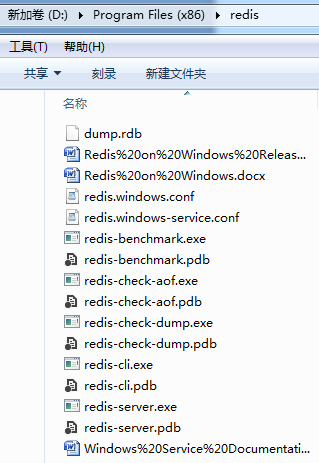
Redis安装及使用

Redis是一个key-value的存储系统，redis和memcached类似，数据保存在内存中，但redis会周期性的将数据写到磁盘上(数据落地)。Redis可以设置数据的过期时间。

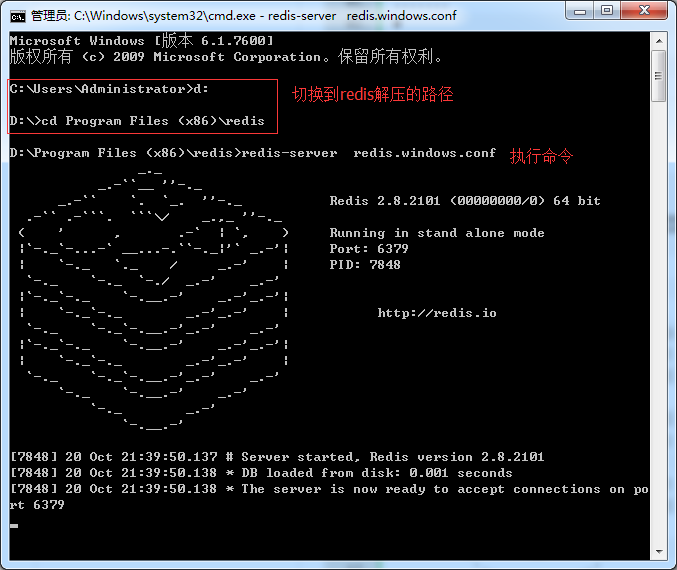
# Redis安装

## Redis 64位系统安装

Redis解压后免安装，注意解压路径不要有中文



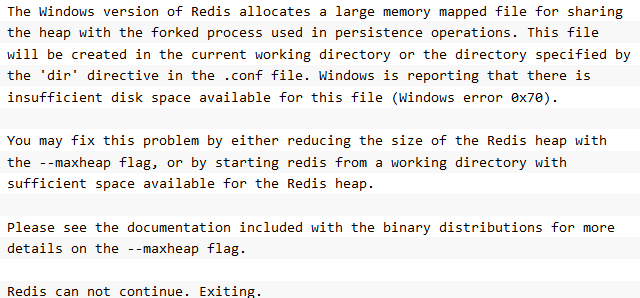
cmd 进入DOS命令：



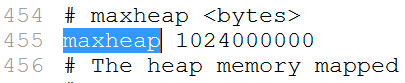
切换到redis解压的路径，运行命令：redis-server redis.windows.conf

启动成功后会看到上面的图形。

备注：如果启动后报如下异常，需要修改maxheap。如果不报错，就不用管它，等报错了再来。



如果看到上面的异常，打开redis.windows.conf，找到maxheap，将数字调大：



再次启动redis即可。注意前面没有空格

可以将redis启动命令写成bat脚本，创建一个txt文件，输入如下内容

d:

