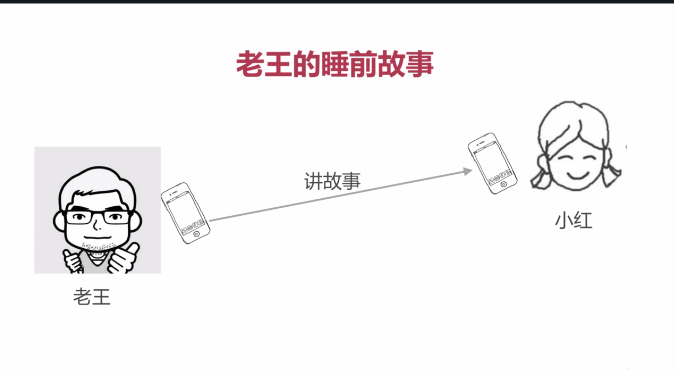
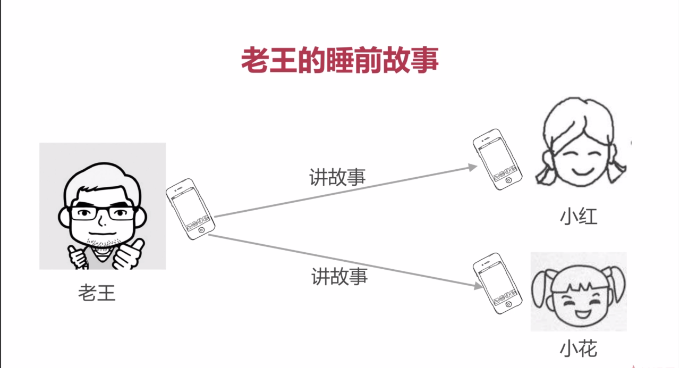
大女儿上大学，老王每天晚上打电话（没信号）



二女儿上大学



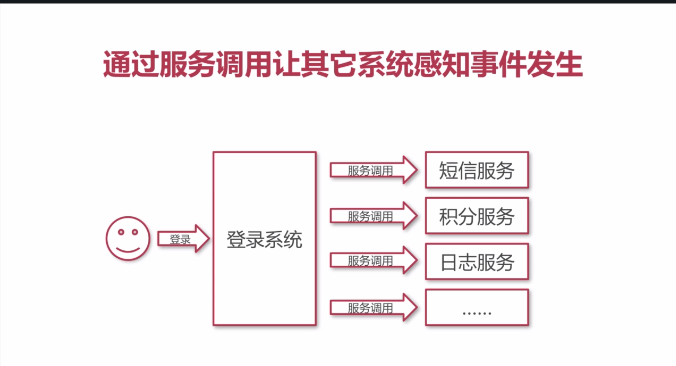
后来使用公众号，老王就不用劳累



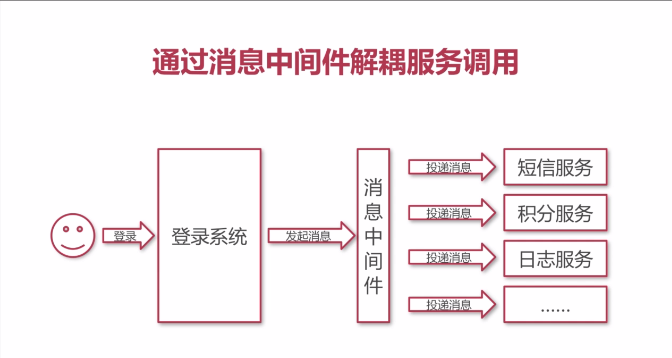
通过服务调用让其他系统感知事件发生

系统之间高耦合

程序执行效率低



通过消息中间件解耦服务调用

生活中的案例微信公众号、黑板、电视

消息中间件带来的好处

* 解耦
* 异步
* 横向扩展
* 安全可靠
* 顺序保证
* 更多

横向扩展解释

当登录系统，需要很多用户登录。这些消息全部需要告知积分系统，去增加积分，而增加积分这个处理过程可能比较麻烦、比较耗时。这个时候，可以启动多台积分系统，来同时消费这个消息中间件里面的登录消息，达到横向扩展的作用。

消息中间件概述

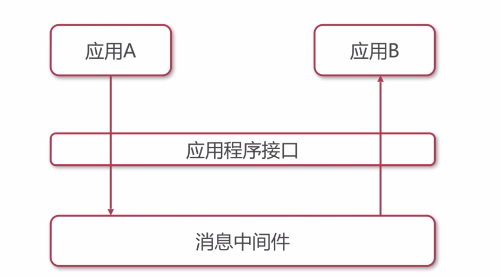
什么是中间件？

非底层操作系统软件，非业务应用软件，不是直接给最终用户使用的，不能直接给客户带来价值的软件统称为中间件。

什么是消息中间件？

关注与数据的发送和接送，利用高小可靠的异步消息传递机制集成分布式系统。

消息中间件示意图



什么是JMS？

Java消息服务（java Message Service）即JMS,是一个java平台中关于面向消息中间件的API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。

