Тема 17. Интеграция MongoDB с приложением.

Цель занятия:

Изучить различные подходы к интеграции MongoDB с приложениями.

Учебные вопросы:

1. Подключение к MongoDB из Python.
2. Основные операции с данными (CRUD).
3. Работа с асинхронными операциями.
4. Интеграция с веб-фреймворками Python.

1. Подключение к MongoDB из Python.

Подключение к MongoDB из Python осуществляется с помощью библиотеки pymongo, которая предоставляет инструменты для работы с MongoDB.

Далее представлен пошаговый процесс подключения, выполнения операций с данными и обработки ошибок.

Установка библиотеки pymongo

Для начала необходимо установить библиотеку pymongo. Это можно сделать с помощью менеджера пакетов pip.

```
pip install pymongo
```

Импорт библиотеки и создание подключения

После установки библиотеки вы можете импортировать ее в своем Python-скрипте и установить соединение с MongoDB.

```python
from pymongo import MongoClient


# Создаем подключение к MongoDB
client = MongoClient("mongodb://username:password@localhost:27017/")


# Указываем базу данных, с которой будем работать
db = client.mydatabase


# Проверка подключения
print("Connected to MongoDB")
```

2. Основные операции с данными (CRUD).

Основные операции с данными (CRUD)

После успешного подключения вы можете выполнять операции создания, чтения, обновления и удаления данных (CRUD) в MongoDB.

Создание документа

```
# Выбираем коллекцию
collection = db.users


# Создаем новый документ
new_user = {"name": "Alice", "age": 25}
result = collection.insert_one(new_user)


# Выводим ID созданного документа
print(f"User created with id: {result.inserted_id}")
```

Чтение документов

```python
# Читаем все документы из коллекции
users = collection.find()


# Выводим пользователей
for user in users:
    print(user)


# Чтение одного документа по условию
user = collection.find_one({"name": "Alice"})
print(user)
```

Обновление документа

```
# Обновляем возраст пользователя с именем "Alice"
collection.update_one({"name": "Alice"}, {"$set": {"age": 26}})


# Проверяем обновление
updated_user = collection.find_one({"name": "Alice"})
print(updated_user)
```

Удаление документа

```
# Удаляем пользователя с именем "Alice"
collection.delete_one({"name": "Alice"})
print("User deleted")
```

Обработка ошибок

Обязательно учитывайте обработку ошибок при работе с MongoDB. Например, можно использовать конструкции try и except для обработки исключений.

```
try:
    client = MongoClient("mongodb://username:password@localhost:27017/")
    db = client.mydatabase
    print("Connected to MongoDB")

    # Пример операции
    collection = db.users
    new_user = {"name": "Bob", "age": 30}
    result = collection.insert_one(new_user)
    print(f"User created with id: {result.inserted_id}")

except Exception as e:
    print(f"An error occurred: {e}")
```

Закрытие подключения

Хотя MongoClient автоматически управляет соединениями, вы можете явно закрыть соединение, если это необходимо.

```
client.close()

print("Connection to MongoDB closed")
```

3. Работа с асинхронными операциями.

Асинхронные операции — это способ выполнения задач, при котором программа продолжает работать, не дожидаясь завершения каждой операции.

Это особенно полезно для операций ввода-вывода (**I/O**), таких как сетевые запросы или чтение и запись в базу данных, которые могут занять время.

Асинхронность позволяет избежать блокировки основного потока, повышая производительность и отзывчивость приложений.

При взаимодействии с MongoDB операции могут занимать время, особенно при работе с большими объемами данных или при удаленных подключениях. Асинхронный доступ позволяет избежать блокировки основного потока.

Motor — это популярная асинхронная библиотека Python для взаимодействия с базами данных MongoDB. Она предоставляет удобный интерфейс для работы с MongoDB в средах, где важна высокая производительность и неблокирующий ввод-вывод, таких как веб-серверы на базе Tornado или asyncio.

Основные преимущества Motor:

- Асинхронность: Позволяет выполнять операции с базой данных асинхронно, не блокируя выполнение других задач. Это особенно полезно для долго выполняющихся операций или при работе с большим количеством запросов.
- Интеграция с asyncio: Хорошо интегрируется с модулем asyncio, что позволяет легко создавать асинхронные приложения на Python.
- Полная функциональность: Предоставляет практически все возможности, доступные в синхронном драйвере PyMongo, включая работу с коллекциями, документами, индексами и т.д.
- Высокая производительность: Благодаря асинхронному характеру работы, Motor позволяет достичь высокой производительности при обработке большого количества запросов.

Основные понятия и использование

- MotorClient: Представляет собой соединение с базой данных MongoDB.
- AsyncIOMotorClient: Аналогичен MotorClient, но использует asyncio для асинхронных операций.
- Database: Представляет собой отдельную базу данных.
- Collection: Представляет собой коллекцию документов.
- Cursor: Итератор для результатов запросов.

