国际物流云商系统第十四天

一.回顾

1.使用 Restful 风格开发海关报运平台的 Webservice

2.Redis 的引入

3.spring 与 redis 整合配置

4.使用 spring 整合 redis 实现权限树的优化

5.spring data redis 配置及使用

二.消息中间件

1.消息中间件介绍

消息队列中间件是分布式系统中重要的组件，主要解决应用耦合，异步消息，流量削锋等问题实现高性能，高可用，可伸缩和最终一致性架构

使用较多的消息队列有 **ActiveMQ**，RabbitMQ，ZeroMQ，Kafka，MetaMQ，RocketMQ

2.消息队列应用场景

以下介绍消息队列在实际应用中常用的使用场景。

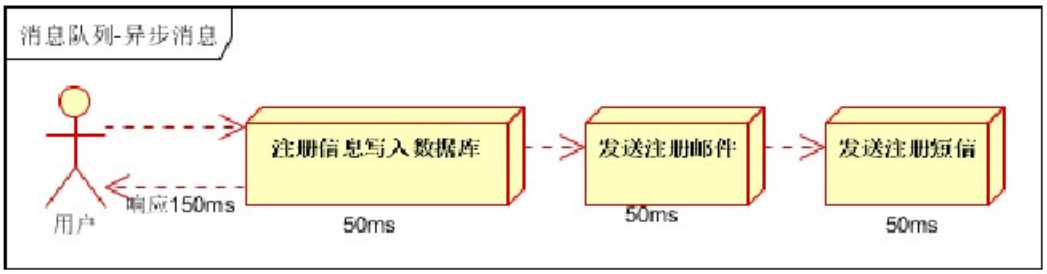
异步处理，应用解耦，流量削锋和消息通讯四个场景

2.1 异步处理

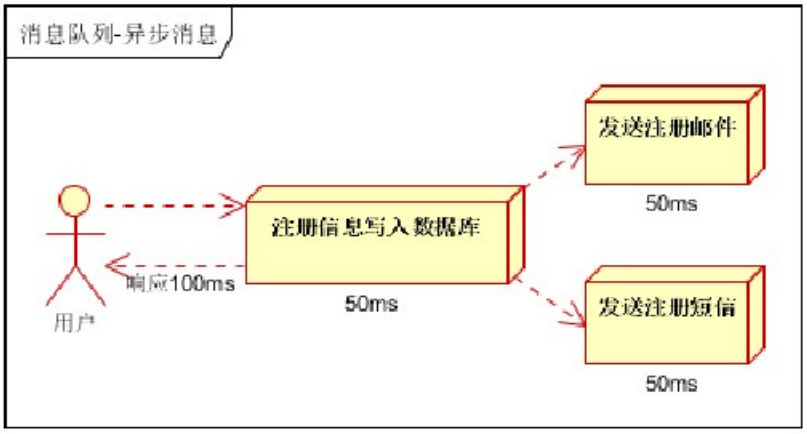
场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种：