什么是AMQP？

AMQP（advanced message queuing protocol）是一个提供同一消息服务的应用层标准协议，基于致协议的客户端与消息中间件可传递消息，并不受客户端/中间件不同产品，不同开发语言等条件的限制。

Jms和AMQP对比



常见消息中间件对比

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J2EE1.4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

* 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

* 完全支持JMS1.1和J2EE1.4规范（持久化、XA消息、事务）
* 虚拟主题、组合目的、镜像队列

RabbitMQ

RabbitMQ是一个开源的AMQP实现，服务器端用Erlang语言编写。用于在分布式系统真储存转发消息，在易用性、扩展性、高可用性等方面表现不俗。

RabbitMQ特性

* 支持多种客户端，如：Python、ruby、.net、java、JMS、c、php、ActionSctipt等
* AMQP的完整实现（vhost、Exchange、Binding、Routing Key等）
* 事务支持/发不确认
* 消息持久化

Kafka

Kafka是一种高吞吐量的分布式发布订阅消息系统，是一个分布式的、分区的、可靠的分布式日志储存服务。它通过一种独一无二的设计提供了一个消息系统的功能。

Kafka特性

* 通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB的消息储存也能够保持长时间的稳定性能
* 高屯吐量：即使是非常普通的硬件Kafka也可以支持每秒数百万的消息。
* Partition、Consumer Group