cd D:\Program Files (x86)\redis

redis-server.exe redis.windows.conf

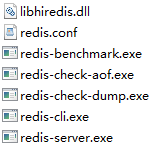
其中第一行和第二行是你自己安装redis的盘符和目录。

保存文件后将拓展名修改成bat，下次双击就可以运行了。



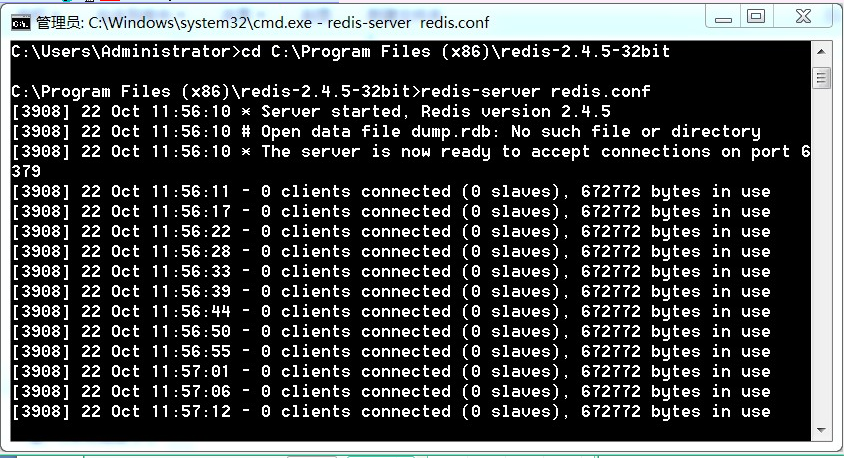
## Redis 32位系统安装

32位系统压缩包解压后目录如下：



cmd进入DOS窗口，运行命令：

redis-server redis.conf



32位客户端没有64位的图形界面，会不断弹出clients connected的信息。

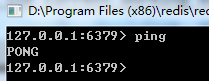
# Redis使用

## redis-cli

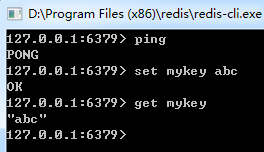
双击打开redis-cli.exe

输入命令：PING

redis响应PONG，证明服务是可用的

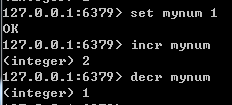


redis存储数据是key-value形式的。通过set命令存值，通过get命令取值

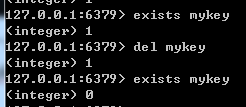


命令“set mykey abc”在redis中存入了一条数据，key是mykey，值是abc。

命令“get mykey”是从redis中取key=mykey的值，取出的是abc



incr命令可以让整数自增，decr可以实现减一的操作。



exists可以判断一个key是否存在，del删除一个key。

常用命令参考网址：<http://doc.redisfans.com/>

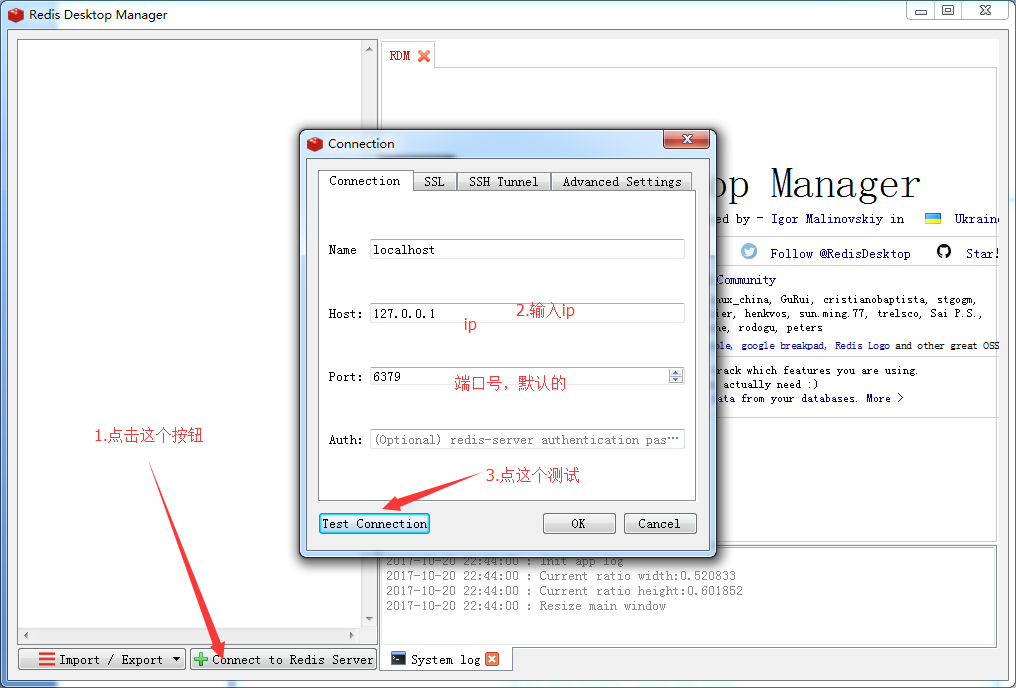
## 可视化工具

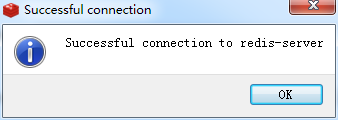
redis-desktop-manager是一个redis可视化工具：



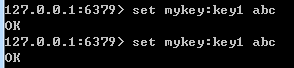
注意安装redis-desktop-manager可能运行不了，报错：应用程序无法正常启动0xc000007b

