

# MQ 안내서

JEUS 9.1

**TMAXSOFT**

## 저작권 공지

Copyright 2025. TmaxSoft Co., Ltd. All Rights Reserved.

## 회사 정보

(주)티맥스소프트

주소 : 경기도 성남시 분당구 정자일로 45, 티맥스소프트타워

기술 서비스 센터: 1544-8629

홈페이지: <https://www.tmaxsoft.com>

## 제한된 권리

이 소프트웨어(JEUS®) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

## 상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

Java, Solaris는 Oracle Corporation 및 그 자회사, 관계회사의 등록 상표입니다.

Microsoft, Windows, Windows NT는 Microsoft Corporation의 등록 상표 또는 상표입니다.

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

Noto는 Google Inc.의 상표입니다. Noto 글꼴은 오픈 소스입니다. 모든 Noto 글꼴은 SIL Open Font License, 버전 1.1에 따라 게시됩니다. (<https://www.google.com/get/noto/>)

기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상호, 상표, 또는 등록 상표입니다.

본 사용설명서에 기재된 회사, 시스템, 제품 이름 등에 반드시 상표 표시(™, ®)를 하지는 않습니다.

## 오픈 소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다.: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. :

`${INSTALL_PATH}/license/oss_licenses`

## 유지 보수

구분	지원항목	서비스 내용
제품지원	패치 & 업그레이드	무상 패치 서비스 제공
		메이저 버전 업그레이드 시 할인 혜택
		웹 지원을 통한 패치 내역 제공
기술 지원 - 기본 서비스	장애 지원	장애 발생 시 원인 분석 및 조치  Service Desk팀 → 기술팀 → R&D의 3단계 장애 분석 및 조치
	일상 지원(온라인 지원)	E-mail, 전화, 원격, 웹 사이트 등 온라인 자원을 통한 질의 응답 서비스
	고객 맞춤 지원(방문 지원)	고객의 요청으로 수행하는 방문 지원 서비스
기술 지원 - 옵션 서비스	예방 지원	정기 점검을 통한 시스템 운영현황 보고 및 장애 예방  ◦ 관리자 또는 운영자의 요구사항 수렴 ◦ 운영 현황(시스템, 엔진 운영) 보고서 제공 ◦ 필요 시 시스템 개선 권장 사항 보고
유지 보수 비용 및 기간	계약 시 별도 협의	계약 시 EOL/EOS 문서 제공

## 안내서 이력

제품 버전	안내서 버전	발행일	비고
JEUS 9.1	3.3.1	2025-09-30	GA 버전
JEUS 9 Fix#1	3.2.2	2025-07-10	JDK 21 지원 내용 추가
JEUS 9 Fix#1	3.2.1	2025-06-30	RC 버전
JEUS 9	3.1.2	2025-01-03	-
JEUS 9	3.1.1	2024-09-27	-

# 목차

1. 소개	1
1.1. Jakarta Messaging(JMS)	1
1.2. JEUS MQ의 특징	1
2. JEUS MQ 클라이언트 프로그래밍	3
2.1. 개요	3
2.2. JMS 관리 객체	5
2.2.1. JNDI 서비스 정의	5
2.2.2. Connection Factory	7
2.2.3. Destination	8
2.3. 커넥션과 세션	10
2.3.1. 커넥션 생성	11
2.3.2. 물리적 연결 공유	11
2.3.3. 세션 생성	13
2.3.4. Client Facility Pooling	14
2.3.5. NONE_ACKNOWLEDGE 모드	14
2.3.6. JMSContext	16
2.4. 메시지	17
2.4.1. 메시지 헤더 필드	17
2.4.2. 메시지 프로퍼티	17
2.4.3. 메시지 바디	19
2.4.4. FileMessage	19
2.5. 트랜잭션	22
2.5.1. 로컬 트랜잭션	22
2.5.2. 분산 트랜잭션	23
3. JEUS MQ 서버 설정	25
3.1. 개요	25
3.1.1. 디렉터리 구조	25
3.2. JMS 리소스 설정	25
3.2.1. Destination 설정	26
3.2.2. Durable Subscription 설정	27
3.3. JMS Quota 설정	28
3.3.1. Quota 설정	28
3.4. JMS 엔진 설정	28
3.4.1. 기본 정보 설정	28
3.4.2. 서비스 채널 설정	30
3.4.3. Connection Factory 설정	31
3.4.4. Persistence Store 설정	31
3.4.5. Message Sort 설정	33
3.5. 서버 관리 및 모니터링	33

3.5.1. 서버 관리	33
3.5.2. 서버 모니터링	34
4. JEUS MQ 클러스터링	36
4.1. 개요	36
4.2. 클러스터링 종류	36
4.2.1. Connection Factory 클러스터링	36
4.2.2. Destination 클러스터링	37
4.3. 클러스터링 사용	38
4.3.1. 서버 설정	38
4.3.2. 클라이언트 설정	38
4.4. 예제	39
4.4.1. 일반적인 사용 예제	39
4.4.2. 잘못된 사용 예제	40
5. JEUS MQ 장애 극복	42
5.1. 개요	42
5.2. 서버 장애 극복	42
5.2.1. 네트워크 구성	42
5.2.2. Connection Factory 설정	44
5.2.3. Persistence Store 설정	45
5.2.4. 자동 복원(Fail-Back)	46
5.3. 클라이언트 장애 극복	46
5.3.1. 재연결	46
5.3.2. Connection Factory 재사용	46
5.3.3. Destination 재사용	47
5.3.4. 응답 대기시간	47
5.3.5. 커넥션 복구	47
5.3.6. 세션 복구	48
5.3.7. 메시지 전송 복구	50
5.3.8. 메시지 수신 복구	51
5.3.9. 메시지 유실 방지와 트랜잭션	52
6. JEUS MQ 특수 기능	54
6.1. JEUS MQ Message Bridge	54
6.1.1. 서버 설정	54
6.2. JEUS MQ Message Sort	56
6.2.1. 서버 설정	56
6.2.2. 클라이언트 설정	57
6.3. JEUS MQ Global Order	58
6.3.1. 클라이언트 설정	58
6.4. JEUS MQ Message Group	58
6.4.1. 서버 설정	58
6.4.2. 클라이언트 설정	59

6.5. JEUS MQ Message Management 기능 .....	60
6.5.1. 메시지 모니터링 .....	61
6.5.2. 메시지 제어 .....	62
6.5.3. Destination 모니터링 .....	63
6.5.4. Destination 제어 .....	64
6.6. JEUS MQ Topic Multicast .....	65
6.6.1. 서버 설정 .....	65
6.7. 신뢰도 높은 메시지 송신 .....	65
6.7.1. LPQ 활성화 .....	66
6.7.2. LPQ 사용 설정 .....	67
6.7.3. LPQ 리스너 설정 .....	68
6.7.4. LPQ 설정 .....	69
Appendix A: Journal Store 추가 속성 .....	72
Appendix B: JDBC Persistence Store 컬럼 .....	74
B.1. Destination Table .....	74
B.2. Durable Subscription Table .....	74
B.3. Message Table .....	74
B.4. MetaInfo Table .....	75
B.5. Subscription Message Table .....	75
B.6. Transaction Table .....	75

# 1. 소개

본 장에서는 Jakarta Messaging(이하 JMS)를 간단히 소개한 뒤 JEUS MQ의 특징에 대해 설명한다.

