УДК 004

Обмен сообщениями Redis с Spring Boot

Семченко Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается создание двух сервисов по отправке и получению данных. Так же описывается как использовать Redis в качестве брокера обмена сообщениями.
Ключевые слова: Redis, Spring Boot, Java, отправка данных, обмен данными

Redis messaging with Spring Boot

Semchenko Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses the creation of two services for sending and receiving data. It also describes how to use Redis as a messaging broker.
Keywords: Redis, Spring Boot, Java, sending data, exchanging data

Redis - это хранилище ключей и значений, которое можно использовать для нескольких целей. Redis можно использовать как базу данных, брокер сообщений и кеш.

Spring обеспечивает встроенную поддержку операций с данными, предоставляя интерфейсы репозитория, которые можно использовать для различных операций, связанных с данными.

Цель данной работы создание двух сервисов по отправке и получению данных.

Р.И.Ибраимов продемонстрировал процесс создания Docker-образа для Spring Boot проекта и развернул его на платформе AWS EC2[3]. Е.О.Кабардинский, А.Г.Ивашко провели сравнительный анализ сервисных шин предприятия, а так же сравнили некоторые ESB, одна из которых Spring

Boot [4]. Так же Р.И.Ибраимов, А.Р.Зайчик, Н.С.Минзатров разработали генеалогическое дерево на языке Java с использованием фреймворка Spring Boot b ,b,kbjntrb gedcom4j[5]. В своей работе А.Б.Джемалетдинов, А.А.Шевченко рассмотрели вопросы создания тестов для Spring Boot mvc контроллеров [1]. В.И.Зарайский провел обзор на разработку модуля автоматизации работы с конференциями в кафедральном приложении [2].

Для пользователей системы «Windows», можно загрузить немного устаревший сервер Redis в zip-архиве. Разархивируем загруженный zip-файл и запустим сервер, запустив файл «redis-server.exe». Откроется окно приложения как на рисунке 1.

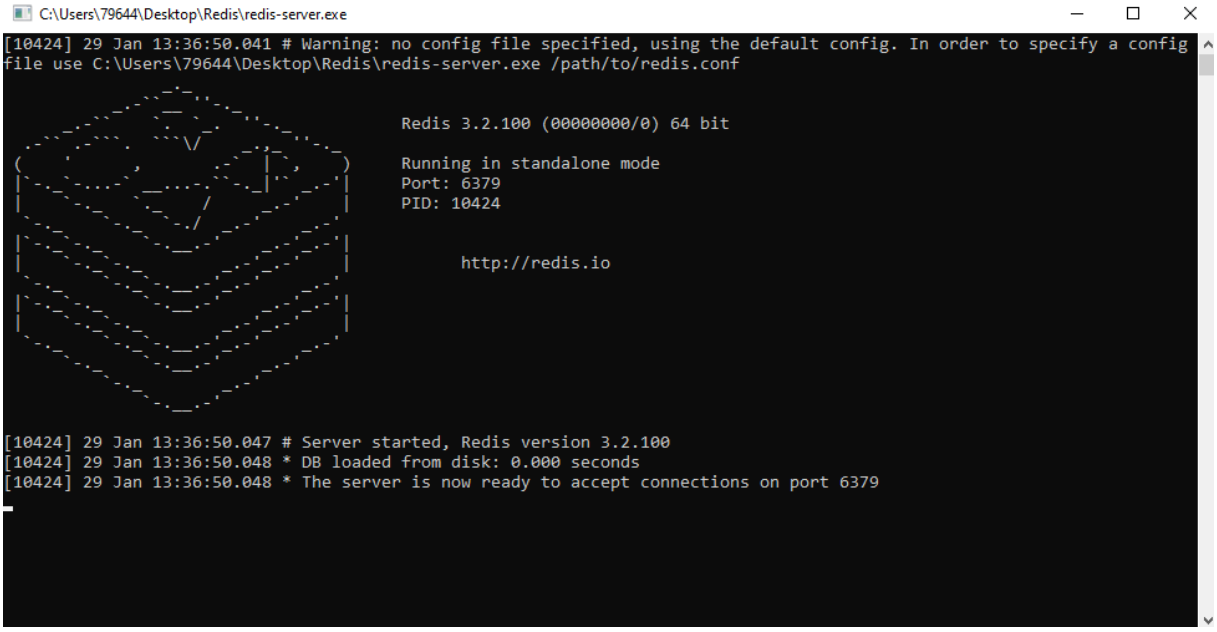

Рисунок 1 – Запуск «redis-server.exe»

