RedisTemplate和StringRedisTemplate的入门使用讲解

如果你的业务不这么复杂，比较简单要求快速开发可以使用 SpringCache它是基于spring-boot-starter-data-redis的依赖开发的框架具体如何使用看我这篇博客https://blog.csdn.net/weixin\_44012722/article/details/105655417

但是如果你的业务比较复杂，涉及SpringCache不能解决的，建议redisRemplate，StringRedisTemplate,Jedis三者中选择

1. 首先我们先了解一下三者区别（这样子我们才可以做出选择）

Jedis是Redis官方推荐的面向Java的操作Redis的客户端，而RedisTemplate,StringRedisTemplate是SpringDataRedis中对JedisApi的高度封装。SpringDataRedis相对于Jedis来说可以方便地更换Redis的Java客户端，比Jedis多了自动管理连接池的特性，方便与其他Spring框架进行搭配使用如：SpringCache，而RedisTemplate,StringRedisTemplate之间的区别：

两者的数据是不共通的；也就是说StringRedisTemplate只能管理StringRedisTemplate里面的数据，RedisTemplate只能管理RedisTemplate中的数据。

其实他们两者之间的区别主要在于他们使用的序列化类:

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

使用时注意事项：

当你的redis数据库里面本来存的是字符串数据或者你要存取的数据就是字符串类型数据的时候，那么你就使用StringRedisTemplate即可。

但是如果你的数据是复杂的对象类型，而取出的时候又不想做任何的数据转换，直接从Redis里面取出一个对象，那么使用RedisTemplate是更好的选择。

下面我将写例子去讲解如何使用这三个框架的API不了解Redis先移步这里https://blog.csdn.net/weixin\_44012722/article/details/105639663

三个框架的API我将以测试类中完成演示 不会SpringBoot项目创建测试类请移步这里https://blog.csdn.net/weixin\_44012722/article/details/105671118

2.RedisTemplate的使用

所需要的依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

1

2

3

4

配置文件

server:

port: 8083

spring:

redis:

database: 0

host: localhost

1

2

3

4

5

6

7

注意如果你的RedisTemplate省略泛型，存入数据库的字符串前面就会有ASCII码的东西，原因上面说了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。 所以RedisTemplate最好操作的数据是java对象。

2.0Reids的Key的操作

public class MyTest0 extends ParentTest{

//因为redistempalte默认装配了泛型为<Object,Object>的redistemplate,

//所以<String,String>也是可以注入的，因为string是object派生类 。

//如果需要要<String,Object>的redistemplate则需要自己装配

@Autowired

private RedisTemplate<String,String> redisTemplate;

@Test

public void test1(){

//给指定键设置过期时间

redisTemplate.expire("lzj",12, TimeUnit.SECONDS);

//删除指定键

redisTemplate.delete("lzj");

//查找 指定的 键

redisTemplate.keys("\*");

//判断是否存在键值

redisTemplate.hasKey("lzj");

//获取过期时间

redisTemplate.getExpire("lzj");

//获取指定格式的过期时间

redisTemplate.getExpire("lzj",TimeUnit.SECONDS);

//获取当前传入的key的值序列化为byte[]类型

redisTemplate.dump("lzj");

//修改指定键的名字 如果该键不存在则报错

redisTemplate.rename("lzj","lyx");

//旧值存在时，将旧值改为新值

redisTemplate.renameIfAbsent("lzj","lyx");

//获取指定键的类型

redisTemplate.type("lzj");

//将指定的键移动到指定的库中

redisTemplate.move("lzj",2);

//随机取一个key

redisTemplate.randomKey();

//将key持久化保存 就是把过期或者设置了过期时间的key变为永不过期

redisTemplate.persist("lzj");

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

2.1Redis的String数据结构

public class MyTest extends ParentTest{

@Autowired

private RedisTemplate<String,String> redisTemplate;

@Test

public void test1(){

//存入数据

redisTemplate.opsForValue().set("name","lzj");

//获取数据

System.out.println(redisTemplate.opsForValue().get("name"));

//获取多个数据

System.out.println(redisTemplate.opsForValue().multiGet(Arrays.asList("lzj","lyx","3333")));

//存入数据并且设置过期时间

redisTemplate.opsForValue().set("num","123",10, TimeUnit.SECONDS);

//给指定键值 4 偏移量的位置开始替换内容

redisTemplate.opsForValue().set("name","lzj",2);

