redis
Clients

Java SpringBoot New
- Introduction
- Strings
- Lists
- Sets
- ZSets
- Hashes
- Streams
- Common Keys
- Pipelining
- Pub/Sub
- Master/Replica
- Sentinel
- Cluster
- ---
- Auto Config
- Manual Config
- Load Balancing (readFrom)
- RedisTemplate
- Connection Pool & Thread
- Async Spring & Lettuce
- DB select
- Spring Multi Data Source
- Lettuce Multi Data Source
- Spring Project Create
- Spring Project Eclipse
- Spring Project IntelliJ
- ---
- Spring Session Standalone
- Spring Session MasterRepli
- Spring Session Sentinel
- Spring Session Cluster

Java Lettuce(Spring)New

Java Lettuce(Plain)

Java Jedis

Java Redisson

C Hiredis

C# StackExchange

PHP PhpRedis

PHP Predis

Redis Admin & Monitoring Tool

Spring Data Redis Common Keys

Spring Data Redis Common Keys

Java Spring Framework를 사용한 레디스 키(Common Keys) 명령 사용법입니다.

Common Keys 소스

Redis07_Common.java

```
package com.redisgate.redis;

import lombok.extern.slf4j.Slf4j;
import org.springframework.data.redis.connection.DataType;
import org.springframework.data.redis.connection.RedisConnection;
import org.springframework.data.redis.core.Cursor;
import org.springframework.data.redis.core.ScanOptions;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import java.time.Duration;
import java.util.HashSet;
import java.util.Set;

@RestController
@Slf4j
public class Redis07_Common {

    private final StringRedisTemplate stringRedisTemplate;
    private final ValueOperations<String,String> valueOperations;

    public Redis07_Common(StringRedisTemplate stringRedisTemplate) {
        this.stringRedisTemplate = stringRedisTemplate;
        this.valueOperations = stringRedisTemplate.opsForValue();
    }
    // 여기에 각 명령(메서드) 별 소스가 들어갑니다.
}
```

각 명령(메서드) 별 표시

EXISTS

```
// 예제 3) EXISTS: 키(key)가 존재하는지 확인
// EXISTS key [key ...]
// http://localhost:8080/exists/key-001, key-900
@GetMapping("/exists/{key}")
public String exists(@PathVariable("key") String key) {
    // Boolean hasKey(K key) -> 키 하나를 확인할 때
    Boolean result = stringRedisTemplate.hasKey(key);
    String msg = "예제 3) EXISTS(hasKey) -> ";
    System.out.println(msg+result);
    // Long countExistingKeys(Collection<K> keys) -> 여러 개를 확인할 때
    Set<String> keys = new HashSet<>();
    keys.add("key-001");
    keys.add("key-002");
    keys.add("key-003");
    Long result2 = stringRedisTemplate.countExistingKeys(keys);
    msg = "예제 3) EXISTS(countExistingKeys) -> ";
    System.out.println(msg+result2);
    return msg+result2;
}
```

KEYS

```
// 예제 1) KEYS: 키(key)들을 pattern으로 조회
// KEYS pattern
// URL에 keys를 입력하면 오류 발생 -> Redis1_String과 충돌임.
// http://localhost:8080/keys/*
@GetMapping("/keys/{pattern}")
public String keys(@PathVariable("pattern") String pattern) {
    String msg = "예제 1) KEYS -> "+pattern;
    System.out.println(msg);
    // Set<K> keys(K pattern)
    Set<String> result = stringRedisTemplate.keys(pattern);
    if (result == null) {
        return msg+"result is null";
    }
    if (result.isEmpty()) { // 결과가 없으면 empty이다.
        return msg+"result is empty";
    }
    int i = 0;
    for (String key: result) {
        System.out.println((++i) +") "+key);
    }
    return result.toString();
}
```

