Лабораторная работа №7: Основы работы с Redis

ФИО: Ляхова Елизавета Олеговна
Группа: РИМ-150950

Цель работы: овладеть базовыми навыками установки, подключения и взаимодействия с Redis в Python. Изучить основные структуры данных Redis и их применение на практике.

Ход работы:

1. Конфигурация Docker с Redis

```
version: '3.8'
services:
  redis:
    image: redis:latest
    container_name: redis
    ports:
      - "6379:6379"
    volumes:
      - redis-data:/data

volumes:
  redis-data:
```

2. Зависимости Python

```
redis==5.0.1
```

3.Основной скрипт работы с Redis

```
import redis

# Подключение к Redis
r = redis.Redis(host='localhost', port=6379, db=0, decode_responses=True)

def test_connection():
    try:
        r.ping()
        print("✅ Успешное подключение к Redis")
```

```python
    except redis.ConnectionError:
        print("❌ Ошибка подключения к Redis")

def demonstrate_strings():
    print("\n=== Работа со строками ===")
    r.set("user:name", "Иван")
    print("Имя пользователя:", r.get("user:name"))

    r.setex("session:123", 10, "active")  # TTL 10 секунд для демо
    print("Сессия:", r.get("session:123"))

    r.set("counter", 0)
    r.incr("counter")
    r.incrby("counter", 5)
    print("Счетчик:", r.get("counter"))

def demonstrate_lists():
    print("\n=== Работа со списками ===")
    r.delete("tasks")  # Очистка перед началом
    r.lpush("tasks", "task1", "task2")
    r.rpush("tasks", "task3", "task4")
    print("Все задачи:", r.lrange("tasks", 0, -1))
    print("Длина списка:", r.llen("tasks"))

def demonstrate_sets():
    print("\n=== Работа с множествами ===")
    r.sadd("tags", "python", "redis", "database")
    r.sadd("languages", "python", "java", "javascript")
    print("Все теги:", r.smembers("tags"))
    print("Пересечение:", r.sinter("tags", "languages"))

def demonstrate_hashes():
```

```
    print("\n=== Работа с хэшами ===")
        r.hset("user:1000", mapping={"name": "Иван", "age": "30", "city":
"Москва"})
    print("Данные пользователя:", r.hgetall("user:1000"))
    print("Имя:", r.hget("user:1000", "name"))

def demonstrate_sorted_sets():
    print("\n=== Работа с упорядоченными множествами ===")
    r.zadd("leaderboard", {"player1": 100, "player2": 200, "player3": 150})
    print("Топ-3 игрока:", r.zrange("leaderboard", 0, 2, withscores=True))
    print("Ранг player1:", r.zrank("leaderboard", "player1"))

def cache_demo():
    print("\n=== Демонстрация кэширования ===")
    # Кэширование пользователя на 5 секунд (для демо)
    user_data = '{"id": 1, "name": "Иван", "email": "ivan@example.com"}'
    r.setex("cached:user:1", 5, user_data)
    print("Данные в кэше:", r.get("cached:user:1"))

    # Очистка кэша
    r.delete("cached:user:1")
    print("После удаления:", r.get("cached:user:1"))

if __name__ == "__main__":
    test_connection()
    demonstrate_strings()
    demonstrate_lists()
    demonstrate_sets()
    demonstrate_hashes()
    demonstrate_sorted_sets()
    cache_demo()
```

4.Модуль кэширования (опционально)

```
import redis
import json

class RedisCache:
    def __init__(self, host='localhost', port=6379, db=0):
        self.client = redis.Redis(
            host=host,
            port=port,
            db=db,
            decode_responses=True
        )

    def cache_user(self, user_id, user_data, ttl=3600):
        """Кэширование данных пользователя на 1 час по умолчанию"""
        key = f"user:{user_id}"
        self.client.setex(key, ttl, json.dumps(user_data))
        return True

    def get_user(self, user_id):
        """Получение пользователя из кэша"""
        key = f"user:{user_id}"
        data = self.client.get(key)
        return json.loads(data) if data else None

    def invalidate_user(self, user_id):
        """Удаление пользователя из кэша"""
        key = f"user:{user_id}"
        return self.client.delete(key)

    def cache_product(self, product_id, product_data, ttl=600):
        """Кэширование данных продукта на 10 минут по умолчанию"""
```

```
        key = f"product:{product_id}"
        self.client.setex(key, ttl, json.dumps(product_data))
        return True

    def update_product_cache(self, product_id, product_data):
        """Обновление кэша продукта"""
        return self.cache_product(product_id, product_data, ttl=600)

# Пример использования
if __name__ == "__main__":
    cache = RedisCache()

    # Кэширование пользователя
    user = {"id": 1, "name": "Иван Иванов", "email": "ivan@example.com"}
    cache.cache_user(1, user)

    # Получение из кэша
    cached_user = cache.get_user(1)
    print("Кэшированный пользователь:", cached_user)
```

