11-redis和缓存

1.redis数据结构与应用场景

一、简介

Redis 是开源免费， key-value 内存数据库，主要解决高并发、大数据场景下，热点数据访问的性能问题，提供高性能的数据快速访问。项目中部分数据访问比较频繁，对下游 DB（例如 MySQL）造成服务压力，这时候可以使用缓存来提高效率。

Redis 的主要特点包括：

- Redis数据存储在内存中，可以提高热点数据的访问效率
- Redis 除了支持 key-value 类型的数据，同时还支持其他多种数据结构的存储；
- Redis 支持数据持久化存储，可以将数据存储在磁盘中，机器重启数据将从磁盘重新加载数据；

Redis 作为缓存数据库和 MySQL 这种结构化数据库进行对比。

- 从数据库类型上，Redis 是 NoSQL 半结构化缓存数据库， MySQL 是结构化关系型数据库；
- 从读写性能上，MySQL 是持久化硬盘存储，读写速度较慢， Redis 数据存储读取都在内存，同时也可以持久化到磁盘，读写速度较快；
- 从使用场景上，Redis 一般作为 MySQL 数据读取性能优化的技术选型，彼此配合使用。Redis用于存储热数据或者缓存数据，并不存在相互替换的关系。

二、Redis 基本数据结构与实战场景

1. redis的数据结构可以理解为Java数据类型中的Map<String,Object>,key是String类型，value是下面的类型。只不过作为一个独立的数据库单独存在，所以Java中的Map怎么用，redis就怎么用，大同小异。
2. 字符串类型的数据结构可以理解为Map<String,String>
3. list类型的数据结构可以理解为Map<String,List<String>>
4. set类型的数据结构可以理解为Map<String,Set<String>>
5. hash类型的数据结构可以理解为Map<String,HashMap<String,String>>

序号	数据结构	常用命令	命令实例
1	字符串	1.set： 设置 key 对应的 value 值 2.get： 获取对应 key 的值，如不存在返回 nil 3.setnx： 只有设置的值不存在，才设置 4.setex： 设置键值，并指定对应的有效期,SETEX key seconds value 5.mset/mget： 一次设置/获取多个 key 的值 6.incr/decr： 对 key 值进行增加 / 减去 1 操作	1.set name “tom” 2.get name ------> 结果： tom 3.setnx name “jim” 4.setex name 10 “tom” 5.mset key1 “hh” key2 “kk” 6. incr/decr key1 ---------->+1/-1
2	list	1.lpush/rpush： 在 key 所对应的 list 左 / 右部添加一个元素 2.lrang/lindex： 获取列表给定范围 / 位置的所有值 3.lset： 设置 list 中指定下表元素的值	1.lpush list*name value1; rpush list*name value2 2.lrang list*name 0 -1 获取列表所有元素* *3.lset list*name 1 valuex
3	set	1.sadd： 向名称 为key 的 set 添加元素 2.smembers： 查看集合中的所有成员 3.spop： 随机返回并删除 set 中一个元素 4.sdiff： 返回所有 set 与第一个 set 的差集 5.sunion： 返回给定集合并集	1.sadd wordset aa; sadd wordiest bb; 2.smembers wordset 3.spop wordset 4.sdiff wordset wordset1 5.sunion wordset wordset1
4	hash	1.hset： 设置一个 hash 的 field 的指定值，如果 key 不存在先创建 2.hget： 获取某个 hash 的某个 filed 值 3.hmset/hmget： 批量设置 / 获取 hash 内容 4.hlen： 返回 hash 表中 key 的数量 5.hkeys/hvals： 返回 hash 表中所有的 key/value	1.hset user name “tom" 2.hget user name 3.hmget user name sex 4.hlen user 5.hkeys user / hvals user
5	Sorted set	1.zadd： 将一个带有给定分值的成员添加到有序集合里面 2.zrange： 取出集合中的元素 3.zcard： 返回集合中所有元素的个数	1.zadd key 1 hello 2.zrang key 0 -1 3.zcard key