## 1.1. Jakarta Messaging(JMS)

JMS는 애플리케이션 간의 통신을 메시지 기반으로 수행하기 위한 Java 표준 API를 정의한 것이다. 여기에는 메시징에 필요한 구성 요소 및 메시지 모델에 해당하는 인터페이스를 정의하고 이들 간의 관계를 설명하고 있다.

JMS는 다음과 같은 특징을 갖는다.

- 느슨한 결합 구조

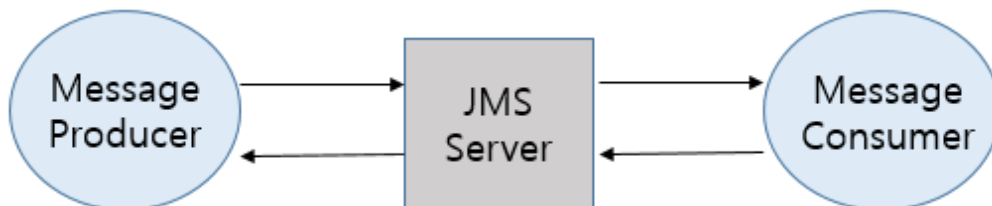
메시지 송신자와 수신자는 서로에 대해 알 필요가 없다.

- 비동기 통신

메시지 수신자는 요청하지 않아도 서버에 도착하는 대로 메시지를 전달받을 수 있다.

- 신뢰성 있는 메시지 전달

메시지가 반드시 한 번(Once-and-Only-Once) 전달되는 것을 보장한다.



JMS Messaging



JMS에 대한 보다 자세한 설명이나 API 사용법에 대해서는 [JMS 스펙 문서](#)를 참고한다.

## 1.2. JEUS MQ의 특징

JEUS MQ(Message Queue)는 TmaxSoft의 Jakarta EE 서버인 JEUS에 포함된 JMS 스펙의 구현체로 Non-Blocking I/O와 XA 트랜잭션을 지원하며 보안이 강화된 가용성 높은 메시지 서비스를 제공한다.

JEUS MQ는 JMS 버전 2.0 스펙을 지원하며 다음과 같은 주요 특징을 갖는다.

- Non-Blocking I/O 지원

Non-Blocking I/O를 사용하여 같은 시스템 리소스로 더 많은 클라이언트를 동시에 처리할 수 있다.

- XA 트랜잭션 지원

JMS 스펙에서 선택 사항으로 정의되어 있는 XA 트랜잭션을 지원한다.

- 보안 강화

SSL(Secure Socket Layer) 통신을 지원하며, JEUS Security를 사용할 수 있다.

- 고가용성

클라이언트는 네트워크나 서버 장애로부터 자동으로 장애가 극복된다.

- 규모 가변성

클라이언트나 메시지 처리량이 증가해도 JEUS MQ 클러스터링을 통해서 부하를 분산할 수 있다.



## 2. JEUS MQ 클라이언트 프로그래밍

본 장에서는 JEUS MQ 클라이언트의 유형, 작성 절차 및 관리 객체에 대해서 설명한다.

### 2.1. 개요

JEUS MQ 클라이언트의 유형과 작성 절차에 대해 설명한다.

#### 클라이언트의 유형

JMS 클라이언트는 메시지를 주고받는 역할에 따라 메시지 송신자와 메시지 수신자로 구분할 수 있으며 하나의 클라이언트가 2가지 역할을 모두 수행하기도 한다. 이와는 별도로 Java 애플리케이션을 배치하고 실행하는 방식에 따라 JEUS MQ 클라이언트를 다음과 같이 분류할 수 있다.

- 독립 애플리케이션

Java SE 환경에서 독립적으로 실행되는 클라이언트이다.

- Jakarta EE 애플리케이션

EJB(Jakarta Enterprise Beans)나 서블릿처럼 Jakarta EE 서버에 deploy되어 실행되는 형태의 클라이언트다. JEUS에 Jakarta EE 애플리케이션을 deploy하고 실행하는 방법에 대해서는 "JEUS Applications & Deployment 안내서"를 참고한다.

- Message Driven Bean

EJB의 한 종류로 Jakarta EE 애플리케이션의 특수한 경우라고 볼 수 있지만, 동작 방식에 있어서는 싱글 스레드 방식으로 작성된 Java SE 애플리케이션을 멀티 스레드 방식으로 확장해준다. Message Driven Bean(이하 MDB)에 대한 자세한 내용은 EJB 관련 서적이나 [Jakarta EE Tutorials](#)를 참고한다.

JEUS에서 MDB를 사용하기 위한 설정 방법은 JEUS EJB 안내서의 "Message Driven Bean(MDB)"를 참고한다.



현재 JEUS MQ에서는 Java 이외의 언어로 작성된 클라이언트는 지원하지 않는다.

#### 클라이언트 작성 절차

JEUS MQ 클라이언트의 유형 중 독립 애플리케이션의 경우를 기준으로 JEUS MQ 클라이언트를 작성하는 전형적인 과정에 대해 설명한다.

Jakarta EE 환경에서 실행되는 클라이언트의 경우에는 XML Deployment Descriptor에 설정해서 초기 JNDI(Java Naming and Directory Interface) Lookup 과정을 대체할 수 있다.

1. JNDI Initial Context를 생성한다.

```
Context context = new InitialContext();
```

JEUS의 JNDI 서비스를 사용하기 위한 환경설정 방법은 [JNDI 서비스 정의](#)를 참고한다.

## 2. JNDI Lookup을 통해 Connection Factory를 얻는다.

```
ConnectionFactory connectionFactory =  
    (ConnectionFactory) context.lookup("jms/ConnectionFactory");
```

JEUS MQ 전용 API를 사용하여 JNDI Lookup 없이 Connection Factory를 얻는 방법도 있다. 자세한 내용은 [Connection Factory](#)를 참고한다.

