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 Strings

Spring Data Redis Strings

Java Spring Framework를 사용한 레디스 스트링(Strings) 명령 사용법입니다.

Strings 소스

Redis01_String.java

```
package com.redisgate.redis;

import lombok.extern.slf4j.Slf4j;
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.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.concurrent.TimeUnit;

@RestController
@Slf4j
public class Redis01_String {

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

   public Redis01_String(StringRedisTemplate stringRedisTemplate) {
      this.stringRedisTemplate = stringRedisTemplate;
      this.valueOperations = stringRedisTemplate.opsForValue();
   }
   // 여기에 각 명령(메서드) 별 소스가 들어간다.
}
```

각 명령(메서드) 별 표시

SET

```
// 예제 1) SET: 키-값 저장
// SET key value [NX|XX] [EX 초] [PX 밀리초] [EXAT timestamp] [PXAT milliseconds-timestamp] [KEEPTTl
// http://localhost:8080/set/key01
@GetMapping("/set/{key}")
public String set(@PathVariable("key") String key) {
    String msg = "예제 1) SET -> OK";
    valueOperations.set(key,"value-"+key);  // return void
    System.out.println(msg);
    return msg;
}
```

GET

```
// 예제 2) GET: 조회
// GET key
// http://localhost:8080/get/key01
@GetMapping("/get/{key}")
public String get(@PathVariable("key") String key) {
    String result = valueOperations.get(key);
    String msg = "예제 2) GET -> "+result;
    System.out.println(msg);
    return msg;
}
```

DEL

```
// 예제 3) DEL: 삭제
// DEL key [key ...]
// http://localhost:8080/del/key01
@GetMapping("/del/{key}")
public String del(@PathVariable("key") String key) {
    // stringRedisTemplate 클래스를 직접 사용한다.
    Boolean result = stringRedisTemplate.delete(key);
    String msg = "예제 3) DEL(delete) -> "+result;
    System.out.println(msg);
    List<String> keys = new ArrayList<>();
    keys.add("key2");
    keys.add("key3");
    Long result2 = stringRedisTemplate.delete(keys);
    msg = "예제 3) DEL(delete) -> "+result2;
    System.out.println(msg);
    return msg;
}
```

SETNX

```
// 예제 4) SETNX: 키가 기존에 없을 경우에만 저장된다.
// SET key value NX
// http://localhost:8080/setnx/key01
@GetMapping("/setnx/{key}")
public String setnx(@PathVariable("key") String key) {
    String msg = "예제 4) SETNX(setIfAbsent) -> ";
    Boolean result;
    result = valueOperations.setIfAbsent(key, "Hello");
    System.out.println(msg+result);
    result = valueOperations.setIfAbsent(key, "Redis");
    System.out.println(msg+result);
    String result2 = valueOperations.get(key);
    System.out.println(msg+result2);
    return msg+result2;
}
```

SETEX

```
// 예제 5) SETEX: 저장하고 10초 후에 자동 삭제된다.
// SETEX key seconds(초) value
// http://localhost:8080/setex/key02
@GetMapping("/setex/{key}")
public String setex(@PathVariable("key") String key) {
    String msg = "예제 5) SETEX(set) -> OK";
    // void set(K key, V value, long timeout, TimeUnit unit)
    int expire = 10;    // 10초
    valueOperations.set(key,"value-"+key, expire, TimeUnit.SECONDS);    // return void
    System.out.println(msg);
    return msg;
}
```

GETEX

```
// 예제 7) GETEX: 조회하고 10초 후에 자동 삭제된다.  Redis 6.2부터 사용 가능
// GETEX key [EX seconds|PX milliseconds|EXAT timestamp|PXAT milliseconds-timestamp|PERSIST]
// http://localhost:8080/getex/key01
@GetMapping("/getex/{key}")
public String getex(@PathVariable("key") String key) {
    String msg = "예제 7) GETEX(getAndExpire) -> ";
    int expire = 10;    // 10초
    String result = valueOperations.getAndExpire(key, expire, TimeUnit.SECONDS);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

GETDEL