//设置键的字符串值并返回其旧值

System.out.println(redisTemplate.opsForValue().getAndSet("name","lyx"));

//给指定键 的值追加字符串

redisTemplate.opsForValue().append("test","Hello");

//存入数据 如果不存在则存入数据返回true 否则不覆盖数据返回false

System.out.println(redisTemplate.opsForValue().setIfAbsent("lzj","1234"));

//存入数据并设置过期时间 如果不存在则存入数据返回true 否则不覆盖数据返回false

System.out.println(redisTemplate.opsForValue().setIfAbsent("lzj","1234",200,TimeUnit.SECONDS));

//存入数据 如果存在键则覆盖数据 返回true 不存在则不作任何操作 返回false

System.out.println(redisTemplate.opsForValue().setIfPresent("lyx","1234"));

//存入数据并设置过期时间 如果存在键则覆盖数据 返回true 不存在则不作任何操作 返回false

System.out.println(redisTemplate.opsForValue().setIfPresent("lyx","1234",200,TimeUnit.SECONDS));

}

@Test

public void test2(){

Map<String,String> map = new HashMap<>();

map.put("1","123");

map.put("2","123");

//多个键值的插入

redisTemplate.opsForValue().multiSet(map);

//多个键值的插入 如果不存在则存入数据返回true 否则不覆盖数据返回false

redisTemplate.opsForValue().multiSetIfAbsent(map);

}

@Test

public void test3(){

//返回键的值的长度

System.out.println(redisTemplate.opsForValue().size("lzj"));

System.out.println(redisTemplate.opsForValue().multiGet(Arrays.asList("lzj","lyx","3333")));

//给指定键 加1如果值不是数字则抛出异常 不存在指定键创建一个初始为0的加1 增加成功则返回增加后的值

System.out.println(redisTemplate.opsForValue().increment("lzj"));

//给指定键 加指定整数如果值不是数字则抛出异常 不存在指定键创建一个初始为0的加指定整数 增加成功则返回增加后的值

System.out.println(redisTemplate.opsForValue().increment("1",1));

//给指定键 加指定浮点数如果值不是数字则抛出异常 不存在指定键创建一个初始为0的加指定浮点数 增加成功则返回增加后的值

System.out.println(redisTemplate.opsForValue().increment("1",1.2));

//给指定键 减1如果值不是数字则抛出异常 不存在指定键创建一个初始为0的减1 减少成功则返回增加后的值

System.out.println(redisTemplate.opsForValue().decrement("1"));

//给指定键 减指定整数如果值不是数字则抛出异常 不存在指定键创建一个初始为0的减指定整数 减少成功则返回增加后的值

System.out.println(redisTemplate.opsForValue().decrement("1",3));

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

2.2Redis的List数据结构

public class MyTest2 extends ParentTest {

@Autowired

private RedisTemplate<String,String> redisTemplate;

@Test

public void test1(){

//存入List数据 做左边推入一个 如果键不存在 则创建一个空的并左推入

redisTemplate.opsForList().leftPush("list1","1");

//存入List数据 做左边推入多个 如果键不存在 则创建一个空的并左推入

redisTemplate.opsForList().leftPushAll("list1","88","999");

//存入List数据 做右边推入一个 如果键不存在 则创建一个空的并右推入

redisTemplate.opsForList().rightPush("list1","3");

//存入List数据 做右边推入多个 如果键不存在 则创建一个空的并右推入

redisTemplate.opsForList().leftPushAll("list1","77","6666");

//返回指定List数据下标的值

System.out.println(redisTemplate.opsForList().index("",2));

//移除2个指定List数据元素内容为1

redisTemplate.opsForList().remove("list1",2,"1");

//左边推出一个

System.out.println(redisTemplate.opsForList().leftPop("list1"));

//移除集合中左边的元素在等待的时间里，如果超过等待的时间仍没有元素则退出。

System.out.println(redisTemplate.opsForList().leftPop("list1",2, TimeUnit.SECONDS));

//右边推出一个

System.out.println(redisTemplate.opsForList().rightPop("list1"));

//移除集合中右边的元素在等待的时间里，如果超过等待的时间仍没有元素则退出。

System.out.println(redisTemplate.opsForList().rightPop("list1",2, TimeUnit.SECONDS));

//给指定键List数据下标为1的元素替换成2

redisTemplate.opsForList().set("list1",1,"2");

//查看指定键List数据元素个数

redisTemplate.opsForList().size("list1");