## 3. Connection Factory를 이용해 커넥션을 맺는다.

```
Connection connection = connectionFactory.createConnection();
```

## 4. 커넥션으로부터 세션을 생성한다.

```
Session session = connection.createSession(false, AUTO_ACKNOWLEDGE);
```

## 5. JNDI Lookup을 통해 Destination을 얻는다.

```
Destination destination = (Destination) context.lookup("ExamplesQueue");
```

Connection Factory의 경우와 마찬가지로 JEUS MQ 전용 API를 사용하면 JNDI Lookup 없이 Destination을 얻을 수 있다. 이 내용은 [Destination](#)에서 설명한다.

## 6. 메시지를 보내는 클라이언트라면 세션으로부터 메시지 송신자를 생성한다.

```
MessageProducer producer = session.createProducer(destination);
```

또는 메시지를 받길 원하는 경우 세션으로부터 메시지 수신자를 생성한다.

```
MessageConsumer consumer = session.createConsumer(destination);
```

메시지를 비동기적으로 수신하여 처리하려면 추가로 메시지 수신자에 `MessageListener` 인터페이스를 구현한 객체를 등록한다.

```
MessageListener listener = new MyMessageListener();  
consumer.setMessageListener(listener);
```

7. 커넥션을 시작한다.

```
connection.start();
```

8. 비즈니스 로직을 수행하면서 메시지를 보내거나 받는다.

메시지는 메시지 송신자를 통해서 보낸다.

```
TextMessage message = session.createTextMessage("Hello, World");  
producer.send(msg);
```

또는 동기적으로 메시지를 수신하는 경우 메시지 수신자의 receive() 메소드를 호출한다.

```
Message message = consumer.receive();
```

반면에 비동기적으로 메시지를 수신하는 경우 수신된 메시지는 **단계 6**에서 등록한 MessageListener 객체의 onMessage() 메소드에서 처리하게 된다.

9. 모든 메시지 송수신이 완료되면 커넥션을 종료한다.

```
connection.close();
```

## 2.2. JMS 관리 객체

JMS 관리 객체(Administered Object)에는 Connection Factory와 Destination의 2가지 종류가 있다.

주로 JMS 서버의 설정에 의해 서버 내에서 생성되고 서버 관리자의 관리를 받는다. JMS 클라이언트는 JMS 서비스를 이용하기 위해 이 관리 객체(또는 그에 대한 Reference)들을 서버로부터 가져와 사용한다. 클라이언트가 JMS 관리 객체를 서버로부터 얻는 방법으로는 JNDI Lookup 방식이 가장 일반적이다.

본 절에서는 JEUS MQ 서버로부터 Connection Factory나 Destination의 Reference를 얻는 방법과 JNDI Lookup을 위해 클라이언트에서 JEUS JNDI 서비스를 정의하는 방법을 설명한다. 또한 클라이언트에서 API를 사용하여 JEUS MQ 서버에 동적으로 Destination을 생성하는 방법과 JEUS MQ 서버의 비정상 메시지 Destination(Dead Message Destination)에 대해서 설명한다.

### 2.2.1. JNDI 서비스 정의

JMS 관리 객체를 JNDI Lookup을 통해 얻기 위해서는 먼저 JNDI 서비스가 정의되어 있어야 한다.

JNDI 서비스는 다음의 3가지 프로퍼티로 정의할 수 있다.

- java.naming.factory.initial
- java.naming.factory.url.pkgs

- java.naming.provider.url

정의된 3가지 프로퍼티는 다음의 방법으로 적용할 수 있다.

#### • JNDI 프로퍼티 파일 작성

JNDI 서비스를 정의하는 가장 편리한 방법은 프로퍼티들의 값을 지정한 jndi.properties 파일을 작성하는 것이다.

다음은 JEUS의 JNDI 서비스를 사용하기 위한 환경을 기술한 jndi.properties 파일의 예제이다.

<jndi.properties>

```
java.naming.factory.initial=jeus.jndi.JEUSContextFactory
java.naming.factory.url.pkgs=jeus.jndi.jns.url
java.naming.provider.url=127.0.0.1:9736
```

이와 같은 jndi.properties 파일을 작성하여 클래스 패스에 위치시키거나 클라이언트 프로그램을 배포할 때 JAR 파일에 클래스 파일들과 함께 포함시키면 InitialContext 객체가 jndi.properties 파일의 설정을 읽을 수 있다.

#### • 시스템 프로퍼티로 전달

클라이언트를 실행하는 시점에 Customizing이 필요한 상황에서는 JAR 파일에 포함된 jndi.properties 파일을 수정하기가 어렵다. 이때에는 클라이언트를 실행할 때 다음과 같이 시스템 프로퍼티로 설정값을 전달하는 방법을 사용할 수 있다.

```
java -Djava.naming.factory.initial=jeus.jndi.JEUSContextFactory \
-Djava.naming.factory.url.pkgs=jeus.jndi.jns.url \
-Djava.naming.provider.url=127.0.0.1:9736 \
. . .
```

#### • 애플릿 프로퍼티로 전달

JEUS MQ 클라이언트가 애플릿 형태로 되어 있을 때는 jndi.properties 파일을 작성하는 것보다 다음과 같이 HTML 파일에서 <applet> 태그에 <param> 태그를 사용하여 넘겨주는 방법이 Customizing에 유리하다.

```
<applet code="JeusMqApplet" width="640" height="480">
  <param name="java.naming.factory.initial"
    value="jeus.jndi.JEUSContextFactory"/>
  <param name="java.naming.factory.url.pkgs" value="jeus.jndi.jns.url"/>
  <param name="java.naming.provider.url" value="127.0.0.1:9736"/>
</applet>
```

애플릿 내에는 Applet.getParameter() 메소드를 사용하여 InitialContext 생성에 필요한 환경을 설정하는 코드를 다음과 같이 작성해야 한다.

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, getParameter("java.naming.factory.initial"));
env.put(Context.URL_PKG_PREFIXES, getParameter("java.naming.factory.url.pkgs"));
```

```
env.put(Context.PROVIDER_URL, getParameter("java.naming.provider.url"));

Context context = new InitialContext(env);
```

## • 코드 삽입

클라이언트 코드 내에서 InitialContext 생성에 필요한 환경을 직접 설정하는 방법이다.

다음과 같이 프로퍼티들을 Hashtable 객체에 담아 InitialContext를 생성할 때 생성자에게 파라미터로 전달한다.

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "jeus.jndi.JEUSContextFactory");
env.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
env.put(Context.PROVIDER_URL, "127.0.0.1:9736");

Context context = new InitialContext(env);
```



이 방법을 사용하면 다른 JNDI 서비스를 사용하거나 서비스 정의가 바뀔 때 소스 코드도 수정해야 하므로 유지보수가 어려워진다.

## 2.2.2. Connection Factory

Connection Factory는 JEUS MQ 서버에 커넥션(Connection)을 맺는 데 필요한 정보를 가지고 있다.

