergradeio/flask-mongo-profiler.git`
install stable	`$ pip install --user flask-mongo-profiler`

install dev	`$ git clone https://github.com/peergradeio/flask-mongo-profiler.git`
	`$ cd ./flask-mongo-profiler`
	`$ virtualenv .venv`
	`$ source .venv/bin/activate`
	`$ pip install -e .`

tests	`$ make test`

Explore:

2.1 About

Profiler for pymongo and MongoEngine. Includes front end support for Flask-Admin.

2.2 API Reference

2.2.1 Utility

flask_mongo_profiler.utils.**combine_events_to_queries**(*_mongo_events*)
Combines pymongo.monitoring events to queries.

CommandStartedEvent has queries information. CommandSuccessfulEvent has timing information.

Iterate over events and map against mongo's request_id (the query).

flask_mongo_profiler.utils.**sanitize_dict**(*d*, *reverse=False*)
ValidationError: ValidationError (ProfilingRequest:None) (Invalid dictionary key name - keys may not contain "." or "$" characters 1.Invalid dictionary key name - keys may not contain "." or "$" characters 2.Invalid dictionary key name - keys may not contain "." or "$" characters: ['queries'])

2.2.2 Helpers

flask_mongo_profiler.helpers.**add_template_dirs**(*app*)
Add flask_mongo_profiler's template directories.

Parameters **app** (*flask.Flask*) –

2.2.3 Constants

flask_mongo_profiler.constants.**MONGO_COMMAND_KEY_DOLLAR_SIGN_REPLACEMENT = '__'**
e.g. $exists, $in

flask_mongo_profiler.constants.**MONGO_COMMAND_KEY_PERIOD_SIGN_REPLACEMENT = '-'**
e.g. parent.ref.$id ($ would be __ thanks to above)

flask_mongo_profiler.constants.**RE_OBJECTID = re.compile('([0-9a-f]{24})')**
Match MongoDB ObjectID pattern

flask_mongo_profiler.constants.**WERKZEUG_ENVIRON_KEY_REPLACEMENT = '-'**
Used to replace keys with . for mongodb compatibility e.g. wsgi.url_scheme

2.2.4 Werkzeug support

See also:

pymongo.monitoring

http //api.mongodb.com/python/current/api/pymongo/monitoring.html

class flask_mongo_profiler.contrib.werkzeug.mongo.**BaseMongoCommandLogger**(*quiet=True, command_filter=['create', 'find', 'update', 'delete']*)

Log Mongo queries we make. Differences from plain-old listener:

1. Turn logging output on/off since there's no public pymongo API to unregister or monitor queries temporarily.
2. Enable logging to various endpoints without having to rewrite everything. It can be valuable to log to standard library logging (stdout, logfile), or database (django-silk style).
3. Can be tailored to output only commands requested.

Parameters

- **quiet** (*bool, optional*) – Silence MongoDB CommandListener from logging (default: True)
- **command_filter** (*list of str, optional*) – MongoDB queries commands to send to logger

See also:

–

failed(*event*)
Abstract method to handle a *CommandFailedEvent*.

Parameters

- *event*: An instance of `CommandFailedEvent`.

start()

started(*event*)
Abstract method to handle a *CommandStartedEvent*.

Parameters

- *event*: An instance of `CommandStartedEvent`.

stop()

succeeded(*event*)
Abstract method to handle a *CommandSucceededEvent*.

Parameters

- *event*: An instance of `CommandSucceededEvent`.

class flask_mongo_profiler.contrib.werkzeug.mongo.**LoggingMongoCommandLogger**(*quiet=True, command_filter=['create', 'find', 'update', 'delete']*)

failed(*event*)
Abstract method to handle a *CommandFailedEvent*.

Parameters

- *event*: An instance of `CommandFailedEvent`.

started(*event*)
Abstract method to handle a *CommandStartedEvent*.

Parameters

- *event*: An instance of `CommandStartedEvent`.

succeeded(*event*)
Abstract method to handle a *CommandSucceededEvent*.