//获取指定健List数据 从开始到结束下标

redisTemplate.opsForList().range("list1",0,-1).forEach(System.out::println);

//移除列表的最后一个元素，并将该元素添加到另一个列表(如果这另一个List不存在就创建一个空的添加)并返回

redisTemplate.opsForList().rightPopAndLeftPush("list1","list2");

// 从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止

redisTemplate.opsForList().rightPopAndLeftPush("list1","list2",80,TimeUnit.SECONDS);

}

@Test

public void test2(){

//指定键List数据右边推出一个元素

System.out.println(redisTemplate.opsForList().rightPop("list1"));

//指定键List数据右边推出一个元素，如果List元素 等待10秒 10秒内没有元素就不操作，有就推出

System.out.println(redisTemplate.opsForList().rightPop("list1",10, TimeUnit.SECONDS));

//指定键List数据左边推出一个元素，如果List元素 等待10秒 10秒内没有元素就不操作，有就推出

System.out.println(redisTemplate.opsForList().leftPop("list1"));

//指定键List数据左边推出一个元素

System.out.println(redisTemplate.opsForList().leftPop("list1",10, TimeUnit.SECONDS));

//给指定键List数据下标为1的元素替换成2

redisTemplate.opsForList().set("list1",1,"2");

//查看指定键List数据元素个数

redisTemplate.opsForList().size("list1");

//如果存在该键的List数据 则向左推入一个元素 不存在的话不操作

redisTemplate.opsForList().leftPushIfPresent("list1","1");

//如果存在该键的List数据 则向右推入一个元素 不存在的话不操作

redisTemplate.opsForList().rightPushIfPresent("list1","1");

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

2.3Redis的Hash数据结构

public class MyTest3 extends ParentTest {

@Autowired

private RedisTemplate<String,String> redisTemplate;

@Test

public void test1(){

//向指定键推入一个元素（指定键不存在就创建一个空的推入）

redisTemplate.opsForHash().put("hash1","lzj","1234");

Map<String,String> map = new HashMap<>();

map.put("lxy","123445");

map.put("lhm","434564");

//向指定键推入多个元素（指定键不存在就创建一个空的推入）

redisTemplate.opsForHash().putAll("hash1",map);

//向指定键推入一个元素（仅当lzj不存在时才设置）

redisTemplate.opsForHash().putIfAbsent("hash1","lzj","1234");

//获取指定键里面单个元素key为lzj的值

System.out.println(redisTemplate.opsForHash().get("hash1","lzj"));

//获取指定键里面多个元素key为特定的值

redisTemplate.opsForHash().multiGet("hash1",Arrays.asList("lzj","num")).forEach(System.out::println);

//查看指定键内有没有元素的key是lzj的

System.out.println(redisTemplate.opsForHash().hasKey("hash1","lzj"));

//查看键所有元素的Key

redisTemplate.opsForHash().keys("hash1").forEach(System.out::println);

//查看键所有的元素

redisTemplate.opsForHash().entries("hash1").forEach((k,v) -> {System.out.println("k"+k+" \_ "+"v"+v);});

//查看键所有元素的值

redisTemplate.opsForHash().values("hash1").forEach(System.out::println);;

//查看指定键的元素的key为lzj的值的长度

System.out.println(redisTemplate.opsForHash().lengthOfValue("hash1","lzj"));

//查看指定键有多少个元素

System.out.println(redisTemplate.opsForHash().size("hash1"));

//指定键的元素的Key为num的值加整数（如果key不存在创建一个初始为0加整数）

redisTemplate.opsForHash().increment("hash1","num",1);

//指定键的元素的Key为num的值加浮点数（如果key不存在创建一个初始为0加浮点数）

redisTemplate.opsForHash().increment("hash1","num",3.2);

//指定键 根据key值删除元素

redisTemplate.opsForHash().delete("hash","lzj");

//获取集合的游标。通过游标可以遍历整个集合。

Cursor<Map.Entry<Object, Object>> curosr = redisTemplate.opsForHash().scan("hash1", ScanOptions.NONE);

while(curosr.hasNext()){

Map.Entry<Object, Object> entry = curosr.next();

System.out.println(entry.getKey()+":"+entry.getValue());

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

2.4Redis的Set数据结构

public class MyTest4 extends ParentTest {

@Autowired

private RedisTemplate<String,String> redisTemplate;

@Test

public void test1(){

//向键为set1的添加元素1（若没有该键，创建一个新的，并加入元素）

redisTemplate.opsForSet().add("set1","1");