Сначала создадим приложение, которое отправит сообщение на сервер Redis. Создадим загрузочное приложение Spring и добавим указанные на рисунке 2 зависимости в файл «pom.xml.

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-redis</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

Рисунок 2 – Зависимости

Создадим DTO-класс «Student.java». И будем передавать этот объект студента брокеру обмена сообщениями Redis (Рис. 3).

```java
@Getter
@Setter
@ToString
public class Student {

    private Integer id;
    private String name;
    @JsonDeserialize(using = LocalDateDeserializer.class)
    @JsonFormat(pattern = "dd/MM/yyyy")
    private LocalDate dob;
}
```

Рисунок 3 – Создание полей передачи

Здесь был использован класс «LocalDateSerializer» для сериализации полей JSON с датой ввода и поддержки формата «dd/mm/yyyy». Spring Boot предоставляет конфигурацию по умолчанию, необходимую для обмена сообщениями Redis.

Следующим шагом создадим класс конфигурации и настроим конфигурацию (Рис. 4).

```
@Configuration
public class RedisProducerConfig {

    @Bean
    RedisTemplate<String, Student> redisTemplate(RedisConnectionFactory connectionFactory,
            Jackson2JsonRedisSerializer<Student> serializer) {
        RedisTemplate<String, Student> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(connectionFactory);
        redisTemplate.setDefaultSerializer(serializer);
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }

    @Bean
    public Jackson2JsonRedisSerializer<Student> jackson2JsonRedisSerializer() {
        return new Jackson2JsonRedisSerializer<>(Student.class);
    }
}
```

Рисунок 4 – Класс конфигурации

Здесь имеются следующие важные моменты:

1. @Configuration: указывает, что это класс конфигурации Spring.

2.Jackson2JsonRedisSerializer: создает настраиваемый сериализатор Redis, который сериализует объект Student.

3.Также настроили bean-компонент «RedisTemplate», настроив его с помощью настраиваемого сериализатора Redis для поддержки объекта Student в качестве тела сообщения.

Далее создадим служебный класс, который будет иметь служебный метод, который использует настраиваемый шаблон Redis для отправки объекта Student на сервер. «@Value» аннотация читает название темы из «application.properties конфигурационного файла». Используем «convertAndSend()» метод «RedisTemplate» который публикует сообщение в Redis (Рис. 5).

```java
@Component
public class StudentProducer {

    @Autowired
    private RedisTemplate<String, Student> redisTemplate;

    @Value("studentTopic")
    private String messageTopic;

    public void sendMessage(Student student) {
        System.out.println("Sending Student details: " + student);
        redisTemplate.convertAndSend(messageTopic, student);
    }
}
```

Рисунок 5 – StudentProducer

Далее создадим контроллер REST с конечной точкой POST. И отправим объект запроса JSON в эту конечную точку, который будет опубликован в брокере сообщений Redis с использованием служебного класса, который создали ранее (Рисунок 6).

```java
@RestController
public class StudentController {

    @Autowired
    private StudentProducer studentProducer;

    @PostMapping("/send-message")
    public ResponseEntity<String> sendMessage(@RequestBody Student student) {
        studentProducer.sendMessage(student);
        return new ResponseEntity<>( body: "Message sent successfully", HttpStatus.OK);
    }
}
```

Рисунок 6 – StudentController

Конечная точка отобразит успешное сообщение после отправки сообщения брокеру Redis.

