

第 4-2 课：Spring Boot 和 Redis 常用操作

Redis 是目前使用最广泛的缓存中间件，相比 Memcached，Redis 支持更多的数据结构和更丰富的数据操作，另外 Redis 有着丰富的集群方案和使用场景，这一课我们一起学习 Redis 的常用操作。

Redis 介绍

Redis 是一个速度非常快的非关系数据库（Non-Relational Database），它可以存储键（Key）与 5 种不同类型的值（Value）之间的映射（Mapping），可以将存储在内存的键值对数据持久化到硬盘，可以使用复制特性来扩展读性能，还可以使用客户端分片来扩展写性能。

为了满足高性能，Redis 采用内存（in-memory）数据集（Dataset），根据使用场景，可以通过每隔一段时间转储数据集到磁盘，或者追加每条命令到日志来持久化。持久化也可以被禁用，如果你只是需要一个功能丰富、网络化的内存缓存。

数据模型

Redis 数据模型不仅与关系数据库管理系统（RDBMS）不同，也不同于任何简单的 NoSQL 键-值数据存储。Redis 数据类型类似于编程语言的基础数据类型，因此开发人员感觉很自然，每个数据类型都支持适用于其类型的操作，受支持的数据类型包括：

- string（字符串）
- hash（哈希）
- list（列表）
- set（集合）
- zset（sorted set：有序集合）

关键优势

Redis 的优势包括它的速度、对富数据类型的支持、操作的原子性，以及通用性：

- 性能极高，它每秒可执行约 100,000 个 SET 以及约 100,000 个 GET 操作；
- 丰富的数据类型，Redis 对大多数开发人员已知的大多数数据类型提供了原生支持，这使得各种问题得以轻松解决；
- 原子性，因为所有 Redis 操作都是原子性的，所以多个客户端会并发地访问一个 Redis 服务器，获取相同的更新值；
- 丰富的特性，Redis 是一个多效用工具，有非常多的应用场景，包括缓存、消息队列（Redis 原生支持发布/订阅）、短期应用程序数据（比如 Web 会话、Web 页面命中计数）等。

spring-boot-starter-data-redis

Spring Boot 提供了对 Redis 集成的组件包：spring-boot-starter-data-redis，它依赖于 spring-data-redis 和 lettuce。Spring Boot 1.0 默认使用的是 Jedis 客户端，2.0 替换成了 Lettuce，但如果你从 Spring Boot 1.5.X 切换过来，几乎感受不大差异，这是因为 spring-boot-starter-data-redis 为我们隔离了其中的差异性。

- Lettuce：是一个可伸缩线程安全的 Redis 客户端，多个线程可以共享同一个 RedisConnection，它利用优秀 Netty NIO 框架来高效地管理多个连接。
- Spring Data：是 Spring 框架中的一个主要项目，目的是为了简化构建基于 Spring 框架应用的数据访问，包括非关系数据库、Map-Reduce 框架、云数据服务等，另外也包含对关系数据库的访问支持。
- Spring Data Redis：是 Spring Data 项目中的一个主要模块，实现了对 Redis 客户端 API 的高度封装，使对 Redis 的操作更加便捷。

可以用以下方式来表达它们之间的关系：

```
Lettuce → Spring Data Redis → Spring Data → spring-boot-starter-data-redis
```

因此 Spring Data Redis 和 Lettuce 具备的功能，spring-boot-starter-data-redis 几乎都会有。

快速上手

相关配置

引入依赖包

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-pool2</artifactId>
</dependency>
```

引入 commons-pool 2 是因为 Lettuce 需要使用 commons-pool 2 创建 Redis 连接池。

application 配置

```
# Redis 数据库索引（默认为 0）
spring.redis.database=0
# Redis 服务器地址
spring.redis.host=localhost
# Redis 服务器连接端口
spring.redis.port=6379
# Redis 服务器连接密码（默认为空）
spring.redis.password=
# 连接池最大连接数（使用负值表示没有限制） 默认 8
spring.redis.lettuce.pool.max-active=8
# 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
spring.redis.lettuce.pool.max-wait=-1
# 连接池中的最大空闲连接 默认 8
spring.redis.lettuce.pool.max-idle=8
# 连接池中的最小空闲连接 默认 0
spring.redis.lettuce.pool.min-idle=0
```

从配置也可以看出 Spring Boot 默认支持 Lettuce 连接池。

缓存配置

在这里可以为 Redis 设置一些全局配置，比如配置主键的生产策略 KeyGenerator，如不配置会默认使用参数名作为主键。