1.串行的方式；2.并行方式

（1）串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端



（2）并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间

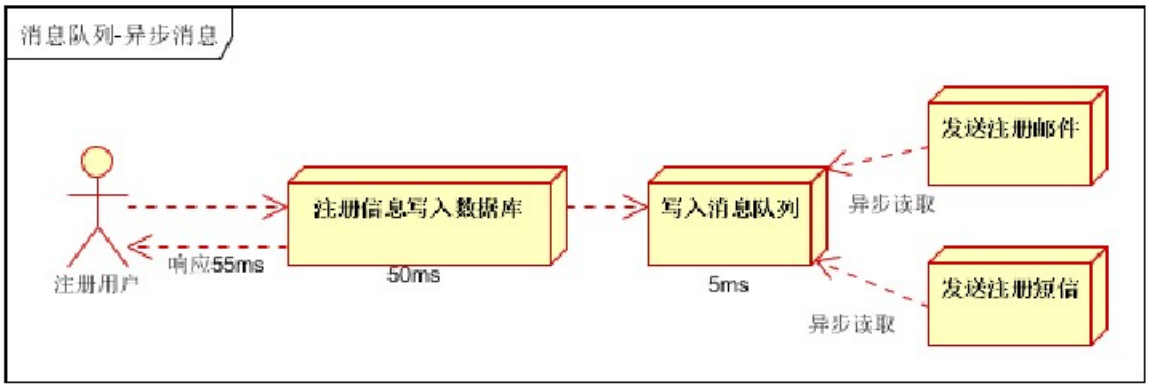


假设三个业务节点每个使用 50 毫秒钟，不考虑网络等其他开销，则串行方式的时间是 150 毫秒，并行的时间可能是 100 毫秒。

因为 CPU 在单位时间内处理的请求数是一定的，假设 CPU 在 1 秒内吞吐量是 100 次。则串行方式 1 秒内 CPU 可处理的请求量是 7 次（1000/150）。并行方式处理的请求量是 10 次（1000/100）

小结：如以上案例描述，传统的方式系统的性能（并发量，吞吐量，响应时间）会有瓶颈。如何解决这个问题呢？

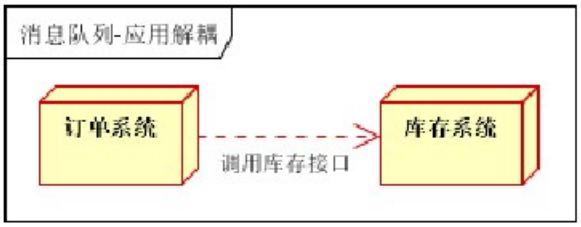
引入消息队列，将不是必须的业务逻辑，异步处理。改造后的架构如下：



按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是 50 毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是 50 毫秒。因此架构改变后，系统的吞吐量提高到每秒 20 QPS。比串行提高了 3 倍，比并行提高了 2 倍

2.2 应用解耦

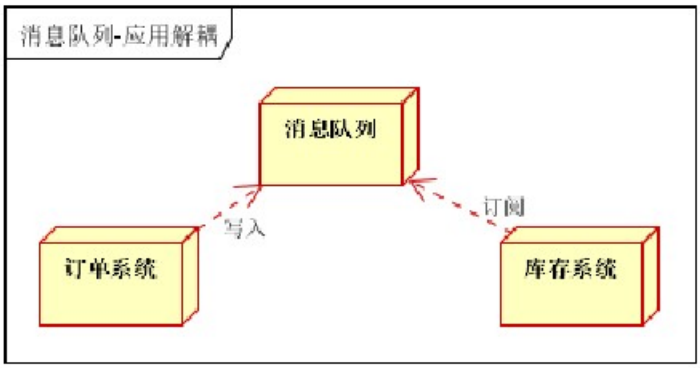
场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。如下图



传统模式的缺点：

- 假如库存系统无法访问，则订单减库存将失败，从而导致订单失败
- 订单系统与库存系统耦合

如何解决以上问题呢？引入应用消息队列后的方案，如下图：



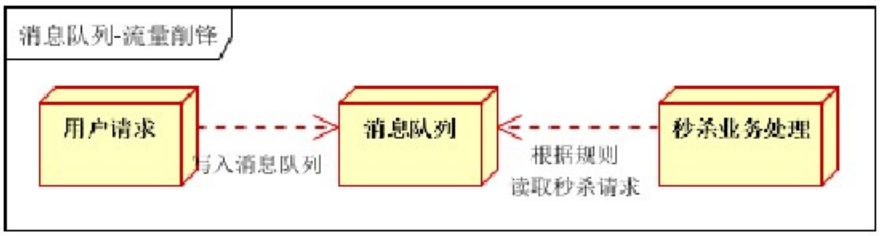
- 订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功
- 库存系统：订阅下单的消息，采用 pub/sub(发布/订阅)的方式，获取下单信息，库存系统根据下单信息，进行库存操作
- 假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦

