Flask-PyMongo Documentation

Release 0.3.0

Dan Crosta

December 01, 2013

Contents

MongoDB is an open source database that stores flexible JSON-like "documents," which can have any number, name, or hierarchy of fields within, instead of rows of data as in a relational database. Python developers can think of MongoDB as a persistent, searchable repository of Python dictionaries (and, in fact, this is how PyMongo represents MongoDB documents).

Flask-PyMongo bridges Flask and PyMongo, so that you can use Flask's normal mechanisms to configure and connect to MongoDB.

CHAPTER 1

Quickstart

First, install Flask-PyMongo:

```
$ pip install Flask-PyMongo
```

Flask-PyMongo depends, and will install for you, recent versions of Flask (0.8 or later) and PyMongo (2.4 or later). Flask-PyMongo is compatible with and tested on Python 2.6, 2.7, and 3.3.

Next, add a PyMongo to your code:

```
from flask import Flask
from flask.ext.pymongo import PyMongo

app = Flask(__name__)
mongo = PyMongo(app)
```

PyMongo connects to the MongoDB server running on port 27017 on localhost, and assumes a default database name of app.name (i.e. whatever name you pass to Flask). This database is exposed as the db attribute.

You can use db directly in views:

```
@app.route('/')
def home_page():
    online_users = mongo.db.users.find({'online': True})
    return render_template('index.html',
        online_users=online_users)
```

CHAPTER 2

Helpers

Flask-PyMongo provides helpers for some common tasks:

`Collection.`**`find_one_or_404`**`(`**args*, ***kwargs*`)`

Find and return a single document, or raise a 404 Not Found exception if no document matches the query spec. See `find_one()` for details.

```
@app.route('/user/<username>')
def user_profile(username):
    user = mongo.db.users.find_one_or_404({'_id': username})
    return render_template('user.html',
        user=user)
```

`PyMongo.`**`send_file`**`(`*filename*, *base='fs'*, *version=-1*, *cache_for=31536000*`)`

Return an instance of the `response_class` containing the named file, and implement conditional GET semantics (using `make_conditional()`).

```
@app.route('/uploads/<path:filename>')
def get_upload(filename):
    return mongo.send_file(filename)
```

Parameters

- **filename** (*str*) – the filename of the file to return
- **base** (*str*) – the base name of the GridFS collections to use
- **version** (*bool*) – if positive, return the Nth revision of the file identified by filename; if negative, return the Nth most recent revision. If no such version exists, return with HTTP status 404.
- **cache_for** (*int*) – number of seconds that browsers should be instructed to cache responses

`PyMongo.`**`save_file`**`(`*filename*, *fileobj*, *base='fs'*, *content_type=None*`)`

Save the file-like object to GridFS using the given filename. Returns `None`.

```
@app.route('/uploads/<path:filename>', methods=['POST'])
def save_upload(filename):
    mongo.save_file(filename, request.files['file'])
    return redirect(url_for('get_upload', filename=filename))
```

Parameters

- **filename** (*str*) – the filename of the file to return
- **fileobj** (*file*) – the file-like object to save
- **base** (*str*) – base the base name of the GridFS collections to use
- **content_type** (*str*) – the MIME content-type of the file. If `None`, the content-type is guessed from the filename using `guess_type()`

class `flask_pymongo.`**`BSONObjectIdConverter`**`(`*map*`)`

A simple converter for the RESTful URL routing system of Flask.

```
@app.route('/<ObjectId:task_id>')
def show_task(task_id):
    task = mongo.db.tasks.find_one_or_404(task_id)
    return render_template('task.html', task=task)
```

Valid object ID strings are converted into `ObjectId` objects; invalid strings result in a 404 error. The converter is automatically registered by the initialization of `PyMongo` with keyword `ObjectId`.

CHAPTER 3

Configuration

PyMongo understands the following configuration directives:

MONGO_URI	A MongoDB URI which is used in preference of the other configuration variables.
MONGO_HOST	The host name or IP address of your MongoDB server. Default: "localhost".
MONGO_PORT	The port number of your MongoDB server. Default: 27017.
MONGO_AUTO_START_REQUEST	Set to False to disable PyMongo 2.2's "auto start request" behavior (see MongoClient). Default: True.
MONGO_MAX_POOL_SIZE	(optional): The maximum number of idle connections maintained in the PyMongo connection pool. Default: PyMongo default.
MONGO_SOCKET_TIMEOUT_MS	(optional): (integer) How long (in milliseconds) a send or receive on a socket can take before timing out. Default: PyMongo default.
MONGO_CONNECT_TIMEOUT_MS	(optional): (integer) How long (in milliseconds) a connection can take to be opened before timing out. Default: PyMongo default.
MONGO_DBNAME	The database name to make available as the db attribute. Default: app.name.
MONGO_USERNAME	The user name for authentication. Default: None
MONGO_PASSWORD	The password for authentication. Default: None
MONGO_REPLICA_SET	The name of a replica set to connect to; this must match the internal name of the replica set (as deteremined by the isMaster command). Default: None.
MONGO_READ_PREFERENCE	Determines how read queries are routed to the replica set members. Must be one of the constants defined on pymongo.read_preferences.ReadPreference or the string names thereof
MONGO_DOCUMENT_CLASS	This tells pymongo to return custom objects instead of dicts, for example bson.son.SON. Default: dict

When PyMongo or init_app() are invoked with only one argument (the Flask instance), a configuration value prefix of MONGO is assumed; this can be overridden with the *config_prefix* argument.

This technique can be used to connect to multiple databases or database servers:

```
app = Flask(__name__)

# connect to MongoDB with the defaults
mongo1 = PyMongo(app)

# connect to another MongoDB database on the same host
app.config['MONGO2_DBNAME'] = 'dbname_two'
mongo2 = PyMongo(app, config_prefix='MONGO2')
```

```
# connect to another MongoDB server altogether
app.config['MONGO3_HOST'] = 'another.host.example.com'
app.config['MONGO3_PORT'] = 27017
app.config['MONGO3_DBNAME'] = 'dbname_three'
mongo3 = PyMongo(app, config_prefix='MONGO3')
```

Some auto-configured settings that you should be aware of are:

`tz_aware`: Flask-PyMongo always uses timezone-aware `datetime` objects. That is, it sets the `tz_aware` parameter to `True` when creating a connection. The timezone of `datetime` objects returned from MongoDB will always be UTC.

`safe`: Flask-PyMongo sets "safe" mode by default, which causes `save()`, `insert()`, `update()`, and `remove()` to wait for acknowledgement from the server before returning. You may override this on a per-call basis by passing the keyword argument `safe=False` to any of the effected methods.

CHAPTER 4

API

4.1 Constants

`flask_pymongo.`**`ASCENDING`** **= 1**

Ascending sort order.

`flask_pymongo.`**`DESCENDING`** **= -1**

Descending sort order.

4.2 Classes

class `flask_pymongo.`**`PyMongo`**(*app=None*, *config_prefix='MONGO'*)

Automatically connects to MongoDB using parameters defined in Flask configuration.

`cx`

The automatically created `Connection` or `ReplicaSetConnection` object.

`db`

The automatically created `Database` object corresponding to the provided `MONGO_DBNAME` configuration parameter.

`init_app`(*app*, *config_prefix='MONGO'*)

Initialize the *app* for use with this `PyMongo`. This is called automatically if *app* is passed to `__init__()`.

The app is configured according to the configuration variables `PREFIX_HOST`, `PREFIX_PORT`, `PREFIX_DBNAME`, `PREFIX_AUTO_START_REQUEST`, `PREFIX_REPLICA_SET`, `PREFIX_READ_PREFERENCE`, `PREFIX_USERNAME`, `PREFIX_PASSWORD`, and `PREFIX_URI` where "PREFIX" defaults to "MONGO". If `PREFIX_URL` is set, it is assumed to have all appropriate configurations, and the other keys are overwritten using their values as present in the URI.

Parameters

- **app** (*flask.Flask*) – the application to configure for use with this `PyMongo`
- **config_prefix** (*str*) – determines the set of configuration variables used to configure this `PyMongo`

save_file (*filename*, *fileobj*, *base='fs'*, *content_type=None*)
Save the file-like object to GridFS using the given filename. Returns `None`.

```
@app.route('/uploads/<path:filename>', methods=['POST'])
def save_upload(filename):
    mongo.save_file(filename, request.files['file'])
    return redirect(url_for('get_upload', filename=filename))
```

Parameters

- **filename** (*str*) – the filename of the file to return
- **fileobj** (*file*) – the file-like object to save
- **base** (*str*) – base the base name of the GridFS collections to use
- **content_type** (*str*) – the MIME content-type of the file. If `None`, the content-type is guessed from the filename using `guess_type()`

send_file (*filename*, *base='fs'*, *version=-1*, *cache_for=31536000*)
Return an instance of the `response_class` containing the named file, and implement conditional GET semantics (using `make_conditional()`).

```
@app.route('/uploads/<path:filename>')
def get_upload(filename):
    return mongo.send_file(filename)
```