上图中命令行更正：lrange，不是lrang

三、 redis应用场景解析

3.1 String 类型使用场景

场景一：商品库存数

从业务上，商品库存数据是热点数据，交易行为会直接影响库存。而 Redis 自身 String 类型提供了：

```bash
incr key      #增加一个库存
decr key     # 减少一个库存
incrby key 10 # 增加20个库存
decrby key 15   # 减少15个库存
```

- set goods_id 10; 设置 id 为 good_id 的商品的库存初始值为 10；
- decr goods_id; 当商品被购买时候，库存数据减 1。

依此类推的场景：商品的浏览次数，问题或者回复的点赞次数等。这种计数的场景都可以考虑利用 Redis 来实现。

场景二：时效信息存储

Redis 的数据存储具有自动失效能力。也就是存储的 key-value 可以设置过期时间，SETEX mykey 60 "value"中的第2个参数就是过期时间。

比如，用户登录某个 App 需要获取登录验证码， 验证码在 30 秒内有效。

- 生成验证码：生成验证码并使用 String 类型在reids存储验证码，同时设置 30 秒的失效时间。如：SETEX validcode 30 "value"
- 验证过程：用户获得验证码之后，我们通过get validcode获取验证码，如果获取不到说明验证码过期了。

3.2 List 类型使用场景

list 是按照插入顺序排序的字符串链表。可以在头部和尾部插入新的元素（双向链表实现，两端添加元素的时间复杂度为 O(1)） 。

场景一：消息队列实现

目前有很多专业的消息队列组件 Kafka、RabbitMQ 等。 我们在这里仅仅是使用 list 的特征来实现消息队列的要求。在实际技术选型的过程中，大家可以慎重思考。

list 存储就是一个队列的存储形式：

- lpush key value; 在 key 对应 list 的头部添加字符串元素；
- rpop key; 移除列表的最后一个元素，返回值为移除的元素。

场景二：最新上架商品

在交易网站首页经常会有新上架产品推荐的模块， 这个模块是存储了最新上架前 100 名。这时候使用 Redis 的 list 数据结构，来进行 TOP 100 新上架产品的存储。

Redis ltrim 指令对一个列表进行修剪（trim），这样 list 就会只包含指定范围的指定元素。

```Bash
ltrim key start end
```

start 和 end 都是由 0 开始计数的，这里的 0 是列表里的第一个元素（表头），1 是第二个元素。

如下伪代码演示：

```Bash
//把新上架商品添加到链表里
ret = r.lpush("new:goods", goodsId)
//保持链表 100 位
ret = r.ltrim("new:goods", 0, 99)
//获得前 100 个最新上架的商品 id 列表
newest_goods_list = r.lrange("new:goods", 0, 99)
```

3.3 set 类型使用场景

set 也是存储了一个集合列表功能。和 list 不同，set 具备去重功能(和Java的Set数据类型一样)。当需要存储一个列表信息，同时要求列表内的元素不能有重复，这时候使用 set 比较合适。与此同时，set 还提供的交集、并集、差集。

例如，在交易网站，我们会存储用户感兴趣的商品信息，在进行相似用户分析的时候， 可以通过计算两个不同用户之间感兴趣商品的数量来提供一些依据。

```bash
//userid 为用户 ID ， goodID 为感兴趣的商品信息。
sadd "user:userId" goodID

sadd "user:101" 1
sadd "user:101" 2
sadd "user:102" 1
Sadd "user:102" 3

sinter "user:101" "user:102"    # 返回值是1
```

获取到两个用户相似的产品， 然后确定相似产品的类目就可以进行用户分析。类似的应用场景还有， 社交场景下共同关注好友， 相似兴趣 tag 等场景的支持。

3.4 Hash 类型使用场景

Redis 在存储对象（例如：用户信息）的时候需要对对象进行序列化转换然后存储，还有一种形式，就是将对象数据转换为 JSON 结构数据，然后存储 JSON 的字符串到 Redis。

对于一些对象类型，还有另外一种比较方便的类型，那就是按照 Redis 的 Hash 类型进行存储。

```bash
hset key field value
```