Установка библиотеки Motor

Чтобы начать работать с motor, установите его через pip:

```
pip install motor
```

Подключение к MongoDB и создание асинхронного клиента.

```python
import asyncio
from motor.motor_asyncio import AsyncIOMotorClient

# Подключение к MongoDB
client = AsyncIOMotorClient("mongodb://localhost:27017")
db = client.mydatabase  # Указываем базу данных
collection = db.books   # Указываем коллекцию

async def test_connection():
    # Проверяем подключение, читая список коллекций
    collections = await db.list_collection_names()
    print(f"Список коллекций: {collections}")

# Запускаем асинхронную функцию
asyncio.run(test_connection())
```

Основные операции с данными (CRUD) в асинхронном режиме. Создание документа (Create)

```python
async def insert_book():
    book = {"title": "Dune", "author": "Frank Herbert", "year": 1965}
    result = await collection.insert_one(book)
    print(f"Книга добавлена с id: {result.inserted_id}")


asyncio.run(insert_book())
```

Чтение документов (Read)

```python
async def find_books():
    # Извлечение всех книг
    async for book in collection.find():
        print(book)

asyncio.run(find_books())
```

Обновление документа (Update)

```python
async def update_book():
    result = await collection.update_one(
        {"title": "Dune"}, {"$set": {"year": 1984}}
    )
    print(f"Модифицировано документов: {result.modified_count}")


asyncio.run(update_book())
```

Удаление документа (Delete)

```python
async def delete_book():
    result = await collection.delete_one({"title": "Dune"})
    print(f"Удалено документов: {result.deleted_count}")


asyncio.run(delete_book())
```

Обработка ошибок в асинхронных операциях

При работе с MongoDB могут возникать различные ошибки (например, ошибки подключения или запросов). Рекомендуется использовать блоки try-except для обработки исключений.

```
async def safe_insert():
    try:
        book = {"title": "The Hobbit", "author": "J.R.R. Tolkien", "year": 1937}
        result = await collection.insert_one(book)
        print(f"Книга добавлена с id: {result.inserted_id}")
    except Exception as e:
        print(f"Ошибка: {e}")

asyncio.run(safe_insert())
```

4. Интеграция с веб-фреймворками Python.

Интеграция с Flask

Flask — легковесный веб-фреймворк, который часто используется для небольших веб-приложений и API.

Установка Flask и pymongo

```
pip install Flask pymongo
```

Пример простого API с Flask и MongoDB:

```python
from flask import Flask, jsonify, request
from pymongo import MongoClient


app = Flask(__name__)
client = MongoClient("mongodb://localhost:27017/")
db = client.mydatabase  # Подключаем базу данных
collection = db.books   # Подключаем коллекцию


@app.route('/books', methods=['GET'])
def get_books():
    books = list(collection.find({}, {"_id": 0}))  # Извлекаем все книги
    return jsonify(books)
```

```python
@app.route('/books', methods=['POST'])
def add_book():
    data = request.json  # Получаем JSON-данные из запроса
    collection.insert_one(data)
    return jsonify({"message": "Book added successfully"}), 201


if __name__ == '__main__':
    app.run(debug=True)
```

Интеграция с Django

Django — мощный фреймворк, который используется для крупных веб-приложений. Для работы с MongoDB в Django используется пакет djongo.

Установка Django и djongo

```
pip install django djongo
```

Настройка подключения к MongoDB в settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'djongo',
        'NAME': 'mydatabase',
        'CLIENT': {
            'host': 'mongodb://localhost:27017',
        }
    }
}
```

Создание модели в Django:

```
from djongo import models


class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=50)
    year = models.IntegerField()

    def __str__(self):
        return self.title
```

Выполните миграции (хотя MongoDB не требует жесткой схемы):

```
python manage.py makemigrations
python manage.py migrate
```

Интеграция с FastAPI

FastAPI — современный и быстрый веб-фреймворк, который поддерживает асинхронные операции и удобен для работы с MongoDB через библиотеку motor.

Установка FastAPI и motor:

```
pip install fastapi motor uvicorn
```

Пример API с FastAPI и MongoDB:

```python
from fastapi import FastAPI, HTTPException
from motor.motor_asyncio import AsyncIOMotorClient


app = FastAPI()
client = AsyncIOMotorClient("mongodb://localhost:27017")
db = client.mydatabase
collection = db.books


@app.get("/books")
async def get_books():
    books = []
    async for book in collection.find({}, {"_id": 0}):
        books.append(book)
    return books
```

```python
@app.post("/books")
async def add_book(book: dict):
    result = await collection.insert_one(book)
    return {"inserted_id": str(result.inserted_id)}


if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Запуск сервера:

```
uvicorn app:app --reload
```

Сравнение Flask, Django и FastAPI

Фреймворк	Когда использовать	Плюсы	Минусы
Flask	Для простых API и небольших приложений	Легковесный и простой в использовании	Не поддерживает ORM по умолчанию
Django	Для крупных и сложных веб-приложений	Мощный, встроенные компоненты и ORM	Более сложен и тяжел
FastAPI	Для высокопроизводительных асинхронных API	Асинхронность, быстрая разработка	Меньше встроенных компонентов

Домашнее задание:

1. Повторить материал лекции.

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com