# 基于Redis的消息队列之生产消费者模式

## 1、pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.leo.common</groupId>  <artifactId>common-parent</artifactId>  <version>1.0-SNAPSHOT</version>  <relativePath>../common-parent</relativePath>  </parent>  <groupId>org.leo.redis</groupId>  <artifactId>redis-procon</artifactId>  <version>${redis-procon.version}</version>  <packaging>war</packaging>  <description>基于Redis的生产消费者模式</description>  <name>redis-procon</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <dependencies>  <dependency>  <groupId>org.leo.common</groupId>  <artifactId>common-config</artifactId>  <version>${common-config.version}</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  </dependency>  <!-- 日志适配器 -->  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j-slf4j-impl</artifactId>  <scope>test</scope>  </dependency>  <!-- 日志实现 -->  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j-core</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  </dependency>  <!-- redis -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-pool2</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-redis</artifactId>  </dependency>  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency>  </dependencies>  </project> |

这里引入了我的一些共通工程，大家做的时候按照自己的实际情况修改即可。

我做成了war包的形式放在Tomcat里跑，大家可以做成jar包。

## 2、web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"2.5"* xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance*  *http://www.springmodules.org/schema/cache/springmodules-cache.xsd*  *http://www.springmodules.org/schema/cache/springmodules-ehcache.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*  *http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd*  *"*>  <display-name>Redis生产消费者</display-name>  <!-- 加载spring容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>**classpath:spring/applicationContext.xml**</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/</url-pattern>  </filter-mapping>  </web-app> |

注意applicationContext.xml配置的路径，按照实际情况修改，我这里是放在src/resources/spring下。

## 3、redis.properties

|  |
| --- |
| ## redis  redis.host=192.168.56.104  redis.port=6379  redis.pwd=111111  redis.maxIdle=5  redis.maxTotal=10  redis.maxWaitMillis=10000  redis.testOnBorrow=true |

里面的配置请按照实际情况修改。

## 4、applicationContext.xml

|  |
| --- |
| <?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:jdbc=*"http://www.springframework.org/schema/jdbc"*  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/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*  *"*>  <context:component-scan base-package=*"org.leo.ssm"* />  <!-- 属性文件读入 -->  <context:property-placeholder location=*"classpath:redis.properties"* />  <!-- spring线程池的配置 -->  <bean id=*"taskExecutor"*  class=*"org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor"*>  <!-- 线程池维护线程的最少数量 -->  <property name=*"corePoolSize"* value=*"5"* />  <!-- 线程池维护线程所允许的空闲时间 -->  <property name=*"keepAliveSeconds"* value=*"300"* />  <!-- 线程池维护线程的最大数量 -->  <property name=*"maxPoolSize"* value=*"10"* />  <!-- 线程池所使用的缓冲队列 -->  <property name=*"queueCapacity"* value=*"25"* />  </bean>  <import resource=*"applicationContext-redis.xml"* />  </beans> |

## 5、applicationContext-redis.xml

|  |
| --- |
| <?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:p=*"http://www.springframework.org/schema/p"*  xmlns:redis=*"http://www.springframework.org/schema/redis"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/redis*  *http://www.springframework.org/schema/redis/spring-redis-1.0.xsd"*>  <bean id=*"poolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>  <property name=*"maxIdle"* value=*"${redis.maxIdle}"* />  <property name=*"maxTotal"* value=*"${redis.maxTotal}"* />  <property name=*"maxWaitMillis"* value=*"${redis.maxWaitMillis}"* />  <property name=*"testOnBorrow"* value=*"${redis.testOnBorrow}"* />  </bean>  <bean id=*"redisConnectionFactory"*  class=*"org.springframework.data.redis.connection.jedis.JedisConnectionFactory"*  p:host-name=*"${redis.host}"* p:port=*"${redis.port}"* p:password=*"${redis.pwd}"*  p:pool-config-ref=*"poolConfig"* />  <bean id=*"redisTemplate"* class=*"org.springframework.data.redis.core.RedisTemplate"*>  <property name=*"connectionFactory"* ref=*"redisConnectionFactory"* />  <property name=*"keySerializer"*>  <bean  class=*"org.springframework.data.redis.serializer.StringRedisSerializer"* />  </property>  <property name=*"valueSerializer"*>  <bean  class=*"org.springframework.data.redis.serializer.StringRedisSerializer"*></bean>  </property>  </bean>  </beans> |

## 6、RedisQueueDaoImpl

|  |
| --- |
| package org.leo.ssm.redis.dao.impl;  import org.leo.ssm.redis.dao.RedisQueueDao;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.data.redis.core.RedisTemplate;  import org.springframework.stereotype.Repository;  @Repository  public class RedisQueueDaoImpl implements RedisQueueDao {  @Autowired  protected RedisTemplate<String, String> redisTemplate;  @Override  public void lpush(String key, String value) {  redisTemplate.opsForList().leftPush(key, value);  }  public String lpop(String key) {  return redisTemplate.opsForList().rightPop(key);  }  } |