2.3 流量削锋

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛

应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

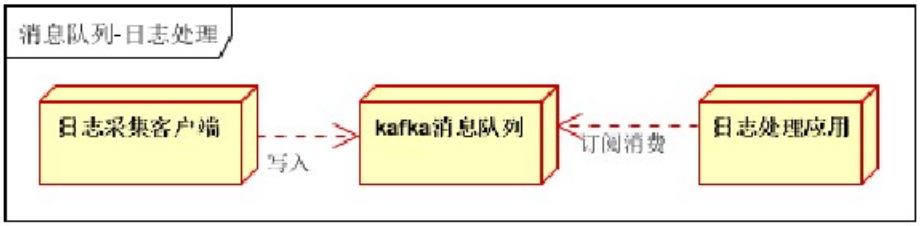
- 可以控制活动的人数
- 可以缓解短时间内高流量压垮应用



- 用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面
- 秒杀业务根据消息队列中的请求信息，再做后续处理

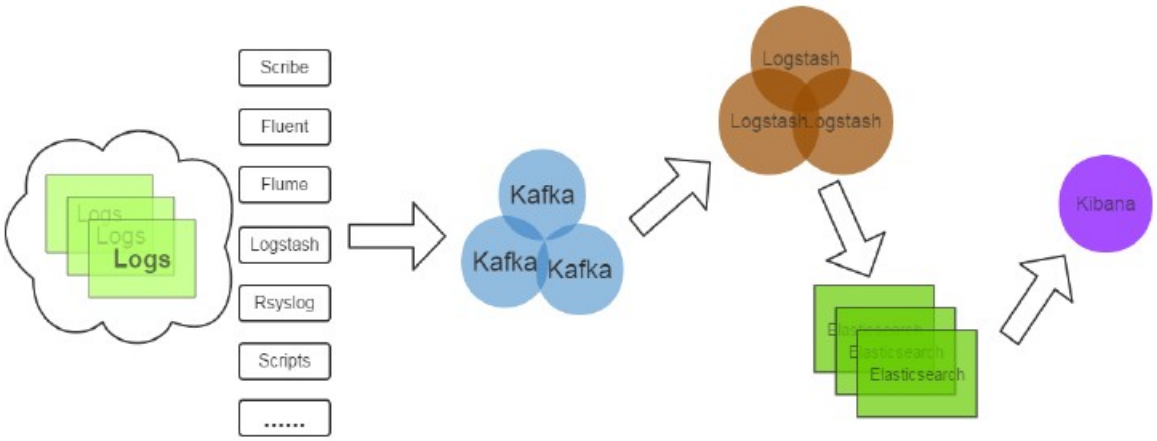
2.4 日志处理

日志处理是指将消息队列用在日志处理中，比如 Kafka 的应用，解决大量日志传输的问题。架构简化如下



- 日志采集客户端，负责日志数据采集，定时写受写入 Kafka 队列
- Kafka 消息队列，负责日志数据的接收，存储和转发

- 日志处理应用：订阅并消费 kafka 队列中的日志数据



(1)Kafka：接收用户日志的消息队列

(2)Logstash：做日志解析，统一成 JSON 输出给 Elasticsearch

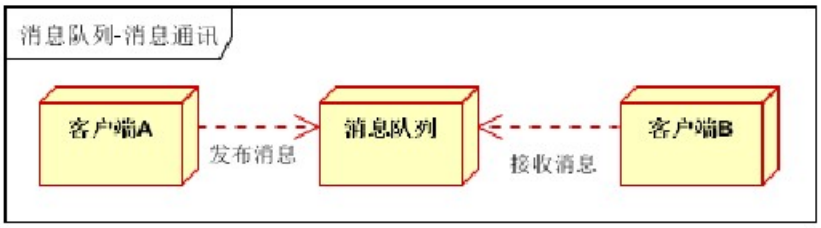
(3)Elasticsearch：实时日志分析服务的核心技术，一个 schemaless，实时的数据存储服务，通过 index 组织数据，兼具强大的搜索和统计功能

(4)Kibana：基于 Elasticsearch 的数据可视化组件，超强的数据可视化能力是众多公司选择 ELK stack 的重要原因

2.5 消息通讯

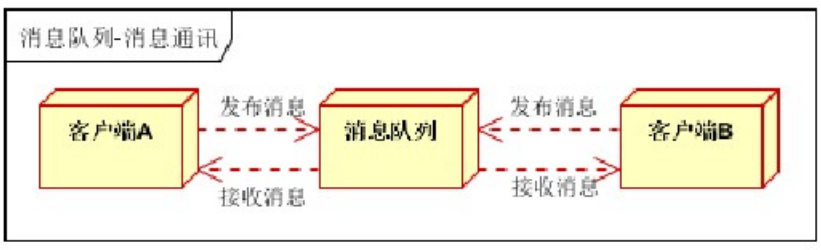
消息通讯是指，消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等