//查询指定键的包含所有元素

System.out.println(redisTemplate.opsForSet().members("set1"));

//查询指定键的包含元素个数

System.out.println(redisTemplate.opsForSet().size("set1"));

//查询指定键是否有该元素

System.out.println(redisTemplate.opsForSet().isMember("set1","1"));

//指定键随机推出一个元素 并返回

System.out.println(redisTemplate.opsForSet().pop("set1"));

//移除指定键里面的指定元素

redisTemplate.opsForSet().remove("set1","1","2");

//将指定键的指定元素移动到指定键中

redisTemplate.opsForSet().move("set1","1","set3");

//获取两个集合的差集

redisTemplate.opsForSet().difference("set1","set2").forEach(System.out::println);

//获取两个集合的差集，并存入一个集合中

redisTemplate.opsForSet().differenceAndStore("set1","set2","set3");

//求指定键与另外一个集合的差集

redisTemplate.opsForSet().difference("set1",Arrays.asList("1","2","3")).forEach(System.out::println);

//求指定键与另外一个集合的差集，并存入一个集合中

redisTemplate.opsForSet().differenceAndStore("set1",Arrays.asList("1","2","3"),"set3");

//获取两个集合的交集

redisTemplate.opsForSet().intersect("set1","set2").forEach(System.out::println);

//获取两个集合的交集，并存入一个集合中

redisTemplate.opsForSet().intersectAndStore("set1","set2","set3");

//求指定键与另外一个集合的交集

redisTemplate.opsForSet().intersect("set1",Arrays.asList("1","2","3"));

//求指定键与另外一个集合的交集，并存入一个集合中

redisTemplate.opsForSet().intersectAndStore("set1",Arrays.asList("1","2","3"),"set3");

//获取两个集合的并集

redisTemplate.opsForSet().union("set1","set2").forEach(System.out::println);

//获取两个集合的并集，并存入一个集合中

redisTemplate.opsForSet().unionAndStore("set1","set2","set3");

//求指定键与另外一个集合的并集

redisTemplate.opsForSet().union("set1",Arrays.asList("1","2","3")).forEach(System.out::println);

//求指定键与另外一个集合的并集，并存入一个集合中

redisTemplate.opsForSet().unionAndStore("set1",Arrays.asList("1","2","3"),"set3");

//随机获取集合中的一个元素

redisTemplate.opsForSet().randomMember("set1");

//随机返回集合中指定数量的元素。随机的元素可能重复

redisTemplate.opsForSet().randomMembers("set1",2);

//随机返回集合中指定数量的元素。随机的元素不会重复

redisTemplate.opsForSet().distinctRandomMembers("set1",2);

//获取集合的游标。通过游标可以遍历整个集合

Cursor<String> curosr = redisTemplate.opsForSet().scan("set1", ScanOptions.NONE);

while(curosr.hasNext()){

String item = curosr.next();

System.out.println(item);

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

2.5Redis的ZSet数据结构

public class MyTest5 extends ParentTest {

@Autowired

private RedisTemplate<String,String> redisTemplate;

@Test

public void test1(){

//向指定键插入元素 和 分数

redisTemplate.opsForZSet().add("zset1","lzj",20.5);

//向指定键插入元素 和 分数

ZSetOperations.TypedTuple<String> objectTypedTuple1 = new DefaultTypedTuple<String>("zset-1",9.6);

ZSetOperations.TypedTuple<String> objectTypedTuple2 = new DefaultTypedTuple<String>("zset-2",9.9);

Set<ZSetOperations.TypedTuple<String>> tuples = new HashSet<ZSetOperations.TypedTuple<String>>();

tuples.add(objectTypedTuple1);

tuples.add(objectTypedTuple2);

redisTemplate.opsForZSet().add("zset1", tuples);

//获取指定键内指定元素的分数

redisTemplate.opsForZSet().score("zset1","zset-1");

//指定键的移除指定元素

redisTemplate.opsForZSet().remove("zset1","lzj","zset-1");

//通过分数返回有序集合指定区间内的成员个数

System.out.println(redisTemplate.opsForZSet().count("zset1",10,20));

//通过索引区间返回有序集合成指定区间内的成员，其中有序集成员按分数值递增(从小到大)排序

redisTemplate.opsForZSet().range("zset1",0,-1).forEach(System.out::println);