5. Конфигурация подключения (опционально)

```
REDIS_CONFIG = {
    'host': 'localhost',
    'port': 6379,
    'db': 0,
    'decode_responses': True,
    'socket_connect_timeout': 5
}

CACHE_TTL = {
    'user': 3600,      # 1 час
    'product': 600,    # 10 минут
    'session': 1800    # 30 минут
```

```
}
```

6. Инструкция по запуску

```
# Лабораторная работа №7: Работа с Redis

## Описание
Демонстрация работы с Redis: подключение, работа с разными типами данных, реализация кэширования.

## Требования
- Docker
- Python 3.8+
- Библиотека redis

## Установка и запуск

1. Клонировать репозиторий
2. Установить зависимости:
  ```bash
  pip install -r requirements.txt
```

7. Запуск Redis

```
C:\Users\lyaho\OneDrive\Рабочий стол\lab7>python -m pip install redis
Collecting redis
  Downloading redis-7.1.0-py3-none-any.whl.metadata (12 kB)
Downloading redis-7.1.0-py3-none-any.whl (354 kB)
Installing collected packages: redis
Successfully installed redis-7.1.0
```

8. Запуск Redis в Docker:

```
C:\Users\lyaho\OneDrive\Рабочий стол\lab7>docker-compose up -d
time="2025-12-07T20:12:52+05:00" level=warning msg="C:\\Users\\lyaho\\OneDrive\\Рабочий стол\\lab7\\docker-compose.yml: the attribute
 `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 1/1
 ✔Container redis  Running                    0.0s
```
```

9. Запуск основного скрипта python main.py

```
C:\Users\lyaho\OneDrive\Рабочий стол\lab7>python main.py
✅ Успешное подключение к Redis

=== Работа со строками ===
Имя пользователя: Иван
Сессия: active
Счетчик: 6

=== Работа со списками ===
Все задачи: ['task2', 'task1', 'task3', 'task4']
Длина списка: 4

=== Работа с множествами ===
Все теги: {'database', 'python', 'redis'}
Пересечение: {'python'}

=== Работа с хэшами ===
Данные пользователя: {'name': 'Иван', 'age': '30', 'city': 'Москва'}
Имя: Иван

=== Работа с упорядоченными множествами ===
Топ-3 игрока: [('player1', 100.0), ('player3', 150.0), ('player2', 200.0)]
Ранг player1: 0

=== Демонстрация кэширования ===
Данные в кэше: {"id": 1, "name": "Иван", "email": "ivan@example.com"}
После удаления: None
```

Ответы на вопросы:

1. Основное преимущество хранения данных в оперативной памяти — высокая скорость чтения и записи, что критично для приложений, требующих быстрого отклика (кэширование, сессии, очереди).
2. Параметр decode_responses=True автоматически декодирует данные из байтовых строк в обычные строки Python, что упрощает работу.
3. TTL (Time To Live) — время жизни ключа. После его истечения ключ автоматически удаляется из Redis. Используется для кэширования, сессий, временных данных.
4. lpush() добавляет элементы в начало списка, rpush() — в конец.
5. Атомарность операций в Redis обеспечивается тем, что каждая команда выполняется как единая операция, без возможности прерывания другими командами.
6. Репликация — создание копий данных на нескольких узлах для повышения отказоустойчивости. Кластеризация — распределение данных между несколькими узлами для повышения производительности и масштабируемости.

Выводы:

В ходе лабораторной работы были освоены базовые навыки работы с Redis: подключение к серверу, работа с основными типами данных, реализация кэширования. Redis является эффективным инструментом для решения задач, требующих высокой производительности и работы с временными данными. Полученные знания могут быть применены в реальных проектах для оптимизации работы с данными.