接口就不附上了。

## 7、RedisQueueServiceImpl

|  |
| --- |
| package org.leo.ssm.redis.service.impl;  import org.leo.ssm.redis.dao.RedisQueueDao;  import org.leo.ssm.redis.service.RedisQueueService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  @Service  public class RedisQueueServiceImpl implements RedisQueueService {  @Autowired  private RedisQueueDao redisQueueDao;  @Override  public void lpush(String key, String value) {  redisQueueDao.lpush(key, value);  }  @Override  public String lpop(String key) {  return redisQueueDao.lpop(key);  }  } |

接口就不附上了。

## 8、RedisRun

消费者具体执行类。UUID只是在本例中做标识用。

|  |
| --- |
| package org.leo.ssm.redis;  import java.util.UUID;  import org.leo.ssm.redis.service.RedisQueueService;  import org.springframework.beans.factory.annotation.Autowired;  public class RedisRun implements Runnable {  @Autowired  private RedisQueueService redisQueueService;  public volatile boolean exit = false;  public RedisRun(RedisQueueService redisQueueService) {  super();  this.redisQueueService = redisQueueService;  }  @Override  public void run() {  String runUUID = UUID.randomUUID().toString().toUpperCase();  int count = 0;  while (!exit) {  // **tq**是本消费者程序从Redis中取消息的key  String result = redisQueueService.lpop("**tq**");  if (null != result) {  // 取出消息之后进行业务处理  System.out.println(System.currentTimeMillis() + ":" + result + "--" + runUUID);  } else {  System.out.println("没有取到信息，休息1秒");  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  System.out.println("=========InterruptedException");  }  count++;  if (count == 10) {  System.out.println("连续10次没有取到信息，休息10秒");  try {  Thread.sleep(10000);  } catch (InterruptedException e) {  System.out.println("=========InterruptedException");  }  count = 0;  }  }  }  System.out.println("Thread Stop:" + runUUID);  }  } |

消费者程序会不停地从Redis中取消息，考虑到如果长时间没有消息进入队列，这样是蛮耗资源的，所以在程序后面加了休息的代码，没有从Redis取出消息，就休息1秒，连续10次没有取出消息，就休息10秒。可以自己修改。

另外还要考虑任务处理失败后消息的重发机制。代码里不再赘述。

## 9、RedisInit

|  |
| --- |
| package org.leo.ssm.redis;  import java.util.UUID;  import org.leo.ssm.redis.service.RedisQueueService;  import org.springframework.beans.factory.DisposableBean;  import org.springframework.beans.factory.InitializingBean;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;  import org.springframework.stereotype.Component;  @Component("redisInit")  public class RedisInit implements InitializingBean, DisposableBean {  @Autowired  private RedisQueueService redisQueueService;  @Autowired  private ThreadPoolTaskExecutor taskExecutor;  private RedisRun redisRun;  private String threadUUID;  public RedisInit() {  threadUUID = UUID.randomUUID().toString().toUpperCase();  System.out.println("------RedisInit-----Start:" + threadUUID);  }  @Override  public void afterPropertiesSet() throws Exception {  redisRun = new RedisRun(redisQueueService);  taskExecutor.execute(redisRun);  }  @Override  public void destroy() throws Exception {  System.out.println("------RedisInit-----Destroy:" + threadUUID);  redisRun.exit = true;  for (;;) {  int count = taskExecutor.getActiveCount();  System.out.println("活跃的线程数 : " + count);  if (count == 0) {  taskExecutor.getThreadPoolExecutor().remove(redisRun);  taskExecutor.shutdown();  break;  } else {  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  }  } |

本类实现了InitializingBean, DisposableBean接口，保证了程序在Tomcat中启动后将RedisRun加载至线程池并运行，以及在销毁时移除并关闭。

## 10、生产者

|  |
| --- |
| package org.leo.ssm;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.leo.ssm.redis.service.RedisQueueService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.test.context.ContextConfiguration;  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration("classpath:spring/applicationContext.xml")  public class TestPush {  @Autowired  private RedisQueueService redisQueueService;  @Test  public void testPush(){  **redisQueueService.lpush("tq", "testQueue");**  }  } |

这是我在工程中写的一个测试类，实际情况中不要这么写，因为要加载applicationContext.xml这导致这个生产者程序在启动的时候，一个消费者也会启动。

这个程序应该写在别的工程中，其实关键代码就一句话：

|  |
| --- |
| **redisQueueService.lpush("tq", "你要发的消息");** |