1. JMS(JAVA MESSAGE SERVICE) :java EE 消息服务规范接口， 用于两个程序之间 或 分布式 系统中 发布消息进行异步通信

JMS提供了应用之间的异步通信机制，当异步发送消息时，客户端不需要等待服务端处理消息结果

1. 适用场景

有A/B 两个应用程序 ， 他们之间也无调用关系 ， 更不知道各自 方法内部的实现 。

B可能对 A 程序的信息感兴趣 。更有可能N个B 对A都感兴趣 。

JMS 就是解决这类事情的, A 发送消息 ，B不需要在线， 当B上线时 jms消息服务能保证发送消息给A

1. 特点 ：

异步 :客户端不用主动发送请求 获取消息 ， 消息服务会自动推送

可靠 ：保证消息只传送一次， 避免消息重复 创建 。

Jms 常用类 ：

1. connectionFactory 连接工厂 接口，定义创建连接规范 ，连接provider服务
2. Connection 连接接口 封装连接信息
3. Session（类似hibernate的session）会话上下文，用于创建队列queue、

创建消息mesage、创建生产者producer、创建消费者comsumer等操作

1. Destination 消息发送和接收的 目的地(保存消息的地方)
2. Acknowledge 签收 ：当消费者接收到消息后 应基于中间件响应 ，当session提 交时会签收
3. Client 用于收 、 发消息的 java程序应用
4. No-jms client 使用jms provider 应用替换jms API实现收发消息的功能 （一般用 于扩展）
5. Administered object ：预定义JMS对象 ，通常在provider 规范中定义 提供给JMS 客户端来访问 ，如 connectionFactoey 、destination

Jms对象 ：（1） Destination：消息被寻址、发送以及接收的对象

p2p queue作为Destination

Pub/Sub topic主题作为Destinatio

1. Provider：实现jms的消息中间件服务（比如 ActiveMQ ）
2. Messsage： jms消息 ，jms消息由三部分组成
   1. 消息头 消息头属性都应有get/set方法
   2. 消息体
3. Producer：消息生产者，创建和发送jms消息的应用
4. Customer 消息的消费者 接受和处理jms消息的应用
   1. 同步消费 ：如果有消息就处理 ，没消息就阻塞 直到消息到达
   2. 异步消费 ：客户端注册监听器，当消息到达的时候 mq会回调监听
5. Domains: jms 规范定义的 消息传递域 ； p2p 和 pub/sub

****点对点（队列）消息传送模型 P2P****

程序由消息队列，发送者，接收者组成。每一个消息发送给一个特殊的消息队列， 该队列保存了所有发送给它的消息(除了被接收者消费掉的和过期的消息)。

特点 ： I、每个消息只能有一个消费者 （一个队列可以保存多个消息）

II、生产者和消费者面向的都是对列 .当消息发送者发送消息的时候，无论接收者程序在不在运行，都能往对列中存消息 ，无论生产者是否运行 只要对列中有消息消费者就可以取消息；

III、当接收者收到消息的时候，会发送确认收到通知,然后消息从队列中删除

III、消息发送者和消息接受者并没有时间依赖性

****发布/订阅消息传送模型**** Publish/Subscribe

发布者发布一个消息，该消息通过topic传递给客户端 。在这种模型中，发布者和订阅者彼此不知道对方，是匿名的且可以动态发布和订阅topic。topic主要用于保存和传递消息，且会一直保存消息直到消息被传递给客户端。

特点 ： I、一个消息可以传递给多个订阅者

II、发布者和订阅者有时间依赖性，只有当客户端创建订阅后才能接受消息，且 订阅者需一直保持活动状态以接收消息。 如果不订阅就不能接收到消息

III、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 。非持久化订阅 ，即使消费者订阅 如果离线 也接受不到消息

IIII、消息发送的方式， 是将消息复制 然后发送给不同的接收者

1. JMS消息的组成
   1. 消息头 ：包含识别信息 和 路由信息

消息头中的属性

* + - 1. JMS Destination 目的地 主要指queue 和 topic ，根据createQueue和createTopic 自动分配 开发者设置
      2. JmsDeliverymode 传送模式 开发者设置（可选） ：

