alchimia Documentation

Release

Alex Gaynor and David Reid

Mar 18, 2018

Contents

1 Getting started **3**

2 Get the code **5**

3 Contents **7**

3.1 DDL 7

3.2 API Reference 7

3.3 Limitations 10

3.4 Contributing 10

`alchimia` lets you use most of the SQLAlchemy-core API with Twisted, it does not allow you to use the ORM.

CHAPTER 1

Getting started

```
from alchimia import wrap_engine

from sqlalchemy import (
    create_engine, MetaData, Table, Column, Integer, String
)
from sqlalchemy.schema import CreateTable

from twisted.internet.defer import inlineCallbacks
from twisted.internet.task import react


@inlineCallbacks
def main(reactor):
    engine = wrap_engine(reactor, create_engine("sqlite://"))

    metadata = MetaData()
    users = Table("users", metadata,
        Column("id", Integer(), primary_key=True),
        Column("name", String()),
    )

    # Create the table
    yield engine.execute(CreateTable(users))

    # Insert some users
    yield engine.execute(users.insert().values(name="Jeremy Goodwin"))
    yield engine.execute(users.insert().values(name="Natalie Hurley"))
    yield engine.execute(users.insert().values(name="Dan Rydell"))
    yield engine.execute(users.insert().values(name="Casey McCall"))
    yield engine.execute(users.insert().values(name="Dana Whitaker"))

    result = yield engine.execute(users.select(users.c.name.startswith("D")))
    d_users = yield result.fetchall()
    # Print out the users
    for user in d_users:
```

```
        print("Username: %s" % user[users.c.name])
    # Queries that return results should be explicitly closed to
    # release the connection
    result.close()

if __name__ == "__main__":
    react(main, [])
```

CHAPTER 2

Get the code

We're on github. Fork us!

CHAPTER 3

Contents

3.1 DDL

Because of some of the limitations in the SQLAlchemy API, it's not possible to create tables using `sqlalchemy.schema.Table.create()` or `sqlalchemy.schema.MetaData.create_all()`. Luckily, SQLAlchemy provides an API that still makes it possible to create tables and perform other DDL operations.

Instead of:

```
users = Table("users", metadata,
    Column("id", Integer(), primary_key=True),
    Column("name", String()),
)

users.create(engine)
```

or

```
metadata.create_all()
```

You can use `sqlalchemy.schema.CreateTable`:

```
from sqlalchemy.schema import CreateTable

d = engine.execute(CreateTable(users))
```

3.2 API Reference

Many of these classes are missing methods from the SQLALchemy API. We encourage you to *file bugs* in those cases.

`alchimia.`**`wrap_engine`**(*reactor*, *engine*, *create_worker*=...)

This returns a *`alchimia.engine.TwistedEngine`*.

The main entry-point to alchimia. To be used like so:

```
from sqlalchemy import create_engine
from alchimia import wrap_engine
from twisted.internet import reactor

underlying_engine = create_engine("sqlite://")
twisted_engine = wrap_engine(reactor, engine)
```

- `reactor` - the Twisted reactor to use with the created `TwistedEngine`.
- `engine` - the underlying `sqlalchemy.engine.Engine`
- **`create_worker` - The object that will coordinate concurrent blocking** work behind the scenes. The default implementation, if nothing is passed, is one which will use a threadpool where each Connection is tied to an individual thread.

  More precisely, this is a callable that is expected to return an object with 2 methods, `do(work)` (expected to call the 0-argument `work` callable in a thread), and `quit()`, expected to stop any future work from occurring. It may be useful to stub out the default threaded implementation for testing purposes.

class alchimia.engine.**TwistedEngine**

Mostly like `sqlalchemy.engine.Engine` except some of the methods return `Deferreds`.

