Flask SQLAlchemy

Quik Reference

Ferreira Juan David

5 de febrero de 2021

Resumen

Si pensamos en utilizar una sola aplicación, podemos saltar este capítulo. Simplemente pasemos nuestra aplicación al constructor `SQLAlchemy` y estará listo. Sin embargo, si deseamos utilizar más de una aplicación o crear la aplicación dinámicamente en una función, debemos seguir leyendo.

Introducción a los contextos

Si definimos la aplicación en una función, pero el objeto `SQLAlchemy` globalmente, *¿cómo aprende este último sobre el primero?*. La respuesta es la función `init_app()`:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

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

Lo que hace es preparar la aplicación para trabajar `SQLAlchemy`. Sin embargo, eso no vincula ahora el objeto `SQLAlchemy` a su aplicación. ¿Por qué no hace eso? Porque puede haber más de una aplicación creada.

Entonces, ¿cómo `SQLAlchemy` se entera de su aplicación? Tendrá que configurar un contexto de aplicación. Si está trabajando dentro de una función de vista *Flask* o un comando CLI, eso sucede automáticamente. Sin embargo, si está trabajando dentro del shell interactivo, tendrá que hacerlo usted mismo (consulte Creación de un contexto de aplicación).

Si intentamos realizar operaciones de base de datos fuera del contexto de una aplicación, verá el siguiente error:
`No application found. Either work inside a view function or push an application context.`
En pocas palabras, haz algo como esto:

```
>>> from yourapp import create_app
>>> app = create_app()
>>> app.app_context().push()
```

Alternativamente, use la declaración with para encargarse de `setup` y `teardown`:

```
def my_function():
    with app.app_context():
        user = db.User(...)
        db.session.add(user)
        db.session.commit()
```

Algunas funciones dentro de `Flask-SQLAlchemy` también aceptan opcionalmente la aplicación para operar:

```
>>> from yourapp import db, create_app
>>> db.create_all(app=create_app())
```

Configuración

Existen los siguientes valores de configuración para `Flask-SQLAlchemy`. `Flask-SQLAlchemy` carga estos valores de su configuración principal de Flask que se puede completar de varias formas. Tenga en cuenta que algunos de ellos no se pueden modificar después de que se creó el motor, así que asegúrese de configurarlos lo antes posible y de no modificarlos en tiempo de ejecución.

Claves de configuración

Una lista de claves de configuración que la extensión comprende actualmente

`SQLALCHEMY_DATABASE_URI`: El `URI` de la base de datos que debemos usar para la conexión. Ejemplos:

- `sqlite:////tmp/test.db`
- `mysql://username:password@server/db`

`SQLALCHEMY_BINDS`: Un diccionario que asigna claves de enlace a `URI` de conexión `SQLAlchemy`. Para obtener más información, consulte sobre Varias bases de datos.

`SQLALCHEMY_ECHO`: Si se establece en `True`, `SQLAlchemy` registrará todas las declaraciones emitidas a `stderr` que pueden ser útiles para la depuración.

`SQLALCHEMY_RECORD_QUERIES`: Puede usarse para deshabilitar o habilitar explícitamente la grabación de consultas. La grabación de consultas ocurre automáticamente en modo de depuración o prueba. Consulte `get_debug_queries()` para obtener más información.

`SQLALCHEMY_NATIVE_UNICODE`: Se puede utilizar para deshabilitar explícitamente la compatibilidad con Unicode nativo. Esto es necesario para algunos adaptadores de base de datos (como `PostgreSQL` en algunas versiones de `Ubuntu`) cuando se usa con valores predeterminados de base de datos incorrectos que especifican bases de datos sin codificación.

En desuso a partir de la `v2.4` y se eliminará en la `v3.0`.

`SQLALCHEMY_POOL_SIZE`: El tamaño del grupo de bases de datos. Por defecto es el motor por defecto (normalmente 5).

En desuso a partir de la `v2.4` y se eliminará en la `v3.0`.

SQLALCHEMY_POOL_TIMEOUT: Especifica el tiempo de espera de la conexión en segundos para el grupo.

En desuso a partir de la v2.4 y se eliminará en la v3.0.

SQLALCHEMY_POOL_RECYCLE: Número de segundos después de los cuales una conexión se recicla automáticamente. Esto es necesario para MySQL, que elimina las conexiones después de 8 horas inactivo de forma predeterminada. Tenga en cuenta que Flask-SQLAlchemy lo establece automáticamente en 2 horas si se usa MySQL. Algunos backends pueden usar un valor de tiempo de espera predeterminado diferente. Para obtener más información sobre los tiempos de espera, consulte Tiempos de espera

En desuso a partir de la v2.4 y se eliminará en la v3.0.