```
@Configuration
@EnableCaching
public class RedisConfig extends CachingConfigurerSupport{

    @Bean
    public KeyGenerator keyGenerator() {
        return new KeyGenerator() {
            @Override
            public Object generate(Object target, Method method, Object... params)
            {
                StringBuilder sb = new StringBuilder();
                sb.append(target.getClass().getName());
                sb.append(method.getName());
                for (Object obj : params) {
                    sb.append(obj.toString());
                }
                return sb.toString();
            }
        };
    }
}
```

注意，我们使用了注解：@EnableCaching 来开启缓存。

测试使用

在单元测试中，注入 RedisTemplate。String 是最常用的一种数据类型，普通的 key/value 存储都可以归为此类，value 其实不仅是 String 也可以是数字。

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class TestRedisTemplate {
    @Autowired
    private RedisTemplate redisTemplate;

    @Test
    public void testString() {
        redisTemplate.opsForValue().set("neo", "ityouknow");
        Assert.assertEquals("ityouknow", redisTemplate.opsForValue().get("neo"));
    }
}
```

在这个单元测试中，我们使用 redisTemplate 存储了一个字符串 "ityouknow"，存储之后获取进行验证，多次进行 set 相同的 key，键对应的值会被覆盖。

从上面的整个流程来看，使用 spring-boot-starter-data-redis 只需要三步就可以快速地集成 Redis 进行操作，下面介绍 Redis 如何操作各种数据类型。

各类型实践

我们知道 Redis 支持多种数据类型，实体、哈希、列表、集合、有序集合，那么在 Spring Boot 体系中都如何使用呢？

实体

先来看 Redis 对 Pojo 的支持，新建一个 User 对象，放到缓存中，再取出来。

```
@Test
public void testObj(){
    User user=new User("ityouknow@126.com", "smile", "youknow", "know","2020");
    ValueOperations<String, User> operations=redisTemplate.opsForValue();
    operations.set("com.neo", user);
    User u=operations.get("com.neo");
    System.out.println("user: "+u.toString());
}
```

输出结果：

```
user: com.neo.domain.User@16fb356[id=<null>,userName=know,passWord=youknow,email=ityouknow@126.com,nickName=smile,regTime=2020]
```

验证发现完美支持对象的存入和读取。

超时失效

Redis 在存入每一个数据的时候都可以设置一个超时时间，过了这个时间就会自动删除数据，这种特性非常适合我们对阶段数据的缓存。

新建一个 User 对象，存入 Redis 的同时设置 100 毫秒后失效，设置一个线程暂停 1000 毫秒之后，判断数据是否存在并打印结果。

```
@Test
public void testExpire() throws InterruptedException {
    User user=new User("ityouknow@126.com", "expire", "youknow", "expire","2020");
    ValueOperations<String, User> operations=redisTemplate.opsForValue();
    operations.set("expire", user,100,TimeUnit.MILLISECONDS);
    Thread.sleep(1000);
    boolean exists=redisTemplate.hasKey("expire");
    if(exists){
        System.out.println("exists is true");
    }else{
        System.out.println("exists is false");
    }
}
```

输出结果：

```
exists is false
```

从结果可以看出，Redis 中已经不存在 User 对象了，此数据已经过期，同时我们在这个测试的方法中使用了 hasKey("expire") 方法，可以判断 key 是否存在。

删除数据

有些时候，我们需要对过期的缓存进行删除，下面来测试此场景的使用。首先 set 一个字符串“ityouknow”，紧接着删除此 key 的值，再进行判断。

```
@Test
public void testDelete() {
    ValueOperations<String, User> operations=redisTemplate.opsForValue();
    redisTemplate.opsForValue().set("deletekey", "ityouknow");
    redisTemplate.delete("deletekey");
    boolean exists=redisTemplate.hasKey("deletekey");
    if(exists){
        System.out.println("exists is true");
    }else{
        System.out.println("exists is false");
    }
}
```