__init__(*pool*, *dialect*, *url*, *reactor=...*, *create_worker=...*)

This constructor is invoked if `TwistedEngine` is created via `create_engine(..., reactor=reactor, strategy=TWISTED_STRATEGY)` rather than called directly. New applications should prefer *alchimia.wrap_engine()*. However, `create_engine` relays its keyword arguments, so the `reactor` and `create_worker` arguments have the same meaning as they do in *alchimia.wrap_engine()*.

classmethod from_sqlalchemy_engine(*reactor*, *engine*, *create_worker=...*)

This is the implementation of `alchimia.wrap_engine`.

connect()

Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with a *TwistedConnection*.

execute(**args*, ***kwargs*)

Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with a *TwistedResultProxy*.

has_table(*table_name*, *schema=None*)

Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with the result.

table_names(*schema=None*, *connection=None*)

Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with the result.

class alchimia.engine.**TwistedConnection**

Mostly like `sqlalchemy.engine.Connection` except some of the methods return `Deferreds`.

execute(**args*, ***kwargs*)

Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with a *TwistedResultProxy*.

close()

Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires when the connection has been closed.

closed
Like the SQLAlchemy attribute of the same name.

begin()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with a *TwistedTransaction*.

begin_nested()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with a *TwistedTransaction*.

in_transaction()
Like the SQLAlchemy method of the same name.

class alchimia.engine.**TwistedTransaction**
Mostly like `sqlalchemy.engine.Transaction` except some of the methods return `Deferreds`.

commit()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires when the transaction has been committed.

rollback()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires when the transaction has been rolled back.

closed()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires when the transaction has been closed.

class alchimia.engine.**TwistedResultProxy**
Mostly like `sqlalchemy.engine.ResultProxy` except some of the methods return `Deferreds`.

fetchone()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with a row.

fetchall()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with a list of rows.

scalar()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with the scalar value.

first()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with the scalar value.

keys()
Like the SQLAlchemy method of the same name, except returns a `Deferred` which fires with the scalar value.

close()
Like the SQLAlchemy method of the same name, it releases the resources used and releases the underlying DB connection.

returns_rows
Like the SQLAlchemy attribute of the same name.

rowcount
Like the SQLAlchemy attribute of the same name.

inserted_primary_key
Like the SQLAlchemy attribute of the same name.

3.3 Limitations

There are two reasons stuff isn't implemented in `alchimia`.

First, because we haven't gotten there yet. For these items you should *file bugs or send pull requests*.

Second, some items can't be implemented because of limitations in SQLAlchemy, there's almost always a workaround though.

- *Table creation*

3.4 Contributing

As an open source project, Alchimia welcomes contributions of many forms.

Examples of contributions include:

- Code patches
- Documentation improvements
- Bug reports and patch reviews

We welcome pull requests and tickets on github!

Index

Symbols

__init__() (alchimia.engine.TwistedEngine method), 8

B

begin() (alchimia.engine.TwistedConnection method), 9
begin_nested() (alchimia.engine.TwistedConnection method), 9

C

close() (alchimia.engine.TwistedConnection method), 8
close() (alchimia.engine.TwistedResultProxy method), 9
closed (alchimia.engine.TwistedConnection attribute), 8
closed() (alchimia.engine.TwistedTransaction method), 9
commit() (alchimia.engine.TwistedTransaction method), 9
connect() (alchimia.engine.TwistedEngine method), 8

E

execute() (alchimia.engine.TwistedConnection method), 8
execute() (alchimia.engine.TwistedEngine method), 8

F

fetchall() (alchimia.engine.TwistedResultProxy method), 9
fetchone() (alchimia.engine.TwistedResultProxy method), 9
first() (alchimia.engine.TwistedResultProxy method), 9
from_sqlalchemy_engine() (alchimia.engine.TwistedEngine class method), 8

H

has_table() (alchimia.engine.TwistedEngine method), 8

I

in_transaction() (alchimia.engine.TwistedConnection method), 9
inserted_primary_key (alchimia.engine.TwistedResultProxy attribute), 9

K

keys() (alchimia.engine.TwistedResultProxy method), 9

R

returns_rows (alchimia.engine.TwistedResultProxy attribute), 9
rollback() (alchimia.engine.TwistedTransaction method), 9
rowcount (alchimia.engine.TwistedResultProxy attribute), 9

S

scalar() (alchimia.engine.TwistedResultProxy method), 9

T

table_names() (alchimia.engine.TwistedEngine method), 8
TwistedConnection (class in alchimia.engine), 8
TwistedEngine (class in alchimia.engine), 8
TwistedResultProxy (class in alchimia.engine), 9
TwistedTransaction (class in alchimia.engine), 9

W

wrap_engine() (in module alchimia), 7