点对点通讯：



客户端 A 和客户端 B 使用同一队列，进行消息通讯。

聊天室通讯：



客户端 A，客户端 B，客户端 N 订阅同一主题，进行消息发布和接收。实现类似聊天室效果。

以上实际是消息队列的两种消息模式，点对点或发布订阅模式。模型为示意图，供参考。

三.JMS 消息服务

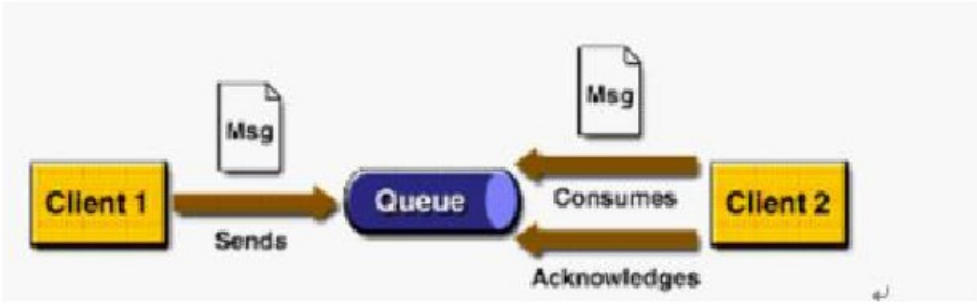
消息队列的 JAVAEE 规范 JMS 。JMS（Java Message Service,java 消息服务）API 是一个消息服务的标准/规范，允许应用程序组件基于 JavaEE 平台创建、发送、接收和读取消息。它使分布式通信耦合度更低，消息服务更加可靠以及异步性。

在 EJB 架构中，有消息 bean 可以无缝的与 JM 消息服务集成。在 J2EE 架构模式中，有消息服务者模式，用于实现消息与应用直接的解耦。

1.消息模型

在 JMS 标准中，有两种消息模型 P2P（Point to Point）,Publish/Subscribe(Pub/Sub)。

2.P2P 模式



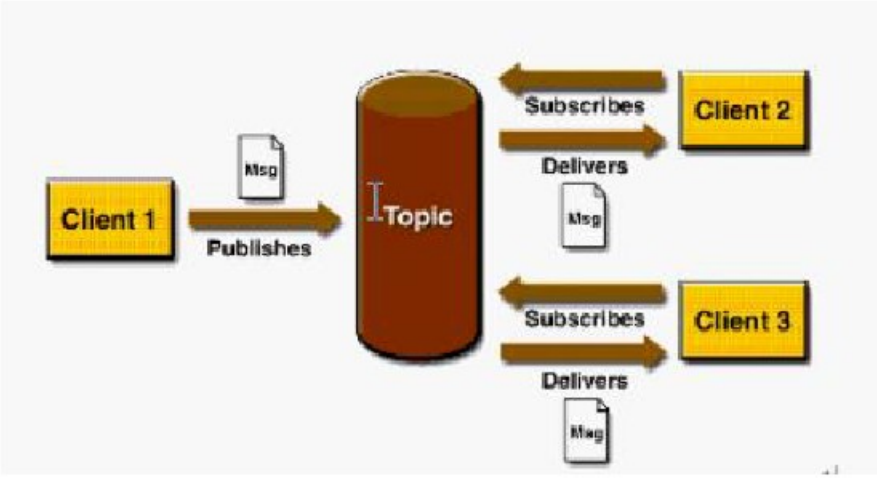
P2P 模式包含三个角色：消息队列（Queue），发送者(Sender)，接收者(Receiver)。每个消息都被发送到一个特定的队列，接收者从队列中获取消息。队列保留着消息，直到他们被消费或超时。

