1.什么是MQ，有什么用?

MQ 是message queue ,消息队列，也叫消息中间件，遵守JMS（java message service）规范的一种软件。(同时还有另一个叫AMQP的应用层协议，语言无关性不受产品 语言等限制，rabbitMQ支持这个 )

是类似于数据库一样需要独立部署在服务器上的一种应用，提供接口给其他系统调用。

主要用于各个系统之间通信的解耦。

举例：

比如登陆系统，在登陆之后需要调用短信系统给用户发短信说已经登陆，同时还需要调用日志系统记录登陆日志，需要调用积分系统对登陆签到的积分进行增加 等等等。

这种情况下，登陆系统和日志系统，短信系统，积分系统等等 强耦合，其中存在可能调用失败，信息丢失等风险，同时会提高系统复杂度。

比如说登陆之后调用日志系统失败，那么该次登陆的日志信息就会丢失，无法再找回。

而且顺序执行，会导致登陆系统运行效率低。

那么如果使用消息中间件,登陆之后只需要将任务推入到消息队列中，就不用去管了。其他系统则从队列中去获取任务。

实现解耦和异步调用 （异步是相对于同步而言，同步是就等待，当系统执行某个任务的时候，一定要等到该任务结束，系统才会继续往下执行，异步则不等待。）

同时还有可以实现横向拓展 安全可靠优点

2.常见MQ的类型

activeMQ 对java支持良好，缺点是对其他语言支持不够友好，适合中小企业系统

rabbitMQ 对java支持良好，对其他语言也支持良好，跨平台，语言无关

kaffka 日志消息中间件 支持大数据场景

3.JMS规范

MQ实现参照了jms规范，（规范就是一种约定）

该规范中包括

提供者：实现jms规范的中间件服务器

客户端：发送或者接受消息的应用程序

生产者/发布者:创建并发送消息的客户端

消费者/订阅者:接受并处理消息的客户端

消息：应用程序之间传递的内容

消息模式：在客户端之间传递消息的方式，jms中定义了主题和队列两种模式

主题模式：

假如发布者发布了100条消息，那么如果有n个订阅者，每个订阅者都可以获取到100条消息。即订阅者可以获取到所有的消息（但如果订阅是在主题发布消息之后，则获取不到任何消息，只能获取到订阅时间之后主题的发布的消息，比如说A订阅了B, B发布了消息，C再订阅了B,那么只有a能够获取到消息，C不能，因为它的订阅行为发生在B发布消息之后）

队列模式:

假如生产者发送了100条消息，如果有n个消费者，那么每个订阅者加起来获取到的消息总数是100。

没有时间上限制。只要队列中有消息，消费者可以任意时间去取消息，一个消息只能被一个消费者消费。

4.jms约定的接口

ConnectionFactory 获取与MQ服务连接的工厂类

Connection 与MQ服务的连接， 由ConnectionFactory 创建

Session 会MQ服务的会话 由Connection 创建

MessageProducer 消息生产者，由Session 创建

MessageConsumer 消息消费者 由Session创建

Message 消息，由Session创建

Desination 消息的目的地

5.原生实际代码例子

一.原生mq用法 队列， 创建生产者消费者模式

生产者发送消息

------------------------

//1.创建connectionFactory 与mq服务器进行连接

ConnectionFactory connectionFactory=new

XXMQConnectionFactory(url) ; //XX 为该mq的实现 比如activeMQ rabbitMQ等

// url为服务器地址 tcp格式 比如 tcp://xxxx:xxx

//2.创建Connection

Connection connection=connectionFactory.createConnection();

//3.启动连接

connection,start();

//4.创建会话

Session session=connection.createSession(XXX); //xx为该session的创建时候的参数 比如设定事务，模式等等，具体依据不同的实现形式

//5.创建一个目标

Destination destination=session.createQueue(queueName);

//6.创建一个生产者

MessageProducer producer=session.createProducer(destination);