SQLALCHEMY_MAX_OVERFLOW: Controla la cantidad de conexiones que se pueden crear después de que el grupo alcanza su tamaño máximo. Cuando esas conexiones adicionales se devuelven al grupo, se desconectan y descartan.

En desuso a partir de la v2.4 y se eliminará en la v3.0.

SQLALCHEMY_TRACK_MODIFICATIONS: Si se establece en True, Flask-SQLAlchemy rastreará las modificaciones de los objetos y emitirá señales. El valor predeterminado es None, que habilita el seguimiento, pero emite una advertencia de que estará deshabilitado de manera predeterminada en el futuro. Esto requiere memoria adicional y debe desactivarse si no es necesario.

SQLALCHEMY_ENGINE_OPTIONS: Un diccionario de argumentos de palabras clave para enviar create_engine(). Consulte también engine_options en SQLAlchemy.

Nuevo en la versión 0.8: Las SQLALCHEMY_NATIVE_UNICODE, SQLALCHEMY_POOL_SIZE, SQLALCHEMY_POOL_TIMEOUT y SQLALCHEMY_POOL_RECYCLE se añadieron claves de configuración.
Nuevo en la versión 0.12: Se *SQLALCHEMY_BINDSagregó* la clave de configuración.
Nuevo en la versión 0.17: El *SQLALCHEMY_MAX_OVERFLOW* se añadió clave de configuración.
Nuevo en la versión 2.0: El *SQLALCHEMY_TRACK_MODIFICATIONS* se añadió clave de configuración.
Modificado en la versión 2.1: *SQLALCHEMY_TRACK_MODIFICATIONS* avisará si no se configura.
Modificado en la versión 2.4:
* SQLALCHEMY_ENGINE_OPTIONS Se agregó la clave de configuración. * Claves obsoletas

- SQLALCHEMY_NATIVE_UNICODE
- SQLALCHEMY_POOL_SIZE
- SQLALCHEMY_POOL_TIMEOUT
- SQLALCHEMY_POOL_RECYCLE
- SQLALCHEMY_MAX_OVERFLOW

Modificado en la versión 2.4.3: obsoleto SQLALCHEMY_COMMIT_ON_TEARDOWN.

Formato de URI de conexión

Para obtener una lista completa de los URI de conexión, consulte la documentación de SQLAlchemy en (Bases de datos compatibles). Aquí se muestran algunas cadenas de conexión comunes.
SQLAlchemy indica el origen de un motor como un URI combinado con argumentos de palabras clave opcionales para especificar opciones para el motor. La forma del URI es:
dialect+driver://username:password@host:port/my_db
Muchas de las partes de la cadena son opcionales. Si no se especifica ningún controlador, se selecciona el predeterminado (asegúrese de no incluirlo +en ese caso).
Postgres: postgresql://scott:tiger@localhost/mydatabase
MySQL: mysql://scott:tiger@localhost/mydatabase
Oracle: oracle://scott:tiger@127.0.0.1:1521/sidname
SQLite (Aquí aplicamos las convenciones según el OS):

- #Unix/Mac (note the four leading slashes)

  sqlite:////absolute/path/to/foo.db

- #Windows (note 3 leading forward slashes and escapes)

  sqlite:///C:\\absolute\\path\\to\\foo.db

- #Windows (alternative using raw string)

  r'sqlite:///C:\absolute\path\to\foo.db

Uso de convenciones de nomenclatura y metadatos personalizados

Opcionalmente, puede construir el objeto SQLAlchemy con un objeto MetaData personalizado . Esto le permite, entre otras cosas, especificar una convención de nomenclatura de restricción personalizada junto con SQLAlchemy 0.9.2 o superior. Hacerlo es importante para lidiar con las migraciones de bases de datos (por ejemplo, usando alambique como se indica aquí . Aquí hay un ejemplo, como lo sugieren los documentos de SQLAlchemy:

```
from sqlalchemy import MetaData
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

convention = {
    "ix": 'ix_%(column_0_label)s',
    "uq": "uq_%(table_name)s_%(column_0_name)s",
    "ck": "ck_%(table_name)s_%(constraint_name)s",
    "fk": "fk_%(table_name)s_%(column_0_name)s_%(referred_table_name)s",
    "pk": "pk_%(table_name)s"
}

metadata = MetaData(naming_convention=convention)
db = SQLAlchemy(app, metadata=metadata)
```

Para obtener más información MetaData, consulte los documentos oficiales al respecto.