Parameters

- **filename** (*str*) – the filename of the file to return
- **base** (*str*) – the base name of the GridFS collections to use
- **version** (*bool*) – if positive, return the Nth revision of the file identified by filename; if negative, return the Nth most recent revision. If no such version exists, return with HTTP status 404.
- **cache_for** (*int*) – number of seconds that browsers should be instructed to cache responses

class `flask_pymongo.wrappers.`**Collection** (*database*, *name*, *create=False*, ***kwargs*)
Custom sub-class of `pymongo.collection.Collection` which adds Flask-specific helper methods.

find_one_or_404 (**args*, ***kwargs*)
Find and return a single document, or raise a 404 Not Found exception if no document matches the query spec. See `find_one()` for details.

```
@app.route('/user/<username>')
def user_profile(username):
    user = mongo.db.users.find_one_or_404({'_id': username})
    return render_template('user.html',
        user=user)
```

4.3 Wrappers

These classes exist solely in order to make expressions such as `mongo.db.foo.bar` evaluate to a `Collection` instance instead of a `pymongo.collection.Collection` instance. They are documented here solely for completeness.

class `flask_pymongo.wrappers.`**`MongoClient`**(*host=None*, *port=None*, *max_pool_size=100*, *document_class=<type 'dict'>*, *tz_aware=False*, *_connect=True*, ***kwargs*)
Returns instances of `flask_pymongo.wrappers.Database` instead of `pymongo.database.Database` when accessed with dot notation.

class `flask_pymongo.wrappers.`**`MongoReplicaSetClient`**(*hosts_or_uri=None*, *max_pool_size=100*, *document_class=<type 'dict'>*, *tz_aware=False*, *_connect=True*, ***kwargs*)
Returns instances of `flask_pymongo.wrappers.Database` instead of `pymongo.database.Database` when accessed with dot notation.

class `flask_pymongo.wrappers.`**`Database`**(*connection*, *name*)
Returns instances of `flask_pymongo.wrappers.Collection` instead of `pymongo.collection.Collection` when accessed with dot notation.

4.4 History and Contributors

Changes:

- 0.3.0: July 4, 2013
  - This is a minor version bump which introduces backwards breaking changes! Please read these change notes carefully.
  - Removed read preference constants from Flask-PyMongo; to set a read preference, use the string name or import contants directly from `pymongo.read_preferences.ReadPreference`.
  - #22 (partial) Add support for `MONGO_SOCKET_TIMEOUT_MS` and `MONGO_CONNECT_TIMEOUT_MS` options (ultrabug).
  - #27 (partial) Make Flask-PyMongo compatible with Python 3 (Vizzy).
- 0.2.1: December 22, 2012
  - #19 Added `MONGO_DOCUMENT_CLASS` config option (jeverling).
- 0.2.0: December 15, 2012
  - This is a minor version bump which may introduce backwards breaking changes! Please read these change notes carefully.
  - #17 Now using PyMongo 2.4's `MongoClient` and `MongoReplicaSetClient` objects instead of `Connection` and `ReplicaSetConnection` classes (tang0th).
  - #17 Now requiring at least PyMongo version 2.4 (tang0th).
  - #17 The wrapper class `flask_pymongo.wrappers.Connection` is renamed to `flask_pymongo.wrappers.MongoClient` (tang0th).
  - #17 The wrapper class `flask_pymongo.wrappers.ReplicaSetConnection` is renamed to `flask_pymongo.wrappers.MongoReplicaSetClient` (tang0th).
  - #18 `MONGO_AUTO_START_REQUEST` now defaults to `False` when connecting using a URI.
- 0.1.4: December 15, 2012
  - #15 Added support for `MONGO_MAX_POOL_SIZE` (Fabrice Aneche)
- 0.1.3: September 22, 2012

  - Added support for configuration from MongoDB URI.
- 0.1.2: June 18, 2012
  - Updated wiki example application
  - #14 Added examples and docs to PyPI package.
- 0.1.1: May 26, 2012
  - Added support for PyMongo 2.2's "auto start request" feature, by way of the `MONGO_AUTO_START_REQUEST` configuration flag.
  - #13 Added BSONObjectIdConverter (Christoph Herr)
  - #12 Corrected documentation typo (Thor Adam)
- 0.1: December 21, 2011
  - Initial Release

Contributors:

- jeverling
- tang0th
- Fabrice Aneche
- Thor Adam
- Christoph Herr