# **Java消息队列--ActiveMQ 实战**

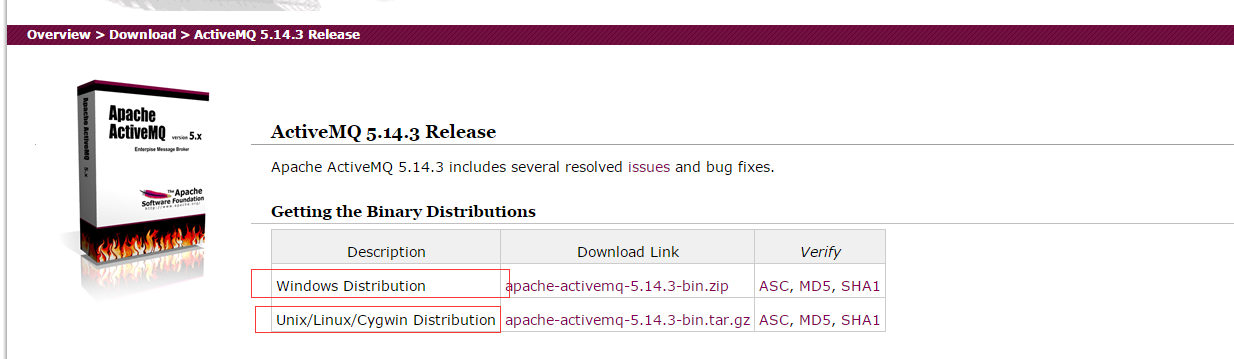
|  |  |  |
| --- | --- | --- |
| [日期：2016-12-27] | 来源：Linux社区  作者：jaycekon | [字体：[大](http://www.linuxidc.com/Linux/2016-12/javascript:ContentSize(16)) [中](http://www.linuxidc.com/Linux/2016-12/javascript:ContentSize(0)) [小](http://www.linuxidc.com/Linux/2016-12/javascript:ContentSize(12))] |

### Java消息队列--ActiveMQ 实战

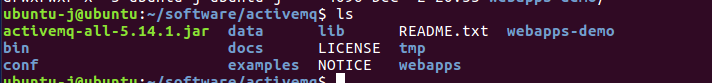
### 1、下载安装ActiveMQ

　　ActiveMQ官网下载地址：[http://activemq.apache.org/download.html](http://activemq.apache.org/download.html" \o "ActiveMQ下载地址" \t "http://www.linuxidc.com/Linux/2016-12/_blank)

　　ActiveMQ 提供了Windows 和Linux、Unix 等几个版本，楼主这里选择了Linux 版本下进行开发。



　　下载完安装包，解压之后的目录：



 　　从它的目录来说，还是很简单的：

* 1. bin存放的是脚本文件
  2. conf存放的是基本配置文件
  3. data存放的是日志文件
  4. docs存放的是说明文档
  5. examples存放的是简单的实例
  6. lib存放的是activemq所需jar包
  7. webapps用于存放项目的目录

