tornado-sqlalchemy Documentation

Release 0.7.0

Siddhant Goel

Jan 05, 2020

Contents

tornado-sqlalchemy is a Python library aimed at providing support for using the SQLAlchemy database toolkit in tornado web applications.

CHAPTER 1

Installation

```
$ pip install tornado-sqlalchemy
```

CHAPTER 2

User Guide

2.1 Motivation

`tornado-sqlalchemy` handles the following problems/use-cases:

- **Boilerplate** - Tornado does not bundle code to handle database connections. So developer building apps using Tornado end up writing their own code - code to establish database connections, initialize engines, get/teardown sessions, and so on.
- **Database migrations** - Since you're using SQLAlchemy, you're probably also using alembic for database migrations. This again brings us to the point about boilerplate. If you're currently using SQLAlchemy with Tornado and have migrations set up using alembic, you likely have custom code written somewhere.
- **Asynchronous query execution** - ORMs are poorly suited for explicit asynchronous programming. You don't know what property access or what method call would end up hitting the database. In such situations, it's a good idea to decide on *exactly what* you want to execute asynchronously.

The intention here is to have answers to all three of these in a standardized library which can act as a central place for all the bugs features, and hopefully can establish best practices.

2.2 Quickstart

First, construct a `SQLAlchemy` object and pass it to your `tornado.web.Application`.

```
from tornado.web import Application
from tornado_sqlalchemy import SQLAlchemy

from my_app.handlers import IndexHandler

app = Application(
    ((r'/', IndexHandler),),
    db=SQLAlchemy(database_url)
 )
```

Next, when defining database models, make sure that your SQLAlchemy models are inheriting from `tornado_sqlalchemy.SQLAlchemy.Model`.

```
from sqlalchemy import Column, BigInteger, String
from tornado_sqlalchemy import SQLAlchemy

db = SQLAlchemy(url=database_url)

class User(db.Model):
    id = Column(BigInteger, primary_key=True)
    username = Column(String(255), unique=True)
```

Finally, add `SessionMixin` to your request handlers, which makes the `make_session` function available in the HTTP handler functions defined in those request handlers.

As a convenience, `SessionMixin` also provides a `self.session` property, which (lazily) builds and returns a new session object. This session is then automatically closed when the request is finished.

```
from tornado_sqlalchemy import SessionMixin

class SomeRequestHandler(SessionMixin, RequestHandler):
    def get(self):
        with self.make_session() as session:
            count = session.query(User).count()

        # alternatively,
        count = self.session.query(User).count()

        self.write('{} users so far!'.format(count))
```

To run database queries in the background, use the `as_future` function to wrap the SQLAlchemy Query into a Future object, which you can `await` on or `yield` to get the result.

```
from tornado.gen import coroutine
from tornado_sqlalchemy import SessionMixin, as_future

class OldCoroutineRequestHandler(SessionMixin, RequestHandler):
    @coroutine
    def get(self):
        with self.make_session() as session:
            count = yield as_future(session.query(User).count)

        self.write('{} users so far!'.format(count))

class NativeCoroutineRequestHandler(SessionMixin, RequestHandler):
    async def get(self):
        with self.make_session() as session:
            count = await as_future(session.query(User).count)

        self.write('{} users so far!'.format(count))
```

For a complete example, please refer to *examples/basic.py*.

2.3 Multiple Databases

The `SQLAlchemy` constructor supports multiple database URLs, using SQLAlchemy "`binds`".

The following example specifies three database connections, with `database_url` as the default, and `foo`/`bar` being the other two connections.

```
from tornado.web import Application
from tornado_sqlalchemy import SQLAlchemy

from my_app.handlers import IndexHandler

app = Application(
    ((r'/', IndexHandler),),
    db=SQLAlchemy(
        database_url, binds={'foo': foo_url, 'bar': bar_url}
    )
)
```

Modify your model definitions with a `__bind_key__` parameter.

```
from sqlalchemy import BigInteger, Column, String
from tornado_sqlalchemy import SQLAlchemy

db = SQLAlchemy(url=database_url, binds={'foo': foo_url, 'bar': bar_url})

class Foo(db.Model):
   __bind_key__ = 'foo'
   __tablename__ = 'foo'

   id = Column(BigInteger, primary_key=True)

class Bar(db.Model):
   __bind_key__ = 'bar'
   __tablename__ = 'bar'

   id = Column(BigInteger, primary_key=True)
```

The request handlers don't need to be modified and can continue working normally. After this piece of configuration has been done, SQLAlchemy takes care of routing the connection to the correct database according to what's being queried.

For a complete example, please refer to *examples/multiple-databases.py*.

2.4 Migrations (using Alembic)

Database migrations are supported using Alembic.

The one piece of configuration that Alembic expects to auto-generate migrations is the `MetaData` object that your app is using. This is provided by the `db.metadata` property.

```
# env.py

from tornado_sqlalchemy import SQLAlchemy

db = SQLAlchemy(database_url)

target_metadata = db.metadata
```

Other than that, the normal Alembic configuration instructions apply.