Flask-Whooshee Documentation

Release 0.4.1

Bohuslav "Slavek" Kabrda

Aug 01, 2017

Contents

Customizable Flask-SQLAlchemy - Whoosh Integration

Flask-Whooshee provides more advanced Whoosh integration into Flask. Its main power is in the ability to index and search joined queries.

CHAPTER 1

Installation

Install the extension with one of the following commands:

```
$ easy_install Flask-Whooshee
```

or alternatively if you have pip installed:

```
$ pip install Flask-Whooshee
```

CHAPTER 2

Set Up

Flask-Whooshee supports two different methods of setting up the extension. You can either initialize it directly, thus binding it to a specific application instance:

```
app = Flask(__name__)
whooshee = Whooshee(app)
```

and the second is to use the factory pattern which will allow you to configure whooshee at a later point:

```
whooshee = Whooshee()
def create_app():
    app = Flask(__name__)
    whooshee.init_app(app)
    return app
```

Following configuration options are available:

Option	Description
WHOOSHEE_DIR	The path for the whoosh index (defaults to **whooshee**)
WHOOSHEE_MIN_STRING_LEN	Min. characters for the search string (defaults to **3**)
WHOOSHEE_WRITER_TIMEOUT	How long should whoosh try to acquire write lock? (defaults to **2**)
WHOOSHEE_MEMORY_STORAGE	Use the memory as storage. Useful for tests. (defaults to **False**)

New in version 0.4.0: It's now possible to register whoosheers before calling `init_app`.

For example:

```
db = SQLAlchemy()
# we don't pass app, but call init_app in create_app below
whooshee = Whooshee()

def create_app():
    app = Flask(__name__)

    db.init_app(app)
    whooshee.init_app(app)
```

```
    return app

@whooshee.register_model('text')
class Article(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.UnicodeText)
```

Changed in version 0.4.0: The `init_app` function now works properly with multiple Flask application objects.

New in version development: Added `WHOOSHEE_MEMORY_STORAGE` config variable.

CHAPTER 3

How It Works

Flask-Whooshee is based on so-called whoosheers. These represent Whoosh indexes and are responsible for indexing new/updated fields. There are two types of whoosheers. The simple *model whoosheers*, that indexes fields from just one index:

```
@whooshee.register_model('title', 'content')
class Entry(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String)
    content = db.Column(db.Text)
```

This will make the columns `title` and `content` searchable.

For more advanced use cases you can create your own custom whoosheers which will allow you to create indexes and search across multiple tables. Create them like this:

```
from flask_sqlalchemy import SQLAlchemy
from flask_whooshee import Whooshee, AbstractWhoosheer

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String)

# you can still keep the model whoosheer
@whooshee.register_model('title', 'content')
class Entry(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String)
    content = db.Column(db.Text)
    user = db.relationship(User, backref=db.backref('entries'))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

Now we create a custom whoosheer class which we will use to update the `User` and `Entry` indexes:

```
@whooshee.register_whoosheer
class EntryUserWhoosheer(AbstractWhoosheer):
```

```
    # create schema, the unique attribute must be in form of
    # model.__name__.lower() + '_' + 'id' (name of model primary key)
    schema = whoosh.fields.Schema(
        entry_id = whoosh.fields.NUMERIC(stored=True, unique=True),
        user_id = whoosh.fields.NUMERIC(stored=True),
        username = whoosh.fields.TEXT(),
        title = whoosh.fields.TEXT(),
        content = whoosh.fields.TEXT())

    # don't forget to list the included models
    models = [Entry, User]

    # create insert_* and update_* methods for all models
    # if you have camel case names like FooBar,
    # just lowercase them: insert_foobar, update_foobar
    @classmethod
    def update_user(cls, writer, user):
        pass  # TODO: update all users entries

    @classmethod
    def update_entry(cls, writer, entry):
        writer.update_document(entry_id=entry.id,
                               user_id=entry.user.id,
                               username=entry.user.name,
                               title=entry.title,
                               content=entry.content)