可以使用DirectX 9.0c修复缺失的dll文件。如果修复后还不能启动，尝试降低redis-desktop-manager的版本。高版本的容易报这个错误

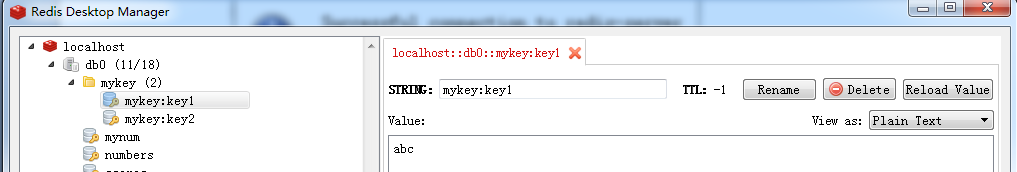




在客户端存两个key



在redis-desktop-manager中查看：

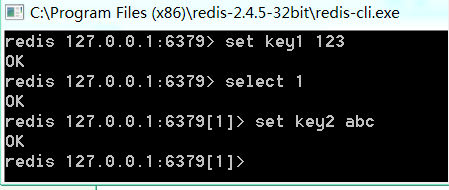


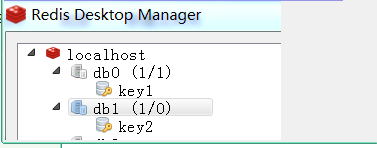
可以看到存数据是，key会按照冒号分组

redis中有很多个库，默认使用0号库。切换库的命令是：

select 库的索引

如select 1就会切换到1号库



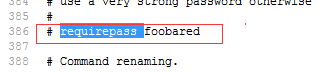


## 设置密码

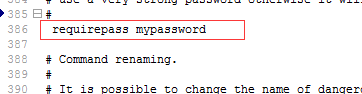
redis可以设置访问密码**(慎用，改错了你自己连不上)**

64位redis打开redis.windows.conf，32位系统打开redis.conf

找到# requirepass foobared这一行

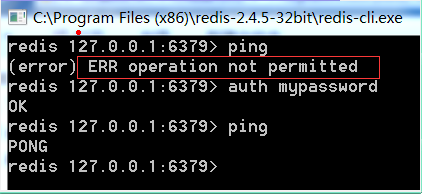


去掉前面的#写你自己的密码



requirepass 后面就是你的密码。重启redis，再启动redis-cli.exe

注意requirepass前面不要有空格



连接上redis后执行命令会报错。

使用auth 后面接访问密码，可以连接上redis

# java操作Redis

java操作Redis使用Jedis。Jedis是Redis的java版本客户端实现。使用Jedis需要在项目中引入jar：

<!-- junit测试包 -->  
<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
</dependency>  
<!--Jedis的jar-->  
<dependency>  
 <groupId>redis.clients</groupId>  
 <artifactId>jedis</artifactId>  
 <version>2.9.0</version>  
</dependency>