//返回有序集中指定成员的排名，其中有序集成员按分数值递增(从小到大)顺序排列

System.out.println(redisTemplate.opsForZSet().rank("zset1","zset-1"));

//返回有序集中指定成员的排名，其中有序集成员按分数值递增(从大到小)顺序排列

redisTemplate.opsForZSet().reverseRank("zset1","zset-1");

//通过分数返回有序集合指定区间内的成员，其中有序集成员按分数值递增(从小到大)顺序排列

redisTemplate.opsForZSet().rangeByScore("zset1",10,20).forEach(System.out::println);

//通过分数返回有序集合指定区间内的成员，其中有序集成员按分数值递增(从小到大)顺序排列 取下标1开始2个元素

redisTemplate.opsForZSet().rangeByScore("zset1",10,20,1,2).forEach(System.out::println);

//通过索引区间返回有序集合成指定区间内的成员，其中有序集成员按分数值递减(从大到小)顺序排列

redisTemplate.opsForZSet().reverseRange("zset1",0,-1).forEach(System.out::println);;

//指定键的分数在10到20之间的元素(从大到小排序)

redisTemplate.opsForZSet().reverseRangeByScore("zset1", 10, 20).forEach(System.out::println);

//指定键的分数在10到20之间的元素(从大到小排序) 取下标1开始2个元素

redisTemplate.opsForZSet().reverseRangeByScore("zset1", 10, 20,1,2).forEach(System.out::println);

//通过索引区间内的成员按分数值递增(从小到大)顺序排列 并且带有分数

Set<ZSetOperations.TypedTuple<String>> zset1 = redisTemplate.opsForZSet().rangeWithScores("zset1", 0, -1);

Iterator<ZSetOperations.TypedTuple<String>> iterator1 = zset1.iterator();

while (iterator1.hasNext())

{

ZSetOperations.TypedTuple<String> typedTuple = iterator1.next();

System.out.println("value:" + typedTuple.getValue() + "score:" + typedTuple.getScore());

}

//指定键的分数在10到20之间的元素(从小到大排序)并且带有分数

Set<ZSetOperations.TypedTuple<String>> zset2 = redisTemplate.opsForZSet().rangeByScoreWithScores("zset1", 10, 20);

Iterator<ZSetOperations.TypedTuple<String>> iterator2 = zset2.iterator();

while (iterator2.hasNext())

{

ZSetOperations.TypedTuple<String> typedTuple = iterator2.next();

System.out.println("value:" + typedTuple.getValue() + "score:" + typedTuple.getScore());

}

//通过索引区间返回有序集合成指定区间内的成员对象，其中有序集成员按分数值递减(从大到小)顺序排列

Set<ZSetOperations.TypedTuple<String>> zset3 = redisTemplate.opsForZSet().reverseRangeWithScores("zset1", 0, -1);

Iterator<ZSetOperations.TypedTuple<String>> iterator3 = zset3.iterator();

while (iterator3.hasNext())

{

ZSetOperations.TypedTuple<String> typedTuple = iterator3.next();

System.out.println("value:" + typedTuple.getValue() + "score:" + typedTuple.getScore());

}

//指定键的分数在10到20之间的元素(从小到大排序)并且带有分数

Set<ZSetOperations.TypedTuple<String>> zset4 = redisTemplate.opsForZSet().reverseRangeByScoreWithScores("zset1", 10,20);

Iterator<ZSetOperations.TypedTuple<String>> iterator4 = zset4.iterator();

while (iterator4.hasNext())

{

ZSetOperations.TypedTuple<String> typedTuple = iterator4.next();

System.out.println("value:" + typedTuple.getValue() + "score:" + typedTuple.getScore());

}

//遍历zset

Cursor<ZSetOperations.TypedTuple<String>> cursor5 = redisTemplate.opsForZSet().scan("zzset1", ScanOptions.NONE);

while (cursor5.hasNext()){

ZSetOperations.TypedTuple<String> item = cursor5.next();

System.out.println(item.getValue() + ":" + item.getScore());

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

3.StringRedisTemplate使用

所需要的依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

1

2

3

4

配置文件

server:

port: 8083

spring:

redis:

database: 0

host: localhost

1

2

3

4

5

6

7

public class MyTest2 extends ParentTest {

@Autowired

private StringRedisTemplate stringRedisTemplate;

@Test

public void test1(){

stringRedisTemplate.opsForValue().set("lzj","123");