일반적으로 JMS 클라이언트는 JNDI Lookup을 통해 Connection Factory를 얻고 이를 사용해서 JMS 서버로의 커넥션을 생성한다.

```
ConnectionFactory connectionFactory =
    (ConnectionFactory) context.lookup("jms/ConnectionFactory");
QueueConnectionFactory queueConnectionFactory =
    (QueueConnectionFactory) context.lookup("jms/QueueConnectionFactory");
TopicConnectionFactory topicConnectionFactory =
    (TopicConnectionFactory) context.lookup("jms/TopicConnectionFactory");
```

JEUS MQ의 분산 트랜잭션을 사용하려면 XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory도 위와 비슷하게 Lookup하여 사용할 수 있다.

Jakarta EE 클라이언트에서는 InitialContext.lookup() 메소드를 호출하는 대신 @Resource Annotation을 사용해서 다음과 같이 Connection Factory를 가져오는 방법도 있다.

```
@Resource(mappedName="jms/ConnectionFactory")
private static ConnectionFactory connectionFactory;
```

JNDI 서비스를 이용할 수 없는 상황에서는 JEUS MQ가 제공하는 전용 API를 사용하여 JNDI Lookup 없이 Connection Factory를 얻을 수 있다.

```
jeus.jms.client.util.JeusConnectionFactoryCreator connectionFactoryCreator =  
    new jeus.jms.client.util.JeusConnectionFactoryCreator();  
connectionFactoryCreator.setFactoryName("ConnectionFactory");  
connectionFactoryCreator.addServerAddress("127.0.0.1", 9741, "internal");  
ConnectionFactory connectionFactory =  
    (ConnectionFactory) connectionFactoryCreator.createConnectionFactory();
```



JNDI Lookup이나 @Resource Annotation을 사용할 때는 Connection Factory를 얻기 위해 Connection Factory의 JNDI 이름(본 절의 예에서는 "jms/ConnectionFactory")을 참조하지만 JeusConnectionFactoryCreator를 사용할 때에는 JEUS MQ 서버가 내부적으로 사용하는 Connection Factory 이름(마지막 예에서 "ConnectionFactory")으로 참조하고 있는 점에 유의한다. Connection Factory 이름과 JNDI 이름의 차이점에 대해서는 [Connection Factory 설정](#)의 예를 참고한다.

## 2.2.3. Destination

Destination은 서버에 의해 관리되는 메시지를 저장하는 공간으로 Queue와 Topic의 2가지 종류가 있다.

Queue는 하나의 메시지를 하나의 클라이언트에만 전달하는 방식(one to one)을 지원하고, Topic은 하나의 메시지를 모든 클라이언트에게 동일하게 전달하는 방식(one to many)을 지원한다.

JMS 클라이언트는 서버에서 관리하는 Destination 객체의 Reference를 얻어서 Destination에 메시지를 송신하거나 Destination으로부터 메시지를 수신한다.

JEUS MQ 클라이언트에서 Destination을 얻는 가장 일반적인 방법은 JEUS MQ 서버가 Destination Reference를 등록한 JNDI 서버를 통해 JNDI 이름으로 Lookup하는 것이다.

```
Queue queue = (Queue) context.lookup("jms/ExamplesQueue");  
Topic topic = (Topic) context.lookup("jms/ExamplesTopic");
```

Jakarta EE 클라이언트에서는 InitialContext.lookup() 메소드를 호출하는 대신 @Resource Annotation을 사용하여 다음과 같이 Destination Reference를 가져오는 방법도 있다.

```
@Resource(mappedName="jms/ExamplesQueue")  
private static Queue queue;
```

ConnectionFactory와 마찬가지로 JEUS MQ 전용 API를 사용하면 JNDI Lookup 없이 Destination Reference를 얻을 수 있다. 이때 Destination을 참조하기 위해서는 JEUS MQ 서버가 내부적으로 사용하는 Destination 이름을 사용해야 한다.

```
jeus.jms.client.util.JeusDestinationCreator destinationCreator =
    new jeus.jms.client.util.JeusDestinationCreator();
destinationCreator.setDestinationName("__ExamplesQueue__");
destinationCreator.setDestinationClass(Destination.class);
Destination destination = (Destination) destinationCreator.createDestination();
```

또는 JMS 구현체마다 다르게 구현하도록 되어 있는 `Session.createQueue(String)`, `Session.createTopic(String)` 메소드를 사용할 수 있다. 앞의 예에서와 마찬가지로 실제 Destination 이름을 파라미터로 넘겨주어야 한다.

```
Queue queue = session.createQueue("ExamplesQueue");
Topic topic = session.createTopic("ExamplesTopic");
```

## Destination 동적 생성

JEUS MQ는 클라이언트가 `Session.createQueue(String)`, `Session.createTopic(String)` 메소드를 사용해서 서버에 동적으로 Destination을 생성할 수 있는 방법을 제공한다.

이때 파라미터로 전달하는 문자열은 생성하려는 Destination 이름 뒤에 물음표(?)를 붙이고, 설정하려는 옵션이 있는 경우 그 뒤에 "param=value" 형태로 원하는 설정을 나열한다. 옵션을 2개 이상 설정할 때에는 앰퍼샌드(&)를 구분자로 사용한다. param 및 value 값으로 domain.xml 환경 파일에 Destination을 설정할 때와 동일한 이름 및 값을 사용할 수 있다.



Destination을 동적으로 생성할 때 설정할 수 있는 옵션은 export-name이 있다.

다음의 예에서는 Destination 이름이 "DynamicQueue", JNDI 이름이 "jms/DynamicQueue"인 Queue를 동적으로 생성하고 있다.

```
QueueConnectionFactory queueConnectionFactory =
    (QueueConnectionFactory) context.lookup("jms/QueueConnectionFactory");
QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();
QueueSession queueSession =
    queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
Queue queue = queueSession.createQueue(
    "DynamicQueue?export-name=jms/DynamicQueue");
```

이 API에는 다음과 같은 특징이 있다.

- 지정한 이름의 Destination이 서버에 이미 존재하거나 지정한 JNDI 이름으로 이미 다른 객체가 등록되어 있으면 `JMSEXception`이 발생한다. JNDI 이름을 설정하지 않은 경우 Destination 이름이 JNDI 이름으로 사용된다.
- JEUS MQ 서버가 보안 기능을 사용하는 경우 커넥션을 생성할 때 사용한 Subject에게 "jeus.jms.destination.creation" 리소스에 대한 "createDestination" 권한이 없으면 `JMSEXception`이 발생한다.
- 그 밖의 설정은 Destination의 기본 설정을 따른다.

- 동적으로 생성한 Destination은 JEUS MQ 서버가 종료될 때 삭제된다.

위의 예에서 생성한 Destination은 JEUS MQ서버가 종료되기 전까지 설정에 의한 Destination들과 동일하게 동작한다.