测试类：

**import org.junit.Test**;  
**import redis.clients.jedis.Jedis**;  
**import redis.clients.jedis.JedisPool**;  
  
**public class JedisTest {  
 @Test  
 public void** testJedisPool() **throws Exception {  
 // 第一步：创建一个JedisPool对象。需要指定服务端的ip及端口。  
 JedisPool jedisPool = new** JedisPool(**"127.0.0.1"**, **6379**);  
 **// 第二步：从JedisPool中获得Jedis对象。  
 Jedis jedis = jedisPool**.getResource();  
 **// 第三步：使用Jedis操作redis服务器。  
 System**.***out***.println(**jedis**.ping());**//输出pong代表服务可用  
 jedis**.set(**"jedis"**, **"test"**);**//向redis中存入一个值  
 String result = jedis**.get(**"jedis"**);**//取值  
 System**.***out***.println(**result**);  
 **// 第四步：操作完毕后关闭jedis对象，连接池回收资源。  
 jedis**.close();  
 **// 第五步：关闭JedisPool对象。  
 jedisPool**.close();  
 **}**

**}**

jedis对象中的方法名对应的就是redis中的命令。

连接池也可以配置初始化参数：

**JedisPoolConfig config = new** JedisPoolConfig();  
**config**.setMaxTotal(**50**);**//最大连接数  
config**.setMaxIdle(**5**);**//最大闲置连接  
config**.setMinIdle(**5**);**//初始化连接数  
config**.setMaxWaitMillis(**10\*1000**);**//连接池中连接用完时,新的请求等待时间,毫秒  
  
JedisPool jedisPool = new** JedisPool(**config**,**"127.0.0.1"**);

其它参数：参考博客：<http://blog.csdn.net/u011731544/article/details/78309133>

# 发布订阅(选学)

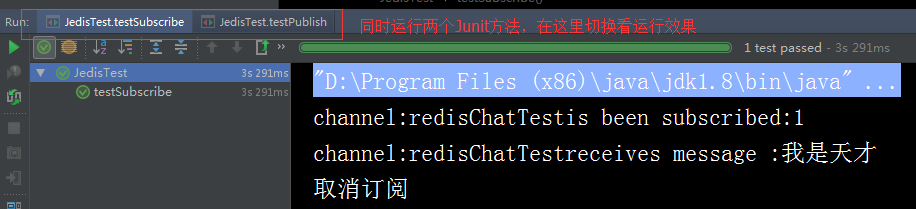
发布者可以向某频道发布信息，订阅者可以订阅该频道，接收到发布者发布的信息：

创建订阅发布工具类，重写父类中的几个方法。

**import redis.clients.jedis.**JedisPubSub;  
  
**public class RedisMsgPubSubListener extends** JedisPubSub **{  
 /\*\*  
 \* 收消息  
 \*  
 \* @param channel  
 \* @param message  
 \*/  
 @Override  
 public void** onMessage(**String channel**, **String message**) **{  
 System**.***out***.println(**"channel:" + channel + "receives message :" + message**);  
 **//this.unsubscribe();//收到信息后取消订阅  
 }  
  
 /\*\*  
 \* 被订阅  
 \*  
 \* @param channel  
 \* @param subscribedChannels  
 \*/  
 @Override  
 public void** onSubscribe(**String channel**, **int subscribedChannels**) **{  
 System**.***out***.println(**"channel:" + channel + "is been subscribed:" + subscribedChannels**);  
 **}  
  
 /\*\*  
 \* 取消订阅  
 \*/  
 @Override  
 public void** unsubscribe() **{  
 System**.***out***.println(**"取消订阅"**);  
 **super**.unsubscribe();  
 **}  
}**

测试方法：

**@Test//订阅  
public void** testSubscribe() **throws Exception {  
 Jedis jedis = new** Jedis(**"localhost"**);  
 **RedisMsgPubSubListener listener = new** RedisMsgPubSubListener();  
 **//收听redisChatTest频道  
 jedis**.subscribe(**listener**, **"redisChatTest"**);  
**}  
  