输出结果：

```
exists is false
```

结果表明字符串“ityouknow”已经被成功删除。

Hash（哈希）

一般我们存储一个键，很自然的就会使用 get/set 去存储，实际上这并不是很好的做法。Redis 存储一个 key 会有一个最小内存，不管你存的这个键多小，都不会低于这个内存，因此合理的使用 Hash 可以帮我们节省很多内存。

Hash Set 就在哈希表 Key 中的域（Field）的值设为 value。如果 Key 不存在，一个新的哈希表被创建并进行 HSET 操作；如果域（field）已经存在于哈希表中，旧值将被覆盖。

```
@Test
public void testHash() {
    HashOperations<String, Object, Object> hash = redisTemplate.opsForHash();
    hash.put("hash", "you", "you");
    String value=(String) hash.get("hash", "you");
    System.out.println("hash value :"+value);
}
```

输出结果：

```
hash value :you
```

根据上面测试用例发现，Hash set 的时候需要传入三个参数，第一个为 key，第二个为 field，第三个为存储的值。一般情况下 Key 代表一组数据，field 为 key 相关的属性，而 value 就是属性对应的值。

List

Redis List 的应用场景非常多，也是 Redis 最重要的数据结构之一。使用 List 可以轻松的实现一个队列，List 典型的应用场景就是消息队列，可以利用 List 的 Push 操作，将任务存在 List 中，然后工作线程再用 POP 操作将任务取出进行执行。

```
@Test
public void testList() {
    ListOperations<String, String> list = redisTemplate.opsForList();
    list.leftPush("list", "it");
    list.leftPush("list", "you");
    list.leftPush("list", "know");
    String value=(String)list.leftPop("list");
    System.out.println("list value :"+value.toString());
}
```

输出结果：

```
list value :know
```

上面的例子我们从左侧插入一个 key 为 "list" 的队列，然后取出左侧最近的一条数据。其实 List 有很多 API 可以操作，比如从右侧进行插入队列从右侧进行读取，或者通过方法 range 读取队列的一部分。接着上面的例子我们使用 range 来读取。

```
List<String> values=list.range("list",0,2);
for (String v:values){
    System.out.println("list range :"+v);
}
```

输出结果：

```
list range :know
list range :you
list range :it
```

range 后面的两个参数就是插入数据的位置，输入不同的参数就可以取出队列中对应的数据。

Redis List 的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis 内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

Set

Redis Set 对外提供的功能与 List 类似是一个列表的功能，特殊之处在于 Set 是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，Set 是一个很好的选择，并且 Set 提供了判断某个成员是否在一个 Set 集合内的重要接口，这个也是 List 所不能提供的。

```
@Test
public void testSet() {
    String key="set";
    SetOperations<String, String> set = redisTemplate.opsForSet();
    set.add(key,"it");
    set.add(key,"you");
    set.add(key,"you");
    set.add(key,"know");
    Set<String> values=set.members(key);
    for (String v:values){
        System.out.println("set value :"+v);
    }
}
```

输出结果：

```
set value :it
set value :know
set value :you
```

通过上面的例子我们发现，输入了两个相同的值“you”，全部读取的时候只剩下了一条，说明 Set 对队列进行了自动的排重操作。

Redis 为集合提供了求交集、并集、差集等操作，可以非常方便的使用。

测试 difference

```
SetOperations<String, String> set = redisTemplate.opsForSet();
String key1="setMore1";
String key2="setMore2";
set.add(key1,"it");
set.add(key1,"you");
set.add(key1,"you");
set.add(key1,"know");
set.add(key2,"xx");
set.add(key2,"know");
Set<String> diffs=set.difference(key1,key2);
for (String v:diffs){
    System.out.println("diffs set value :"+v);
}
```

输出结果：

```
diffs set value :it
diffs set value :you
```


根据上面这个例子可以看出，difference() 函数会把 key 1 中不同于 key 2 的数据对比出来，这个特性适合我们在金融场景中对账的时候使用。

测试 unions

```
SetOperations<String, String> set = redisTemplate.opsForSet();
String key3="setMore3";
String key4="setMore4";
set.add(key3,"it");
set.add(key3,"you");
set.add(key3,"xx");
set.add(key4,"aa");
set.add(key4,"bb");
set.add(key4,"know");
Set<String> unions=set.union(key3,key4);
for (String v:unions){
    System.out.println("unions value :"+v);
}
```