```
// 예제 8) GETDEL: 조회하고 삭제한다.  Redis 6.2부터 사용 가능
// GETDEL key
// http://localhost:8080/getdel/keyM01
@GetMapping("/getdel/{key}")
public String getdel(@PathVariable("key") String key) {
    String msg = "예제 8) GETDEL(getAndDelete) -> ";
    String result = valueOperations.getAndDelete(key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

INCR

```
// 예제 11) INCR: 숫자(값)을 1씩 증가한다.
// INCR key
// http://localhost:8080/incr/key01
@GetMapping("/incr/{key}")
public String incr(@PathVariable("key") String key) {
    String msg = "예제 11) INCR(increment) -> ";
    Long result = valueOperations.increment(key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

DECR

```
// 예제 12) DECR: 숫자(값)을 1씩 감소한다.
// DECR key
// http://localhost:8080/decr/key01
@GetMapping("/decr/{key}")
public String decr(@PathVariable("key") String key) {
    String msg = "예제 12) DECR(decrement) -> ";
    Long result = valueOperations.decrement(key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

MSET

```
// 예제 6) MSET: 여러 개 키-값 쌍을 저장한다.
// MSET key value [key value ...]
// http://localhost:8080/mset/keyM
@GetMapping("/mset/{key}")
public String mset(@PathVariable("key") String key) {
    String msg = "예제 6) MSET(multiSet) -> OK";
    HashMap<String, String> keyValues = new HashMap<>();
    keyValues.put(key+"01", "value01");
    keyValues.put(key+"02", "value02");
    keyValues.put(key+"03", "value03");
    valueOperations.multiSet(keyValues);    // return void
    System.out.println(msg);
    return msg;
}
```

MGET

```
// 예제 9) MGET: 여러 개 키를 조회한다.
// MGET key [key ...]
// http://localhost:8080/mget/keyM
@GetMapping("/mget/{key}")
public String mget(@PathVariable("key") String key) {
    String msg = "예제 9) MGET(multiGet) -> ";
    List<String> keys = new ArrayList<>();
    keys.add(key+"01");
    keys.add(key+"02");
    keys.add(key+"03");
    List<String> values = valueOperations.multiGet(keys);
    msg += values;
    System.out.println(msg);
    return msg;
}
```

SETRANGE

```
// 예제 14) SETRANGE: 값의 일부를 저장(변경)한다.
// 큰 값의 일부를 변경할 경우 사용한다.
// SETRANGE key offset value
// http://localhost:8080/setrange/keyM02
@GetMapping("/setrange/{key}")
public String setrange  (@PathVariable("key") String key) {
    String msg = "예제 13) SETRANGE(set) -> OK";
    valueOperations.set(key,"MySQL",6); // return void
    System.out.println(msg);
    return msg;
}
```

GETRANGE

```
// 예제 13) GETRANGE: 값의 일부를 조회한다.
// 값이 클 경우 값 전체를 가져와서 substring() 하는 것보다 성능면에서 효율적이다.
// GETRANGE key start end
// http://localhost:8080/getrange/keyM02
@GetMapping("/getrange/{key}")
public String getrange (@PathVariable("key") String key) {
    String msg = "예제 13) GETRANGE(get) -> ";
    String result = valueOperations.get(key,6,10);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

GETSET

```
// 예제 10) GETSET: 값을 조회하고 새 값을 저장한다.
// GETSET key value
// http://localhost:8080/getset/keyM02
@GetMapping("/getset/{key}")
public String getset(@PathVariable("key") String key) {
    String msg = "예제 10) GETSET(getAndSet) -> ";
    String result = valueOperations.getAndSet(key, "Hello-Redis");
    msg += result;
    System.out.println(msg);
    return msg;
}
```

<< Spring Introduction Strings Lists >>

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