例如，我们存储一些网站用户的基本信息， 我们可以使用：

```bash
hset user101 name "小明"
hset user101 phone "123456"
hset user101 sex "男"
```

这样就存储了一个用户基本信息，存储信息有：{name : 小明， phone : “123456”，sex : “男”}

当然这种类似场景还非常多， 比如存储订单的数据，产品的数据，商家基本信息等。大家可以参考来进行存储选型。但是不适合存储关联关系比较复杂的数据，那种场景还得用关系型数据库比较方便。

3.5 Sorted Set 类型使用场景

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 可以通过提供一个 score 参数来为存储数据排序，并且是自动排序，插入既有序。**业务中如果需要一个有序且不重复的集合列表，就可以选择 sorted set 这种数据结构。**

比如：商品的购买热度可以将购买总量 num 当做商品列表的 score，这样获取最热门的商品时就是可以自动按售卖总量排好序。

2.整合单例模式

redis集群模式和哨兵模式高可用的安装与运维，需要去专门的redis课程学习。这里主要面向Spring Boot整合redis来开发，不涉及redis集群高可用及运维知识。

一、spring-data-redis简介

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， spring-boot-starter-data-redis 为我们隔离了其中的差异性。

1. 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。
2. 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。
3. 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。

二、整合spring data redis

引入依赖包

```xml
<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全局配置redis的单节点实例：

```yaml
spring:
  redis:
    database: 0     # Redis 数据库索引（默认为 0）
    host: 127.0.0.1 # Redis 服务器地址
    port: 6379 # Redis 服务器连接端口
    password: 123456 # Redis 服务器连接密码（默认为空）
    timeout:  5000  # 连接超时，单位ms
    lettuce:
      pool:
        max-active: 8 # 连接池最大连接数（使用负值表示没有限制） 默认 8
        max-wait: -1 # 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
        max-idle: 8 # 连接池中的最大空闲连接 默认 8
        min-idle: 0 # 连接池中的最小空闲连接 默认 0
```

3.使用redisTemplate操作数据

一、redis模板封装类

RedisTemplate 的封装使我们能够更方便的进行redis数据操作，比直接使用Jedis或者Lettuce的java SDK要方便很多。RedisTemplate作为java 操作redis数据库的API模板更通用，可以操作所有的redis数据类型。

```java
// 注入RedisTemplate，更通用
@Resource
private RedisTemplate<String, Object> redisTemplate;

ValueOperations<String,Object> ValueOperations = redisTemplate.opsForValue();//操作字符串
HashOperations<String, String, Object> hashOperations = redisTemplate.opsForHash();//操作 hash
ListOperations<String, Object> listOperations = redisTemplate.opsForList();//操作 list
SetOperations<String, Object> setOperations = redisTemplate.opsForSet();//操作 set
ZSetOperations<String, Object> zSetOperations = redisTemplate.opsForZSet();//操作有序 set
```

ListOperations、ValueOperations、HashOperations、SetOperations、ZSetOperations等都是针对专有数据类型进行操作，使用起来更简洁。

```java
@Resource(name = "redisTemplate")
private ValueOperations<String,Object> valueOperations;   //以redis string类型存取Java Object(序列化反序列化)

@Resource(name = "redisTemplate")
private HashOperations<String, String, Object> hashOperations; //以redis的hash类型存储java Object

@Resource(name = "redisTemplate")
private ListOperations<String, Object> listOperations; //以redis的list类型存储java Object

@Resource(name = "redisTemplate")
private SetOperations<String, Object> setOperations;   //以redis的set类型存储java Object

@Resource(name = "redisTemplate")
    private ZSetOperations<String, Object> zSetOperations;  //以redis的zset类型存储java Object
```

二、基础数据Java类

为了方便后面写代码解释API的使用方法，写测试用例。我们需要先准备数据对象Person，注意要实现Serializable接口，为什么一定要实现这个接口？我们下文解释。

```java
@Data
public class Person implements Serializable {

  private static final long serialVersionUID = -8985545025228238754L;

