Spring data redis

1. 简介

Spring Data Redis, part of the larger Spring Data family, provides easy configuration and access to Redis from Spring applications. It offers both low-level and high-level abstractions for interacting with the store, freeing the user from infrastructural concerns.

1. Redis安装

1 安装环境

Redis版本：3.0.0

环境：Linux

2 安装步骤

安装gcc编译器

yum install gcc-c++

解压源码压缩包

tar -zxf redis-3.0.0.tar.gz

进入解压目录进行编译

cd redis-3.0.0

编译

make

将redis安装到指定目录

make PREFIX=/usr/local/redis install

启动redis

前置启动

默认的是前置启动，/usr/local/redis/bin/redis-server

后置启动

先将redis.conf文件拷贝到redis的安装目录

cd redis.conf /usr/local/redis/bin

编辑redis.conf文件修改:daemonize yes

./redis-server redis.conf

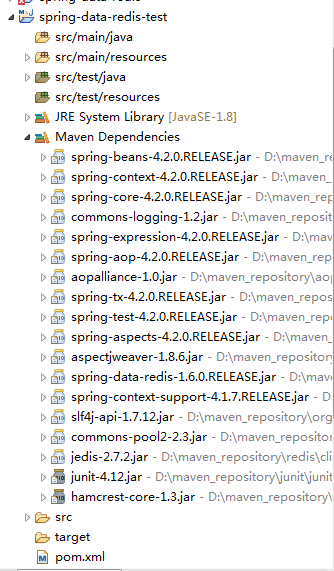
查看redis进程

ps aux|grep redis

关闭后置启动的redis

./redis-cli shutdown

1. 搭建整合环境
2. 创建项目



1. 整合配置

配置文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*    xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx.xsd"*>    <!-- 配置读取properties文件的工具类 -->  <context:property-placeholder location=*"classpath:redis.properties"*/>    <!-- Jedis连接池 -->  <bean id=*"poolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>  <property name=*"maxTotal"* value=*"${redis.pool.maxTotal}"*/>  <property name=*"maxIdle"* value=*"${redis.pool.maxIdel}"*/>  <property name=*"minIdle"* value=*"${redis.pool.minIdle}"*/>  </bean>    <!-- Jedis的连接工厂 -->  <bean id=*"jedisConnectionFactory"* class=*"org.springframework.data.redis.connection.jedis.JedisConnectionFactory"*>  <property name=*"hostName"* value=*"${redis.conn.hostName}"*/>  <property name=*"hostName"* value=*"${redis.conn.port}"*/>  <property name=*"poolConfig"* ref=*"poolConfig"*/>  </bean>    <!-- Redis模板对象 -->  <bean id=*"redisTemplate"* class=*"org.springframework.data.redis.core.RedisTemplate"*>  <property name=*"connectionFactory"* ref=*"jedisConnectionFactory"*/>  <!-- 序列化器：能够把我们存储的key与value做序列化处理的对象 -->  <!-- 配置默认的序列化器 -->  <!-- keySerializer、valueSerializer 配置Redis中的String类型key与value的序列化器-->  <!-- hashKeySerializer、hashValueSerializer 配置Redis中的Hash类型key与value的序列化器-->  <property name=*"keySerializer"*>  <bean class=*"org.springframework.data.redis.serializer.StringRedisSerializer"*/>  </property>  <property name=*"valueSerializer"*>  <bean class=*"org.springframework.data.redis.serializer.StringRedisSerializer"*/>  </property>  </bean>    </beans> |

1. 测试整合环境

测试代码

|  |
| --- |
| **package** com.github.jihch;  **import** org.junit.Test;  **import** org.junit.runner.RunWith;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.data.redis.core.RedisTemplate;  **import** org.springframework.test.context.ContextConfiguration;  **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  /\*\*  \*  \* **@author** 姬鸿昌  \* 2020年3月21日  \*/  @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration("classpath:applicationContext.xml")  **public** **class** RedisTest {    @Autowired  **private** RedisTemplate<String, Object> redisTemplate;  /\*\*  \* 添加键值对  \*/  @Test  **public** **void** test1() {  redisTemplate.opsForValue().set("key", "test");  }    /\*\*  \* 获取redis中的数据  \*/  @Test  **public** **void** test2() {  String str = (String) redisTemplate.opsForValue().get("key");  System.***out***.println(str);  }    } |

1. Spring Data Redis存储实体对象

测试代码

|  |
| --- |
| @Test  **public** **void** test3() {  User user = **new** User();  user.setAge(30);  user.setId(1);  user.setName("张三");  //更换序列化器  redisTemplate.setValueSerializer(**new** JdkSerializationRedisSerializer());  redisTemplate.opsForValue().set("user", user);  }    /\*\*  \* 获取 User  \*/  @Test  **public** **void** test4() {  //更换序列化器  redisTemplate.setValueSerializer(**new** JdkSerializationRedisSerializer());  User user = (User) redisTemplate.opsForValue().get("user");  System.***out***.println(user);  } |

1. Spring Data Redis以JSON的格式存储实体对象

|  |
| --- |
| /\*\*  \* 添加 User JSON格式  \*/  @Test  **public** **void** test5() {  User user = **new** User();  user.setAge(23);  user.setId(2);  user.setName("李四");  redisTemplate.setValueSerializer(**new** Jackson2JsonRedisSerializer<>(User.**class**));  redisTemplate.opsForValue().set("user", user);  }    @Test  **public** **void** test6() {  redisTemplate.setValueSerializer(**new** Jackson2JsonRedisSerializer<>(User.**class**));  User user = (User) redisTemplate.opsForValue().get("user");  System.***out***.println(user);  } |

1. Spring Data Redis 管道技术

|  |
| --- |
| /\*\*  \* 通过RedisTemplate测试pipeline  \*/  @Test  **public** **void** test7() {  Long start = System.*currentTimeMillis*();  RedisSerializer<String> stringSerializer = **new** StringRedisSerializer();  RedisSerializer<User> valueSerializer = **new** Jackson2JsonRedisSerializer<>(User.**class**);  Map<**byte**[], **byte**[]> tuple = **new** HashMap<>();  **for** (**int** i = 0; i < 10000; i++) {  User user = **new** User();  user.setAge(23);  user.setId(2);  user.setName("李四");  **byte**[] key = stringSerializer.serialize("user:"+i);  **byte**[] value = valueSerializer.serialize(user);  tuple.put(key, value);  }  Long end = System.*currentTimeMillis*();  System.***out***.println("cost: " + (end - start) + " ms" );    redisTemplate.execute(**new** RedisCallback<Long>() {  @Override  **public** Long doInRedis(RedisConnection connection) **throws** DataAccessException {  connection.openPipeline();  connection.mSet(tuple);  connection.closePipeline();  **return** **null**;  }  });  end = System.*currentTimeMillis*();  System.***out***.println("cost: " + (end - start) + " ms" );  } |