输出结果：

```
unions value :know
unions value :you
unions value :xx
unions value :it
unions value :bb
unions value :aa
```

根据例子我们发现，unions 会取两个集合的合集，Set 还有其他很多类似的操作，非常方便我们对集合进行数据处理。

Set 的内部实现是一个 Value 永远为 null 的 HashMap，实际就是通过计算 Hash 的方式来快速排重，这也是 Set 能提供判断一个成员是否在集合内的原因。

ZSet

Redis Sorted Set 的使用场景与 Set 类似，区别是 Set 不是自动有序的，而 Sorted Set 可以通过用户额外提供一个优先级（Score）的参数来为成员排序，并且是插入有序，即自动排序。

在使用 Zset 的时候需要额外的输入一个参数 Score，Zset 会自动根据 Score 的值对集合进行排序，我们可以利用这个特性来做具有权重的队列，比如普通消息的 Score 为1，重要消息的 Score 为 2，然后工作线程可以选择按 Score 的倒序来获取工作任务。

```
@Test
public void testZset(){
    String key="zset";
    redisTemplate.delete(key);
    ZSetOperations<String, String> zset = redisTemplate.opsForZSet();
    zset.add(key,"it",1);
    zset.add(key,"you",6);
    zset.add(key,"know",4);
    zset.add(key,"neo",3);

    Set<String> zsets=zset.range(key,0,3);
    for (String v:zsets){
        System.out.println("zset value :"+v);
    }

    Set<String> zsetB=zset.rangeByScore(key,0,3);
    for (String v:zsetB){
        System.out.println("zsetB value :"+v);
    }
}
```

输出结果：

```
zset value :it
zset value :neo
zset value :know
zset value :you
zsetB value :it
zsetB value :neo
```

通过上面的例子我们发现插入到 Zset 的数据会自动根据 Score 进行排序，根据这个特性我们可以做优先队列等各种常见的场景。另外 Redis 还提供了 rangeByScore 这样的方法，可以只获取 Score 范围内排序后的数据。

Redis Sorted Set 的内部使用 HashMap 和跳跃表（SkipList）来保证数据的存储和有序，HashMap 里放的是成员到 Score 的映射，而跳跃表里存放的是所有的成员，排序依据是 HashMap 里存的 Score，使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

封装

在我们实际的使用过程中，不会给每一个使用的类都注入 redisTemplate 来直接使用，一般都会对业务进行简单的包装，最后提供出来对外使用。

我们举两个例子说明。

首先定义一个 RedisService 服务，将 RedisTemplate 注入到类中。

```

@Service
public class RedisService {
    @Autowired
    private RedisTemplate redisTemplate;
}

```

封装简单插入操作：

```

public boolean set(final String key, Object value) {
    boolean result = false;
    try {
        ValueOperations<Serializable, Object> operations = redisTemplate.opsForValue();
        operations.set(key, value);
        result = true;
    } catch (Exception e) {
        logger.error("set error: key {}, value {}",key,value,e);
    }
    return result;
}

```

会对其中出现的异常继续处理，反馈给调用方。

比如我们想删除某一类的 Key 的值。

```

public void removePattern(final String pattern) {
    Set<Serializable> keys = redisTemplate.keys(pattern);
    if (keys.size() > 0)
        redisTemplate.delete(keys);
}

```

使用 Redis 的 Pattern 来匹配出一批符合条件的缓存，然后批量进行删除。

还有其他封装方法，比如删除的时候先判断 Key 是否存在等，这些简单的业务判断都应该封装在 RedisService，对外提供最简单的 API 调用即可。

```

@Autowired
private RedisService redisService;

@Test
public void testString() throws Exception {
    redisService.set("neo", "ityouknow");
    Assert.assertEquals("ityouknow", redisService.get("neo"));
}

```

在其他服务使用的时候将 RedisService 注入其中，调用对应的方法来操作 Redis，这样会更优雅简单一些。

总结

Redis 是一款非常优秀的高性能缓存中间件，被广泛的使用在各互联网公司中，Spring Boot 对 Redis 的操作提供了很多支持，可以非常方便的去集成。Redis 拥有丰富的数据类型，方便我们在不同的业务场景中去使用，特别是提供了很多内置的高效集合操作，在业务中使用非常方便。

[点击这里下载源码。](#)