  String id;
  String firstname;
  String lastname;
  Address address;   //注意这里，不是基础数据类型

  public Person(String firstname, String lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
}
```

准备数据对象Address

```java
@Data
public class Address implements Serializable {

  private static final long serialVersionUID = -8985545025228238771L;

  String city;
  String country;

  public Address(String city, String country) {
    this.city = city;
    this.country = country;
  }
}
```

三、StringRedisTemplate

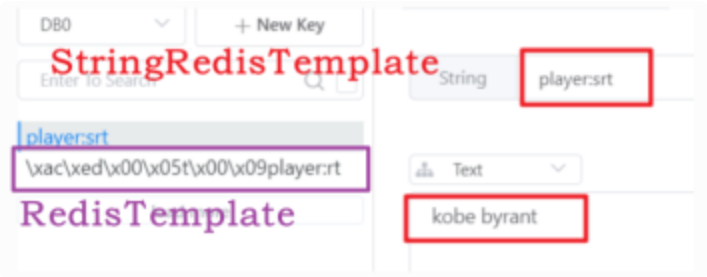
除了RedisTemplate模板类，还有另一个模板类叫做StringRedisTemplate 。二者都提供了用来操作redis数据库的API。

```java
@SpringBootTest
public class RedisConfigTest {

    @Resource
    private StringRedisTemplate stringRedisTemplate;   //以String序列化方式保存数据的通用模板类

    @Resource
    private RedisTemplate<String, Person> redisTemplate;   //默认以JDK二进制方式保存数据的通用模板类

    @Test
    public void stringRedisTemplate() {
        Person person = new Person("zhang","san");
        person.setAddress(new Address("南京","中国"));
        //将数据存入redis数据库
        stringRedisTemplate.opsForValue().set("player:srt","zhangsan",20,TimeUnit.SECONDS);
        redisTemplate.opsForValue().set("player:rt",person,20,TimeUnit.SECONDS);
    }
}
```

二者的区别在于

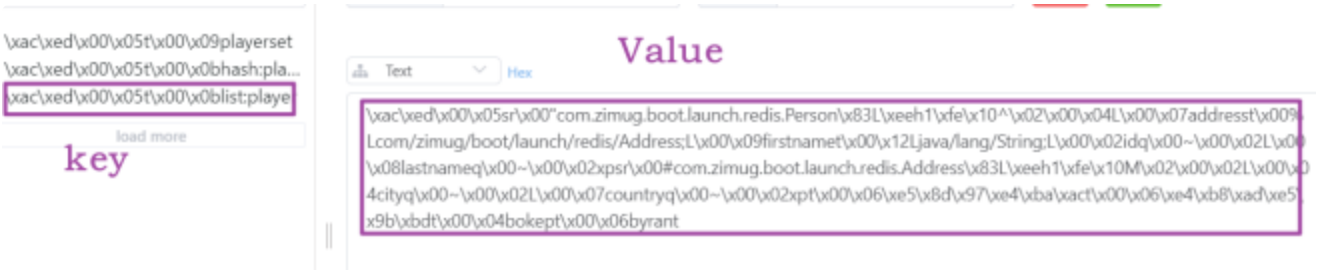
- 操作的数据类型不同，以List类型为例：RedisTemplate操作List<Object>,StringRedisTemplate操作List<String>
- 序列化数据的方式不同，RedisTemplate使用的是JdkSerializationRedisSerializer 存入数据会将数据先序列化成字节数组然后在存入Redis数据库。 StringRedisTemplate使用的是StringRedisSerializer

redis持久化的java数据类为什么要实现Serializable接口？因为RedisTemplate默认使用的是JdkSerializationRedisSerializer，也就是使用Java JDK默认的序列化方式存储数据。如果不实现Serializable接口，JDK序列化就会报错，这是java基础知识。如果我们可以不使用JDK默认的序列化方式，就不需要实现这个Serializable接口。



需要注意的是因为RedisTemplate和StringRedisTemplate的默认序列化存储方式不一样，所以二者存储的数据并不能通用。也就是说RedisTemplate存的数据只能用RedisTemplate去取，对于StringRedisTemplate也是一样。

四、解决key-value乱码问题



其实这个不是严格意义上的乱码，是JDK的二进制序列化之后的存储方式。

如何解决？看下文的配置类代码

- 采用StringRedisSerializer对key进行序列化（字符串格式）
- 采用Jackson2JsonRedisSerializer对value将进行序列化（JSON格式）