P2P 的特点

- 每个消息只有一个消费者（Consumer）(即一旦被消费，消息就不再在消息队列中)
- 发送者和接收者之间在时间上没有依赖性，也就是说当发送者发送了消息之后，不管接收者有没有正在运行，它不会影响到消息被发送到队列
- 接收者在成功接收消息之后需向队列应答成功

如果希望发送的每个消息都会被成功处理的话，那么需要 P2P 模式。

3.Pub/Sub 模式



包含三个角色主题（Topic），发布者（Publisher），订阅者（Subscriber） 多个发布者将消息发送到 Topic,系统将这些消息传递给多个订阅者。

Pub/Sub 的特点

- 每个消息可以有多个消费者
- 发布者和订阅者之间有时间上的依赖性。针对某个主题（Topic）的订阅者，它必须创建一个订阅者之后，才能消费发布者的消息
- 为了消费消息，订阅者必须保持运行的状态

为了缓和这样严格的时间相关性，JMS 允许订阅者创建一个可持久化的订阅。这样，即使订阅者没有被激活（运行），它也能接收到发布者的消息。

如果希望发送的消息可以不被做任何处理、或者只被一个消息者处理、或者可以被多个消费者处理的话，那么可以采用 Pub/Sub 模型。

4.消息消费

在 JMS 中，消息的产生和消费都是异步的。对于消费来说，JMS 的消息者可以通过两种方式来消费消息。

（1）同步

订阅者或接收者通过 receive 方法来接收消息，receive 方法在接收到消息之前（或超时之

前）将一直阻塞；

（2）异步

订阅者或接收者可以注册为一个消息监听器。当消息到达之后，系统自动调用监听器的 onMessage 方法。

JNDI：Java 命名和目录接口,是一种标准的 Java 命名系统接口。可以在网络上查找和访问服务。通过指定一个资源名称，该名称对应于数据库或命名服务中的一个记录，同时返回资源连接建立所必须的信息。

JNDI 在 JMS 中起到查找和访问发送目标或消息来源的作用。

5.JMS 编程模型

(1) ConnectionFactory

创建 Connection 对象的工厂，针对两种不同的 jms 消息模型，分别有 QueueConnectionFactory 和 TopicConnectionFactory 两种。可以通过 JNDI 来查找 ConnectionFactory 对象。

(2) Destination

Destination 的意思是消息生产者的消息发送目标或者说消息消费者的消息来源。对于消息生产者来说，它的 Destination 是某个队列（Queue）或某个主题（Topic）;对于消息消费者来说，它的 Destination 也是某个队列或主题（即消息来源）。

所以，Destination 实际上就是两种类型的对象：Queue、Topic 可以通过 JNDI 来查找 Destination。

(3) Connection

Connection 表示在客户端和 JMS 系统之间建立的链接（对 TCP/IP socket 的包装）。Connection 可以产生一个或多个 Session。跟 ConnectionFactory 一样，Connection 也有两种类型：QueueConnection 和 TopicConnection。

(4) Session

Session 是操作消息的接口。可以通过 session 创建生产者、消费者、消息等。Session 提供了事务的功能。当需要使用 session 发送/接收多个消息时，可以将这些发送/接收动作放到一个事务中。同样，也分 QueueSession 和 TopicSession。

(5) 消息的生产者

消息生产者由 Session 创建，并用于将消息发送到 Destination。同样，消息生产者分两种类型：QueueSender 和 TopicPublisher。可以调用消息生产者的方法（send 或 publish 方法）发送消息。

(6) 消息消费者

消息消费者由 Session 创建，用于接收被发送到 Destination 的消息。两种类型：QueueReceiver 和 TopicSubscriber。可分别通过 session 的 createReceiver(Queue)或 createSubscriber(Topic)来创建。当然，也可以 session 的 creatDurableSubscriber 方法来创建持久化的订阅者。

(7) MessageListener

消息监听器。如果注册了消息监听器，一旦消息到达，将自动调用监听器的 onMessage 方法。EJB 中的 MDB（Message-Driven Bean）就是一种 MessageListener。