综合评价



JMS规范

JMS相关概念

提供者：实现JMS规范的消息中间件服务器

客户端：发送或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

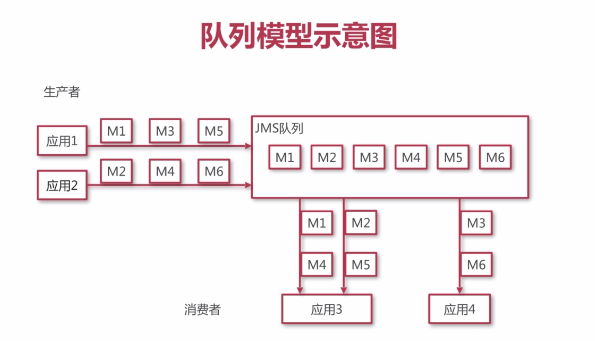
消息：应用程序之间传递的数据内容

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

JMS消息模式

队列模型

* 客户端包括生产者和消费者
* 队列中的消息只能被一个消费者消费
* 消费者可以随时消费队列中的消息



主题模型

* 客户端包括发布者和订阅者
* 主题中的消息被所有订阅者消费
* 消费者不能消费订阅之前就发送到主题职工的消息



提前订阅：要不收不到消息

JMS编码接口

ConnectionFactory用于创建连接到消息中间件的连接工厂

Connection代表了应用程序和消息服务器之间的通信链路

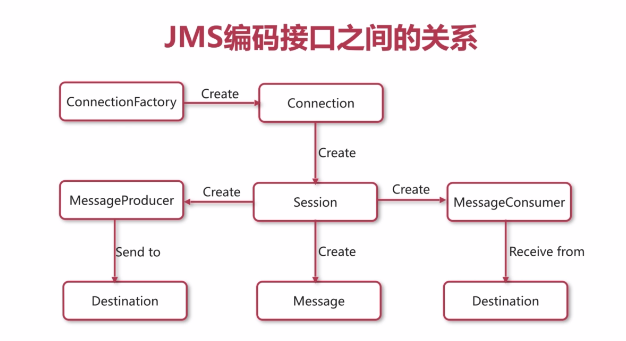
Destination指消息发布和接收的地点，包括队列和主题

Session表示一个单线程的上下文，用于发送和接收消息

MessageConsumer由会话创建，用于接收发送到目标的消息

MessageProducer由会话创建，用于发送消息到目标

Message是在消费者和生产者之间传送的对象，消息头，一组消息属性、一个消息体



用使ActiveMQ

Win安装ActiveMQ（类似与tomcat）

下载安装包

直接启动

使用服务启动

两个bat文件（一个是命令框，一个是启动服务）

安装验证

访问地址：http://127.0.0.1:8161/

默认用户：admin

默认密码：admin

Linux安装ActiveMQ

下载并解压安装包

Wget 地址

tar –zxvf

启动

启动验证

进入到bin目录，使用命令./activemq start启动服务

使用命令ps -ef |grep activemq查看进程是否存在

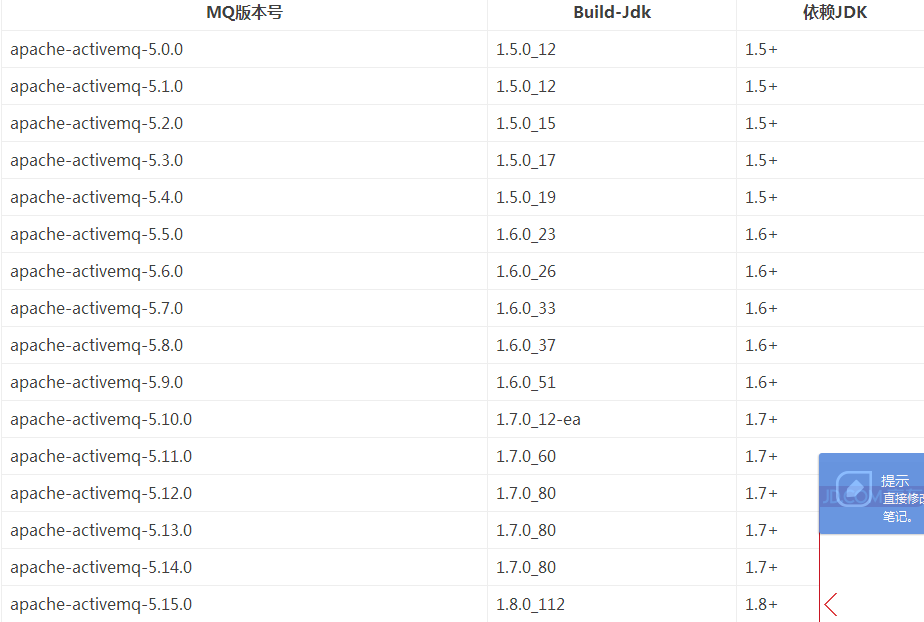
使用命令./activemq stop关闭服务

安装验证

访问地址：http://Linux主机IP:8161/

默认用户：admin

默认密码：admin



使用JMS接口规范连接ActiveMQ

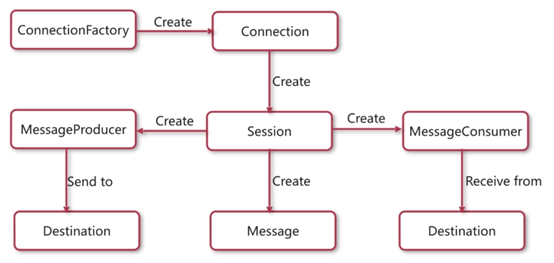
创建生产者

创建消费者

创建发布者

创建接受者

回顾JMS编码接口之间的关系



队列模式的消息演示

代码演示

1.编写AppProducer类

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.MessageProducer;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

/\*\*

\* App 生产者-队列模式

\* @author ZhangCheng on 2017-07-22

\*

\*/

public class AppProducer {

/\*\* 指定ActiveMQ服务的地址 \*/

private static final String URL = "tcp://127.0.0.1:61616";

/\*\* 指定队列的名称 \*/

private static final String QUEUE\_NAME = "queue-test";

public static void main(String[] args) throws JMSException {

// 1.创建ConnectionFactory

ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(URL);

// 2.创建Connection

Connection connection = connectionFactory.createConnection();

// 3.启动连接

connection.start();

// 4.创建会话（第一个参数：是否在事务中处理）

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 5. 创建一个目标

Destination destination = session.createQueue(QUEUE\_NAME);

// 6.创建一个生产者

MessageProducer producer = session.createProducer(destination);

for (int i = 0; i < 100; i++) {

// 7.创建消息

TextMessage textMessage = session.createTextMessage("test" + i);

// 8.发布消息

producer.send(textMessage);

System.out.println("消息发送：" + textMessage.getText());

}

// 9.关闭连接

connection.close();

}

}

2.编写AppConsumer类

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageConsumer;

import javax.jms.MessageListener;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

/\*\*

\* App 消费者-队列模式

\* @author ZhangCheng on 2017-07-22

\*

\*/

public class AppConsumer {

/\*\* 指定ActiveMQ服务的地址 \*/

private static final String URL = "tcp://127.0.0.1:61616";

/\*\* 指定队列的名称 \*/

private static final String QUEUE\_NAME = "queue-test";

public static void main(String[] args) throws JMSException {

// 1.创建ConnectionFactory

ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(URL);

// 2.创建Connection

Connection connection = connectionFactory.createConnection();

// 3.启动连接

connection.start();

// 4.创建会话（第一个参数：是否在事务中处理）

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 5.创建一个目标

Destination destination = session.createQueue(QUEUE\_NAME);

// 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;

try {

System.out.println("接收消息：" + textMessage.getText());

} catch (JMSException e) {

System.out.println("接收消息异常：");

e.printStackTrace();