Добавим указанные ниже свойства конфигурации в файл «application.properties» приложения в каталоге «/src/main/resources/» (Рис. 7).

```
spring.redis.host=localhost
spring.redis.port=6379
redis.student.topic=studentTopic
```

Рисунок 7 – application.properties

Здесь указали конфигурацию Redis, указав хост сервера Redis, порт и тему топика.

Теперь создадим приложение, которое будет получать сообщения с сервера Redis, которые были ранее опубликованы приложением-отправителем. Зависимости будут аналогичны как на рисунке 2. Так же идентичным будет DTO-класс.

Создадим класс конфигурации получателя Redis (Рис. 8).

```
@Bean
public RedisMessageListenerContainer listenerContainer(MessageListenerAdapter listenerAdapter,
        RedisConnectionFactory connectionFactory) {
    RedisMessageListenerContainer container = new RedisMessageListenerContainer();
    container.setConnectionFactory(connectionFactory);
    container.addMessageListener(listenerAdapter, new PatternTopic(studentTopic));
    return container;
}
@Bean
public MessageListenerAdapter listenerAdapter(StudentConsumer consumer) {
    MessageListenerAdapter messageListenerAdapter = new MessageListenerAdapter(consumer);
    messageListenerAdapter.setSerializer(new Jackson2JsonRedisSerializer<>(Student.class));
    return messageListenerAdapter;
}
@Bean
RedisTemplate<String, Student> redisTemplate(RedisConnectionFactory connectionFactory,
        Jackson2JsonRedisSerializer<Student> serializer) {
    RedisTemplate<String, Student> redisTemplate = new RedisTemplate<>();
    redisTemplate.setConnectionFactory(connectionFactory);
    redisTemplate.setDefaultSerializer(serializer);
    redisTemplate.afterPropertiesSet();
    return redisTemplate;
}
@Bean
public Jackson2JsonRedisSerializer<Student> jackson2JsonRedisSerializer() {
    return new Jackson2JsonRedisSerializer<>(Student.class);
}
```

Рисунок 8 – RedisListenerConfig

Здесь так же имеются следующие важные моменты:

1.RedisMessageListenerContainer: этот компонент полезен для настройки темы, которую должно использовать приложение-получатель.

2.MessageListenerAdapter: этот bean-компонент настроенный с помощью настраиваемого сериализатора «Jackson2JsonRedisSerializer». Этот класс также ожидает, что определится «handleMethod()», который принимает сообщение Redis.

3.Был создан специальный bean-компонент «Jackson2JsonRedisSerializer» для поддержки объекта student. Эта конфигурация bean-компонента аналогична конфигурации приложения-отправителя.

4.Также создали собственный объект Bean «RedisTemplate».

Создадим теперь класс компонента «StudentConsumer.java». Этот класс будет классом получателем Redis.

По умолчанию необходимо создать метод с именем «handleMessage()». Этот метод будет прослушивать темы, указанные в конфигурации. Параметр

метода должен содержать тип объекта, который он использует. В примере метод принимает объект Student (Рис. 9).

```
@Component
public class StudentConsumer {

    public void handleMessage(Student student) { System.out.println("Consumer> " + student); }
}
```

Рисунок 9 – StudentConsumer

Свойства конфигурации будут идентичные приложению отправителю, с единственным отличием, мы поменяем порт на 8081.

Теперь для проведения тестирования запустим оба приложения и перейдем в postman для отправки запросов. Отправим запрос идентичный (Рисунок 10).