JEUS MQ 클러스터링을 사용하고 있을 때에는 Destination 동적 생성을 사용할 수 없다.

## 비정상 메시지 Destination

비정상 메시지 Destination은 특별한 이유로 정상적으로 처리되지 못한 메시지들을 보관하는 시스템 Destination이다. 메시지에 설정된 Destination을 찾을 수 없거나 지정한 횟수 이상 반복해서 메시지가 복구되는 경우가 여기에 포함된다.

메시지 수신자에게 전달된 메시지가 서버 측 Destination으로 복구되는 상황은 다음과 같다.

- 메시지 수신자에서 Session.recover() 등을 호출한 경우
- Message Listener의 onMessage() 메소드를 수행하는 중에 예외가 발생한 경우 등

복구된 메시지는 메시지 수신자에게 다시 전달되는데 만일 비즈니스 로직이나 클라이언트의 오류로 인해 반복적으로 특정 메시지 처리에 실패한다면 계속해서 Destination에는 메시지가 쌓이고 클라이언트는 메시지를 수신하지 못하게 된다. 이런 문제를 피하기 위해 JEUS MQ는 메시지가 복구되는 최대 횟수를 제한할 수 있는 기능을 제공한다. "JMS\_JEUS\_RedeliveryLimit" 메시지 프로퍼티 값을 설정하고 메시지를 송신한다.

```
message.setIntProperty("JMS_JEUS_RedeliveryLimit", <integer value>)
```

메시지 프로퍼티의 기본값은 메시지 송신자 클라이언트를 실행할 때 다음과 같이 "jeus.jms.client.default-redelivery-limit" 시스템 프로퍼티를 설정하여 변경할 수 있다.

```
-Djeus.jms.client.default-redelivery-limit=<integer value>
```

별도로 설정하지 않으면 3이 사용된다. JEUS MQ 서버는 기동할 때 "JEUSMQ\_DLQ"라는 이름의 비정상 메시지 Destination을 생성한다. 비정상 메시지 Destination에 쌓인 메시지는 보통 시스템 관리 툴에서 처리하지만 일반 클라이언트도 이 이름으로 비정상 메시지 Destination에 접근하여 원하는 작업을 할 수 있다.



비정상 메시지 Destination에 메시지가 너무 많이 쌓일 경우, OutOfMemoryError의 원인이 될 수도 있다. 별도의 메시지 수신자 클라이언트를 연결해놓거나 일정 주기로 시스템 관리툴로 쌓인 메시지를 삭제하여 비정상메시지 Destination에 메시지가 과도하게 쌓이지 않도록 유의해야 한다.

## 2.3. 커넥션과 세션



JMS 클라이언트가 서버와 메시지를 주고받기 위해서는 서버와 커넥션을 맺고 세션을 생성해야 한다. 커넥션과 세션은 JMS에서 매우 중요한 클라이언트 리소스로 시스템의 성능 및 클라이언트가 장애 상황에 대처하는 방식에 영향을 미친다. JEUS MQ는 커넥션의 성능에 대한 몇 가지 옵션을 제공한다.

본 절에서는 이런 옵션들을 설정하는 방법과 옵션을 설정했을 때 JEUS MQ 클라이언트가 어떻게 동작하는지를 설명한다.

### 2.3.1. 커넥션 생성

커넥션은 클라이언트가 JMS 서버와 작업하기 위해 최초로 생성하는 객체이며 JMS 서버와의 물리적 혹은 논리적 연결을 나타낸다. 커넥션은 Connection Factory의 `createConnection()` 메소드를 호출하여 생성한다.

```
public Connection createConnection()  
    throws JMSException;  
public Connection createConnection(String userName, String password)  
    throws JMSException;
```

JEUS MQ 서버가 보안 기능을 사용하는 경우 `userName/password`에 해당하는 Subject에게 "jeus.jms.client.connectionFactory" 리소스에 대한 "createConnection" 권한이 없으면 `JMSException`이 발생한다.

JEUS MQ 클라이언트를 실행할 때 다음과 같은 옵션을 시스템 프로퍼티로 설정할 수 있다.

```
-Djeus.jms.client.connect.timeout=<long value>
```

커넥션이 맺어질 때까지 `ConnectionFactory.createConnection()` 메소드가 대기하는 시간이다. 이 시간 동안 연결되지 않으면 `ConnectionFactory.createConnection()` 메소드에서 `JMSException`이 발생한다. 0으로 설정하면 연결에 성공할 때까지 무한 대기한다. (기본값: 5초, 단위: ms)

클라이언트 코드에 다음과 같은 문장을 추가하면 런타임에도 설정 변경이 가능하다.

```
System.setProperty("jeus.jms.client.connect.timeout", <long value>);
```

또는

```
System.setProperty(jeus.jms.common.JeusJMSProperties.KEY_CLIENT_CONNECT_TIMEOUT, <long value>);
```

### 2.3.2. 물리적 연결 공유

일반적으로 애플리케이션이 메시지를 전송하는 과정은 다음과 같다.

#### 1. 커넥션 생성

2. 세션 생성
3. Message Producer 생성
4. 메시지 전송
5. 커넥션 종료

만약 JMS 커넥션이 물리적 커넥션(Socket)과 1대 1의 관계를 갖고 있다면 위와 같은 사용 패턴에서는 메시지를 한 번 보낼 때마다 매번 새로운 물리적 연결을 맺어야 하므로 성능이 많이 저하된다. 실제 메시지를 전송하는 시간보다 물리적 연결을 맺기 위한 시간이 더 많이 걸리게 된다. 뿐만 아니라 물리적 연결이 증가하면 file descriptor의 사용이 많아져 IOException을 발생시키기도 하므로 안정성도 떨어진다.

JEUS 6 Fix#6 이후부터는 물리적 커넥션을 공유함으로써 1대 1 관계에서 성능, 안전성 상에 발생할 수 있는 문제들을 해결하였다. 그러나 일부 환경에서는 물리적 연결을 공유하는 것보다 매번 연결을 새로 생성하는 것이 나을 수 있으므로 이 기능은 클라이언트 옵션을 설정하여 적용할 수 있도록 하였다.

기본적으로 물리적 연결은 ConnectionFactory와 1대 1 관계를 갖으나 JEUS MQ 장애 극복을 설정하거나 "jeus.jms.client.use-single-server-entry" 시스템 속성을 false로 설정하는 경우 커넥션과 물리적 연결이 1대 1 관계가 된다. JEUS MQ 장애 극복에 관한 자세한 사항은 [JEUS MQ 장애 극복](#)을 참고한다.

다음은 물리적 연결 공유와 관련된 JVM 옵션들이다.