Parameters

- *event*: An instance of `CommandSucceededEvent`.

class flask_mongo_profiler.contrib.werkzeug.mongo.**QueuedMongoCommandLogger**(*quiet=True, command_filter=['create', 'find', 'update', 'delete']*)
Record the commands in self.collector list.

failed(*event*)
Abstract method to handle a *CommandFailedEvent*.

Parameters

- *event*: An instance of `CommandFailedEvent`.

start()

started(*event*)
Abstract method to handle a *CommandStartedEvent*.

Parameters

- *event*: An instance of `CommandStartedEvent`.

stop()

succeeded(*event*)
Abstract method to handle a *CommandSucceededEvent*.

Parameters

- *event*: An instance of `CommandSucceededEvent`.

class flask_mongo_profiler.contrib.werkzeug.werkzeug_middleware.**ProfilerMiddleware**(*app, pymongo_log ignored_url_*

2.2.5 MongoEngine support

class flask_mongo_profiler.contrib.mongoengine.mixins.**FlaskAdminURLMixin**
Add get_admin_url to models supported by Flask-Admin

Flask-Admin creates URL's namespaced w/ model class name, lowercase.

classmethod get_admin_list_url(*_external=False*)

get_admin_url(*_external=False*)

class flask_mongo_profiler.contrib.mongoengine.mixins.**GetAttributeMixin**

get(*attr*, *default=None*)

class flask_mongo_profiler.contrib.mongoengine.profiling.**ProfilingQuery**(**args*, ***values*)

exception DoesNotExist

exception MultipleObjectsReturned

command
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

command_name
A unicode string field.

connection
An embedded document field - with a declared document_type. Only valid values are subclasses of EmbeddedDocument.

database_name
A unicode string field.

duration
Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

failure
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

id
A field wrapper around MongoDB's ObjectIds.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet` , probably the same one that was passed in, but modified in some way.

operation_id
32-bit integer field.

request
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

request_id
A unicode string field.

class flask_mongo_profiler.contrib.mongoengine.profiling.**ProfilingQueryConnection**(**args*, ***kwargs*)

Examples

Unpack ('localhost', 27017) tuple to dictionary:

```
>>> params = dict(
>>>     connection=model.ProfilingQueryConnection(*event.connection_id),
>>> )
```

All pymongo.monitoring event types include connection_id. The address (host, port) of the server this command was sent to.

References

http://api.mongodb.com/python/current/api/pymongo/monitoring.html

host
A unicode string field.

port
32-bit integer field.

class flask_mongo_profiler.contrib.mongoengine.profiling.**ProfilingRequest**(**args*, ***values*)

exception DoesNotExist

exception MultipleObjectsReturned

duration
Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

environ
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

id
A field wrapper around MongoDB's ObjectIds.

method
A unicode string field.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet` , probably the same one that was passed in, but modified in some way.

path
A unicode string field.

pyprofile
A unicode string field.

query_count
32-bit integer field.

query_duration
Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

referrer
A unicode string field.

status
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

time
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

2.2.6 Flask Admin support

Helpers

flask_mongo_profiler.contrib.flask_admin.helpers.**get_list_url_filtered_by_field_value**(*view*, *model*, *name*, *reverse=*

Get the URL if a filter of model[name] value was appended.

This allows programatically adding filters. This is used in the specialized case of filtering deeper into a list by a field's value.

For instance, since there can be multiple assignments in a list of handins. The assignment column can have a URL generated by get_filter_url to filter the handins to show only ones for that assignment.

Parameters

- **view** (*View instance*) –
- **model** (*document (model instance, not the class itself)*) –
- **name** (*field name*) –
- **reverse** (*bool*) – Whether to *remove* an applied filter from url

Returns string

Return type URL of current list args + filtering on field value

flask_mongo_profiler.contrib.flask_admin.helpers.**search_relative_field**(*model_class*, *fields*, *term*)

Searches a ReferenceField's fields, returning ID's to be used in __in

There is no JOIN, so no Assignment.objects.filter(course__title='My Course'). To get around this, we return a list of ID's.

Since this is an internal tool, we allow multiple fields to AND/OR group.