```java
@Configuration
public class RedisConfig {

    @Bean
    public RedisTemplate redisTemplate(RedisConnectionFactory redisConnectionFactory) {
        RedisTemplate redisTemplate = new RedisTemplate();
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);

        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);

 objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);

        jackson2JsonRedisSerializer.setObjectMapper(objectMapper);

        //重点在这四行代码
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());

 redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);

        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }
}
```

乱码问题的症结在于对象的序列化问题：RedisTemplate默认使用的是JdkSerializationRedisSerializer（二进制存储），StringRedisTemplate默认使用的是StringRedisSerializer（redis字符串格式存储）。

序列化方式对比：

- JdkSerializationRedisSerializer: 使用JDK提供的序列化功能。 优点是反序列化时不需要提供类型信息(class)，但缺点是需要实现Serializable接口，还有序列化后的结果非常庞大，是JSON格式的5倍左右，这样就会消耗redis服务器的大量内存。而且是以二进制形式保存，自然人无法理解。
- Jackson2JsonRedisSerializer： 使用Jackson库将对象序列化为JSON字符串。优点是速度快，序列化后的字符串短小精悍，不需要实现Serializable接口。似乎没啥缺点。
- StringRedisSerializer序列化之后的结果，自然人也是可以理解，但是value只能是String类型，不能

是Object。

五、使用redisTemplate存取redis各种数据类型

下面的各种数据类型操作的api和redis命令行api的含义几乎是一致的。

Java | 复制代码

```java
@SpringBootTest
public class RedisConfigTest2 {

    @Resource(name = "redisTemplate")
    private ValueOperations<String,Object> valueOperations;   //以redis string类型存取Java Object(序列化反序列化)

    @Resource(name = "redisTemplate")
    private HashOperations<String, String, Object> hashOperations; //以redis的hash类型存储java Object

    @Resource(name = "redisTemplate")
    private ListOperations<String, Object> listOperations; //以redis的list类型存储java Object

    @Resource(name = "redisTemplate")
    private SetOperations<String, Object> setOperations;   //以redis的set类型存储java Object

    @Resource(name = "redisTemplate")
    private ZSetOperations<String, Object> zSetOperations;  //以redis的zset类型存储java Object


    @Test
    public void testValueObj() {
        Person person = new Person("张","三");
        person.setAddress(new Address("南京","中国"));
        //向redis数据库保存数据(key,value),数据有效期20秒
        valueOperations.set("player:1",person,20, TimeUnit.SECONDS); //20秒之后数据消失
        //根据key把数据取出来
        Person getBack = (Person)valueOperations.get("player:1");
        System.out.println(getBack);
    }

    @Test
    public void testSetOperation() {
        Person person = new Person("zhang","san");
        Person person2 = new Person("张","三");

        setOperations.add("playerset",person,person2);  //向Set中添加数据项
        //members获取Redis Set中的所有记录
        Set<Object> result = setOperations.members("playerset");
        System.out.println(result);  //包含kobe和curry的数组
```