}

}

1

2

3

4

5

6

7

8

9

10

11

他的api几乎和RedisTemplate相同 用法相同不多讲解，看完RedisTemplate相信你也有了对他们的区别，还有什么时候用什么有了进一步了解,我们通常会把这些方法封装起来 成一个工具类方便我们使用，下面我提供了一个工具类给大家

RedisTemplate封装的工具类(你也可以不要这个SpringUtils直接给RedisUtils标注上@Component，自动注入RedisTemplate使用，在要用到RedisUtils的地方注入RedisUtils即可)

@Component

public final class SpringUtils implements BeanFactoryPostProcessor

{

/\*\* Spring应用上下文环境 \*/

private static ConfigurableListableBeanFactory beanFactory;

@Override

public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException

{

SpringUtils.beanFactory = beanFactory;

}

/\*\*

\* 获取对象

\*

\* @param name

\* @return Object 一个以所给名字注册的bean的实例

\* @throws org.springframework.beans.BeansException

\*

\*/

@SuppressWarnings("unchecked")

public static <T> T getBean(String name) throws BeansException

{

return (T) beanFactory.getBean(name);

}

/\*\*

\* 获取类型为requiredType的对象

\*

\* @param clz

\* @return

\* @throws org.springframework.beans.BeansException

\*

\*/

public static <T> T getBean(Class<T> clz) throws BeansException

{

T result = (T) beanFactory.getBean(clz);

return result;

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

public class RedisUtils {

private RedisUtils() {

}

@SuppressWarnings("unchecked")

private static RedisTemplate<String, Object> redisTemplate = SpringUtils.getBean("redisTemplate");

/\*\*

\* 设置有效时间

\*

\* @param key Redis键

\* @param timeout 超时时间

\* @return true=设置成功；false=设置失败

\*/

public static boolean expire(final String key, final long timeout) {

return expire(key, timeout, TimeUnit.SECONDS);

}

/\*\*

\* 设置有效时间

\*

\* @param key Redis键

\* @param timeout 超时时间

\* @param unit 时间单位

\* @return true=设置成功；false=设置失败

\*/

public static boolean expire(final String key, final long timeout, final TimeUnit unit) {

Boolean ret = redisTemplate.expire(key, timeout, unit);

return ret != null && ret;

}

/\*\*

\* 删除单个key

\*

\* @param key 键

\* @return true=删除成功；false=删除失败

\*/

public static boolean delKey(final String key) {

Boolean ret = redisTemplate.delete(key);

return ret != null && ret;

}

/\*\*

\* 删除多个key

\*

\* @param keys 键集合

\* @return 成功删除的个数

\*/

public static long delKeys(final Collection<String> keys) {

Long ret = redisTemplate.delete(keys);

return ret == null ? 0 : ret;

}

/\*\*

\* 存入普通对象

\*

\* @param key Redis键

\* @param value 值

\*/

public static void setValue(final String key, final Object value) {

redisTemplate.opsForValue().set(key, value, 1, TimeUnit.MINUTES);

}

// 存储普通对象操作

/\*\*

\* 存入普通对象

\*

\* @param key 键

\* @param value 值

\* @param timeout 有效期，单位秒

\*/

public static void setValueTimeout(final String key, final Object value, final long timeout) {

redisTemplate.opsForValue().set(key, value, timeout, TimeUnit.SECONDS);

}

/\*\*

\* 获取普通对象

\*

\* @param key 键

\* @return 对象

\*/

public static Object getValue(final String key) {

return redisTemplate.opsForValue().get(key);

}

// 存储Hash操作

/\*\*

\* 确定哈希hashKey是否存在

\*

\* @param key 键

\* @param hkey hash键

\* @return true=存在；false=不存在

\*/

public static boolean hasHashKey(final String key,String hkey) {

Boolean ret = redisTemplate.opsForHash().hasKey(key,hkey);

return ret != null && ret;

}

/\*\*

\* 往Hash中存入数据

\*

\* @param key Redis键

\* @param hKey Hash键

\* @param value 值

\*/

public static void hashPut(final String key, final String hKey, final Object value) {

redisTemplate.opsForHash().put(key, hKey, value);

}

/\*\*

\* 往Hash中存入多个数据

\*

\* @param key Redis键

\* @param values Hash键值对

\*/

public static void hashPutAll(final String key, final Map<String, Object> values) {

redisTemplate.opsForHash().putAll(key, values);