### 2、启动ActiveMQ

 　　进入到ActiveMQ 安装目录的Bin 目录，linux 下输入 ./activemq start 启动activeMQ 服务。

　　 输入命令之后，会提示我们创建了一个进程IP 号，这时候说明服务已经成功启动了。

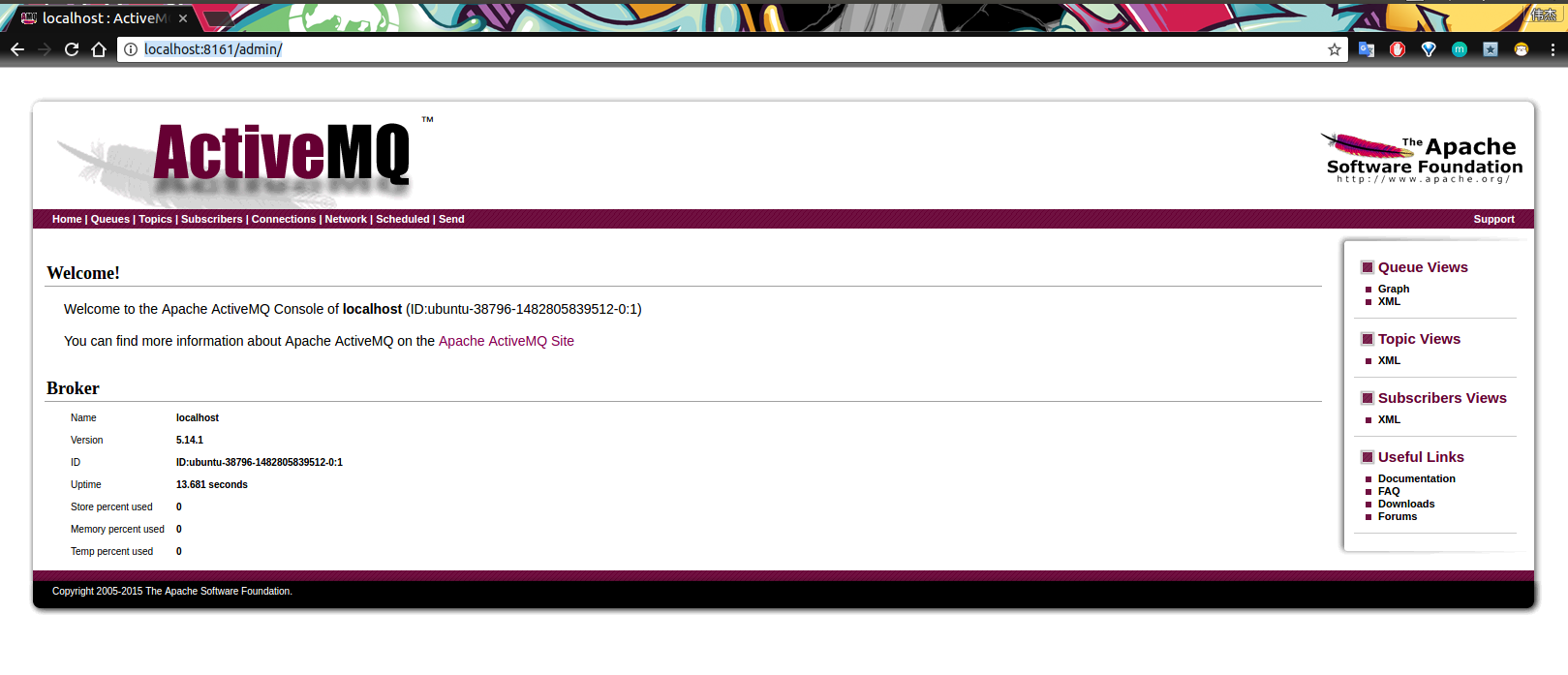
### IMG_258

　　ActiveMQ默认启动时，启动了内置的jetty服务器，提供一个用于监控ActiveMQ的admin应用。   
　　admin：http://127.0.0.1:8161/admin/

　　我们在浏览器打开链接之后输入账号密码（这里和tomcat 服务器类似）

　　默认账号：admin

　　密码：admin

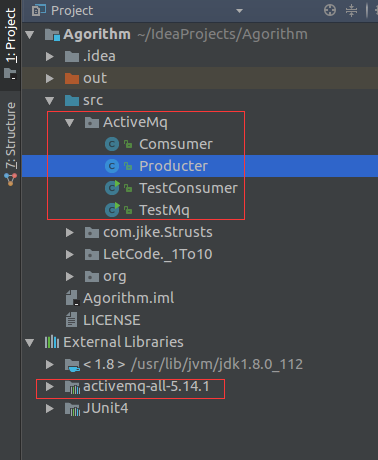


　　　到这里为止，ActiveMQ 服务端就启动完毕了。

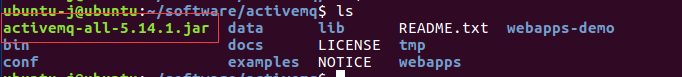
　　　ActiveMQ 在linux 下的终止命令是 ./activemq stop