}

}

});

// 8.关闭连接

//connection.close();

}

}

## 主题模式的消息演示

代码演示

1.编写AppProducer类

package com.myimooc.jms.topic;

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.MessageProducer;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

/\*\*

\* App 生产者-主题模式

\* @author ZhangCheng on 2017-07-22

\*

\*/

public class AppProducer {

/\*\* 指定ActiveMQ服务的地址 \*/

private static final String URL = "tcp://127.0.0.1:61616";

/\*\* 指定主题的名称 \*/

private static final String TOPIC\_NAME = "topic-test";

public static void main(String[] args) throws JMSException {

// 1.创建ConnectionFactory

ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(URL);

// 2.创建Connection

Connection connection = connectionFactory.createConnection();

// 3.启动连接

connection.start();

// 4.创建会话（第一个参数：是否在事务中处理）

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 5. 创建一个目标

Destination destination = session.createTopic(TOPIC\_NAME);

// 6.创建一个生产者

MessageProducer producer = session.createProducer(destination);

for (int i = 0; i < 100; i++) {

// 7.创建消息

TextMessage textMessage = session.createTextMessage("test" + i);

// 8.发布消息

producer.send(textMessage);

System.out.println("消息发送：" + textMessage.getText());

}

// 9.关闭连接

connection.close();

}

}

2.编写AppConsumer类

package com.myimooc.jms.topic;

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageConsumer;

import javax.jms.MessageListener;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

/\*\*

\* App 消费者-主题模式

\* @author ZhangCheng on 2017-07-22

\*

\*/

public class AppConsumer {

/\*\* 指定ActiveMQ服务的地址 \*/

private static final String URL = "tcp://127.0.0.1:61616";

/\*\* 指定主题的名称 \*/

private static final String TOPIC\_NAME = "topic-test";

public static void main(String[] args) throws JMSException {

// 1.创建ConnectionFactory

ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(URL);

// 2.创建Connection

Connection connection = connectionFactory.createConnection();

// 3.启动连接

connection.start();

// 4.创建会话（第一个参数：是否在事务中处理）

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 5.创建一个目标

Destination destination = session.createTopic(TOPIC\_NAME);

// 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;

try {

System.out.println("接收消息：" + textMessage.getText());

} catch (JMSException e) {

System.out.println("接收消息异常：");

e.printStackTrace();

}

}

});

// 8.关闭连接

//connection.close();

}

}

使用Spring集成JMS连接ActiveMQ

ConnectionFactory 用于管理连接的连接工厂

* 一个Spring为我们提供的连接池
* JmsTemplate每次发消息都会重新创建连接，会话和productor
* Spring 中提供了SingleConnectionFactory（单例）和CachingConnectionFactory（继承single缓存）

JmsTemplate 用于发送和接收消息的模板类

* 是spring提供的，只需向spring容器内注册这个类就可以使用JmsTemplate方便的操作jms
* JmsTemplate类是线程安全的，可以在整个应用范围使用。

MessageListerner 消息监听器

* 实现一个OnMessage方法，该方法只接收一个Message参数。

## Spring jms演示

代码演示

1.创建名为jmsspring的maven项目POM文件如下

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.myimooc</groupId>

<artifactId>jmsspring</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>jmsspring</name>

<url>http://maven.apache.org</url>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jms</artifactId>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.activemq/activemq-all -->