1.持久模式 为了保证数据一定可以发送或接受 数据需要被持久 化 ，即使即使发生故障 也能接受到

2 .非持久模式

c.JMSExpiration 消息过期时间 ，发送 send方法 可以设置过期时间如果timeToLive 为0 表示永不过期 开发者设置（可选）

d.JMSpriority 消息发送优先级 ，发送 send方法 可以设置优先级

0 - 4 普通消息 ，5 -9 加急消息 值越大 ，优先级越高 默认值是 4 ， 自动分配 开发者设置（可选）

e.JMSMesageId ： 每个消息的 唯一标识 自动设置

f.JmsTimestamp : 当provider 调send方法时 的时间 和 接收者接收到时间 的差值，自动设置

g.JMSCorrelationID :用来连接到另一个消息，用于 回复消息时 连接到原消息

h.JMS replayTo :回复消息的地址

可以通过replayTo(Destination对象);getreplayTo()接收消息体 返回 Destination 还可以继续发送消息

I.JMSType :消息类型的识别符号 开发者设置

J.JMSDelivered 重新投递 ：如果需要消息由于其他原因未被签收，可以设置消息从新投递

JMSDelivered = true ，JMSDelivered = false

* 1. 消息体 ：就是针对发送的数对创建不同的消息体 。JMS API定义了5种消息格式，也称消息题 可以使用不同形式发送 接收数据

注意：再接收者接收消息时 接收到的消息应转换成对应的消息格式

* + - * 1. TextMEssage 普通文本信息
        2. MapMessage map 适合发送对象
        3. ByteMessage 数组
        4. StreamMessage 流式消息
        5. ObjectMessage 对象

* 1. 消息体属性
     + - 1. 发送消息时messageSetXXX(); 为待发送的消息 设置附加属性
         2. 接收消息时messageGetXXX(); 载接收消息时，获取相应的属性

1. JMS机制
   1. JMS机制可靠性：防止发送重复发送 同样的消息

消息接收确认 :消息确认之后才认为是被成功消费了 。如果不确认中间件 下一次还会发 同样的消息 。 消息何时被确认 取决于 创建会话时的应答 模式 。

消息成功消费包含3个阶段 ：客户接收消息、客户处理消息、消息被确认

* + - 1. session.AUTO\_ACKNOWWLEDGE在事务性会话中 当事物被提交时 session.commint()，确认默认发生。
      2. Session.CLIENT\_ACKNOWWLEDGE 通过消费方 对象调 acknowledge 方法手动确认消息

注意： 由于确认是在会话层进行的 ，一旦确认 会将之前所有已经消费的消息确认 ,一旦使用 这种方式 ，就将是否支持是否设置为false 和 不用session.commint() ，因为session.commint()是配合 session.AUTO\_ACKNOWWLEDGE使用

* + - 1. session.DUPS\_ACKNOWWLEDGE 迟钝的方式进行消息确认 ,性能稍好 。但是特殊情况 由于是延迟确认消息 ，就会存在 重复 收到消息的情况 ，比如 provider 服务突然挂了 ，重新开机 由于消息 还未确认 还会继续发送 那些未 确认过的消息 。

将JMS消息头 JMSRedelivered 设置为 true ，来对重复发送的消息进行检测

（种方式 ，就将是否支持是否设置为false 和 不用session.commint() ）

* 1. JMS消息提交模式
     + 1. PERSISTANT持久保存消息 ，provider服务可能会故障 消息会丢失

(也是可靠性的体现)

* + - 1. Non\_PERSISTANT 不持久化消息 ，要求消费者 先在线 ，生产者再发送
  1. Session.createTemporaryXX(); 创建临时目的地 他们存在的时间只限于创 建他们的连接所保持的时间 ， 只有创建临时目的地的消费者 才能从临 时目的中取消息
  2. 持久订阅 session.createDurableSubscriber(); provider会存储发布到 topic上的消息 ，当消费者 再次连接到 topic 的时候 ，发送者就会把订阅过后 消费者没有收到的消息再次发送给 消费端 。持久订阅在某个时刻 只能有一个激活的订阅者， 持久订阅被创建后会一直保留 ，直到调用 ununsubsciber方法

使用条件

A） 发送者 ：

producer.setDeliveryMode(DeliveryMode.PERSISTENT);第一个参数

ii、connection.start() 在持久化设置完再启动

1. 接收者

I、再连接之后需要创建消费者id : connection.setClientId(“”) ，

Ii、然后创建session.createDurableSubscriber()进行持久化订阅

Iii、最后再连接再 connection.start()

Iiii、注意 ：必须先运行 消费者 将设置的 connection Id，注册到服务器 ，然后 才能实现 持久化订阅

* 1. 本地事物 在创建session时可以设置事物是否开启

注意1、 ：一旦事物开启 ，业务需求又需要 有请求回复机制 rePlay(); 才能实现的功能 是比较耗资源的，因为 再回复之前 事物不会被提交 只会阻塞挂起直到回复完毕才会 提交事物