四.消息队列 ActiveMQ

一般商用的容器，比如 WebLogic，JBoss，都支持 JMS 标准，开发上很方便。但免费的比如 Tomcat，Jetty 等则需要使用第三方的消息中间件。本部分内容介绍常用的消息中间件（Active MQ,Rabbit MQ，Zero MQ,Kafka）以及他们的特点。

1.ActiveMQ

ActiveMQ 是 Apache 出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现，尽管 JMS 规范出台已经是很久的事情了，但是 JMS 在当今的 J2EE 应用中间仍然扮演着特殊的地位。

ActiveMQ 特性如下：

1. 多种语言和协议编写客户端。语言: Java,C,C++,C#,Ruby,Perl,Python,PHP。应用协议： OpenWire,Stomp REST,WS Notification,XMPP,AMQP

2. 完全支持 JMS1.1 和 J2EE 1.4 规范 （持久化，XA 消息，事务)

3. 对 spring 的支持，ActiveMQ 可以很容易内嵌到使用 Spring 的系统里面去，而且也支持 Spring2.0 及更高版本的特性

4. 通过了常见 J2EE 服务器（如 Geronimo,JBoss 4,GlassFish,WebLogic)的测试，其中通过 JCA 1.5 resource adaptors 的配置，可以让 ActiveMQ 可以自动的部署到任何兼

容 J2EE 1.4 商业服务器上

5. 支持多种传送协议：in-VM,TCP,SSL,NIO,UDP,JGroups,JXTA
6. 支持通过 JDBC 和 journal 提供高速的消息持久化
7. 从设计上保证了高性能的集群，客户端-服务器，点对点
8. 支持 Ajax
9. 支持与 Axis 的整合
10. 可以很容易得调用内嵌 JMS provider，进行测试

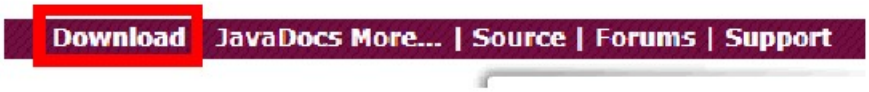
2.ActivieMQ 的下载及安装

Apache ActiveMQ ™ is the most popular and powerful open source messaging and Integration Patterns server.

Apache ActiveMQ is fast, supports many Cross Language Clients and Protocols, comes with easy to use Enterprise Integration Patterns and many advanced features while fully supporting JMS 1.1 and J2EE 1.4. Apache ActiveMQ is released under the Apache 2.0 License

Grab yourself a Download, try our Getting Started Guide, surf our FAQ or start Contributing and join us on our Discussion Forums.

在官网进行下载



进入下载页面

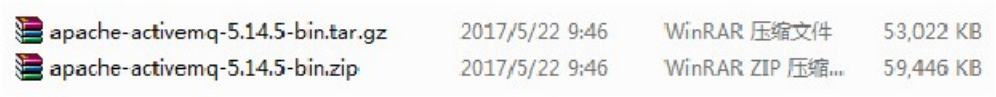
- ActiveMQ 5.4.1 Release
- ActiveMQ 5.4.2 Release
- ActiveMQ 5.4.3 Release
- ActiveMQ 5.5.0 Release
- ActiveMQ 5.5.1 Release
- ActiveMQ 5.6.0 Release
- ActiveMQ 5.7.0 Release
- ActiveMQ 5.8.0 Release
- ActiveMQ 5.9.0 Release
- ActiveMQ 5.9.1 Release

点击后进入下载页面

Getting the Binary Distributions

Description	Download Link	PGP Signature file of download
Windows Distribution	apache-activemq-5.9.0-bin.zip	apache-activemq-5.9.0-bin.zip.asc
Unix/Linux/Cygwin Distribution	apache-activemq-5.9.0-bin.tar.gz	apache-activemq-5.9.0-bin.tar.gz.asc

下载后的资源如下：



我们为了简化安装过程，直接解压 windows 版本的安装包并启动 redis 服务器

启动后进入 activeMQ 的后台管理页面

至此我们成功完成了 actvieMQ 的安装与启动。

3.项目中引入 ActiveMQ

1.引入 ActiveMQ 开发的 maven 坐标

```
<!-- activemq  start -->
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-all</artifactId>
  <version>5.2.0</version>
</dependency>
<!-- activemq  end -->

<!-- spring 与 mq整合 start -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>4.2.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.apache.xbean</groupId>
    <artifactId>xbean-spring</artifactId>
    <version>3.7</version>
</dependency>
<!-- spring 与 mq整合 end -->
```

