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.

API Reference: Configuración

`get_tables_for_bind(bind=None)`: Devuelve una lista de todas las tablas relevantes para un enlace.

`init_app(app)`: Esta devolución de llamada se puede utilizar para inicializar una aplicación para su uso con esta configuración de base de datos. Nunca use una base de datos en el contexto de una aplicación que no se haya inicializado de esa manera o las conexiones se filtrarán.

`make_connector(app=None, bind=None)`: Crea el conector para un estado y enlace determinados.

`metadata`: Los metadatos asociados con `db.Model`.

`make_declarative_base(model, metadata=None)`: Crea la base declarativa de la que heredarán todos los modelos.

Parámetros

- **modelo**: clase de modelo base (o una tupla de clases base) a la que pasar `declarative_base()`. O una clase devuelta `declarative_base`, en cuyo caso no se crea una nueva clase base.
- **metadatos**: `MetaData` instancia para usar, o ninguna para usar el valor predeterminado de `SQLAlchemy`.

`reflect(bind = '__all__',app = None)`: Refleja tablas de la base de datos.

Modificado en la versión 0.12: se agregaron parámetros.

Models

`class flask_sqlalchemy.Model`
Clase base para el modelo base declarativo de `SQLAlchemy`.
Para definir modelos, subclase `db.Model`, no esta clase. Para personalizar `db.Model`, subclase esto y pasarlo como `model_class` a `SQLAlchemy`.

`__bind_key__`: Opcionalmente declara el enlace que se utilizará. `None` se refiere al enlace predeterminado. Para obtener más información, consulte Varias bases de datos con enlaces.

`__tablename__`: El nombre de la tabla en la base de datos. Esto es requerido por `SQLAlchemy`; sin embargo, `Flask-SQLAlchemy` lo configurará automáticamente si un modelo tiene una clave primaria definida. Si `__table__` o `__tablename__` se establece explícitamente, se utilizará en su lugar.

`class flask_sqlalchemy.BaseQuery(entities, session = None )` Subclase `SQLAlchemy Query` con métodos convenientes para realizar consultas en una aplicación web.
Este es el objeto `query` predeterminado que se usa para los modelos y se expone como `Query`. Reemplace la clase de consulta para un modelo individual subclasificando esto y estableciendo `query_class`.

`first_or_404(description=None)`: Busca `first()` pero aborta con 404 si no se encuentra en lugar de regresar `None`.

`get_or_404(ident, description=None)`: Busca `get()` pero aborta con 404 si no se encuentra en lugar de regresar `None`.

`paginate(page=None,per_page=None,error_out=True,max_per_page=None))`: Devuelve artículos `per_page` de la página `page`.

Si `page` o `per_page` son `None`, se recuperarán de la consulta de solicitud. Si `max_per_page` se especifica, `per_page` se limitará a ese valor. Si no hay ninguna solicitud o no están en la consulta, se establecen de forma predeterminada en 1 y 20 respectivamente.

Cuando `error_out` es `True` (predeterminado), las siguientes reglas causarán una respuesta 404:

- No se encuentran elementos y page no es 1.
- page es menor que 1 o per_page es negativo.
- page o per_page no son ints.

Cuando error_out es False, page y el per_page valor predeterminado es 1 y 20 respectivamente.
Devuelve un objeto Pagination.

Sesiones

class flask_sqlalchemy.SignallingSession(db,autocommit=False, autoflush=True, **options)
La sesión de señalización es la sesión predeterminada que usa Flask-SQLAlchemy. Amplía el sistema de sesión predeterminado con selección de enlaces y seguimiento de modificaciones.
Si desea utilizar una sesión diferente, puede anular la función SQLAlchemy.create_session().
Nuevo en la versión 2.0.
Nuevo en la versión 2.1: La liga se añadió opción, que permite que una sesión se unió a una transacción externa.

get_bind(mapper=None, clause=None): Devuelva el motor o la conexión para un modelo o tabla determinados, utilizando el __bind_key__ si está configurado.

Utilities

class flask_sqlalchemy.Pagination(query, page, per_page, total, items)
Del tipo helper class interna devuelta por BaseQuery.paginate(). También puede construirlo a partir de cualquier otro objeto de consulta SQLAlchemy si está trabajando con otras bibliotecas. Además, es posible pasar Ninguno como objeto de consulta, en cuyo caso prev() y next() dejarán de funcionar.
has_next: True si existe una página siguiente.

has_prev: True si existe una página anterior

items=None: los elementos de la página actual

iter_pages(left_edge=2,left_current=2,right_current=5,right_edge=2): Repite los números de página en la paginación. Los cuatro parámetros controlan los umbrales cuántos números deben producirse desde los lados. Los números de página omitidos se representan como None. Así es como podría representar dicha paginación en las plantillas:

```
{% macro render_pagination(pagination, endpoint) %}
  <div class=pagination>
  {%- for page in pagination.iter_pages() %}
    {% if page %}
      {% if page != pagination.page %}
        <a href="{{ url_for(endpoint, page=page) }}">{{ page }}</a>
      {% else %}
        <strong>{{ page }}</strong>
      {% endif %}
    {% else %}
      <span class=ellipsis></span>
    {% endif %}
  {%- endfor %}
  </div>
{% endmacro %}
```

next(error_out=False): Devuelve un objeto Pagination para la página siguiente.

next_num: Número de la página siguiente.

page= None: el número de página actual (*1* indexado).

pages: El número total de páginas.

per_page= None: el número de elementos que se mostrarán en una página.

prev(error_out = False): Devuelve un objeto Pagination de la página anterior.

prev_num: Número de la página anterior.

query=None: el objeto de consulta ilimitado que se utilizó para crear este objeto Pagination.

total=None: el número total de elementos que coinciden con la consulta.

flask_sqlalchemy.get_debug_queries()
En el modo de depuración, Flask-SQLAlchemy registrará todas las consultas SQL enviadas a la base de datos. Esta información está disponible hasta el final de la solicitud lo que permite asegurar fácilmente que el SQL generado es el esperado en errores o en pruebas unitarias. Si no desea habilitar el modo DEBUG para sus pruebas unitarias, también puede habilitar la grabación de consultas estableciendo la variable de configuración 'SQLALCHEMY_RECORD_QUERIES' en True. Esto se habilita automáticamente si Flask está en modo de prueba.
El valor devuelto será una lista de tuplas con nombre con los siguientes atributos: statement, parameters, start_time/end_time, duration y context (ver descripción en la documentación oficial).