<dependency>

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

<artifactId>activemq-all</artifactId>

<version>5.9.0</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

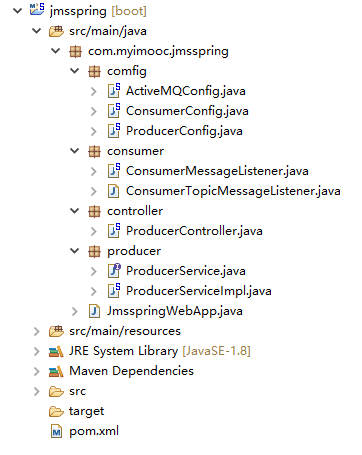
</plugin>

</plugins>

</build>

</project>

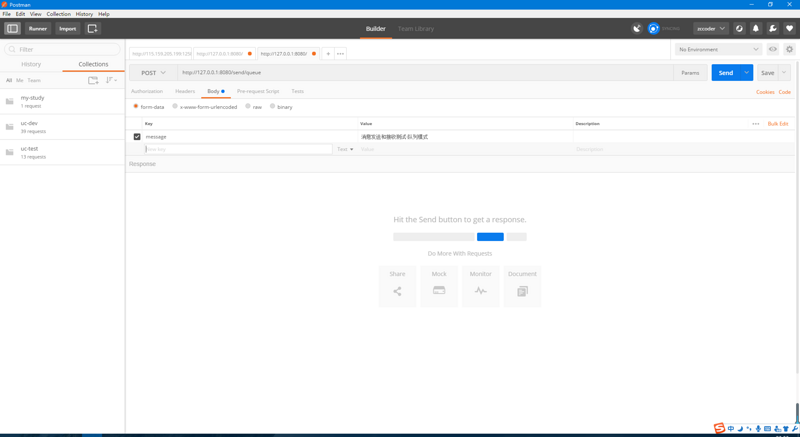
2.完成后的目录结构如下



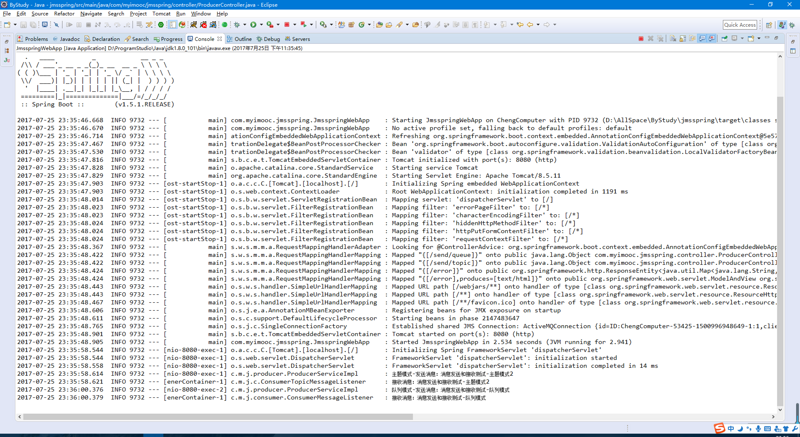
源码请到我的github地址查看

3.测试

使用Postman向ProducerController发起请求，将消息发送出去



对应的ConsumerTopicMessageListener 和 ConsumerMessageListener接收到消息



ActiveMQ集群配置

为什么要对消息中间件集群？

* 实现高可用，已排除单点故障引起的服务中断
* 实现负载均衡，以提升效率为更多客户提供服务

集群方式：

客户端集群：让多个消费者消费同一个队列

Broker clusters：多个Broker之间同步消息

MAster Slave：实现高可用

ActiveMQ失效转移（failover）-客户端配置

允许当其中一台消息服务器宕机时，客户端在传输层上重新连接到其它消息服务器

语法：failover:(uri1,…,uriN)?transportOptions

transportOptions参数说明

randomize默认为true，表示在URI列表中选择URI连接时是否采用随机策略

initialReconnectDelay默认为10，单位毫秒，表示第一次尝试重连之间等待的时间

maxReconnectDelay默认为30000，单位毫秒，最长重连的时间间隔

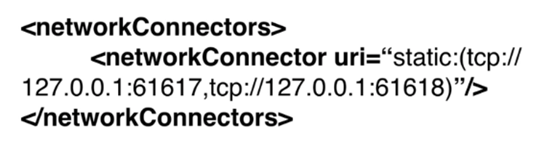
Broker cluster集群配置-原理

 NetworkConnector（网络连接器）

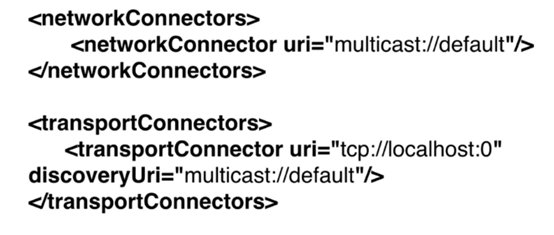
网络连接器主要用于配置ActiveMQ服务器与服务器之间的网络通讯方式，用于服务器透传消息

网络连接器分为静态连接器和动态连接器

静态连接器



动态连接器



## 5-2 ActiveMQ集群理论

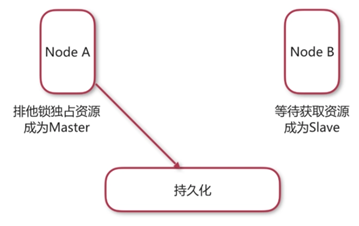
ActiveMQ Master Slace集群方案

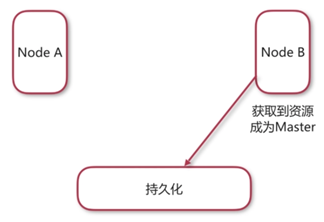
Share nothing storage master/slave（已过时，5.8+后移除）

Shared storage master/slave 共享存储

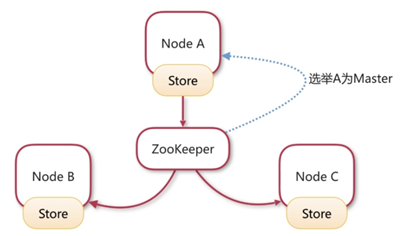
Replicated LevelDB Store基于负责的LevelDB Store

