Datos de primavera Redis

Spring Data Redis es una herramienta fantástica para integrar Redis, un almacén de datos en memoria de alto rendimiento, en tu aplicación Spring. Simplifica el trabajo con Redis proporcionando una interfaz de estilo repositorio familiar y abstracciones sobre operaciones Redis de bajo nivel. Vamos a ver cómo configurarlo y usarlo paso a paso.

1. Configurar tu Proyecto

Primero, incluye la dependencia de Spring Data Redis en tu proyecto. Si estás usando Maven, añade esto a tu `pom.xml`:

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

Para Gradle, usa:

```
implementation 'org.springframework.boot:spring-boot-starter-data-redis'
```

Esto incluye el cliente Redis (Jedis o Lettuce—Lettuce es el predeterminado). Si prefieres Jedis, puedes añadirlo explícitamente y excluir Lettuce:

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
    <exclusions>
        <exclusion>
            <groupId>io.lettuce</groupId>
            <artifactId>lettuce-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

2. Configurar Redis

Configura tu conexión Redis en `application.properties` o `application.yml`. Para una instancia Redis local ejecutándose en el puerto predeterminado (6379):

```
spring.redis.host=localhost
spring.redis.port=6379
spring.redis.password= # Opcional, si tu servidor Redis tiene una contraseña
spring.redis.database=0 # Índice de base de datos predeterminado
```

Si estás usando un servidor Redis remoto o un servicio como AWS ElastiCache, actualiza el host y las credenciales en consecuencia.

3. Uso Básico con RedisTemplate

Spring Data Redis proporciona `RedisTemplate` para operaciones de bajo nivel. Puedes autoinyectarlo en tu servicio o componente:

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

@Service
public class RedisService {
    private final RedisTemplate<String, String> redisTemplate;

    @Autowired
    public RedisService(RedisTemplate<String, String> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    public void saveData(String key, String value) {
        redisTemplate.opsForValue().set(key, value);
    }

    public String getData(String key) {
        return redisTemplate.opsForValue().get(key);
    }
}
```

- `RedisTemplate` es genérico: `<String, String>` significa que las claves y los valores son cadenas. Puedes usar otros tipos (por ejemplo, `<String, Object>`).

- `opsForValue()` es para operaciones clave-valor simples. Otros métodos incluyen `opsForList()`, `opsForSet()`, `opsForHash()`, etc., para diferentes estructuras de datos Redis.

4. Uso con Objetos

Para almacenar y recuperar objetos Java, configura `RedisTemplate` con serializadores. Spring Boot lo configura automáticamente, pero puedes personalizarlo si es necesario:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
public class RedisConfig {
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionFactory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(connectionFactory);
        template.setKeySerializer(new StringRedisSerializer());
        template.setValueSerializer(new Jackson2JsonRedisSerializer<>(Object.class));
        template.afterPropertiesSet();
        return template;
    }
}
```

Ahora puedes almacenar y recuperar objetos:

```
public class Person {
    private String firstName;
    private String lastName;

    // Constructor predeterminado (para deserialización)
    public Person() {}

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
```

```
    // Getters y setters
    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }
    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
}

@Service
public class PersonRedisService {
    private final RedisTemplate<String, Object> redisTemplate;

    @Autowired
    public PersonRedisService(RedisTemplate<String, Object> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    public void savePerson(String key, Person person) {
        redisTemplate.opsForValue().set(key, person);
    }

    public Person getPerson(String key) {
        return (Person) redisTemplate.opsForValue().get(key);
    }
}
```

5. Enfoque de Repositorio

Para una abstracción de nivel superior, usa los repositorios de Spring Data Redis. Define una entidad y un repositorio:

```
import org.springframework.data.annotation.Id;
import org.springframework.data.redis.core.RedisHash;

@RedisHash("Person") // Mapa a un hash Redis con prefijo "Person"
public class Person {
    @Id
    private String id; // La clave Redis será "Person:<id>"
    private String firstName;
    private String lastName;
```

```
    // Constructores, getters, setters (como arriba)
}

import org.springframework.data.repository.CrudRepository;

public interface PersonRepository extends CrudRepository<Person, String> {
}
```

Úsalo de esta manera:

```
@Service
public class PersonService {
    private final PersonRepository repository;

    @Autowired
    public PersonService(PersonRepository repository) {
        this.repository = repository;
    }

    public void savePerson() {
        Person person = new Person("John", "Doe");
        repository.save(person);
        System.out.println("Saved person with ID: " + person.getId());
    }

    public void findPerson(String id) {
        Person person = repository.findById(id).orElse(null);
        if (person != null) {
            System.out.println(person.getFirstName() + " " + person.getLastName());
        }
    }
}
```

- `@RedisHash` mappa la entidad a un hash Redis.
- El repositorio proporciona operaciones CRUD de inmediato.

6. Ejecutar tu Aplicación

Asegúrate de que Redis esté ejecutándose localmente (por ejemplo, mediante Docker: `docker run -d -p 6379:6379 redis`) o conecta a una instancia remota. Luego, inicia tu aplicación Spring Boot:

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

7. Características Avanzadas

- **TTL (Time-to-Live):** Establece la expiración en las claves con `redisTemplate.expire(key, 10, TimeUnit.MINUTES)`.
- **Hashes:** Usa `opsForHash()` para trabajar con hashes Redis directamente.
- **Pub/Sub:** Usa `RedisMessageListenerContainer` para mensajería.

Flujo de Trabajo de Ejemplo

1. Guarda un objeto `Person` con `repository.save()` o `redisTemplate.opsForValue().set()`.
2. Recupéralo con `repository.findById()` o `redisTemplate.opsForValue().get()`.
3. Verifica tu instancia Redis (por ejemplo, con `redis-cli`) para ver los datos.

Esto cubre lo esencial. Si quieres explorar el almacenamiento en caché, pub/sub, o algo específico, házmelo saber y profundizaré más. ¿Cómo encaja esto con lo que estás tratando de hacer?