@Test//发布  
public void** testPublish() **throws Exception {  
 Jedis jedis = new** Jedis(**"localhost"**);  
 **//发布消息  
 jedis**.publish(**"redisChatTest"**, **"我是天才"**);  
 **Thread**.sleep(**3000**);  
 **jedis**.publish(**"redisChatTest"**, **"哈哈"**);  
**}**



# Spring整合Redis

## 普通的整合

可以将常用的Jedis命令封装成工具类，由Spring注入JedisPool连接池对象。

**import redis.clients.jedis.Jedis**;  
**import redis.clients.jedis.JedisPool**;  
  
**public class JedisClientUtil {  
  
 private JedisPool jedisPool**;  
  
 **public** JedisClientUtil(**JedisPool jedisPool**) **{  
 this**.**jedisPool = jedisPool**;  
 **}  
  
 /\*\*  
 \* 存数据  
 \*  
 \* @param key  
 \* @param value  
 \* @return  
 \*/  
 public String** set(**String key**, **String value**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **String result = jedis**.set(**key**, **value**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* 取数据  
 \*  
 \* @param key  
 \* @return  
 \*/  
 public String** get(**String key**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **String result = jedis**.get(**key**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* 判断key是否存在  
 \*  
 \* @param key  
 \* @return  
 \*/  
 public Boolean** exists(**String key**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **Boolean result = jedis**.exists(**key**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* 更新生存期  
 \*  
 \* @param key  
 \* @param seconds  
 \* @return  
 \*/  
 public Long** expire(**String key**, **int seconds**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **Long result = jedis**.expire(**key**, **seconds**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* 返回可以的剩余生存时间，单位秒  
 \*  
 \* @param key  
 \* @return  
 \*/  
 public Long** ttl(**String key**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **Long result = jedis**.ttl(**key**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* 数字自增  
 \*  
 \* @param key  
 \* @return  
 \*/  
 public Long** incr(**String key**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **Long result = jedis**.incr(**key**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* HSET key field value  
 \* 哈希表存值  
 \*  
 \* @param key  
 \* @param field  
 \* @param value  
 \* @return  
 \*/  
 public Long** hset(**String key**, **String field**, **String value**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **Long result = jedis**.hset(**key**, **field**, **value**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* 哈希表取值  
 \* HGET key field  
 \*  
 \* @param key  
 \* @param field  
 \* @return  
 \*/  
 public String** hget(**String key**, **String field**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **String result = jedis**.hget(**key**, **field**);  
 **jedis**.close();  
 **return result**;  
 **}  
  
 /\*\*  
 \* 哈希表删除一个域或多个域  
 \*  
 \* @param key  
 \* @param field  
 \* @return  
 \*/  
 public Long** hdel(**String key**, **String**... **field**) **{  
 Jedis jedis = jedisPool**.getResource();  
 **Long result = jedis**.hdel(**key**, **field**);  
 **jedis**.close();  
 **return result**;  
 **}  
}**

spring需要导入基础jar包：

<!--Spring基础jar包-->  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.11.RELEASE</version>  
</dependency>

在resources目录创建redis.properties

redis.host=127.0.0.1  
redis.port=6379   
redis.maxIdle=300   
redis.maxActive=600   
redis.maxWait=1000   
redis.testOnBorrow=true

在resources目录下创建spring-config.xml，配置：

*<?*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"  
 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-4.2.xsd">  
 <context:property-placeholder location="classpath:redis.properties" ignore-unresolvable="true"/>  
  
 <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">  
 <property name="maxIdle" value="${redis.maxIdle}"/>  
 <property name="maxWaitMillis" value="${redis.maxWait}"/>  
 <property name="maxTotal" value="${redis.maxActive}"/>  
 <property name="testOnBorrow" value="${redis.testOnBorrow}"/>  
 </bean>  
  
 <bean id="jedisPool" class="redis.clients.jedis.JedisPool" destroy-method="close">  
 <constructor-arg type="java.lang.String" value="${redis.host}"/><!--host-->  
 <constructor-arg type="int" value="${redis.port}"/><!--port-->  
 </bean>  
  
 <bean id="jedisClientUtil" class="JedisClientUtil">  
 <constructor-arg index="0" ref="jedisPool"/>  
 </bean>  
</beans>