共享存储集群的原理





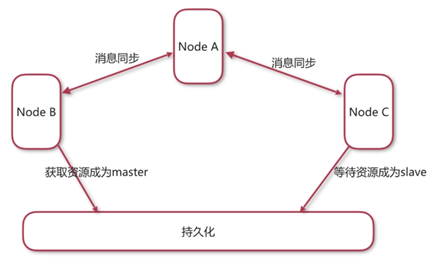
基于复制的LevelDB Store的原理



两种集群方式对比

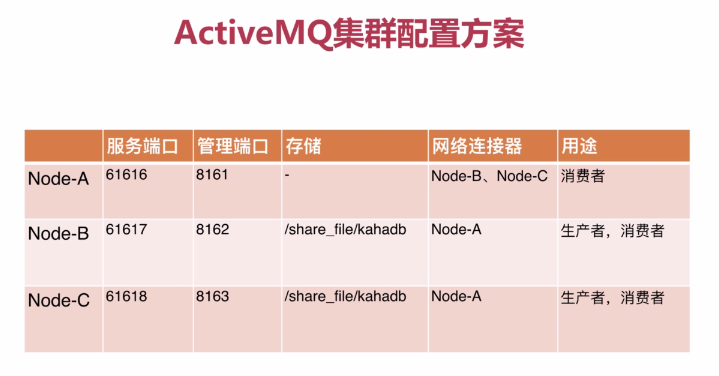


三台服务器的完美集群方案



## ActiveMQ集群实践

ActiveMQ集群配置方案

配置过程

1.节点准备

mkdir activemq创建目录

cp -rf apache-activemq-5.15.0 activemq/activemq-a

cp -rf apache-activemq-5.15.0 activemq/activemq-b

cp -rf apache-activemq-5.15.0 activemq/activemq-c

cd activemq

mkdir kahadb

2.配置a节点

cd activemq-a/

cd conf/

vim activemq.xml

<networkConnectors>

<networkConnector name="local\_network" uri="static:(tcp://127.0.0.1:61617,tcp://127.0.0.1:61618)" />

</networkConnectors>

vim jetty.xml：配置管理端口号，a节点使用默认端口，无须配置

3.配置b节点

vim activemq.xml

配置网络连接器

<networkConnectors>

<networkConnector name="network\_a" uri="static:(tcp://127.0.0.1:61616)" />

</networkConnectors>

配置持久化存储路径

<persistenceAdapter>

<kahaDB directory="/studio/activemq/kahadb"/>

</persistenceAdapter>

配置服务端口

<transportConnector name="openwire" uri="tcp://0.0.0.0:61617?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

vim jetty.xml

配置管理端口号

<bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">

<!-- the default port number for the web console -->

<property name="host" value="0.0.0.0"/>

<property name="port" value="8162"/>

</bean>

4.配置c节点

vim activemq.xml

配置网络连接器

<networkConnectors>

<networkConnector name="network\_a" uri="static:(tcp://127.0.0.1:61616)" />

</networkConnectors>

配置持久化存储路径

<persistenceAdapter>

<kahaDB directory="/studio/activemq/kahadb"/>

</persistenceAdapter>

配置服务端口

<transportConnector name="openwire" uri="tcp://0.0.0.0:61618?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

vim jetty.xml

配置管理端口号

<bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">

<!-- the default port number for the web console -->

<property name="host" value="0.0.0.0"/>

<property name="port" value="8163"/>

</bean>

5.启动服务  
回到activemq目录，分别启动a，b，c三个节点

./activemq-a/bin/activemq start

./activemq-b/bin/activemq start

./activemq-c /bin/activemq start

检查是否都启动成功

ps -ef |grep activemq

检查是否对外提供服务，即端口是否被监听（占用）

netstat -anp |grep 61616

netstat -anp |grep 61617

netstat -anp |grep 61618

检查发现61618即c节点没有提供服务，但是c节点的进程是启动成功了的。因为b节点和c点击是master/slave配置，现在b节点获取到了共享文件夹的所有权，所以c节点正在等待获得资源，并且提供服务。即c节点在未获得资源之前，是不提供服务的。

测试，把b节点杀掉，看c节点能不能提供61618的服务

./activemq-b/bin/activemq stop

netstat -anp |grep 61618

./activemq-b/bin/activemq start

netstat -anp |grep 61617

检查发现，重新启动b节点后，b节点61617端口并没有提供服务，是因为现在b节点成为了slave节点，而c节点成为了master节点。所以，现在b节点启动了，但是它并不对外提供服务。只有当c节点出现问题后，b节点才对外提供服务。