### 3、创建一个ActiveMQ工程

 　　项目目录结构：



　　上述在官网下载ActiveMq 的时候，我们可以在目录下看到一个jar包：



　　这个jar 包就是我们需要在项目中进行开发中使用到的相关依赖。

#### 3.1 创建生产者

public class Producter {

//ActiveMq 的默认用户名

private static final String USERNAME = ActiveMQConnection.DEFAULT\_USER;

//ActiveMq 的默认登录密码

private static final String PASSWORD = ActiveMQConnection.DEFAULT\_PASSWORD;

//ActiveMQ 的链接地址

private static final String BROKEN\_URL = ActiveMQConnection.DEFAULT\_BROKER\_URL;

AtomicInteger count = new AtomicInteger(0);

//链接工厂 ConnectionFactory connectionFactory;

//链接对象 Connection connection;

//事务管理 Session session;

ThreadLocal<MessageProducer> threadLocal = new ThreadLocal<>();

public void init(){

try {

//创建一个链接工厂

connectionFactory = new ActiveMQConnectionFactory(USERNAME,PASSWORD,BROKEN\_URL);

//从工厂中创建一个链接

connection = connectionFactory.createConnection();

//开启链接 connection.start();

//创建一个事务（这里通过参数可以设置事务的级别）

session = connection.createSession(true,Session.SESSION\_TRANSACTED);

} catch (JMSException e) {

e.printStackTrace();

}

}

public void sendMessage(String disname){

try {

//创建一个消息队列

Queue queue = session.createQueue(disname);

//消息生产者

MessageProducer messageProducer = null;

if(threadLocal.get()!=null){

messageProducer = threadLocal.get();

}else{

messageProducer = session.createProducer(queue);

threadLocal.set(messageProducer);

}

while(true){

Thread.sleep(1000);

int num = count.getAndIncrement();

//创建一条消息

TextMessage msg = session.createTextMessage(Thread.currentThread().getName()+

"productor:我是大帅哥，我现在正在生产东西！,count:"+num);

System.out.println(Thread.currentThread().getName()+

"productor:我是大帅哥，我现在正在生产东西！,count:"+num);

//发送消息 messageProducer.send(msg);

//提交事务 session.commit();

}

} catch (JMSException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

#### 3.2 创建消费者

public class Comsumer {

private static final String USERNAME = ActiveMQConnection.DEFAULT\_USER;

private static final String PASSWORD = ActiveMQConnection.DEFAULT\_PASSWORD;

private static final String BROKEN\_URL = ActiveMQConnection.DEFAULT\_BROKER\_URL;

ConnectionFactory connectionFactory;

Connection connection;

Session session;

ThreadLocal<MessageConsumer> threadLocal = new ThreadLocal<>();

AtomicInteger count = new AtomicInteger();

public void init(){

try {

connectionFactory = new ActiveMQConnectionFactory(USERNAME,PASSWORD,BROKEN\_URL);

connection = connectionFactory.createConnection();

connection.start();

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

} catch (JMSException e) {

e.printStackTrace();

}

}

public void getMessage(String disname){

try {

Queue queue = session.createQueue(disname);

MessageConsumer consumer = null;

if(threadLocal.get()!=null){

consumer = threadLocal.get();

}else{

consumer = session.createConsumer(queue);

threadLocal.set(consumer);

}

while(true){

Thread.sleep(1000);

TextMessage msg = (TextMessage) consumer.receive();

if(msg!=null) {

msg.acknowledge();

System.out.println(Thread.currentThread().getName()+": Consumer:我是消费者，我正在消费Msg"+msg.getText()+"--->"+count.getAndIncrement());

}else {

break;

}

}

} catch (JMSException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

### 4、运行ActiveMQ项目

#### 4.1 生产者开始生产消息

public class TestMq {

public static void main(String[] args){

Producter producter = new Producter();

producter.init();