```
40        }
41
42        @Test
43        public void HashOperations() {
44            Person person = new Person("kobe","byrant");
45            //使用hash的方法存储对象数据（一个属性一个属性的存，下节教大家简单的方法）
46
      hashOperations.put("hash:player","firstname",person.getFirstname());
47
      hashOperations.put("hash:player","lastname",person.getLastname());
48            hashOperations.put("hash:player","address",person.getAddress());
49            //取出一个对象的属性值，有没有办法一次将整个对象取出来？有，下节介绍
50            String firstName =
     (String)hashOperations.get("hash:player","firstname");
51            System.out.println(firstName);   //kobe
52        }
53
54        @Test
55        public void  ListOperations() {
56            //将数据对象放入队列
57            listOperations.leftPush("list:player",new Person("张","三"));
58            listOperations.leftPush("list:player",new Person("张","三丰"));
59            listOperations.leftPush("list:player",new Person("张","三风"));
60            //从左侧存，再从左侧取，所以取出来的数据是后放入的curry
61            Person person = (Person) listOperations.leftPop("list:player");
62            System.out.println(person); //curry对象
63        }
64    }
```

4.使用Redis Repository操作数据

通过集成spring-boot-starter-data-redis之后一共有三种redis hash数据操作方式可以供我们选择

- 一个属性、一个属性的存取
- 使用Jackson2HashMapper存取对象
- 使用RedisRepository的对象操作（本节核心内容）

一、一个属性、一个属性的存取

这种方式在上一节中的代码，已经得以体现。

```java
@Test
public void HashOperations() {
    Person person = new Person("zhang","san");
    person.setAddress(new Address("南京","中国"));
    //使用hash的方法存储对象数据（一个属性一个属性的存，下节教大家简单的方法）
    hashOperations.put("hash:player","firstname",person.getFirstname());
    hashOperations.put("hash:player","lastname",person.getLastname());
    hashOperations.put("hash:player","address",person.getAddress());

    String firstName =
(String)hashOperations.get("hash:player","firstname");
    System.out.println(firstName);
}
```

- 一个hash代表一个对象的数据
- 一个对象有多个属性key、value键值对数据，每一组键值对都可以单独存取

二、使用Jackson2HashMapper存取对象

上一小节我们操作hash对象的时候是一个属性一个属性设置的，那我们有没有办法将对象一次性hash入库呢？可以使用jacksonHashOperations和Jackson2HashMapper

```java
@SpringBootTest
@ExtendWith(SpringExtension.class)
public class RedisConfigTest3 {

    @Resource(name="redisTemplate")
    private HashOperations<String, String, Object> jacksonHashOperations;

    //注意这里的false，下文会讲解
    private HashMapper<Object, String, Object> jackson2HashMapper = new Jackson2HashMapper(false);

    @Test
    public void testHashPutAll(){

        Person person = new Person("zhang","san");
        person.setId("zhang");
        person.setAddress(new Address("洛杉矶","美国"));

        //将对象以hash的形式放入redis数据库
        Map<String,Object> mappedHash = jackson2HashMapper.toHash(person);
        jacksonHashOperations.putAll("player:" + person.getId(), mappedHash);

        //将对象从数据库取出来
        Map<String,Object> loadedHash = jacksonHashOperations.entries("player:" + person.getId());
        Object map = jackson2HashMapper.fromHash(loadedHash);
        Person getback = new ObjectMapper().convertValue(map,Person.class);

        //Junit5,验证放进去的和取出来的数据一致
        assertEquals(person.getFirstname(),getback.getFirstname());
    }
}
```

使用这种方式可以一次性存取 Java 对象为redis数据库的hash数据类型。需要注意的是：执行上面的测试用例，Person和Address一定要有public无参构造方法，在将map转换成Person或Address对象的时候用到，如果没有的话会报错。

三、使用RedisRepository的对象操作

使用RedisRepository进行redis数据操作，它不只是能简单地存取数据，还可以做很多CURD操作。使用起来和用JPA进行关系型数据库的单表操作，几乎是一样的。

首先，我们需要在需要操作的java实体类上面加上@RedisHash注解，并使用@Id为该实体类指定id。

Java | 复制代码

```java
@RedisHash("people")   //注意这里的person，下文会说明
public class Person {
  @Id
  String id;

  //其他和上一节代码一样

}
```

然后写一个PersonRepository ，继承CrudRepository，是不是也和JPA差不多?

Java | 复制代码

```java
//泛型第二个参数是id的数据类型
public interface PersonRepository extends CrudRepository<Person, String>
{
 // 继承CrudRepository，获取基本的CRUD操作
}
```

CrudRepository默认为我们提供了下面的这么多方法，我们直接调用就可以了。

