RabbitMq小Demo

安装好RabbitMq后，配合Spring使用RabbitMq

1. 引入依赖

```
<!--rabbit mq-->
<dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>5.3.0</version>
</dependency>
```

2. 配置RedisPool

```
package com.sm.redis;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Primary;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;
import redis.clients.jedis.Jedis;

@Component
@Primary
public class JedisPool extends redis.clients.jedis.JedisPool {
    static String  host = "127.0.0.1";
    static int port = 6379;
    public JedisPool() {
        this.port = 6379;
        this.host = "127.0.0.1";
    }
    public JedisPool(String host, int port) {
        super(host, port);
    }
    public static Jedis getJedis() {
        return new Jedis(host, port);
    }
}
```

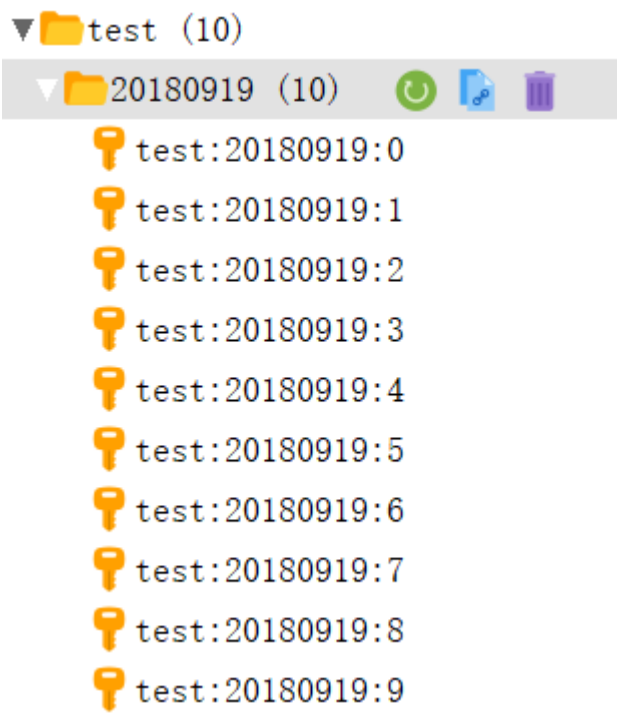
3. 编写RabbitMq测试

```
package com.sm.Component;

import com.rabbitmq.client.*;
import com.sm.redis.JedisPool;
```

```
import redis.clients.jedis.Jedis;

import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeoutException;

/**
 * @Auth justinniu
 * @Date 2018/9/19
 * @Desc
 */
public class RedisTest {
    private static final String EXCHANGE_NAME2 = "task_exchange2";
    private static final String TASK_QUEUE_NAME2 = "task_queue2";
    private static DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    //配置RabbitMq，并返回一个Channel对象
    private Channel getChannel() throws IOException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        Connection connection = null;
        try {
            connection = factory.newConnection();
        } catch (TimeoutException e) {
            e.printStackTrace();
        }
       return connection.createChannel();

    }
    //生产者
    public void Producer() {
        try {
            Channel channel = getChannel();
            /*
            *   声明Exchange，以Topic形式发布，Topic形式是指RoutKey以key.*的形式发布，
            *   绑定了Bindingkey .* Queue 都可以获取到消息 这些消息
            */
            channel.exchangeDeclare(EXCHANGE_NAME2, BuiltinExchangeType.TOPIC);
            /*
            * 模拟，把消息对应的数据放到Redis里，消费者根据推入Mq的key 来获取数据
            */
            Jedis jedis = JedisPool.getJedis();
            String key = "test:20180919:";
            for (int i = 0; i < 10; i++) {
                jedis.set(key + i, i + "");
                channel.basicPublish(EXCHANGE_NAME2, "zwt.123",
MessageProperties.PERSISTENT_TEXT_PLAIN, (key + i).getBytes("UTF-8"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }

    public void Consumer() {
        try {
            Channel channel = getChannel();
            //声明Exchange
            channel.exchangeDeclare(EXCHANGE_NAME2, BuiltinExchangeType.TOPIC);
            //声明queue
            channel.queueDeclare(TASK_QUEUE_NAME2, true, false, false,null);
            //声明queue绑定的Key， 已经对应的Exchange
            channel.queueBind(TASK_QUEUE_NAME2, EXCHANGE_NAME2, "zwt.*");
            //创建一个消费者
            Consumer consumer = new DefaultConsumer(channel) {
                @Override
                public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties,
                                           byte[] body) throws IOException {
                     //获取消息
                    String message = new String(body, "UTF-8");

                    System.out.println(df.format(new Date()) + " [x] Received '"  + "
Consumer1 "+ message + "'");
                    try {
                        System.out.println(JedisPool.getJedis().get(message));
                    } finally {
                        System.out.println(" [x] Done");
                        channel.basicAck(envelope.getDeliveryTag(), false);
                    }
                }
            };
            //声明消费者
            channel.basicConsume(TASK_QUEUE_NAME2, false, consumer);

        } catch (Exception e) {
            e.printStackTrace();
        }

    }
    public static void main(String[] args) throws InterruptedException {
        RedisTest redisTest = new RedisTest();
        redisTest.Producer();
        Thread.sleep(1000);
        redisTest.Consumer();
    }

}
```

测试结果如下




```
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:0'
0
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:1'
1
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:2'
2
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:3'
3
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:4'
4
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:5'
5
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:6'
6
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:7'
7
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:8'
8
 [x] Done
2018-09-19 11:39:18 [x] Received ' Consumer1 test:20180919:9'
9
 [x] Done
```

我的理解是一个Exchange相等与一个收费站，这个收费站在多条隧道前面，隧道相当于queue.

如果是work模式，隧道抢着让你走，但你只能走一条隧道

如果是fanout模式，相当于国庆节高速公路限时免费，所有的queue都能走，

如果是routing模式，这时候大型车只能走声明了大型车的queue，小型车只能走声明小型车的queue，车的类型可以有很多很多种，queue也可以绑定很多很多车类型。

如果是topic模式，类似限行，只能是车牌号以什么什么开头的才能走某个隧道