SCAN

```
// 예제 2) SCAN: 키(key)들을 일정 단위 개수 만큼씩 조회
// SCAN cursor [MATCH pattern] [COUNT count]
// http://localhost:8080/scan/*
@GetMapping("/scan/{pattern}")
public String scan(@PathVariable("pattern") String pattern) {
    String msg = "예제 2) SCAN -> ";
    System.out.println(msg+"0");
    // Cursor<K> scan(ScanOptions options)
    // ScanOptions -> COUNT(limit)와 MATCH(pattern) 설정
    ScanOptions scanOptions = ScanOptions.scanOptions().count(20).match(pattern).build();
    Cursor<String> result = stringRedisTemplate.scan(scanOptions);
    long cursor = result.getCursorId();
    while (result.hasNext()) {
        if (cursor != result.getCursorId()) {
            System.out.println("Next cursor: "+result.getCursorId());
            cursor = result.getCursorId();
        }
        String key = result.next();
        System.out.println("  "+ result.getPosition() +") "+key);
    }
    return msg+result.getCursorId();
}
```

RENAME

```
// 예제 4) RENAME: 키(key) 이름 변경, RENAMENX
// RENAME key new_key, RENAMENX key new_key
// http://localhost:8080/rename/key-100:key-900
@GetMapping("/rename/{key}")
public String rename(@PathVariable("key") String key) {
    String[] keys = key.split(":");
    stringRedisTemplate.rename(keys[0],keys[1]);    // return void
    String msg = "예제 4) RENAME -> OK";
    System.out.println(msg);
    // RENAMENX: 기존에 키가 없을 경우에만 이름 변경
    Boolean result = stringRedisTemplate.renameIfAbsent("key-099",keys[1]);
    msg = "예제 4) RENAMENX(renameIfAbsent) -> "+result;
    System.out.println(msg);
    return msg;
}
```

EXPIRE

```
// 예제 5) EXPIRE: 지정된 시간(초) 후 키(key) 자동 삭제
// EXPIRE key seconds
// http://localhost:8080/expire/key-900:180
@GetMapping("/expire/{key}")
public String expire(@PathVariable("key") String key) {
    String[] keyExpire = key.split(":");
    Boolean result = stringRedisTemplate.expire(keyExpire[0],
            Duration.ofSeconds(Long.parseLong(keyExpire[1])));
    String msg = "예제 5) EXPIRE -> "+result;
    System.out.println(msg);
    return msg;
}
```

TTL

```
// 예제 6) TTL: 남은 expire time(seconds)을 조회
// TTL key
// http://localhost:8080/ttl/key-900
@GetMapping("/ttl/{key}")
public String ttl(@PathVariable("key") String key) {
    Long result = stringRedisTemplate.getExpire(key);
    String msg = "예제 6) TTL(getExpire) -> "+result;
    System.out.println(msg);
    return msg;
}
```

PERSIST

```
// 예제 7) PERSIST: 설정된 Expire time을 삭제
// PERSIST key
// http://localhost:8080/persist/key-900
@GetMapping("/persist/{key}")
public String persist(@PathVariable("key") String key) {
    Boolean result = stringRedisTemplate.persist(key);
    String msg = "예제 7) PERSIST -> "+result;
    System.out.println(msg);
    return msg;
}
```

TYPE

```
// 예제 8) TYPE: 키(key)의 data type을 조회
// TYPE key
// http://localhost:8080/persist/key-001
@GetMapping("/type/{key}")
public String type(@PathVariable("key") String key) {
    String msg = "예제 8) TYPE -> ";
    DataType result = stringRedisTemplate.type(key);
    if (result == null) { //
        return msg+"result is null";
    }
    msg += result.name();
    System.out.println(msg);
    return msg;
}
```

FLUSHDB

```
// 예제 9) FLUSHDB: 현재 사용중인(선택된) DB의 모든 데이터(키와 값)를 삭제
// FLUSHDB [async], FLUSHALL [async]
// http://localhost:8080/flushdb
@GetMapping("/flushdb")
public String flushdb() {
    RedisConnection redisConnection = stringRedisTemplate.getConnectionFactory().getConnection();
    redisConnection.serverCommands().flushDb(); // return void
    String msg = "예제 9) FLUSHDB -> OK";
    System.out.println(msg);
    return msg;
}
```

테스트 데이터 입력

```
// 예제 10) 테스트 데이터 입력
// http://localhost:8080/input
@GetMapping("/input")
public String input() {
    // String: key* 100개,
    for (int i=1; i<=100; i++) {
        valueOperations.set("key-"+String.format("%03d",i), "value-"+String.format("%03d",i));
    }
    String msg = "예제 10) 테스트 데이터 입력 -> OK";
    System.out.println(msg);
    return msg;
}
```

<< Streams Common Keys Pipelining >>

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