- `-Djeus.jms.client.use-single-server-entry=<boolean value>`

ConnectionFactory 당 하나의 물리적 연결을 생성하고 커넥션끼리 공유할지 여부를 설정한다. 이 값을 false로 할 경우 커넥션과 물리적 연결은 1대 1 관계를 갖는다. (기본값: true)

클라이언트 코드에 다음과 같은 문장을 추가하면 런타임에도 설정 변경이 가능하다.

```
System.setProperty("jeus.jms.client.use-single-server-entry", <boolean value>);
```

또는

```
System.setProperty(jeus.jms.common.JeusJMSProperties.USE_SINGLE_SERVER_ENTRY, <boolean value>);
```

- `-Djeus.jms.client.single-server-entry.shutdown-delay=<long value>`

이 값을 공유하고 있는 물리적 연결을 사용하는 커넥션들이 0개일 때 물리적 연결을 언제 끊을지를 결정한다. 물리적 연결을 JVM이 종료할 때까지 유지하지 않고 idle 시간에는 리소스를 시스템에 반환할 수 있도록 한다. (기본값: 600000(10분), 단위: ms)

클라이언트 코드에 다음과 같은 문장을 추가하면 런타임에도 설정 변경이 가능하다.

```
System.setProperty("jeus.jms.client.single-server-entry.shutdown-delay", <long value>);
```

또는

```
System.setProperty(jeus.jms.common.JeusJMSProperties.SINGLE_SERVER_ENTRY_SHUTDOWN_DELAY, <long value>);
```



커넥션의 종료를 MessageListener의 onMessage 안에서 하는 것은 deadlock의 원인이 될수 있기 때문에 JMS 스펙에서 제한하고 있다.

### 2.3.3. 세션 생성

JMS 세션은 메시지를 생성하여 Destination에 송신하거나 Destination으로부터 메시지를 수신하는 것과 같은 모든 메시징 작업의 기본 단위가 된다. 또한 세션은 JMS 클라이언트가 로컬 및 XA 트랜잭션에 참여하는 단위이기도 하다.

세션과 세션을 거치는 모든 작업들은 기본적으로 단일 스레드 컨텍스트에 의해 처리되어야 하며 이는 세션 객체 자체가 멀티 스레드에 안전하지 않다는 것을 의미한다.



JEUS MQ 클라이언트는 동시에 여러 스레드가 하나의 세션을 사용하는 경우에 대해 안전한 동작을 보장하지 않는다. 세션을 사용하여 동시에 실행되어야 하는 스레드의 수만큼 세션을 생성하여 사용할 것을 권장한다. 덧붙여 여러 세션을 생성하는 것은 곧, 여러 스레드를 사용한다는 의미이기 때문에, JMS 스펙에서는 Jakarta EE Web이나 EJB Container 위에서 동작하는 클라이언트는 하나 이상의 세션을 생성하는 것을 제한하고 있다.

세션은 커넥션 객체로부터 다음 API를 사용하여 생성한다.

```
public Session createSession(boolean transacted, int acknowledgeMode)
    throws JMSException;
```

JMS 스펙은 다음의 4가지 ACKNOWLEDGE 모드를 정의하고 있다.

```
Session.AUTO_ACKNOWLEDGE = 1
Session.CLIENT_ACKNOWLEDGE = 2
Session.DUPS_OK_ACKNOWLEDGE = 3
Session.SESSION_TRANSACTED = 0
```

JEUS MQ는 메시지 송수신 성능을 극대화시켜야 하는 경우를 위해 추가적인 ACKNOWLEDGE 모드를 지원한다.

```
jeus.jms.JeusSession.NONE_ACKNOWLEDGE = -1
```



JMS 스펙에서는 로컬 트랜잭션이 Jakarta EE Web이나 EJB Container 위에서 동작하는 클라이언트에서 사용되는 것을 제한하고 있다. 따라서, 이러한 클라이언트에서는 트랜잭션

처리된 세션을 생성하는 것이 제한되어있다. 또한, ACKNOWLEDGE 모드 중 CLEINT\_ACKNOWLEDGE 모드를 사용하는 것이 제한되어 있다.

## 2.3.4. Client Facility Pooling

앞서 커넥션 공유에서 설명했듯이 일반적인 애플리케이션 사용 패턴에서는 Client Facility들을(Connection, Session, Message Producer) 반복적으로 생성하여 사용한다. 그런데 이런 객체들은 한 번 생성될 때마다 서버와 관리 메시지를 주고받게 되고 사용자 메시지를 한 번 보내기 위해서 실제로는 더 많은 수의 관리 메시지를 주고받게 되어 성능에 영향을 미친다.

이런 문제를 해결하기 위해서 JEUS MQ에서는 Client Facility들을 Pooling하는 기능을 제공한다. 이 기능은 트랜잭션이 아니거나 JEUS MQ 장애 극복이 설정되지 않은 환경에서 Message Producer를 사용하는 경우에만 적용되며 메시지 수신자나 트랜잭션이 사용될 때는 Pooling을 하지 않고 사용되었던 Client Facility들은 close할 때 바로 제거한다.

이 기능은 선택적으로 적용할 수 있으며 다음과 같은 클라이언트 JVM 옵션을 설정할 수 있다.

- `-Djeus.jms.client.use-pooled-connection-factory=<boolean value>`

Client Facility Pooling을 사용할지 여부를 설정한다. (기본값: true)

- `-Djeus.jms.client.pooled-connection.check-period=<long value>`

사용되지 않는 Pooling된 객체들을 삭제할 검사 주기를 설정한다. (기본값: 60000(1분), 단위: ms )

- `-Djeus.jms.client.pooled-connection.unused-timeout=<long value>`

현재 Pooling된 객체들 중에 사용되지 않은 시간이 이 시간보다 길면 제거한다. (기본값: 120000(2분), 단위: ms)

## 2.3.5. NONE\_ACKNOWLEDGE 모드

트랜잭션 처리된 세션을 제외하면 JMS의 기본 ACKNOWLEDGE 모드는 메시지를 수신하는 경우에만 의미가 있지만 NONE\_ACKNOWLEDGE 모드는 메시지 송신 동작에도 영향을 미친다.

이 ACKNOWLEDGE 모드를 사용하면 성능은 높아지지만 JMS의 중요한 특징 중 하나인 신뢰성있는 메시지의 송수신은 어렵게 된다. 그러므로 해당 세션을 통해 클라이언트가 다룰 메시지의 특성과 성능 및 신뢰도 요구 사항, 그리고 기타 주변 상황을 고려하여 NONE\_ACKNOWLEDGE 모드 사용 여부를 판단해야 한다.



FileMessage를 주고받거나 트랜잭션 처리된 세션 내에서는 NONE\_ACKNOWLEDGE 모드도 AUTO\_ACKNOWLEDGE 모드와 동일하게 동작한다.

