一、ActiveMQ结合Spring开发

发送端

引入相关jar包

Spring提供了对JMS的支持，需要添加Spring 支持JMS的包

```
      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-jms</artifactId>
         <version>${spring.version}</version>
       </dependency>
       <dependency>
         <groupId>org.apache.activemq</groupId>
         <artifactId>activemq-all</artifactId>
         <version>5.15.0</version>
       </dependency>

       <dependency>
         <groupId>org.apache.commons</groupId>
         <artifactId>commons-pool2</artifactId>
         <version>2.4.2</version>
       </dependency>
```

配置Spring文件

```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:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://code.alibabatech.com/schema/dubbo
        http://code.alibabatech.com/schema/dubbo/dubbo.xsd" default-autowire="byName">

    <bean id="connectionFactory" class="org.apache.activemq.pool.PooledConnectionFactory"
destroy-method="stop">
        <property name="connectionFactory">
            <bean class="org.apache.activemq.ActiveMQConnectionFactory">
                <property name="brokerURL">
                    <value>tcp://192.168.190.101:61616</value>
                </property>
            </bean>
        </property>
        <property name="maxConnections" value="50"/>
    </bean>
    <bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">
        <constructor-arg index="0" value="register-queue"/>
    </bean>

    <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
        <property name="connectionFactory" ref="connectionFactory"/>
        <property name="defaultDestination" ref="destination"/>
        <property name="messageConverter">
            <bean class="org.springframework.jms.support.converter.SimpleMessageConverter"/>
        </property>
    </bean>
```

编写发送端代码

```java
package com.softwore.zgd.dubbo.order.jms;

import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import javax.jms.TextMessage;

/**
 * @author 风骚的GRE
 * @Description JMS发送端代码
 * @date 2018/1/30.
 */
public class SpringJmsSender {

    public static void main(String[] args) {
        ClassPathXmlApplicationContext context=
                new ClassPathXmlApplicationContext(
                        "classpath:META-INF/spring/service-jms.xml");

        JmsTemplate jmsTemplate=(JmsTemplate) context.getBean("jmsTemplate");

        jmsTemplate.send(new MessageCreator() {
            public Message createMessage(Session session) throws JMSException {
                TextMessage message=session.createTextMessage();
                message.setText("Hello,风骚的GRE");
                return message;
            }
        });
    }
}
```

接收端

前面两步一致，这里不再累述

编写接收端代码

```java
package com.softwore.zgd.dubbo.order.jms;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

/**
 * @author 风骚的GRE
 * @Description TODO
 * @date 2018/1/30.
 */
public class SpringJmsReceiver {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context=
                new ClassPathXmlApplicationContext(
                        "classpath:META-INF/spring/service-jms.xml");
        JmsTemplate jmsTemplate=(JmsTemplate) context.getBean("jmsTemplate");

        String msg=(String)jmsTemplate.receiveAndConvert();

        System.out.println(msg);
    }

}
```

Spring发布订阅配置

以事件通知方式来配置消费者

更改发送端和消费端的xml

```xml
<bean id="jmsContainer"
class="org.springframework.jms.listener.DefaultMessageListenerContainer">
        <property name="connectionFactory" ref="connectionFactory"/>
        <property name="destination" ref="destination"/>
        <property name="messageListener" ref="messageListener"/>
    </bean>

    <bean id="messageListener"
class="com.softwore.zgd.dubbo.order.jms.RegisterQueueMessageListener"/>
</beans>
```

```
<bean id="jmsContainer"
class="org.springframework.jms.listener.DefaultMessageListenerContainer">
        <property name="connectionFactory" ref="connectionFactory"/>
        <property name="destination" ref="destination"/>
        <property name="messageListener" ref="messageListener"/>
    </bean>

    <bean id="messageListener" class="com.softwore.zgd.dubbo.order.jms.SpringJmsListener"/>
```

增加监听类

发送端：

```
package com.softwore.zgd.dubbo.user.jms;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

/**
 * @author 风骚的GRE
 * @Description 注册队列消息监听器
 * @date 2018/1/30.
 */
@Component
public class RegisterQueueMessageListener implements MessageListener {
    Logger logger= LoggerFactory.getLogger(RegisterQueueMessageListener.class);

    public void onMessage(Message message) {
        TextMessage message1=(TextMessage) message;
        try {
            logger.info(message1.getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }

}
```

消费端：

```java
package com.softwore.zgd.dubbo.order.jms;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

/**
 * @author 风骚的GRE
 * @Description 注册队列消息监听器
 * @date 2018/1/30.
 */
@Component
public class SpringJmsListener implements MessageListener {
    Logger logger= LoggerFactory.getLogger(SpringJmsListener.class);

    public void onMessage(Message message) {
        TextMessage message1=(TextMessage) message;
        try {
            logger.info(message1.getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

启动容器

```java
package com.softwore.zgd.dubbo.order.jms;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

/**
 * @author 风骚的GRE
 * @Description TODO
 * @date 2018/1/30.
 */
public class SpringJmsReceiver {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context=
                new ClassPathXmlApplicationContext(
                        "classpath:META-INF/spring/service-jms.xml");
        try {
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        }

        /*JmsTemplate jmsTemplate=(JmsTemplate) context.getBean("jmsTemplate");

        String msg=(String)jmsTemplate.receiveAndConvert();

        System.out.println(msg);*/
    }

}
```

二、ActiveMQ支持的传输协议

client端和broker端的通讯协议

TCP、UDP 、NIO、SSL、Http（s）、vm****

三、ActiveMQ持久化存储

kahaDB 默认的存储方式

```
<persistenceAdapter>
    <kahaDB directory="${activemq.data}/kahadb"/>
 </persistenceAdapter>
```

中指定了kahaDB，并表明数据存储在 "activemq-data"目录下，日志文件最大长度是32MB。

比如一个实际的ActiveMQ的KahaDB存储方式下的数据目录如下：

```
[root@localhost kahadb]# ll -s -t -h
总用量 11M
32K -rw-r--r-- 1 root root 32K 1月  30 12:23 db.data
36K -rw-r--r-- 1 root root 33K 1月  30 12:23 db.redo
11M -rw-r--r-- 1 root root 32M 1月  30 12:23 db-1.log
  0 -rw-r--r-- 1 root root   0 1月  23 08:23 lock
```

可以看出，上面directory一共有四个文件：

- db.data

它是消息的索引文件。本质上是B-Tree的实现，使用B-Tree作为索引指向db-*.log里面存储的消息。

- db.redo

主要用来进行消息恢复。

- db-*.log 存储消息的内容。对于一个消息而言，不仅仅有消息本身的数据(message data)，而且还有(Destinations、订阅关系、事务...)

AMQ 基于文件的存储方式

写入速度很快，容易恢复。 文件默认大小是32M

JDBC 基于数据库的存储

ACTIVEMQ_ACKS： 存储持久订阅的信息 ACTIVEMQ_LOCK： 锁表（用来做集群的时候，实现master选举的表） ACTIVEMQ_MSGS： 消息表

LevelDB

5.8以后引入的持久化策略。通常用于集群配置

四、LevelDB

五、ActiveMQ的网络连接