}

/\*\*

\* 获取Hash中的数据

\*

\* @param key Redis键

\* @param hKey Hash键

\* @return Hash中的对象

\*/

public static Object hashGet(final String key, final String hKey) {

return redisTemplate.opsForHash().get(key, hKey);

}

/\*\*

\* 获取Hash中的数据

\*

\* @param key Redis键

\* @return Hash对象

\*/

public static Map<Object, Object> hashGetAll(final String key) {

return redisTemplate.opsForHash().entries(key);

}

/\*\*

\* 获取多个Hash中的数据

\*

\* @param key Redis键

\* @param hKeys Hash键集合

\* @return Hash对象集合

\*/

public static List<Object> hashMultiGet(final String key, final Collection<Object> hKeys) {

return redisTemplate.opsForHash().multiGet(key, hKeys);

}

/\*\*

\* 删除Hash中的数据

\*

\* @param key Redis键

\* @param hKeys Hash键集合

\* @return Hash对象集合

\*/

public static long hashDeleteKeys(final String key, final Collection<Object> hKeys) {

return redisTemplate.opsForHash().delete(key,hKeys);

}

// 存储Set相关操作

/\*\*

\* 往Set中存入数据

\*

\* @param key Redis键

\* @param values 值

\* @return 存入的个数

\*/

public static long setSet(final String key, final Object... values) {

Long count = redisTemplate.opsForSet().add(key, values);

return count == null ? 0 : count;

}

/\*\*

\* 删除Set中的数据

\*

\* @param key Redis键

\* @param values 值

\* @return 移除的个数

\*/

public static long setDel(final String key, final Object... values) {

Long count = redisTemplate.opsForSet().remove(key, values);

return count == null ? 0 : count;

}

/\*\*

\* 获取set中的所有对象

\*

\* @param key Redis键

\* @return set集合

\*/

public static Set<Object> getSetAll(final String key) {

return redisTemplate.opsForSet().members(key);

}

// 存储ZSet相关操作

/\*\*

\* 往ZSet中存入数据

\*

\* @param key Redis键

\* @param values 值

\* @return 存入的个数

\*/

public static long zsetSet(final String key, final Set<ZSetOperations.TypedTuple<Object>> values) {

Long count = redisTemplate.opsForZSet().add(key, values);

return count == null ? 0 : count;

}

/\*\*

\* 删除ZSet中的数据

\*

\* @param key Redis键

\* @param values 值

\* @return 移除的个数

\*/

public static long zsetDel(final String key, final Set<ZSetOperations.TypedTuple<Object>> values) {

Long count = redisTemplate.opsForZSet().remove(key, values);

return count == null ? 0 : count;

}

// 存储List相关操作

/\*\*

\* 往List中存入数据

\*

\* @param key Redis键

\* @param value 数据

\* @return 存入的个数

\*/

public static long listPush(final String key, final Object value) {

Long count = redisTemplate.opsForList().rightPush(key, value);

return count == null ? 0 : count;

}

/\*\*

\* 往List中存入多个数据

\*

\* @param key Redis键

\* @param values 多个数据

\* @return 存入的个数

\*/

public static long listPushAll(final String key, final Collection<Object> values) {

Long count = redisTemplate.opsForList().rightPushAll(key, values);

return count == null ? 0 : count;

}

/\*\*

\* 往List中存入多个数据

\*

\* @param key Redis键

\* @param values 多个数据

\* @return 存入的个数

\*/

public static long listPushAll(final String key, final Object... values) {

Long count = redisTemplate.opsForList().rightPushAll(key, values);

return count == null ? 0 : count;

}

/\*\*

\* 从List中获取begin到end之间的元素

\*

\* @param key Redis键

\* @param start 开始位置

\* @param end 结束位置（start=0，end=-1表示获取全部元素）

\* @return List对象

\*/

public static List<Object> listGet(final String key, final int start, final int end) {

return redisTemplate.opsForList().range(key, start, end);

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

4.Jedis的使用

导入依赖

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>3.1.0</version>

</dependency>

1

2

3

4

5

4.1Jedis基本使用

public class MyTest7 extends ParentTest {

@Test

public void test1(){

Jedis jedis = new Jedis("localhost", 6379);

jedis.set("error","come");

jedis.close();

}

}

1

2

3

4

5

6

7

8

9

10

4.2JedisPool连接池使用

public class MyTest7 extends ParentTest {

@Test

public void test1() {

Jedis jedis = new Jedis("localhost", 6379);

jedis.set("error", "come");

jedis.close();