## 메시지 송신 고려 사항

클라이언트가 송신한 메시지가 JMS 서버에 도착했다는 것을 보장하기 위해서는 서버로부터 어떤 응답이 올 때까지 클라이언트 스레드는 `MessageProducer.send()` 메소드를 호출한 시점에서 멈춰 있어야 한다.

반면에 `NONE_ACKNOWLEDGE` 모드를 사용하면 `MessageProducer.send()` 메소드가 서버로 JMS 메시지를 전송한 뒤 바로 리턴하므로 대기시간을 줄일 수 있어 메시지 송신 성능을 높일 수 있다.

다음은 일반적인 경우와 `NONE_ACKNOWLEDGE` 모드일 때 메시지 송신 방식의 차이를 보여준다. 회색 상자는 서버에 메시지가 도착했음을 기록으로 남기는 작업을 나타낸다.

다음과 같은 경우 메시지가 유실될 수 있다.

- 클라이언트에서 서버로 메시지가 전송되는 과정에서 네트워크 장애가 발생하는 경우
- JEUS MQ 서버에 도착한 메시지가 `Destination`에 추가되기 전에 JEUS MQ 서버가 장애를 일으키는 경우

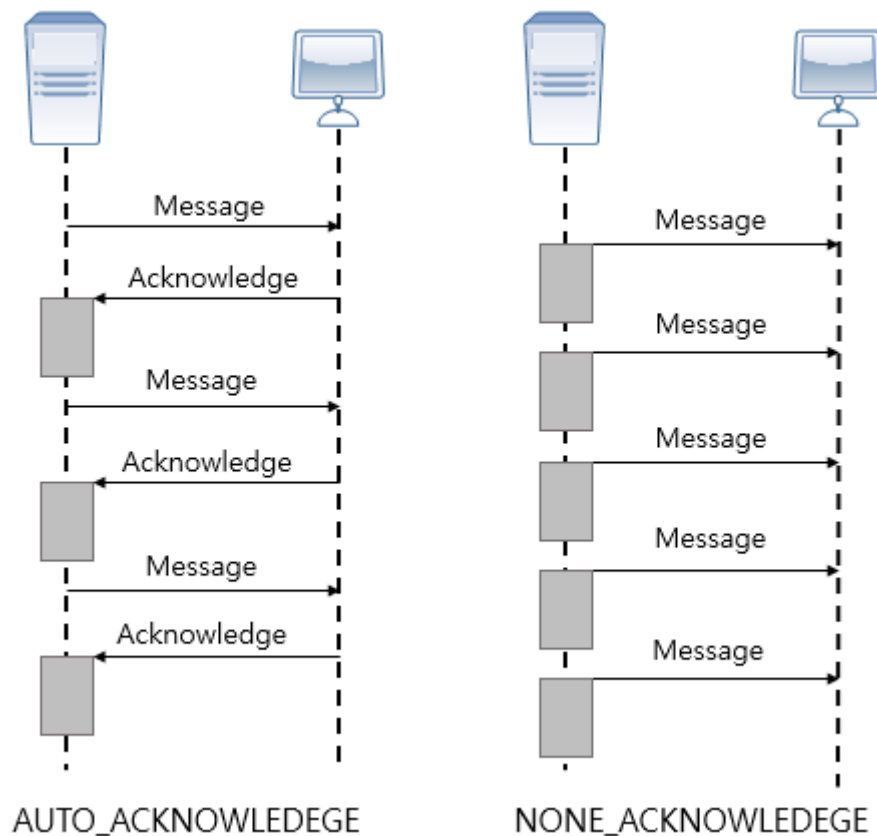
이렇게 유실된 메시지는 메시지 전달 방식을 `DeliveryMode.PERSISTENT`로 지정한 경우에도 복구되지 않는다.

## 메시지 수신 고려 사항

JMS에서는 메시지 전달을 보장하기 위해 메시지를 수신한 클라이언트로부터 메시지가 도착했다는 `Acknowledge`를 받을 때까지 서버에서 메시지 정보를 삭제하지 않는다. 이 방식은 메시징의 신뢰성은 높여주지만 네트워크 통신이 한 번 더 발생하고 JMS 서버가 메시지 상태를 복잡하게 관리해야 하므로 성능면에서는 좋지 않다.

세션의 `ACKNOWLEDGE` 모드를 `NONE_ACKNOWLEDGE` 모드로 설정하면 JEUS MQ 서버는 네트워크의 메시지를 수신하는 클라이언트로 JMS 메시지를 전송한 뒤 클라이언트의 `Acknowledge` 없이 서버에서 바로 메시지 정보를 삭제하므로 결과적으로 메시지 수신 속도를 더 빠르게 할 수 있다.

다음은 `AUTO_ACKNOWLEDGE` 모드일 때와 `NONE_ACKNOWLEDGE` 모드일 때 메시지 수신 방식의 차이를 보여준다. 회색 상자는 클라이언트에 전송한 메시지 정보를 서버에서 삭제하는 작업을 나타낸다.



AUTO\_ACKNOWLEDGE 모드와 NONE\_ACKNOWLEDGE 모드에서의 메시지 수신

다음과 같은 경우 메시지가 유실될 수 있다.

- 서버에서 클라이언트로 메시지가 전송되는 과정에서 네트워크 장애가 발생하는 경우
- JEUS MQ 클라이언트 라이브러리에서 메시지를 처리하는 도중 클라이언트가 오류를 일으키는 경우
- 클라이언트가 등록한 MessageListener 객체의 onMessage() 메소드 내에서 예외가 발생하는 경우

유실된 메시지는 Session.recover() 메소드를 호출해도 다시 수신할 수 없다.

## 2.3.6. JMSContext

JMS 스펙에서 좀더 손쉬운 사용을 위해 커넥션과 세션을 통합시킨 형태의 JMSContext라는 인터페이스를 제공하고 있다. JMSContext는 커넥션과 세션의 기능을 모두 하고 있으며, 커넥션과 비슷하게 ConnectionFactory에 다음과 같이 정의된 API로 생성한다.

```
public Connection createContext()
    throws JMSException;
public Connection createContext(int sessionMode)
    throws JMSException;
public Connection createContext(String userName, String password)
    throws JMSException;
public Connection createConnection(String userName, String password, int sessionMode)
    throws JMSException;
```

세션을 생성할 때와 마찬가지로 ACKNOWLEDGE 모드를 설정하고 있는데, 세션 생성과는 달리 트랜잭션 여부는 별도의 파라미터가 아닌 ACKNOWLEDGE 모드에 의해 결정된다. JEUS MQ에서는 JMSContext 또한 커넥션과

세션처럼 Client Facility Pooling에 포함되며, 각각 커넥션과 세션의 pool을 공유해 사용한다. 따라서, 이에 관한 세부 설정은 커넥션과 세션의 pooling을 따른다.