2、生产者和消费者 不能在同一个事物中，只有生产者 提交了事物 消息才会 发送到 目的地 ，然后消费端的事物开启接收消息

1. JMS消息传送模型
   1. ****点对点（队列）消息传送模型 P2P****

不相互独立 一个点 依赖一个点

****特点：****

* + 1. 如过消费端sesison关闭之前没有 签收 acknowledge（没有commint） ，那么消息对列中的消息就不被销毁 下次还会发送
    2. 如果用户在receive中设置了消息的选择条件 ，那么不符合条件的会留在队列中，不会被接收
    3. 队列可以长久的保存消息 ，即使消费端处于离线状态 ，重新上线 也会接收到消息 。除非消息过期
  1. Publis/subscribe

Topic 类似报纸 ， 消息的发布者类似出版社 ， 消息的接收者类似 读者

这使得 发送者 和 接收者 相互独立 ， 因此一个 主题 可以有多个程序订阅

****特点：****

* + 1. 消息订阅分为 持久订阅 和 非持久订阅
       1. 只要是非 持久订阅 只有客户端是 在线状态才能接收到消息，即使上线也收不到 ， 即使重新订阅也接收不到消息 ，除非 订阅后 客户端重新发送消息
       2. 持久订阅 消费端离线 ， 在上线时也可以接收到消息 。持久订阅时 消费端向jms中间件 注册了一个省份id， jms会根据id 为这个消费者保存消息， 当客户端再次连接上 jms服务时 消息就会发送过来
       3. 如果用户在receive中设置了消息的选择条件 ,那么不符合的消息 消费端不会接收
       4. 当消息 比较重要 必须接收 就是用持久订阅， 当消息可以适度丢失 可以使用非持久订阅

1. 接受消息方式
   1. 同步接收
   2. 异步接收

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

1、是什么activityMQ

Apache下实现 消息推送的中间件 MOM (消息中间键).

什么是消息中间件 ： 当 A 方法调用本地项目 或者 远程项目中的方法 B ， 他们之间就形成了 强依赖的 关系。 这时可以添加消息中间件， A向 中间件发送消息 ， B 去取消息 ，当取到消息 就去处理 相应的功能 。 通过中间件 进行数据的传递来 解耦合 。

本质 ： 将信息 以消息的 形式 从一个应用程序 传送到 多个应用程序

2、activityMQ的作用

用于实现 JMS 的规范，完成相应功能 。用来实现 高可用、高性能、可伸缩 和 安全的 企业级 面向消息的 服务系统

1. activityMQ的特点

(1)、完全支持javaEE jms规 范

(2)、异步：即使B程序关闭，也可以在启动的时候接收到消息 ，也就是A无需等待B的处理 （类似 手机接收短信 ，作用：提高性能）

(3)、解耦 （作用：减少分布式的 多模块之间的依赖）

(4)、能够持久化消息

(5)、支持事物

(6)、支持多传输协议

(7)、和tomcat 易集成

1. 使用
2. bin 下启动 默认端口 61616
3. 进入 ActivityMQ 管理页面 http://localhost:8161/admin/

用户名/密码 ： admin

1. 创建消息队列的步骤

A/获取jms工厂

B/使用连接工厂创建连接

C/启动连接

D/从连接创建会话 session 三个参数。参数1：是否支持事物 a/性能搞 b/如果一次性 发送或接受多条消息 失败可以回滚

E/创建 Destination （p2p创建queue，Pub/Sub创建topic）

F/创建 producer

G/producer 发送消息到 destination

H/消息端 取消息

I/释放资源 关闭连接

1. 使用ActivieMQ 构建Broker(mq服务) 实例 ，java程序嵌入broker服务

两种方式

1. 、borker服务

BrokerService broker = new BrokerService();

broker.setUseJmx(true); //启动broker JMX监控功能

broker.setBrokerName("myBroker"); //设置名子

broker.addConnector("localhost:61616"); //连接地址

broker.start(); //启动

1. 、brock工厂启动broker （读取配置文件）

String url = "properties：borker.properties";

BrokerService broker = BrokerFactory.createBroker(url);

broker.addConnector("tcp://localhost:61616");