```
count()                                                  long
findAll()                                  Iterable<Person>
delete(Person t)                                         void
deleteAll()                                              void
deleteAll(Iterable<? extends Person> iterable)           void
deleteById(String id)                                    void
existsById(String id)                                 boolean
findAllById(Iterable<String> iterable)     Iterable<Person>
findById(String id)                        Optional<Person>
save(S s)                                                   S
saveAll(Iterable<S> iterable)                  Iterable<S>
```

然后进行下面的测试

```java
@SpringBootTest
@ExtendWith(SpringExtension.class)
public class RedisRepositoryTest {

    @Resource
    PersonRepository personRepository;

    @Test
    public void test(){

        Person rand = new Person("zhang", "san");
        rand.setAddress(new Address("南京", "中国"));
        //存
        personRepository.save(rand);

        //取
        Optional<Person> op = personRepository.findById(rand.getId());
        Person p2 = op.get();

        //统计Person的数量
        personRepository.count();
        //删除person对象rand
        personRepository.delete(rand);

    }

}
```

测试结果：需要注意的是RedisRepository在存取对象数据的时候，实际上使用了redis的2种数据类型

- 第一种是Set类型，用于保存每一个存入redis的对象（Person）的id。我们可以利用这个Set实现person对象集合类的操作，比如说：count()统计redis数据库中一共保存了多少个person。
- 第二种是Hash类型，是用来保存Java对象的，id是RedisRepository帮我们生成的。

5.spring cache缓存基本用法

一、为什么要做缓存

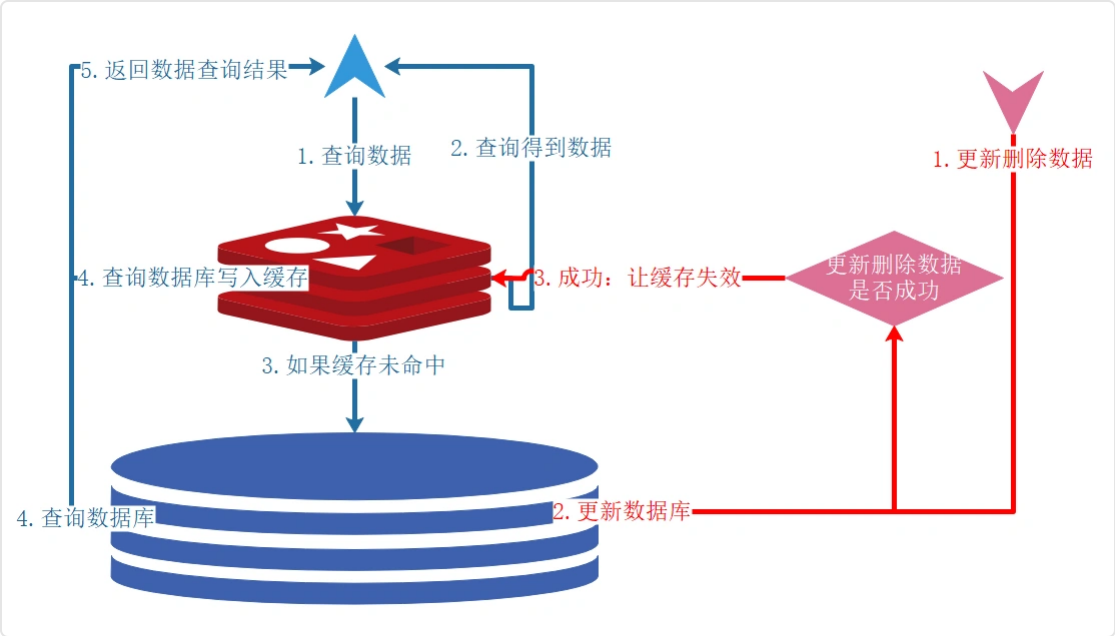
- 提升性能绝大多数情况下，关系型数据库select查询是出现性能问题最大的地方。一方面，select 会有很多像 join、group、order、like 等这样丰富的语义，而这些语义是非常耗性能的；另一方面，大

多数应用都是读多写少，所以加剧了慢查询的问题。分布式系统中远程调用也会耗很多性能，因为有网络开销，会导致整体的响应时间下降。为了挽救这样的性能开销，在业务允许的情况（不需要太实时的数据）下，使用缓存是非常必要的事情。

- 缓解数据库压力当用户请求增多时，数据库的压力将大大增加，通过缓存能够大大降低数据库的压力。

二、常用缓存操作流程

使用缓存最关键的一点就是保证：缓存与数据库的数据一致性，该怎么去做？下图是一种最常用的缓存操作模式，来保证数据一致性。



- **更新写数据**：先把数据存到数据库中，然后再让缓存**失效或更新**。缓存操作失败，数据库事务回滚。
- **删除写数据**: 先从数据库里面删掉数据，再从缓存里面删掉。缓存操作失败，数据库事务回滚。
- **查询读数据**
  - **缓存命中**：先去缓存 cache 中取数据，取到后返回结果。
  - **缓存失效**：应用程序先从 cache 取数据，没有得到，则从数据库中取数据，成功后，在将数据放到缓存中。

如果上面的这些更新、删除、查询操作流程全都由程序员通过编码来完成的话

- 因为加入缓存层，程序员的编码量大大增多
- 缓存层代码和业务代码耦合，造成难以维护的问题。

三、整合Spring Cache

我们可以使用Spring cache解决上面遇到的两个问题，Spring cache通过注解的方式来操作缓存，一定程度上减少了程序员缓存操作代码编写量。注解添加和移除都很方便，不与业务代码耦合，容易维护。

第一步：pom.xml 添加 Spring Boot 的 jar 依赖：

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

第二步：添加入口启动类 @EnableCaching 注解开启 Caching

在Spring Boot中通过@EnableCaching注解自动化配置合适的缓存管理器（CacheManager），Spring Boot根据下面的顺序去侦测缓存提供者，也就是说Spring Cache支持下面的这些缓存框架：

- Generic
- JCache (JSR-107) (EhCache 3, Hazelcast, Infinispan, and others)
- EhCache 2.x
- Hazelcast
- Infinispan
- Couchbase
- Redis
- Caffeine
- Simple

四、在ArticleController类上实现一个简单的例子

第一次访问走数据库，第二次访问就走缓存了，可以自己打日志试一下。