TestMq testMq = new TestMq();

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

//Thread 1

new Thread(testMq.new ProductorMq(producter)).start();

//Thread 2

new Thread(testMq.new ProductorMq(producter)).start();

//Thread 3

new Thread(testMq.new ProductorMq(producter)).start();

//Thread 4

new Thread(testMq.new ProductorMq(producter)).start();

//Thread 5

new Thread(testMq.new ProductorMq(producter)).start();

}

private class ProductorMq implements Runnable{

Producter producter;

public ProductorMq(Producter producter){

this.producter = producter;

}

@Override

public void run() {

while(true){

try {

producter.sendMessage("Jaycekon-MQ");

Thread.sleep(10000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

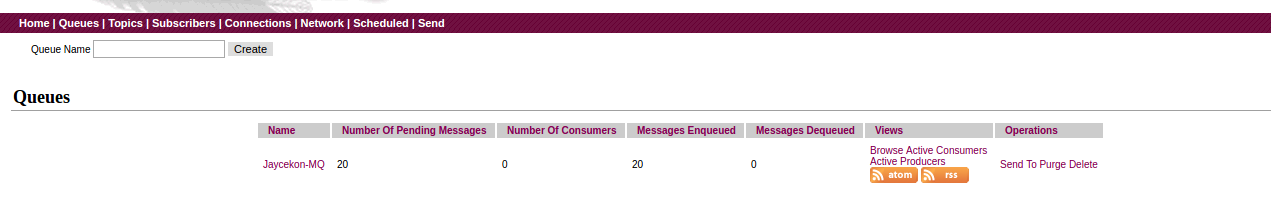
}

}

}

　　　运行结果：

INFO | Successfully connected to tcp://localhost:61616  
Thread-6productor:我是大帅哥，我现在正在生产东西！,count:0  
Thread-4productor:我是大帅哥，我现在正在生产东西！,count:1  
Thread-2productor:我是大帅哥，我现在正在生产东西！,count:3  
Thread-5productor:我是大帅哥，我现在正在生产东西！,count:2  
Thread-3productor:我是大帅哥，我现在正在生产东西！,count:4  
Thread-6productor:我是大帅哥，我现在正在生产东西！,count:5  
Thread-3productor:我是大帅哥，我现在正在生产东西！,count:6  
Thread-5productor:我是大帅哥，我现在正在生产东西！,count:7  
Thread-2productor:我是大帅哥，我现在正在生产东西！,count:8  
Thread-4productor:我是大帅哥，我现在正在生产东西！,count:9  
Thread-6productor:我是大帅哥，我现在正在生产东西！,count:10  
Thread-3productor:我是大帅哥，我现在正在生产东西！,count:11  
Thread-5productor:我是大帅哥，我现在正在生产东西！,count:12  
Thread-2productor:我是大帅哥，我现在正在生产东西！,count:13  
Thread-4productor:我是大帅哥，我现在正在生产东西！,count:14  
Thread-6productor:我是大帅哥，我现在正在生产东西！,count:15  
Thread-3productor:我是大帅哥，我现在正在生产东西！,count:16  
Thread-5productor:我是大帅哥，我现在正在生产东西！,count:17  
Thread-2productor:我是大帅哥，我现在正在生产东西！,count:18  
Thread-4productor:我是大帅哥，我现在正在生产东西！,count:19



#### 4.2 消费者开始消费消息

public class TestConsumer {

public static void main(String[] args){

Comsumer comsumer = new Comsumer();

comsumer.init();

TestConsumer testConsumer = new TestConsumer();

new Thread(testConsumer.new ConsumerMq(comsumer)).start();

new Thread(testConsumer.new ConsumerMq(comsumer)).start();

new Thread(testConsumer.new ConsumerMq(comsumer)).start();

new Thread(testConsumer.new ConsumerMq(comsumer)).start();