}

@Test

public void test2() {

//1 获得连接池配置对象，设置配置项

JedisPoolConfig config = new JedisPoolConfig();

// 1.1 最大连接数

config.setMaxTotal(30);

//1.2 最大空闲连接数

config.setMaxIdle(10);

//获得连接池

JedisPool jedisPool = new JedisPool(config, "localhost", 6379);

//获得连接

Jedis jedis = jedisPool.getResource();

jedis.set("error2", "come on");

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

JedisPoolConfig Redis连接池的配置对象

里面属性介绍

maxActive:(高版本改名为MaxTotal) ---- 控制一个pool可分配多少个jedis实例，如果赋值为-1则为不限制

maxWait:(高版本改名为MaxWaitMillis) ---- 表示当借一个jedis实例，最大等待时间，如果超过等待事件，则抛出JedisConnectionException

maxIdle: ---- 控制一个pool最少有多少个状态为(Idle)空闲的实例时就要挂了

whenExhaustedAction: ---- 表示当pool中的jedis实例被借用完，pool要采取的的操作，默认有三种：

-WHEN\_EXHAUSTED-FAIL -> 表示无jedis实例时，直接抛出NoSuchElementException

-WHEN\_EXHAUSTED-BLOCK -> 表示阻塞住或达到maxwait直接抛出JedisConnectionException

-WHEN\_EXHAUSTED-GROW -> 表示新建一个jedis实例，也就是说设置的maxactive无用

testOnBorrow ---- 获得一个jedis实例的时候是否检查可用性(ping()) 如果为true 则得到的jedis实例时可用的

testOnReturn ---- 还给一个jedis实例的时候是否检查可用性(ping())

所以我们封装了一个jedis连接池工具类方便我们使用

//JedisPool工具类

public class JedisPoolUtils {

private static volatile JedisPool jedisPool = null;

public static JedisPool getJedisPoolInstance() {

synchronized (JedisPoolUtils.class) {

if(jedisPool == null) {

JedisPoolConfig poolConfig = new JedisPoolConfig();

poolConfig.setMaxWaitMillis(100\*1000);

poolConfig.setMaxIdle(32);

poolConfig.setTestOnBorrow(true);

poolConfig.setMaxTotal(1000);

jedisPool = new JedisPool(poolConfig,"47.102.197.194",6379);

}

}

return jedisPool;

}

public static void release(Jedis jedis) {

if(null != jedisPool) {

/\*

高版本2.9以上弃用了

jedisPool.returnBrokenResource(jedis);

jedisPool.returnResource(jedis);

jedisPool.returnResourceObject(jedis);

\*/

jedis.close();

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

关于Jedis API

因为他所有API几乎和原生的redis命令一样，所以我在这我多做解释给几个例子给大家伙康康

public void keyTest() throws UnsupportedEncodingException {

System.out.println(jedis.flushDB());// 清空数据

System.out.println(jedis.echo("hello"));

// 判断key否存在

System.out.println(jedis.exists("foo"));

jedis.set("key", "values");

jedis.set("key2", "values");

System.out.println(jedis.exists("key"));// 判断是否存在

// 如果数据库没有任何key，返回nil，否则返回数据库中一个随机的key。

String randomKey = jedis.randomKey();

System.out.println("randomKey: " + randomKey);

// 设置60秒后该key过期

jedis.expire("key", 60);

// key有效毫秒数

System.out.println(jedis.pttl("key"));

// 移除key的过期时间

jedis.persist("key");

// 获取key的类型, "string", "list", "set". "none" none表示key不存在

System.out.println("type: " + jedis.type("key"));

// 导出key的值

byte[] bytes = jedis.dump("key");

System.out.println(new String(bytes));

// 将key重命名

jedis.renamenx("key", "keytest");

System.out.println("key是否存在: " + jedis.exists("key"));// 判断是否存在

System.out.println("keytest是否存在: " + jedis.exists("keytest"));// 判断是否存在

// 查询匹配的key

// KEYS \* 匹配数据库中所有 key 。

// KEYS h?llo 匹配 hello ， hallo 和 hxllo 等。

// KEYS h\*llo 匹配 hllo 和 heeeeello 等。

// KEYS h[ae]llo 匹配 hello 和 hallo ，但不匹配 hillo 。

// 特殊符号用 \ 隔开。

Set<String> set = jedis.keys("k\*");

System.out.println(set);

// 删除key

jedis.del("key");

System.out.println(jedis.exists("key"));

}