6.通过代码测试集群配置是否生效

生产者

package com.myimooc.jms.queue;

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.MessageProducer;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

/\*\*

\* App 生产者-队列模式-集群配置测试

\* @author ZhangCheng on 2017-07-25

\*

\*/

public class AppProducerTest {

/\*\* failover 为状态转移的存在部分

\* 因a节点只作为消费者使用，所以这里不配置61616节点了。

\* \*/

private static final String URL = "failover:(tcp://127.0.0.1:61617,tcp://127.0.0.1:61618)?randomize=true";

/\*\* 指定队列的名称 \*/

private static final String QUEUE\_NAME = "queue-test";

public static void main(String[] args) throws JMSException {

// 1.创建ConnectionFactory

ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(URL);

// 2.创建Connection

Connection connection = connectionFactory.createConnection();

// 3.启动连接

connection.start();

// 4.创建会话（第一个参数：是否在事务中处理）

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 5. 创建一个目标

Destination destination = session.createQueue(QUEUE\_NAME);

// 6.创建一个生产者

MessageProducer producer = session.createProducer(destination);

for (int i = 0; i < 100; i++) {

// 7.创建消息

TextMessage textMessage = session.createTextMessage("test" + i);

// 8.发布消息

producer.send(textMessage);

System.out.println("消息发送：" + textMessage.getText());

}

// 9.关闭连接

connection.close();

}

}

消费者

package com.myimooc.jms.queue;

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageConsumer;

import javax.jms.MessageListener;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

/\*\*

\* App 消费者-队列模式-集群配置测试

\* @author ZhangCheng on 2017-07-22

\*

\*/

public class AppConsumerTest {

/\*\* failover 为状态转移的存在部分

\* \*/

private static final String URL = "failover:(tcp://127.0.0.1:61616,tcp://127.0.0.1:61617,tcp://127.0.0.1:61618)?randomize=true";

/\*\* 指定队列的名称 \*/

private static final String QUEUE\_NAME = "queue-test";

public static void main(String[] args) throws JMSException {

// 1.创建ConnectionFactory

ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(URL);

// 2.创建Connection

Connection connection = connectionFactory.createConnection();

// 3.启动连接

connection.start();

// 4.创建会话（第一个参数：是否在事务中处理）

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 5.创建一个目标

Destination destination = session.createQueue(QUEUE\_NAME);

// 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;

try {

System.out.println("接收消息：" + textMessage.getText());

} catch (JMSException e) {

System.out.println("接收消息异常：");

e.printStackTrace();

}

}

});

// 8.关闭连接

//connection.close();

}

}

运行生产者，然后到管理界面查看消息发送到了那里

http://127.0.0.1:8161

http://127.0.0.1:8162

http://127.0.0.1:8163

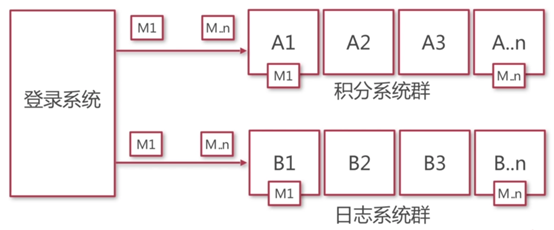
查看发现，8162无法访问，是因为b节点是slave节点，不提供服务，消息都发送到了c节点

把8163即c节点宕掉后，运行消费者，查看消息是否能够使用

./activemq-c/bin/activemq stop

## 企业级系统中的最佳实践

实际业务场景分析



实际业务场景特点：

子业务系统都有集群的可能性

同一个消息会广播给关注该类消息的所有子业务系统

同一类消息在集群中被负载消费

业务的发生和消息的发布最终一致性

需要解决的问题：

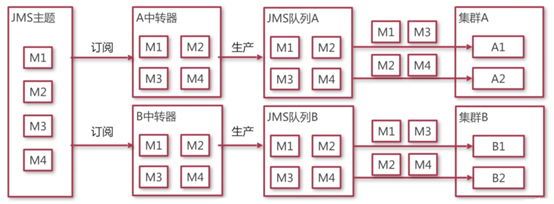
不同业务系统分别处理同一个消息，同一业务系统负载处理同类消息

解决消息发送时的一致性问题

解决消息处理的幂等性问题

基于消息机制建立事件总线

集群系统处理消息方案-使用JMS级联的解决方案



1.集群系统处理消息方案-使用ActiveMQ的虚拟主题解决方案

发布者：将消息发布到一个主题中，主题名以VirtualTopic开头，如VirtualTopic.TEST

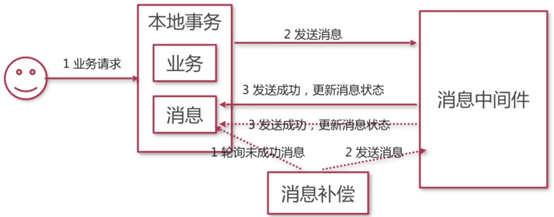
消费者：从队列中获取消息，在队列名中表名自己身份，如Consumer.A.VirtualTopic.TEST

