Spring Data Redis Lists

Spring Data Redis Lists

Java Spring Framework를 사용한 레디스 리스트(Lists) 명령 사용법입니다.

Lists 소스

Redis02_List.java

```
package com.redisgate.redis;

import lombok.extern.slf4j.Slf4j;
import org.springframework.data.redis.core.ListOperations;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

@RestController
@Slf4j
public class Redis02_List {
   private final StringRedisTemplate stringRedisTemplate;
   private final ListOperations<String,String> listOperations;
   public Redis02_List(StringRedisTemplate stringRedisTemplate) {
      this.stringRedisTemplate = stringRedisTemplate;
      this.listOperations = stringRedisTemplate.opsForList();
   }
   // 여기에 각 명령(메서드) 별 소스가 들어간다.
}
```

각 명령(메서드) 별 표시

LPUSH

```
// 예제 1) LPUSH: 리스트의 왼쪽에 데이터를 저장, 리턴값은 리스트에 포함된 값의 개수이다.
// 입력된 개수가 아님.
// LPUSH key value [value ...]
// http://localhost:8080/lpush/mylist1
@GetMapping("/lpush/{key}")
public String lpush(@PathVariable("key") String key) {
    String msg = "예제 1) LPUSH(leftPush) -> ";
    Long result = 0L;    // list의 값의 총 개수를 리턴한다.
    result = listOperations.leftPush(key, "value1");    // 하나 입력
    System.out.println(msg+result);
    result = listOperations.leftPushAll(key, "value2", "value3");   // 여러 개 입력
    System.out.println(msg+result);
    // 리스트를 입력하는 방법
    ArrayList<String> list = new ArrayList<>();
    list.add("value4");
    list.add("value5");
    list.add("value6");
    result = listOperations.leftPushAll(key, list);   // 여러 개 입력
    System.out.println(msg+result);
    return msg+result;
}
```

RPOP

```
// 예제 4) RPOP: 리스트 오른쪽에서 데이터(값)를 꺼내온다.
// RPOP key [count]
// http://localhost:8080/rpop/mylist1
@GetMapping("/rpop/{key}")
public String rpop(@PathVariable("key") String key) {
    String msg = "예제 4) RPOP(rightPop) -> ";
    String result = listOperations.rightPop(key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

LPOP

```
// 예제 5) LPOP: 리스트 왼쪽에서 데이터를 꺼내온다.
// LPOP key [count]
// http://localhost:8080/lpop/mylist1
@GetMapping("/lpop/{key}")
public String lpop(@PathVariable("key") String key) {
    String msg = "예제 5) LPOP(leftPop) -> ";
    String result = listOperations.leftPop(key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

RPUSH

```
// 예제 2) RPUSH: 리스트의 오른쪽에 데이터를 저장, 리턴값은 리스트에 포함된 값의 개수이다.
// RPUSH key value [value ...]
// http://localhost:8080/rpush/mylist1
@GetMapping("/rpush/{key}")
public String rpush(@PathVariable("key") String key) {
    String msg = "예제 2) RPUSH(rightPush) -> ";
    Long result = 0L;    // list의 값의 총 개수를 리턴한다.
    result = listOperations.rightPush(key, "Rvalue1");    // 하나 입력
    System.out.println(msg+result);
    result = listOperations.rightPushAll(key, "Rvalue2", "Rvalue3");   // 여러 개 입력
    System.out.println(msg+result);
    ArrayList<String> list = new ArrayList<>();
    list.add("Rvalue4");
    list.add("Rvalue5");
    list.add("Rvalue6");
    result = listOperations.rightPushAll(key, list);
    System.out.println(msg+result);
    return msg+result;
}
```

LRANGE

```
// 예제 3) LRANGE: 리스트 조회, 범위(인덱스) 지정, 전체를 조회할 경우 start에 0, stop에 -1을 입력한다.
// LRANGE key start stop
// http://localhost:8080/lrange/mylist1
@GetMapping("/lrange/{key}")
public String lrange(@PathVariable("key") String key) {
    String msg = "예제 3) LRANGE(range) -> ";
    List<String> result = listOperations.range(key, 0, -1);
    System.out.println(msg+result);
    result =  listOperations.range(key, 0, 2);
    System.out.println(msg+result);
    result = listOperations.range(key, -1, -3);
    System.out.println(msg+result);
    return msg+result;
}
```

LLEN

```
// 예제 6) LLEN: 리스트에서 value의 개수를 조회
// LLEN key
// http://localhost:8080/llen/mylist1
@GetMapping("/llen/{key}")
public String llen(@PathVariable("key") String key) {
    String msg = "예제 6) LLEN(size) -> ";
    Long result = listOperations.size(key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

RPOPLPUSH