4.编写 JMS 测试类发送消息

使用 JMS 原生 API 编写测试类，向消息中间件写入消息。

```
@Test
public void test(){
    try {
        //1.建立连接工厂
        ConnectionFactory factory   = new ActiveMQConnectionFactory();
        //2.创建连接
        Connection connection = factory.createConnection();
        connection.start();//开启连接
        //建立会话 第一个参数，代表是否使用事务，true代表使用事务，就要使用session.commit()进行提交
        Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
        //创建对象
        Queue queue = session.createQueue("helloworld");

        //创建一个生产者
        MessageProducer proceducer = session.createProducer(queue);

        //消息发送
        proceducer.send(session.createTextMessage("Hello,i am girl"));

        //提交
        session.commit();
        session.close();
        connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

发送成功后，控制台显示连接上服务器。

```
[  ActiveMQ Task] INFO  ort.failover.FailoverTransport  - Successfully connected to
tcp://localhost:61616
```

进入后台查看消息如下：

Queues

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued
helloworld	1	0	1	0

5.编写 JMS 消息的消费者

```
@Test
public void testConsumer() throws Exception{
    //1.连接工厂
    ConnectionFactory factory = new ActiveMQConnectionFactory();
    //2.创建连接
    Connection conn = factory.createConnection();
    conn.start();
    //3.创建会话对象 第一个参数，代表是否使用事务，true代表使用事务，就要使用session.commit()进行提交
    Session session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
    //4.创建Queue或Topic对象
    Queue queue = session.createQueue("helloworld");
    //5.创建消费者
    MessageConsumer consumer = session.createConsumer(queue);
    //6.消费
    while(true){
        TextMessage message = (TextMessage)consumer.receive(10000);
        if(message!=null){
            System.out.println(message.getText());
        }else{
            break;
        }
    }
    session.close();
    conn.close();
}
```

采用监听器的方式实现消息的消费

```
@Test  //使用监听器来实现消费
public void testConsumer02() throws Exception{
    //1.连接工厂
    ConnectionFactory factory = new ActiveMQConnectionFactory();
    //2.创建连接
    Connection conn = factory.createConnection();
    conn.start();
    //3.创建会话对象  第一个参数，代表是否使用事务，true代表使用事务，就要使用session.commit()进行提交
    Session session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
    //4.创建Queue或Topic对象
    Queue queue = session.createQueue("helloworld");
    //5.创建消费者
    MessageConsumer consumer = session.createConsumer(queue);
    //6.消费
    consumer.setMessageListener(new MessageListener() {
        //每次接收消息，自动调用onMessage方法
        public void onMessage(Message message) {
            TextMessage msg = (TextMessage) message;
            try {
                System.out.println(msg.getText());
            } catch (JMSException e) {
                e.printStackTrace();
            }
        }
    });
    while(true){
    }
    //session.close();
    //conn.close();
```

五．实现 Spring 与 ActiveMQ 整合

1.引入 Spring 与 ActiveMQ 整合的约束文件

```
xmlns:amq="http://activemq.apache.org/schema/core"
xmlns:jms="http://www.springframework.org/schema/jms"
http://www.springframework.org/schema/jms
http://www.springframework.org/schema/jms/spring-jms.xsd
http://activemq.apache.org/schema/core
http://activemq.apache.org/schema/core/activemq-core.xsd
```

2.编写 ActiveMQ 连接工厂

```
<!-- ActiveMQ 连接工厂 -->
<!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->
<!-- 如果连接网络：tcp://ip:61616；未连接网络：tcp://localhost:61616 以及用户名，密码-->
<amq:connectionFactory id="amqConnectionFactory"
    brokerURL="tcp://localhost:61616" userName="admin" password="admin"  />
```

3.编写 Spring Caching 连接工厂