2.解决消息发送时的一致性问题-使用JMS中XA系列接口保证强一致性

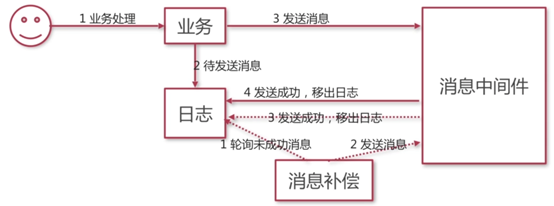
引入分布式事务

要求业务操作必须支持XA协议

解决消息发送时的一致性问题-使用消息表的本地事务解决方案



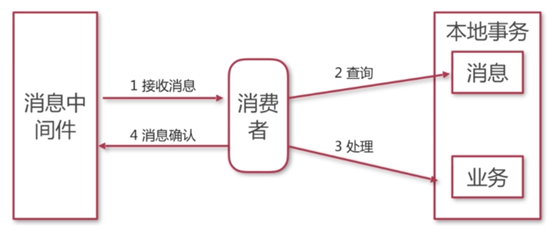
3．解决消息发送时的一致性问题-使用内存日志的解决方案



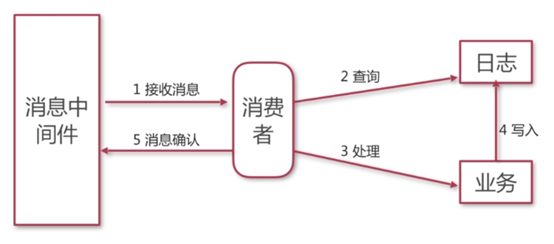
4.解决消息处理的幂等性问题

所谓幂等性问题，是指多次执行所产生的影响（结果）与一次执行所产生的影响（结果）一样。比如：支付成功后，支付宝会发起多次通知给业务系统，要求业务系统能够处理这些重复的消息，但是又不重复处理订单。如果在消息处理系统中保证幂等性，会增加系统复杂度，我们可以统一处理幂等性后，再将消息发送给消息处理系统。

解决消息处理的幂等性问题-使用消息表的本地事务解决方案



解决消息处理的幂等性问题-使用内存日志的解决方案

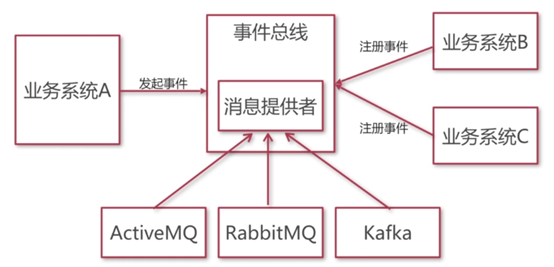


**基于消息机制的事件总线-什么是事件驱动架构**

事件驱动架构（**Event** Driven Architecture，EDA）定义了一个设计和实现一个应用系统的方法学，在这个系统里事件可传输于松散耦合的组件和服务之间。

特点：有事我叫你，没事别烦我

事件驱动架构模型



该教师正在开发该事件总线的框架，github地址<https://github.com/jovezhao/nest>。

## 6-1 使用其它消息中间件

企业开发需要解决的问题：

1.不同业务系统分别处理同一个消息，同一业务系统负载处理同类消息

2.解决消息发送时的一致性问题

3.解决消息处理的幂等性问题

4.基于消息机制建立事件总线（可以处理2和3）

分析需要做的事：

解决各业务系统集群处理同一条消息

实现自己的消息提供者

常用消息中间件：

ActiveMQ

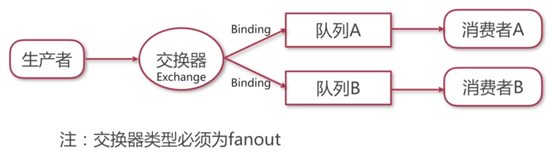
RabbitMQ

Kafka

集成RabbitMQ

RabbitMQ：使用交换器绑定到队列

示意图



RabbitMQ消息提供者源码解析

创建ConnectionFactory

创建Connection

创建Channel

定义Exchange

定义Queue并且绑定队列

集成Kafka

Kafka使用group.id分组消费者

配置消息者参数group.id相同时对息进行负载处理

配置服务器partitions参数，控制同一个group.id下的consumer数量小于partitions

Kafka只保证同一个partition下的消息是有序的

Kafka消息提供者源码解析

创建生产者

创建消费者

源码

<https://github.com/jovezhao/nest>