    @classmethod
    def insert_user(cls, writer, user):
        pass  # nothing, user doesn't have entries yet

    @classmethod
    def insert_entry(cls, writer, entry):
        writer.add_document(entry_id=entry.id,
                            user_id=entry.user.id,
                            username=entry.user.name,
                            title=entry.title,
                            content=entry.content)

    @classmethod
    def delete_user(cls, writer, user):
        pass  # TODO: delete all users entries

    @classmethod
    def delete_entry(cls, writer, entry):
        writer.delete_by_term('entry_id', entry.id)
```

To register all whoosheers in one place, just call the *Whooshee.register_whoosheer()* method like this:

```
whooshee.register_whoosheer(EntryUserWhoosheer)
```

CHAPTER 4

Writing Queries

After the whoosheers have been registered, you can leverage the query_class provided by Whooshee and write queries like this:

```
# will find entries whose title or content matches 'chuck norris'
Entry.query.\
    whooshee_search('chuck norris').\
    order_by(Entry.id.desc()).\
    all()
```

You can even join queries and search them like this (using the advanced example from above):

```
# will find any joined entry<->query,
# whose User.name or Entry.title or Entry.content
# matches 'chuck norris'
Entry.query.join(User).\
    whooshee_search('chuck norris').\
    order_by(Entry.id.desc()).\
    all()
```

The whoosheer that is used for searching is, by default, selected based on the models participating in the query. This set of models is compared against the value of *models* attribute of each registered whoosheer and the one with an exact match is selected. You can override this behaviour by explicitly passing whoosheer that should be used for searching to the *WhoosheeQuery.whooshee_search()* method. This is useful if you don't want to join on all the models that form the search index. For example:

```
Entry.query.\
    whooshee_search('chuck norris', whoosheer=EntryUserWhoosheer).\
    order_by(Entry.id.desc()).\
    all()
```

If there exists an entry of a user called 'chuck norris', this entry will be found because the custom whoosheer, that contains field *username*, will be used. But without the whoosheer option, that entry won't be found (unless it has 'chuck norris' in content or title) because the model whoosheer will be used.

Search Result Ordering

By default only first 10 (for optimization reasons) search results are sorted by relevance. You can modify this behaviour by explicitly setting the value of *order_by_relevance* parameter of the *whooshee_search* method.

Return all search results sorted by relevance (only Chuck Norris can do this):

```
Entry.query.join(User).\
    whooshee_search('chuck norris', order_by_relevance=-1).\
    all()
```

Return first 25 rows sorted by their relevance:

```
Entry.query.join(User).\
    whooshee_search('chuck norris', order_by_relevance=25).\
    all()
```

Disable sorting altogether:

```
Entry.query.join(User).\
    whooshee_search('chuck norris', order_by_relevance=0).\
    all()
```

CHAPTER 5

Reindexing

If you lost your search index data and you need to recreate it or you are introducing Flask-Whooshee to an existing application and need to index already existing data, you can use the *Whooshee.reindex()* method to reindex your data:

```
from flask_whooshee import Whooshee
whooshee = Whooshee(app)
whooshee.reindex()
```

New in version v0.0.9.

CHAPTER 6

API

class flask_whooshee.**Whooshee**(*app=None*)

A top level class that allows to register whoosheers and adds an on_commit hook to SQLAlchemy.

There are two different methods on setting up Flask-Whooshee for your application. The first one would be to initialize it directly, thus binding it to a specific application instance:

```
app = Flask(__name__)
whooshee = Whooshee(app)
```

and the second is to use the factory pattern which will allow you to configure whooshee at a later point:

```
whooshee = Whooshee()
def create_app():
    app = Flask(__name__)
    whooshee.init_app(app)
    return app
```

Please note that Whooshee will replace the Flask-SQLAlchemy's *db.Model.query_class* with a whoosh specific query class, *WhoosheeQuery* which will enable full-text search on the registered model.

classmethod camel_to_snake(*s*)

Constructs nice dir name from class name, e.g. FooBar => foo_bar.

Parameters **s** – The string which should be converted to snake_case.

classmethod create_index(*app*, *wh*)

Creates and opens an index for the given whoosheer and app. If the index already exists, it just opens it, otherwise it creates it first.

Parameters

- **app** – The application instance.
- **wh** – The whoosheer instance for which a index should be created.

classmethod get_or_create_index(*app*, *wh*)

Gets a previously cached index or creates a new one for the given app and whoosheer.

Parameters

- **app** – The application instance.
- **wh** – The whoosheer instance for which the index should be retrieved or created.

init_app(*app*)
Initialize the extension. It will create the *index_path_root* directory upon initalization but it will **not** create the index. Please use *reindex()* for this.

Parameters **app** – The application instance for which the extension should be initialized.

on_commit(*changes*)
Method that gets called when a model is changed. This serves to do the actual index writing.

register_model(**index_fields*, ***kw*)
Registers a single model for fulltext search. This basically creates a simple Whoosheer for the model and calls *register_whoosheer()* on it.

register_whoosheer(*wh*)
This will register the given whoosher on *whoosheers*, create the neccessary SQLAlchemy event listeners, replace the *query_class* with our own query class which will provide the search functionality and store the app on the whoosheer, so that we can always work with that.

Parameters **wh** – The whoosher which should be registered.

reindex()
Reindex all data

This method retrieves all the data from the registered models and calls the `update_<model>()` function for every instance of such model.

class flask_whooshee.**WhoosheeQuery**(*entities*, *session=None*)
An override for SQLAlchemy query used to do fulltext search.

whooshee_search(*search_string*, *group=<class 'whoosh.qparser.syntax.OrGroup'>*, *whoosheer=None*, *match_substrings=True*, *limit=None*, *order_by_relevance=10*)
Do a fulltext search on the query. Returns a query filtered with results of the fulltext search.

Parameters

- **search_string** – The string to search for.
- **group** – The whoosh group to use for searching. Defaults to `whoosh.qparser.OrGroup` which searches for all words in all columns.
- **match_substrings** – `True` if you want to match substrings, `False` otherwise
- **limit** – The number of the top records to be returned. Defaults to `None` and returns all records.

class flask_whooshee.**AbstractWhoosheer**
A superclass for all whoosheers.

Whoosheer is basically a unit of fulltext search. It represents either of:

•One table, in which case all given fields of the model is searched.

•More tables, in which case all given fields of all the tables are searched.

classmethod **prep_search_string**(*search_string*, *match_substrings*)
Prepares search string as a proper whoosh search string.

Parameters

- **search_string** – The search string which should be prepared.
- **match_substrings** – `True` if you want to match substrings, `False` otherwise.

classmethod search(*search_string*, *values_of=''*, *group=<class 'whoosh.qparser.syntax.OrGroup'>*, *match_substrings=True*, *limit=None*)

Searches the fields for given search_string. Returns the found records if 'values_of' is left empty, else the values of the given columns.

Parameters

- **search_string** – The string to search for.
- **values_of** – If given, the method will not return the whole records, but only values of given column. Defaults to returning whole records.
- **group** – The whoosh group to use for searching. Defaults to `whoosh.qparser.OrGroup` which searches for all words in all columns.
- **match_substrings** – `True` if you want to match substrings, `False` otherwise.
- **limit** – The number of the top records to be returned. Defaults to `None` and returns all records.

CHAPTER 7

Changelog

0.4.1

- SQLAlchemy's aliased entities are now recognized by `whooshee_search`.
- Added support for `RamStorage`. Can be enabled by setting `WHOOSHEE_MEMORY_STORAGE` to `True`.

0.4.0

- `init_app` now properly registers multiple Flask applications.
- It's now possible to register whoosheers before calling `init_app`.

CHAPTER 8

Additional Information

License

```
Copyright (c) 2016, Slavek Kabrda and individual contributors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of the <organization> nor the
      names of its contributors may be used to endorse or promote products
      derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

- search

Python Module Index

f

Index