//7.创建消息

TextMessage textMessage=session.createTextMessage("text");

//8.发送消息

producer.send(textMessage);

//9.关闭连接

connection.close();

----------------

消费者获取消息

//1.创建connectionFactory 与mq服务器进行连接

ConnectionFactory connectionFactory=new

XXMQConnectionFactory(url) ; //XX 为该mq的实现 比如activeMQ rabbitMQ等

// url为服务器地址 tcp格式 比如 tcp://xxxx:xxx

//2.创建Connection

Connection connection=connectionFactory.createConnection();

//3.启动连接

connection,start();

//4.创建会话

Session session=connection.createSession(XXX); //xx为该session的创建时候的参数 比如设定事务，模式等等，具体依据不同的实现形式

//5.创建一个目标

Destination destination=session.createQueue(queueName);

//6.创建一个消费者

MessageConsumer consumer=session.createConsumer(destination);

//7.创建一个监听器

consumer.setMessageListener(

new MessageListener(){

public void onMessage(Message message){

TextMessage textMessage=(TextMessage)message;

//8.获取消息

textMessage.getText();

}});

//9.关闭连接

connection.close();

二.原生mq用法 主题， 创建发布者 订阅者模式

发布者----

//1.创建connectionFactory 与mq服务器进行连接

ConnectionFactory connectionFactory=new

XXMQConnectionFactory(url) ; //XX 为该mq的实现 比如activeMQ rabbitMQ等

// url为服务器地址 tcp格式 比如 tcp://xxxx:xxx

//2.创建Connection

Connection connection=connectionFactory.createConnection();

//3.启动连接

connection,start();

//4.创建会话

Session session=connection.createSession(XXX); //xx为该session的创建时候的参数 比如设定事务，模式等等，具体依据不同的实现形式

//5.创建一个目标

Destination destination=session.createTopic(topicName);

//6.创建一个生产者

MessageProducer producer=session.createProducer(destination);

//7.创建消息

TextMessage textMessage=session.createTextMessage("text");

//8.发送消息

producer.send(textMessage);

//9.关闭连接

connection.close();

---订阅者

//1.创建connectionFactory 与mq服务器进行连接

ConnectionFactory connectionFactory=new

XXMQConnectionFactory(url) ; //XX 为该mq的实现 比如activeMQ rabbitMQ等

// url为服务器地址 tcp格式 比如 tcp://xxxx:xxx

//2.创建Connection

Connection connection=connectionFactory.createConnection();

//3.启动连接

connection,start();

//4.创建会话

Session session=connection.createSession(XXX); //xx为该session的创建时候的参数 比如设定事务，模式等等，具体依据不同的实现形式

//5.创建一个目标

Destination destination=session.createTopic(topicName);

//6.创建一个消费者

MessageConsumer consumer=session.createConsumer(destination);

//7.创建一个监听器

consumer.setMessageListener(

new MessageListener(){

public void onMessage(Message message){

TextMessage textMessage=(TextMessage)message;

//8.获取消息

textMessage.getText();

}});

//9.关闭连接

connection.close();

6.spring集成下的代码例子

ConnectionFactory用于管理连接的连接工厂 这个是

spring提供地连接池，spring提供了SingleConnectionFactory和

CachingConnectionFactory

JmsTemplate用于发送和接受消息的模板类，spring提供的 只要注入这个bean，既可以

用jmsTempalate方便的操作jms 不需要像之前那样写一堆重复代码

这个是线程安全的

MessageListerner 消息监听器，实现了一个onMessage方法，该方法只接收一个Message参数

发送消息的方法

@Autowired

JmsTemplate jmsTemplate;

@Autowired

Destination destination;

jmsTemplate.send(destination,new MessgaeCreator(){

public Message createMessage(Session session) {

TextMessage textMessage=session.createTextMessage("message");

return textMessage;

}});

JmsTemplate实际上就是做了一层封装而已 简化了用法 ，可以参考里面的源代码

---------------------