```
<!-- Spring Caching连接工厂 -->
<!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->
<bean id="connectionFactory" class="org.springframework.jms.connection.CachingConnectionFactory">
    <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->
    <property name="targetConnectionFactory" ref="amqConnectionFactory"></property>
    <!-- Session缓存数量 -->
    <property name="sessionCacheSize" value="100" />
</bean>
```

4.编写 JmsTemplate 模板

```
<!-- 定义JmsTemplate的Queue类型 -->
<bean id="jmsQueueTemplate" class="org.springframework.jms.core.JmsTemplate">
    <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->
    <constructor-arg ref="connectionFactory" />
    <!-- 非pub/sub模型（发布/订阅），即队列模式 -->
    <property name="pubSubDomain" value="false" />
</bean>

<!-- 定义JmsTemplate的Topic类型 -->
<bean id="jmsTopicTemplate" class="org.springframework.jms.core.JmsTemplate" >
    <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->
    <constructor-arg ref="connectionFactory" />
    <!-- pub/sub模型（发布/订阅） -->
    <property name="pubSubDomain" value="true"  />
</bean>
```

5.编写发送消息的测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:spring/applicationContext-mq.xml")
public class SpringMQTest {
    @Autowired
    @Qualifier("jmsQueueTemplate")
    private JmsTemplate jmsQueueTemplate;

    @Autowired
    @Qualifier("jmsTopicTemplate")
    private JmsTemplate jmsTopicTemplate;
    @Test
    public void testMq(){
        //发送队列消息
        jmsQueueTemplate.send("spring_queue", new MessageCreator() {
            public Message createMessage(Session session) throws JMSException {
                return session.createTextMessage("你好，队列消息");
            }
        });
        //发送Topic消息
        jmsTopicTemplate.send("spring_topic", new MessageCreator() {
            public Message createMessage(Session session) throws JMSException {
                return session.createTextMessage("你好，订阅消息");
            }
        });
    }
}
```

6.定义消息的消费者

1.定义 Queue 消息的消费者

```
@Component("queueConsumer1")
public class QueueConsumer1 implements MessageListener {

    public void onMessage(Message message) {
        try {
            System.out.println("消费者QueueCustomer1:"+((TextMessage)message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

2.定义 Topic 消息的消息者

```
@Component("topicConsumer1")
public class TopicConsumer1 implements MessageListener {

    public void onMessage(Message message) {
        try {
            System.out.println("消费者TopicConsumer1:"+((TextMessage)message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }

}
```

7. 配置 spring 与 activeMQ 整合的配置文件

```
<!-- 扫描包 -->
<context:component-scan base-package="com.itcast.jms.consumer" />

<!-- ActiveMQ 连接工厂 -->
<amq:connectionFactory id="amqConnectionFactory"
    brokerURL="tcp://localhost:61616" userName="admin" password="admin"  />

<!-- Spring Caching连接工厂 -->
<bean id="connectionFactory" class="org.springframework.jms.connection.CachingConnectionFactory">
    <property name="targetConnectionFactory" ref="amqConnectionFactory"></property>
    <property name="sessionCacheSize" value="100" />
</bean>

 <!-- Spring JmsTemplate 的消息生产者 start-->
<jms:listener-container  destination-type="queue" container-type="default" connection-factory="connectionFactory">
   <jms:listener destination="spring_queue" ref="queueConsumer1"/>
</jms:listener-container>

 <jms:listener-container  destination-type="topic" container-type="default" connection-factory="connectionFactory">
   <jms:listener destination="spring_topic" ref="topicConsumer1"/>
   <jms:listener destination="spring_topic" ref="topicConsumer2" />
</jms:listener-container>
```

8.测试 Spring 与 activeMQ 整合的消息消费

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:spring/applicationContext-mq-consumer.xml")
public class SpringMQConsumerTest2 {

    @Test
    public void testMq(){
        while(true){
            //为了防止进程死掉
        }
    }
}
```

六.总结

1.消息中间件的作用
2.消息中间件的使用场景
3.JMS 的消息模型
4.消息中间件 ActiveMQ 介绍
5.ActiveMQ 的引入
6.Spring 与 ActiveMQ 整合开发