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 Streams

레디스 개발자 교육 신청

레디스 정기점검/기술지원
Redis Technical Support

레디스 엔터프라이즈 서버
Redis Enterprise Server

Spring Data Redis Streams

Java Spring Framework를 사용한 레디스 스트림(Streams) 명령 사용법입니다.

Streams 소스

Redis06_Stream.java

```
package com.redisgate.redis;

import lombok.extern.slf4j.Slf4j;
import org.springframework.data.redis.connection.stream.*;
import org.springframework.data.redis.core.StreamOperations;
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 org.springframework.data.domain.Range;
//import io.lettuce.core.Range; 이것을 사용하지 마시고 위 domain.Range를 사용하세요.

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@Slf4j
public class Redis06_Stream {

    private final StringRedisTemplate stringRedisTemplate;
    private final StreamOperations<String,String,String> streamOperations;

    public Redis06_Stream(StringRedisTemplate stringRedisTemplate) {
        this.stringRedisTemplate = stringRedisTemplate;
        this.streamOperations = stringRedisTemplate.opsForStream();
    }
    // 여기에 각 명령(메서드) 별 소스가 들어갑니다.
}
```

각 명령(메서드) 별 표시

XADD

```
// 예제 1) XADD: 스트림에 데이터를 추가
// XADD key ID field value [field2 value2 ...]
// XADD key [NOMKSTREAM] [MAXLEN|MINID [=|~] threshold [LIMIT count]] *|ID field value [field valu
// http://localhost:8080/xadd/sensor-1234
@GetMapping("/xadd/{key}")
public String xadd(@PathVariable("key") String key) {
    String msg = "예제 1) XADD(add) -> ";
    System.out.println(msg);
    // 온도 입력
    // RecordId add(K key, Map<? extends HK, ? extends HV> content)
    Map<String,String> data = new HashMap<>();
    data.put("temperature","98.7");
    RecordId recordId  = streamOperations.add(key,data);
    System.out.println(recordId);
    // 온도, 습도 입력
    data.clear();
    data.put("temperature","97.7");
    data.put("humidity","50.5");
    recordId  = streamOperations.add(key,data);
    System.out.println(recordId);
    // 온도, 습도, 압력 입력
    data.clear();
    data.put("temperature", "90.5");
    data.put("humidity", "50.5");
    data.put("pressure", "10.5");
    // RecordId add(Record<K, ?> record) -> key와 data를 같이 입력.
    StringRecord record = StreamRecords.string(data).withStreamKey(key);
    recordId = streamOperations.add(record);
    System.out.println(recordId);
    return msg+recordId;
}
```

XLEN

```
// 예제 3) XLEN: 스트림의 엔트리(ID) 개수 리턴
// XLEN key
// http://localhost:8080/xlen/sensor-1234
@GetMapping("/xlen/{key}")
public String xlen(@PathVariable("key") String key) {
    String msg = "예제 3) XLEN(size) -> ";
    Long result = streamOperations.size(key);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

XRANGE

```
// 예제 2) XRANGE: 데이터 조회
// XRANGE key start end [COUNT count], XREVRANGE key end start [COUNT count]
// http://localhost:8080/xrange/sensor-1234
@GetMapping("/xrange/{key}")
public String xrange(@PathVariable("key") String key) {
    String msg = "예제 2) XRANGE(range) -> ";
    String[] keyId = key.split(":");
    Range<String> range;
    if (keyId.length == 1) {
        range = Range.unbounded();
        msg += "전체 조회";
    } else {
        range = Range.rightUnbounded(Range.Bound.inclusive(keyId[1]));
        msg += "지정 id 부터";
    }
    System.out.println(msg);
    // List<MapRecord<K, HK, HV>> range(K key, Range<String> range, Limit limit)
    List<MapRecord<String, String, String>> result = streamOperations.range(keyId[0],range);
    if (result == null) return null;
    for (MapRecord<String, String, String> record: result) {
        System.out.println("ID: "+record.getId());
        for (Map.Entry<String,String> entry : record.getValue().entrySet()) {
            System.out.println("  "+entry.getKey()+" "+entry.getValue());
        }
    }
    return msg;
}
```

XREAD

```
// 예제 4) XREAD: 데이터 읽기
// XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
// http://localhost:8080/xread/sensor-1234:0   -> 처음부터 읽을 때는 0을 지정
// http://localhost:8080/xread/sensor-1234:1704983789750  -> 지정한 ID 이후 데이터를 읽어온다.
// http://localhost:8080/xread/sensor-1234:>  -> '>' 는 XREADGROUP 명령에서만 사용할 수 있다.
@GetMapping("/xread/{key}")
@SuppressWarnings("unchecked")
public String xread(@PathVariable("key") String key) {
    String msg = "예제 4) XREAD(read) -> ";
    String[] keyId = key.split(":");
    // List<MapRecord<K, HK, HV>> read(StreamReadOptions readOptions, StreamOffset<K>... streams)
    List<MapRecord<String, String, String>> result = streamOperations.read(StreamReadOptions.empty
            StreamOffset.create(keyId[0],ReadOffset.from(keyId[1])));
    if (result == null) return null;
    for (MapRecord<String, String, String> record: result) {
        System.out.println("ID: "+record.getId());
        for (Map.Entry<String,String> entry : record.getValue().entrySet()) {
            System.out.println("  "+entry.getKey()+" "+entry.getValue());
        }
    }
    return msg;
}
```

XDEL

```
// 예제 5) XDEL: 데이터 삭제
// XDEL key ID [ID ...]
// http://localhost:8080/xdel/sensor-1234:1704983736197
@GetMapping("/xdel/{key}")
public String xdel(@PathVariable("key") String key) {
    String msg = "예제 5) XDEL -> ";
    String[] keyId = key.split(":");
    // 삭제한 id 개수를 리턴한다.
    // Long delete(K key, String... recordIds)
    Long result = streamOperations.delete(keyId[0], keyId[1]);

    // RecordId id = RecordId.of(keyId[1]);
    // Long delete(K key, RecordId... recordIds)
    // Long result = streamOperations.delete(keyId[0], id);
    msg += result;
    System.out.println(msg);
    return msg;
}
```

<< Hashes Streams Common Keys >>

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

Copyright © 2014-2024 redisGate
All right reserved