}

private class ConsumerMq implements Runnable{

Comsumer comsumer;

public ConsumerMq(Comsumer comsumer){

this.comsumer = comsumer;

}

@Override

public void run() {

while(true){

try {

comsumer.getMessage("Jaycekon-MQ");

Thread.sleep(10000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

　　运行结果：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | INFO | Successfully connected to tcp://localhost:61616  Thread-2: Consumer:我是消费者，我正在消费MsgThread-5productor:我是大帅哥，我现在正在生产东西！,count:4--->0  Thread-3: Consumer:我是消费者，我正在消费MsgThread-4productor:我是大帅哥，我现在正在生产东西！,count:36--->1  Thread-4: Consumer:我是消费者，我正在消费MsgThread-3productor:我是大帅哥，我现在正在生产东西！,count:38--->2  Thread-5: Consumer:我是消费者，我正在消费MsgThread-6productor:我是大帅哥，我现在正在生产东西！,count:37--->3  Thread-2: Consumer:我是消费者，我正在消费MsgThread-6productor:我是大帅哥，我现在正在生产东西！,count:2--->4  Thread-3: Consumer:我是消费者，我正在消费MsgThread-5productor:我是大帅哥，我现在正在生产东西！,count:40--->5  Thread-4: Consumer:我是消费者，我正在消费MsgThread-6productor:我是大帅哥，我现在正在生产东西！,count:42--->6  Thread-5: Consumer:我是消费者，我正在消费MsgThread-4productor:我是大帅哥，我现在正在生产东西！,count:41--->7  Thread-2: Consumer:我是消费者，我正在消费MsgThread-3productor:我是大帅哥，我现在正在生产东西！,count:1--->8  Thread-3: Consumer:我是消费者，我正在消费MsgThread-2productor:我是大帅哥，我现在正在生产东西！,count:44--->9  Thread-4: Consumer:我是消费者，我正在消费MsgThread-4productor:我是大帅哥，我现在正在生产东西！,count:46--->10  Thread-5: Consumer:我是消费者，我正在消费MsgThread-5productor:我是大帅哥，我现在正在生产东西！,count:45--->11  Thread-2: Consumer:我是消费者，我正在消费MsgThread-2productor:我是大帅哥，我现在正在生产东西！,count:3--->12  Thread-3: Consumer:我是消费者，我正在消费MsgThread-3productor:我是大帅哥，我现在正在生产东西！,count:48--->13  Thread-4: Consumer:我是消费者，我正在消费MsgThread-5productor:我是大帅哥，我现在正在生产东西！,count:50--->14  Thread-5: Consumer:我是消费者，我正在消费MsgThread-2productor:我是大帅哥，我现在正在生产东西！,count:49--->15  Thread-4: Consumer:我是消费者，我正在消费MsgThread-2productor:我是大帅哥，我现在正在生产东西！,count:54--->16  Thread-2: Consumer:我是消费者，我正在消费MsgThread-5productor:我是大帅哥，我现在正在生产东西！,count:6--->17  Thread-3: Consumer:我是消费者，我正在消费MsgThread-6productor:我是大帅哥，我现在正在生产东西！,count:52--->18  Thread-5: Consumer:我是消费者，我正在消费MsgThread-3productor:我是大帅哥，我现在正在生产东西！,count:53--->19  Thread-4: Consumer:我是消费者，我正在消费MsgThread-3productor:我是大帅哥，我现在正在生产东西！,count:58--->20 |

　　查看运行结果，我们可以做ActiveMQ 服务端：<http://127.0.0.1:8161/admin/> 里面的Queues 中查看我们生产的消息。



### 5、ActiveMQ的特性

#### 5.1 ActiveMq 的特性

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

#### 5.2 什么情况下使用ActiveMQ?

1. 多个项目之间集成   
   (1) 跨平台   
   (2) 多语言   
   (3) 多项目
2. 降低系统间模块的耦合度，解耦   
   (1) 软件扩展性
3. 系统前后端隔离   
   (1) 前后端隔离，屏蔽高安全区