Formatters

flask_mongo_profiler.contrib.flask_admin.formatters.date.**date_formatter**(*view*, *value*)

We appear to store naive datetimes in our database.

Formats the date, and if the user has a timezone that's different from the naive date, will append the naive date below appended with UTC.

flask_mongo_profiler.contrib.flask_admin.formatters.lookup.**search_field_formatter**(*view*, *context*, *model*, *name*)

Formatters that do polymorphic relation resolution against GenericReferienceField. They will look up the original model in the field, then lookup like a normal relational formatter.

See also:

`model.AdminLog.document`, `model.Answer.parent`

References

flask_mongo_profiler.contrib.flask_admin.formatters.polymorphic_relations.**generic_document_**

Return AdminLog.document field wrapped in URL to its list view.

flask_mongo_profiler.contrib.flask_admin.formatters.polymorphic_relations.**generic_lazy_ref_**

See also:

diff_formatter()

flask_mongo_profiler.contrib.flask_admin.formatters.polymorphic_relations.**generic_ref_forma**

For GenericReferenceField and LazyGenericReferenceField

See also:

diff_formatter()

flask_mongo_profiler.contrib.flask_admin.formatters.polymorphic_relations.**generic_ref_list_**

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**http_method_formatter**(*view*, *context*, *model*, *name*)

Wrap HTTP method value in a bs3 label.

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**map_to_bootstrap_column_forma**

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**mongo_command_name_formatter**

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**profile_pyprofile_formatter**(v

Format pyprofile output

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**profiling_pure_query_formatte**

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**profiling_query_formatter**(*view*

Format a ProfilingQuery entry for a ProfilingRequest detail field

Parameters **query_document** (*model.ProfilingQuery*) –

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**profiling_query_list_formatte**

Parameters **queryset** (*mongoengine.Queryset of model.ProfilingQuery*) –

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**profiling_request_formatter**(v c t r r

Wrap HTTP method value in a bs3 label.

flask_mongo_profiler.contrib.flask_admin.formatters.profiling.**request_environ_formatter**(*viev con text mod nan*

flask_mongo_profiler.contrib.flask_admin.formatters.relational.**qs_field**(*model_class*, *field*, *filters=None*, *formatter=<function queryset_formatter>*, *manager_name='objects'*)

Show computed fields based on QuerySet's.

This is a workaround since sometimes some filtering is involved to see if a user owns and object, is a student, etc.

Example

class MyModel(ModelView):

details_extra_columns = [('courses_owned', 'Courses (Owner of)'),

] column_formatters_detail = {

'courses_owner': qs_field(model.Course, 'owner'),

]

flask_mongo_profiler.contrib.flask_admin.formatters.relational.**queryset_formatter**(*queryset*)

This is used for custom detail fields returning a QuerySet of admin objects.

Views

class flask_mongo_profiler.contrib.flask_admin.views.base.**BaseModelView**(**args*, ***kwargs*)

action_view()

Mass-model action view.

ajax_lookup()

ajax_update()
Edits a single column of a record in list view.

api_file_view()

create_view()
Create model view

delete_view()
Delete model view. Only POST method is allowed.

details_view()
Details model view

edit_view()
Edit model view

export(*export_type*)

index_view()
List view

named_filter_urls = True

class flask_mongo_profiler.contrib.flask_admin.views.base.**ColumnFieldTypeFormattersMixin**(*

class flask_mongo_profiler.contrib.flask_admin.views.base.**ExtraDetailColumnsMixin**

details_extra_columns = []

get_details_columns()
Add details_extra_columns. (Not in normal Flask-Admin)

class flask_mongo_profiler.contrib.flask_admin.views.base.**PrettyDatesMixin**

column_type_formatters = {<class 'datetime.date'>: <function date_formatter>}

column_type_formatters_detail = {<class 'datetime.date'>: <function date_formatter>}

class flask_mongo_profiler.contrib.flask_admin.views.base.**ReadOnlyMixin**

can_create = False

can_delete = False

can_edit = False

can_view_details = True

class flask_mongo_profiler.contrib.flask_admin.views.base.**RelationalFieldMixin**

allowed_search_types = (<class 'mongoengine.fields.StringField'>, <class 'mongoengine.

column_field_type_formatters = {<class 'mongoengine.fields.ReferenceField'>: <functio

column_field_type_formatters_detail = {<class 'mongoengine.fields.ReferenceField'>: <

class flask_mongo_profiler.contrib.flask_admin.views.base.**RelationalSearchMixin**

column_searchable_refs = {}

flask_mongo_profiler.contrib.flask_admin.views.date.**date_formatter**(*view*, *value*)

We appear to store naive datetimes in our database.