测试代码：

**@Test  
public void** testSpring() **{** ApplicationContext **ac = new** ClassPathXmlApplicationContext(**"spring-config.xml"**);  
 **JedisClientUtil client =** (**JedisClientUtil**) **ac**.**getBean**(**"jedisClientUtil"**);  
 **client**.hset(**"hkey1"**, **"f1"**, **"123"**);**//赋值  
 System**.***out***.println(**client**.hget(**"hkey1"**, **"f1"**));**//取值**

((**ClassPathXmlApplicationContext**) **ac**).close(); **}**

## 使用Spring Data Redis

使用Spring Data Redis需要导入jar

<dependency>  
 <groupId>org.springframework.data</groupId>  
 <artifactId>spring-data-redis</artifactId>  
 <version>1.8.7.RELEASE</version>  
</dependency>

Spring Data Redis通过RedisTemplate操作redis

RedisTemplate和StringRedisTemplate允许用户使用一个回调方法直接操作RedisConnection。

简单的封装一个get和set的工具类：

**import org.springframework.dao.**DataAccessException;  
**import org.springframework.data.redis.connection.**RedisConnection;  
**import org.springframework.data.redis.connection.**StringRedisConnection;  
**import org.springframework.data.redis.core.**RedisCallback;  
**import org.springframework.data.redis.core.RedisTemplate**;  
**import org.springframework.data.redis.serializer.**RedisSerializer;  
  
**public class RedisDAO {  
 //由Spring注入  
 private RedisTemplate<String**, **String> redisTemplate**;  
  
 **public void** setRedisTemplate(**RedisTemplate<String**, **String> redisTemplate**) **{  
 this**.**redisTemplate = redisTemplate**;  
 **}  
  
 /\*\*  
 \* 存值  
 \*  
 \* @param key  
 \* @param value  
 \*/  
 public void** set(**String key**, **String value**) **{  
 redisTemplate**.execute(**new** RedisCallback**<Long>**() **{  
 public Long** doInRedis(RedisConnection **connection**)  
 **throws** DataAccessException **{  
 try {  
// RedisSerializer<String> redisSerializer  
// = redisTemplate.getStringSerializer();  
// connection.set(redisSerializer.serialize(key),  
// redisSerializer.serialize(value));**

**//如果使用的是StringRedisTemplate,可以直接转换成StringRedisConnection** ((StringRedisConnection) **connection**).**set**(key, value);  
 **return 1L**;  
 **} catch** (**Exception e**) **{  
 e**.printStackTrace();  
 **}  
 return null**;  
 **}  
 }**);  
 **}  
  
 /\*\*  
 \* 取值  
 \*  
 \* @param key  
 \* @return  
 \*/  
 public String** get(**String key**) **{  
 return redisTemplate**.execute(**new** RedisCallback**<String>**() **{  
 public String** doInRedis(RedisConnection **connection**) **throws** DataAccessException **{  
 try {  
// RedisSerializer<String> redisSerializer  
// = redisTemplate.getStringSerializer();  
// byte[] value = connection.get(redisSerializer.serialize(key));  
// return redisSerializer.deserialize(value);  
 return** ((StringRedisConnection) **connection**).**get**(key);  
 **} catch** (**Exception e**) **{  
 e**.printStackTrace();  
 **}  
 return null**;  
 **}  
 }**);  
 **}  
}**

spring-config.xml中的配置：

<bean id="connectionFactory"  
 class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"  
 destroy-method="destroy">  
 <property name="poolConfig" ref="jedisPoolConfig"/>  
 <property name="hostName" value="${redis.host}"/>  
 <property name="port" value="${redis.port}"/>  
</bean>  
  
<bean id="redisTemplate"  
 class="org.springframework.data.redis.core.StringRedisTemplate">  
 <property name="connectionFactory" ref="connectionFactory"/>  
</bean>  
  
<bean id="redisDAO" class="RedisDAO">  
 <property name="redisTemplate" ref="redisTemplate"/>  
</bean>