配置文件

useJmx = true

brokerName = myBroker

persistent = false

1. spring集成ActiveMQ
2. 、添加spring提供对jms的支持

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-pool</artifactId>

<version>5.7.0</version>

</dependency>

1. 、ActiveMQ连接池 pool

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-pool</artifactId>

<version>5.7.0</version>

</dependency>

配置：

<bean id="activeMQDataSource" class="org.apache.activemq.pool.PooledConnectionFactory">

<property name="maxConnections" value="100"/>

<property name="connectionFactory">

<bean class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="userName" value="admin"></property>

<property name="password" value="admin"></property>

<property name="brokerURL" value="tcp://localhost:61616"/>

</bean>

</property>

</bean>

1. 、配置jmsTemplate （为使用mq开发时提供方便 类似jdbcTemplate）
2. 依赖 连接池
3. 依赖 destination
4. Destination

Queue配置：<bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">

<constructor-arg index = "0" value="spring-queue"/>

</bean>

Topoic配置 <bean id="destinationTopic" class="org.apache.activemq.command.ActiveMQTopic">

<constructor-arg index = "0" value="spring-topic"/>

</bean>

配置：<bean class="org.springframework.jms.core.JmsTemplate">

<property name="connectionFactory" ref="activeMQDataSource"/>

<property name="defaultDestination" ref="destination"/>

<property name="messageConverter"> <!-- 消息转化 -->

<bean class="org.springframework.jms.support.converter.SimpleMessageConverter"/>

</property>

</bean>

8、spring内嵌broker

<bean id="myBroker" class="org.apache.activemq.broker.BrokerService" init-method="start" destroy-method="stop">

<property name="useJmx" value="true"/>

<property name="brokerName" value="broker"/>

<property name="persistent" value="false"></property>

<property name="transportConnectorURIs">

<list>

<value>tcp://localhost:61616</value>

</list>

</property>

</bean>

9、可以在spring中配合消费者监听 MessageListener ， 自动接收消息 ，无需再单独的写一个消费者进行接收消息，当发送者发送消息时 ，监听将自动接收消息

（1）<!-- 自定义消息接收监听 类，实现Messagelistener接口实现onMessage方法--> onMessage()：用于自动接收的方法

<bean id="messageListenerMQ" class="com.it.listener.MQMessageListener"/>

（2）<!-- 配置接收消费者监听类用户自动接收消息 -->

<bean id="MessageListenerContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer">

<property name="connectionFactory" ref="activeMQDataSource"/>

<property name="destination" ref="destinationTopic"/>

<property name="messageListener" ref="messageListenerMQ"/>

</bean>

1. 与spring整合开发的注意事项
   1. 、发送接收消息的核心 jmsTemplate对象， 隔离了打开连接 ，关闭sesion和producer 等频繁的操作 。让开发者只关注核心逻辑
   2. 、不建议使用jmsTemplate的receive方法来接收消息， 因为是同步的 阻塞性能降低 ，

建议使用自定义监听实现messageListener接口异步自动接收消息

(3)、建议配置DefaultMessageListenerContainer监听容器 ，它允许异步缓存session还可以 根据消息的数量合理的增加监听器和缩减监听器数量

1. ActiveMQ传输协议
2. 、connetor：activeMQ提供的用来实现通讯功能的一个组件。 两种情况
   * 1. Client-to-broker 客户端连接mq服务
     2. Broker-tobroker mq服务连接mq服务
3. 、Active允许多种协议 可以连接到 mq服务，可以在 conf下的 activemq.xml中配置