```java
@Cacheable(value="article")
@GetMapping( "/article/{id}")
public @ResponseBody  AjaxResponse getArticle(@PathVariable Long id) {

}
```

使用redis缓存，被缓存的对象（函数返回值）有几个非常需要注意的点：

1. 必须实现无参的构造函数
2. 需要实现Serializable 接口和定义serialVersionUID （因为缓存需要使用JDK的方式序列化和反序列化）。

五、更改Redis缓存的序列化方式

让缓存使用JDK默认的序列化和反序列化方式非常不友好，我们可以修改为使用JSON序列化与反序列化的方式，可读性更强，体积更小，速度更快。

Java | 复制代码

```java
@Configuration
public class RedisConfig {
    //这个函数是上一节的内容
    @Bean
    public RedisTemplate redisTemplate(RedisConnectionFactory redisConnectionFactory) {
        RedisTemplate redisTemplate = new RedisTemplate();
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);

        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);

        objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);

        jackson2JsonRedisSerializer.setObjectMapper(objectMapper);

        //重点在这四行代码
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);

        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }

     //本节的重点配置，让Redis缓存的序列化方式使用redisTemplate.getValueSerializer()
    //不在使用JDK默认的序列化方式
    @Bean
    public RedisCacheManager redisCacheManager(RedisTemplate redisTemplate) {
        RedisCacheWriter redisCacheWriter = RedisCacheWriter.nonLockingRedisCacheWriter(redisTemplate.getConnectionFactory());
        RedisCacheConfiguration redisCacheConfiguration = RedisCacheConfiguration.defaultCacheConfig()

.serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(redisTemplate.getValueSerializer()));
        return new RedisCacheManager(redisCacheWriter, redisCacheConfiguration);
```

```
34        }
35    }
```