```
// 예제 7) RPOPLPUSH: 리스트 오른쪽에서 데이터를 꺼내서 왼쪽에 넣는다.
// RPOPLPUSH source destination
// http://localhost:8080/rpoplpush/mylist2
@GetMapping("/rpoplpush/{key}")
public String rpoplpush(@PathVariable("key") String key) {
    String msg = "예제 7) RPOPLPUSH(rightPopAndLeftPush) -> ";
    String result = listOperations.rightPopAndLeftPush("mylist1", key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

BLPOP

레디스 Lists에는 4개의 Blocking(대기) 명령이 있습니다. BLPOP/BRPOP/BRPOPLPUSH/BLMOVE
레터스(Lettuce)에서는 메서드명을 레디스 명령과 동일하게 해서 구분합니다.
스프링(Spring)에서는 메서드명은 일반 메서드명과 같이 사용하면서 인자로 대기 명령을 구분합니다.
예를 들어, leftPop(key)는 LPOP 명령이고 leftPop(key, long timeout, TimeUnit unit)는 BLPOP 명령입니다.
다음은 레디스 명령과 스프링 메서드 목록입니다.

BLPOP
leftPop(key, long timeout, TimeUnit unit)
leftPop(K key, Duration timeout)
BRPOP
rightPop(K key, long timeout, TimeUnit unit)
rightPop(K key, Duration timeout)
BRPOPLPUSH
rightPopAndLeftPush(K sourceKey, K destinationKey, long timeout, TimeUnit unit)
rightPopAndLeftPush(K sourceKey, K destinationKey, Duration timeout)
BLMOVE
move(MoveFrom from, MoveTo to, Duration timeout)
move(K sourceKey, Direction from, K destinationKey, Direction to, Duration timeout)
move(K sourceKey, Direction from, K destinationKey, Direction to, long timeout, TimeUnit unit)

레터스와 스프링이 대기 명령을 처리하는 방식에는 다음과 같은 차이점이 있습니다.
레터스에서는 대기 명령을 한(같은) 연결(connection)에서 비동기로 실행합니다.
스프링에서는 대기 명령 실행마다 새로 연결(connection)을 맺어서 처리하고 끊습니다.
이것은 LettuceConnectionFactory 의 shareNativeConnection 설정과 관계없습니다.
새로 연결을 맺고 끊는데 따른 컴퓨터 자원 사용과 처리 시간을 고려하시기 바랍니다.

예제 8) BLPOP(leftPop) 메서드 실행

```
// 예제 8) BLPOP/BRPOP/BLMOVE/BRPOPLPUSH: Blocking(대기) 명령
// BLPOP: 데이터가 들어오면 리스트의 왼쪽에서 꺼내온다.
// BLPOP key [key ...] timeout, BRPOP key [key ...] timeout
// ** Spring에서는 BLPOP 명령 실행마다 새로 연결(connection)을 맺어서 처리하고 끊는다. **
// 이것은 LettuceConnectionFactory 의 shareNativeConnection 설정과 관계없다.
// http://localhost:8080/blpop/mylist2
@GetMapping("/blpop/{key}")
public String blpop(@PathVariable("key") String key) {
    String msg = "예제 8) BLPOP(leftPop) -> ";
    String result = listOperations.leftPop(key,0, TimeUnit.SECONDS);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

예제 9) connection 사용을 확인하기 위해서 리스트에 100개 데이터 입력

```
// 예제 9) List에 100개 값 입력 -> BLPOP multi connection Test
// http://localhost:8080/listinput/mylist9
@GetMapping("/listinput/{key}")
public String listinput(@PathVariable("key") String key) {
   String msg = "예제 9) List Input -> ";
   Long result = 0L;
   for (int i = 1; i < 101; i++) {
      result += listOperations.leftPush(key,"value-"+i);
   }
   msg += result;
   System.out.println(msg);
   return msg;
}
```

예제 10) BLPOP(leftPop) 메서드를 100번 반복 실행하면서 "redis-cli --stat"로 connection 확인

```
// 예제 10) BLPOP 명령 반복 실행해서 connection 확인
// http://localhost:8080/multiblpop/mylist9
// redis-cli --stat       -> connections 확인
@GetMapping("/multiblpop/{key}")
public String multiblpop(@PathVariable("key") String key) {
   String msg = "예제 10) BLPOP(leftPop) -> ";
   System.out.println(msg);
   for (int i = 1; i < 101; i++) {
      String result = listOperations.leftPop(key,0, TimeUnit.SECONDS);
      System.out.println(result);
   }
   return msg;
}
```


redisgate@gmail.com 02.503.2235
서울시 강남구 강남대로 342 역삼빌딩 5층 (역삼동) 우 06242