<transportConnectors>

<transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

transportConnectors标签用于定义 ： 客户端如何连接到 broker上去

支持的协议包括 ： tcp、NIO、ssl、http（s）、udp 、 vm

1. 、tcp：是连接activemq的默认通讯协议 端口默认是61616 ，依赖openwire

用于流化数据 便于数据交互 。往后是地址加端口号 再往后是连接参数 最大连接数和数据传输大小

2）、Vm:如果客户端和broker在一个虚拟机下的话可以通过vm协议进行连接

如：内嵌到java虚拟机中 ，就是通过vm协议通讯

3）、扩展 ：如果有大量的cilent连接到 broker时 ，可以是使用NIO协议场景

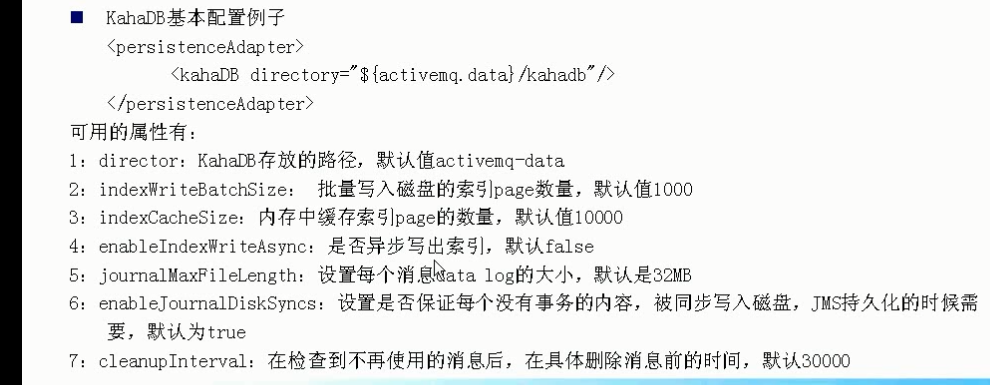
一般client 连接broker时 ，是被操作系统的线程数控制的 ，NIO能使用更少的线程数与broker建立连接

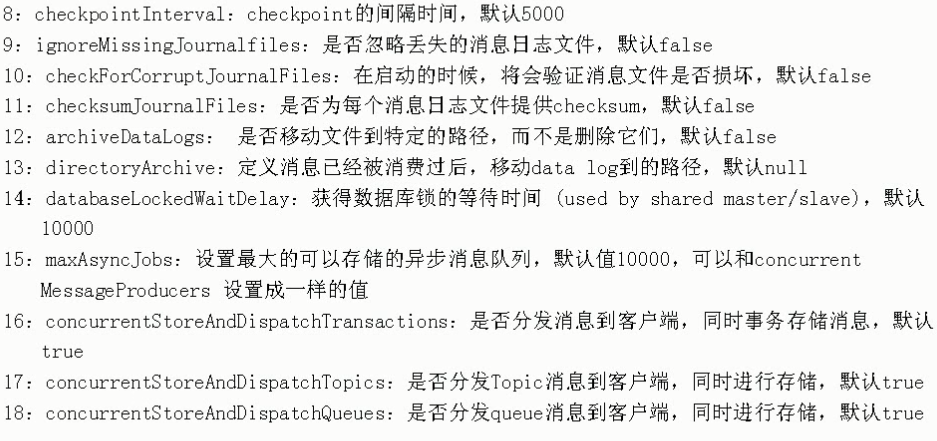
<transportConnector name=”nio” uri=”nio://localhost:端口”/>

1. ActiveMQ消息存储持久化

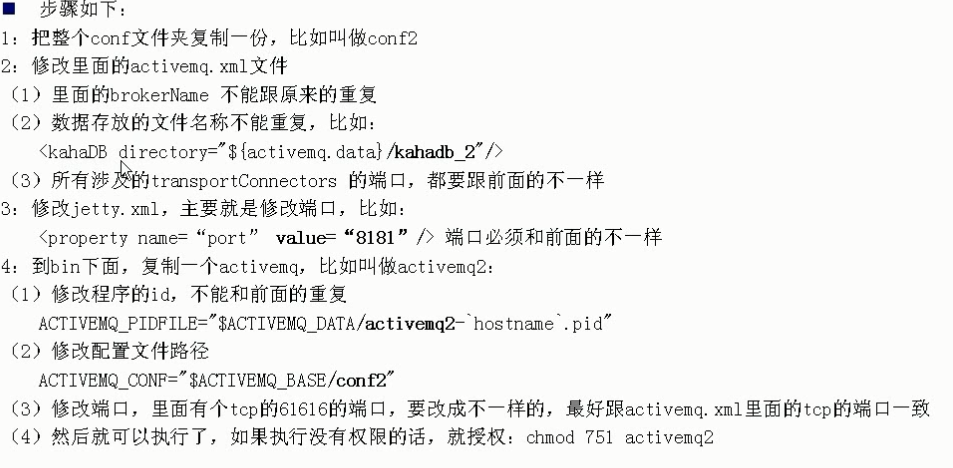
消息存储的几种方式 ：

1. 、AMQ消息存储， 基于文件的存储方式 是5.0版本以前默认的消息存储
2. 、KahaDB消息存储 是现在默认的存储方式（特点：提升了数据的容量和恢复能力）
   1. 可以用于任何场景
   2. 性能高
   3. 配置详情





1. 、JDBC消息存储 ，将消息持久化到表中
2. 、Merroy 基于内存的消息存储
3. 一台服务器集成多台 broker



Java底层如何实现热加载

本质 ：将最新的java文件编译然后运行

Jvm ：如何加载class