Formats the date, and if the user has a timezone that's different from the naive date, will append the naive date below appended with UTC.

class flask_mongo_profiler.contrib.flask_admin.views.profiling.**ProfilingCommonView**(**args*, ***kwargs*)

action_view()
Mass-model action view.

ajax_lookup()

ajax_update()
Edits a single column of a record in list view.

api_file_view()

clear_entities()

create_view()
Create model view

delete_view()
Delete model view. Only POST method is allowed.

details_view()
Details model view

edit_view()
Edit model view

export(*export_type*)

index_view()
List view

list_template = 'admin/model/profiling-list.html'

class flask_mongo_profiler.contrib.flask_admin.views.profiling.**ProfilingQueryView**(**args*, ***kwargs*)

action_view()
Mass-model action view.

ajax_lookup()

ajax_update()
Edits a single column of a record in list view.

api_file_view()

clear_entities()

column_filters = ['command_name', 'request']

column_formatters = {'command': <function profiling_pure_query_formatter>, 'command_n

column_formatters_detail = {'command_name': <function mongo_command_name_formatter>,

column_labels = {'duration': 'Duration (ms)'}

column_list = ['command_name', 'request', 'duration', 'command']

column_searchable_list = ['command_name', 'request']

create_view()
Create model view

delete_view()
Delete model view. Only POST method is allowed.

details_extra_columns = [('command_pretty', 'Command (pretty)')]

details_view()
Details model view

edit_view()
Edit model view

export(*export_type*)

index_view()
List view

class flask_mongo_profiler.contrib.flask_admin.views.profiling.**ProfilingRequestView**(**args*, ***kwargs*

action_view()
Mass-model action view.

ajax_lookup()

ajax_update()
Edits a single column of a record in list view.

api_file_view()

clear_entities()

column_descriptions = {'referrer': 'Page requesting API endpoint. If empty, is the pa

column_details_exclude_list = ['pyprofile']

column_filters = ['method', 'path', 'referrer', 'duration', 'query_duration', 'query_c

column_formatters = {'method': <function http_method_formatter>, 'path': <function s

column_formatters_detail = {'environ': <function request_environ_formatter>, 'method'

column_labels = {'duration': 'Duration (ms)', 'query_duration': 'Query Duration (ms)

column_list = ['method', 'path', 'referrer', 'duration', 'query_duration', 'query_coun

column_searchable_list = ['path', 'method']

create_view()
Create model view

delete_view()
Delete model view. Only POST method is allowed.

details_extra_columns = [('queries', 'Queries'), ('pyprofile', 'Profile')]

details_view()
Details model view

edit_view()
Edit model view

export(*export_type*)

index_view()
List view

list_template = 'admin/model/profilingrequest-list.html'

2.3 History

2.3.1 Upcoming

2.3.2 0.5.0 alpha 3

December 15th, 2018

- Fixes for PyPI Packaging

  Required directories underneath `flask_mongo_profiler` weren't being included in the distribution.

2.3.3 0.5.0 alpha 2

October 28th, 2018

- Add example project
- Add tests for example project
- Test middleware, admin panel, coverage
- Fix date module
- Docs: Add API docs
- Python 2/3 fixes
- Template helper

2.3.4 0.5.0 alpha 1

October 27th, 2018

- README
- Metadata
- Packaging
- Formatting / linting
- Initial README
- Ported over initial Flask-Admin helpers, models, etc.
- Sphinx documentation for ReadTheDocs

Python Module Index

f

Index

A

B

C

D

E

S

T

W