Flask-Whooshee Documentation

Release 0.7.0

Bohuslav "Slavek" Kabrda

Apr 16, 2023

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**)
`WHOOSHEE_ENABLE_INDEXING`	Specify whether or not to actually do any operations with the Whoosh index (defaults to **True**).

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():
```

(continues on next page)

(continued from previous page)

```
    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&nbsp;norris' in content or title) because the model whoosheer will be used.

4.1 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

Manual index updates

If your application depends heavily on write operations and there are lots of concurrent search-index updates, you might want opt for a cron job invoking `whooshee.reindex()` periodically instead of employing the default index auto-updating mechanism.

This is especially recommended, if you encouter `LockError` raised by python-whoosh module and setting `WHOOSHEE_WRITER_TIMEOUT` to a higher value (default is 2) does not help.

To disable index auto updating, set `auto_update` class property of a Whoosheer to `False`:

```
@whooshee.register_whoosheer
class NewEntryUserWhoosheer(EntryUserWhoosheer):
    auto_update = False
```

New in version v0.5.0.

CHAPTER 7

Enabling/disabling indexing

By setting the configuration option `WHOOSHEE_ENABLE_INDEXING` to `False`, you can turn of any operations with the Whoosh index (creating, updating and deleting entries). This can be useful e.g. when mass-importing large amounts of entries for testing purposes, but you don't actually need the whooshee fulltext search for these tests to pass.

Note, that once the `Whooshee(app)` call is done, the value of this configuration setting can only be changed by using `app.extensions['whooshee']['enable_indexing'] = <value>` (where `value` is either `True` or `False`).

New in version v0.5.0.

CHAPTER 8

API

CHAPTER 9

Changelog

9.1 0.8.2

- Fixed compatibility with the latest flexmock
- Switched from nose to pytest

9.2 0.8.1

- SQLAlchemy 1.4+ compat fix included (missing *_join_entities* attribute)

9.3 0.7.0

- Dropped support for Python 3.3 and 3.4, added support for Python 3.7.
- Added support for PK's of type BigInteger. Thanks to Andrew Henry for contributing the fix.
- Index writers now properly cancel Whoosh transaction and release index lock on exceptions.

9.4 0.6.0

- Fixed searching for unicode strings in Python 2. Thanks to Grey Li for contributing the fix.
- Added support for registering models with non-int primary keys.

9.5 0.5.0

- Added configuration option `WHOOSHEE_ENABLE_INDEXING` that allows turning off indexing (useful when importing large test sets that don't require indexing in order to actually execute the tests).
- Fixed whooshee search for *str* objects containing unicode characters on Python 2.7.
- Python 2.6 is no longer officially supported, although flask-whooshee should keep working on it.
- Added the option to do manual index updates (through `AbstractWhoosheer.auto_update` attribute).

9.6 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`.

9.7 0.4.0

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

CHAPTER 10

Additional Information

10.1 License

- search

Python Module Index

Index