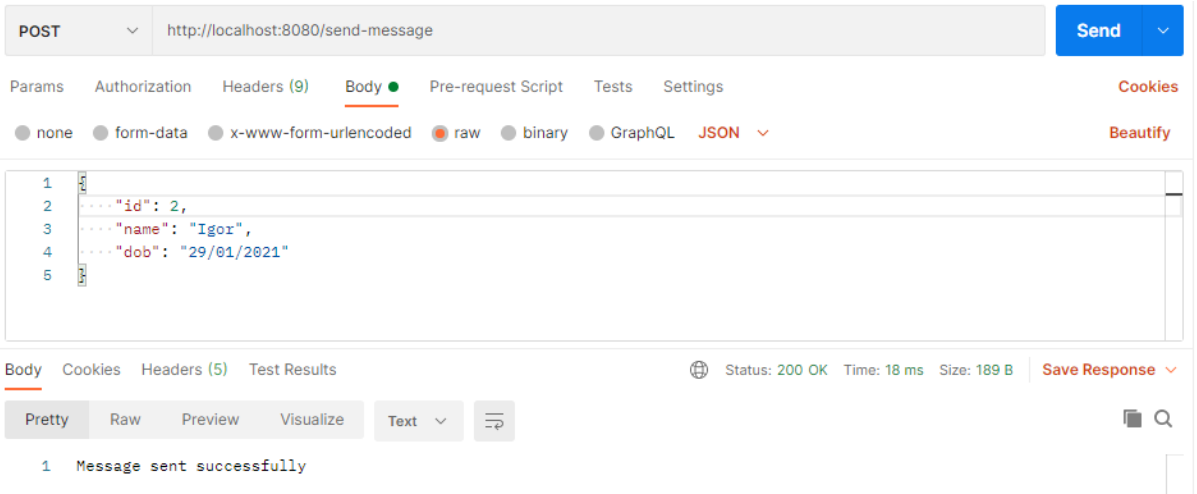

Рисунок 10 – Отправка запросов

Теперь данные отображаются в консолях двух приложениях и выводятся в консоль Redis (Рисунок 11).

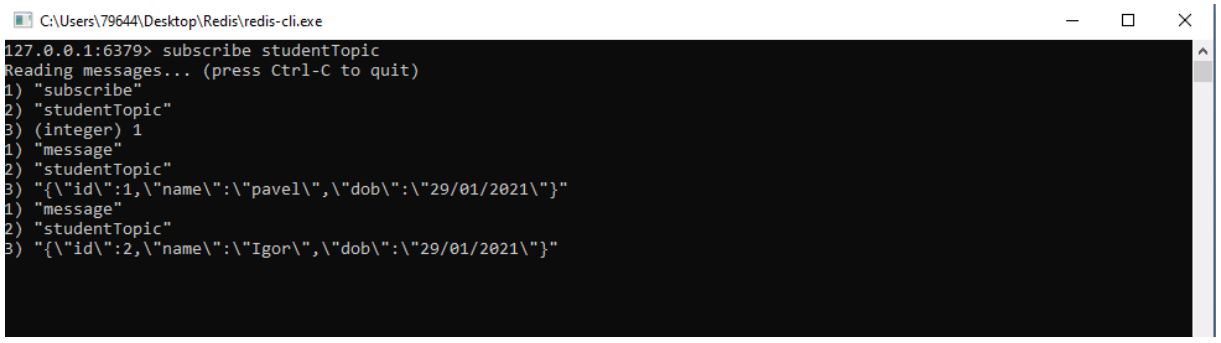

Рисунок 11 – Получение данных в консоли Redis

В данной статье было рассмотрено, как создать и настроить приложения отправления и получения данных и выводить их в консоль Redis.

Библиографический список

1. Ибраимов Р.И. Развертывание spring приложения с помощью сервиса aws ec2 и docker-контейнеров // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2020. №1(27). С. 138-147.
2. Кабардинский Е.О., Ивашко А.Г. Сравнительный анализ сервисных шин предприятия (esb) // Математическое и информационное моделирование. 2017. №10. С. 177-185.
3. Ибраимов Р.И., Зайчик А.Р., Минзатров Н.С. Разработка генеалогического дерева средствами фреймвока spring boot // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2017. №4(18). С. 18-23.
4. Джемалетдинов А.Б., Шевченко А.А. Spring boot: создание тестов для spring mvc контроллеров // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2017. №4(18). С. 104-111.
5. Зарайский В.И. Разработка модуля автоматизации работы с конференциями в кафедральном приложении // Вестник Ульяновского государственного технического университета. 2019. №3. С. 74-82.