Atitit redisTemplate 使用

## 单独使用不使用注入模式启动慢

初始化

## 二进制数值不好调试 使用string序列化

edisTemplate中定义了对5种数据结构操作

redisTemplate.opsForValue();//操作字符串

redisTemplate.opsForHash();//操作hash

redisTemplate.opsForList();//操作list

redisTemplate.opsForSet();//操作set

redisTemplate.opsForZSet();//操作有序set

StringRedisTemplate与RedisTemplate

两者的关系是StringRedisTemplate继承RedisTemplate。

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。

SDR默认采用的序列化策略有两种，一种是String的序列化策略，一种是JDK的序列化策略。

StringRedisTemplate默认采用的是String的序列化策略，保存的key和value都是采用此策略序列化保存的。

RedisTemplate默认采用的是JDK的序列化策略，保存的key和value都是采用此策略序列化保存的

public static void delX(String **keyid**) {

*getRedis*().delete(**keyid**);

}

*@SuppressWarnings*("all")

public static void setX(String **key**, String **jsonString**, int **timeout**) {

Map m=new LinkedHashMap<>();m.put("k", **key**);m.put("v", **jsonString**);m.put("timeSec", **timeout**);

*log*.info("SSSet rds ---> "+ JSON.*toJSONString*(m));

RedisTemplate redis = *getRedis*();

// redis.opsForValue().set

redis.opsForValue().set(**key**, **jsonString**, **timeout**, *TimeUnit*.***SECONDS***);

}

public static String getX(String **key**) {

RedisTemplate redis = *getRedis*();

return (String) redis.opsForValue().get(**key** );

}

public static Set<Object> zrevrange(String **key**, int **start**, int **end**) {

*log*.info("query rds zrevrange----key:"+**key**+",start:"+**start**+",end:"+**end**);

return *getRedis*() .opsForZSet().reverseRange(**key**, **start**, **end**);

}

public static boolean zadd(String **key**, Object **jsonString**,Long **getOpenTimeStamp**) {

Map m = Maps.*newLinkedHashMap*();

m.put("k", **key**);

try {

Object obj = JSONObject.*parse*(**jsonString**.toString());

m.put("v", obj);

} catch (Exception e) {

m.put("v", **jsonString**);

}

m.put("score", **getOpenTimeStamp**);

*log*.info("rds zadd:" + JSON.*toJSONString*(m));

*log*.info("rds zadd readable:" + JSON.*toJSONString*(m, true));

// log.info("zadd ret:"+jedis.zadd(key, getOpenTimeStamp, jsonString));

try {

//log.info(redisTemplate);

//jedis.zadd(key, getOpenTimeStamp, jsonString)

*log*.info("zadd ret:"+ *getRedis*() .opsForZSet().add(**key**, **jsonString**, **getOpenTimeStamp**));

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

public static RedisTemplate getRedis() {

ResourceBundle resource = ResourceBundle.*getBundle*("redis");

// InputStream inStream = OpenServerController.class.getClassLoader().getResourceAsStream("redis.properties");

// Properties prop = new Properties();

// try {

// prop.load(inStream);

// } catch (IOException e) {

// throw new RuntimeException(e);

// }

String host = resource.getString("spring.redis.host");

String pwd = resource.getString("spring.redis.password");

// jedis.auth(pwd);

// jedis.select(2);

// return jedis;

RedisStandaloneConfiguration rsc = new RedisStandaloneConfiguration();

rsc.setPort(6379);

rsc.setPassword(pwd);

rsc.setHostName(host);

rsc.setDatabase(5); //jei g haosyo mayon

// JedisConnectionFactory fac = new JedisConnectionFactory(rsc);

// DefaultJedisClientConfigurationBuilder JedisClientConfiguration1=new DefaultJedisClientConfiguration();

LettuceConnectionFactory conn = new LettuceConnectionFactory(rsc);

// conn.set

conn.setDatabase(7); //jeig ok

// conn.setHostName(host);

// conn.setPort(6379);

// conn.setPassword(pwd);

// conn.setUsePool(true);

conn.afterPropertiesSet();

// StringRedisTemplate template

RedisTemplate<String,String> template = new RedisTemplate<>();

template.setConnectionFactory(conn);

RedisSerializer<String> redisSerializer = new StringRedisSerializer();

//key序列化方式

template.setKeySerializer(redisSerializer);

//value序列化

template.setValueSerializer(redisSerializer);

// template.setValueSerializer(StringRedisSerializer.class);

template.afterPropertiesSet();

return template;

}