## 2.4. 메시지

본 절에서는 JEUS MQ가 JMS 스펙 이외의 부가 기능을 제공하기 위해 JMS 메시지를 확장한 부분에 대해 설명한다.

### 2.4.1. 메시지 헤더 필드

JMS는 다음과 같은 메시지 헤더 필드들을 정의하고 있다.

- JMSDestination
- JMSDeliveryMode
- JMSMessageID
- JMSTimestamp
- JMSCorrelationID
- JMSReplyTo
- JMSRedelivered
- JMSType
- JMSExpiration
- JMSPriority

JEUS MQ는 메시지에 유일한 메시지 ID를 부여하므로 다음과 같은 기능은 제공하지 않는다.

- MessageProducer.setDisableMessageID(boolean) 메소드를 사용하여 JMS 메시지에 메시지 ID를 할당하지 않는 기능
- MessageProducer.setDisableTimestamp(boolean) 메소드를 사용하여 JMS 메시지에 Timestamp를 부여하지 않는 기능
- 관리자 설정에 의해 클라이언트에서 설정한 JMSDeliveryMode, JMSExpiration, JMSPriority 필드값을 덮어쓰는 기능

승인 모드(Acknowledge Mode)를 NONE\_ACKNOWLEDGE 모드로 설정한 세션을 통해 메시지를 송신한 뒤 Message.getJMSMessageID()를 호출하면 메시지 ID가 null로 나타난다. 이는 MessageProducer.send() 메소드가 서버로부터 응답을 받기 전에 리턴하기 때문이다. 자세한 내용은 [NONE\\_ACKNOWLEDGE 모드](#)를 참고한다.

### 2.4.2. 메시지 프로퍼티

JMS는 다음과 같이 "JMSX?"로 시작하는 프로퍼티 이름을 정의하고 있다.

- JMSXUserID

- JMSXAppID
- JMSXDeliveryCount
- JMSXGroupID
- JMSXGroupSeq
- JMSXProducerTXID
- JMSXConsumerTXID
- JMSXRcvTimestamp
- JMSXState



"JMSX?" 메시지 프로퍼티에 대한 지원은 JMSXDeliveryCount를 제외하고 필수 사항이 아니며, JEUS MQ에서는 이 프로퍼티들을 사용하지 않는다.

다음은 JEUS MQ가 지원하는 전용 메시지 프로퍼티이다.

- JMS\_JEUS\_Schedule

JMS에서 정의하는 Message Delivery Delay와 동일한 기능의 프로퍼티로 JEUS MQ 서버가 메시지 수신자에게 메시지 전달을 미루는 시간이다. JEUS MQ 서버는 송신된 메시지가 서버에 도착한 시각(JMSTimestamp 필드값)으로부터 이 시간만큼이 지나기 전에는 메시지 수신자에게 메시지를 전달하지 않는다. 설정값은 long 형태로 설정한다. (단위: ms)

```
Message.setLongProperty("JMS_JEUS_Schedule", <long value>);
```



Message Delivery Delay와 함께 설정된 경우 Message Delivery Delay의 값이 우선시 된다.

- JMS\_JEUS\_Compaction

메시지 바디를 압축해서 전송할지 여부를 설정한다. true로 설정하면 네트워크를 통해 메시지를 전송할 때 ZLIB 라이브러리를 이용하여 메시지의 바디 부분을 압축한다.

```
Message.setBooleanProperty("JMS_JEUS_Compaction", <boolean value>);
```

- JMS\_JEUS\_RedeliveryLimit

메시지를 수신자에 재전송할 수 있는 최대 횟수이다. 이 횟수를 넘기면 메시지는 더 이상 재전송되지 않고 비정상 메시지 Destination(비정상 메시지 Destination)에 저장된다.

```
Message.setIntProperty("JMS_JEUS_RedeliveryLimit", <integer value>);
```



### 2.4.3. 메시지 바디

JMS 스펙은 메시지 바디의 형태에 따라 5가지 타입의 메시지를 정의하고 있다.

- StreamMessage
- MapMessage
- TextMessage
- ObjectMessage
- BytesMessage

각 타입의 메시지는 다음 API를 사용해서 세션 객체로부터 생성할 수 있다.

```
public StreamMessage createStreamMessage()
    throws JMSException;
public MapMessage createMapMessage()
    throws JMSException;
public TextMessage createTextMessage()
    throws JMSException;
public TextMessage createTextMessage(String text)
    throws JMSException;
public ObjectMessage createObjectMessage()
    throws JMSException;
public ObjectMessage createObjectMessage(Serializable object)
    throws JMSException;
public BytesMessage createBytesMessage()
    throws JMSException;
```

### 2.4.4. FileMessage

JEUS MQ는 JMS 기본 메시지 타입 외에 FileMessage를 추가로 지원한다. JMS는 메시지 기반으로 동작하기 때문에 메모리에 메시지 내용이 모두 올라와 있어야만 메시지 송수신이 가능하다. 이는 메시지의 크기가 매우 클 경우 클라이언트나 서버에서 메모리 Overflow가 발생할 수 있으므로 문제가 된다. JEUS MQ가 제공하는 FileMessage를 사용하면 파일의 내용을 블록 단위로 전송하기 때문에 이와 같은 문제를 피할 수 있다.

#### 메시지 생성

jeus.jms.JeusSession 클래스에 정의되어 있는 다음 메소드를 사용해 FileMessage를 생성한다.

```
public jeus.jms.FileMessage createFileMessage()
    throws jakarta.jms.JMSException;
public jeus.jms.FileMessage createFileMessage(java.net.URL url)
    throws jakarta.jms.JMSException;
```

JEUS MQ 클라이언트 라이브러리를 통해 생성한 Session, QueueSession, TopicSession 객체는 각각 jeus.jms.JeusSession, jeus.jms.JeusQueueSession, jeus.jms.JeusTopicSession으로 캐스팅할 수 있다.

## FileMessage 인터페이스

jeus.jms.FileMessage 인터페이스 정의는 다음과 같다.

```
public interface FileMessage extends jakarta.jms.Message {
    public java.net.URL getURL();
    public void setURL(java.net.URL url)
        throws jakarta.jms.MessageNotWriteableException;
    public boolean isURLOnly();
    public void setURLOnly(boolean urlOnly);
}
```

메시지를 송신하기 전에 setURL() 메소드로 전송할 파일의 URL을 지정할 수 있다. 또는 메시지를 생성할 때 JeusSession.createFileMessage() 메소드에 파라미터로 넘겨줘도 된다. urlOnly 속성은 메시지 수신자에게 서버에 위치한 파일의 URL만 전송할지 여부를 결정한다. 이 속성값에 따라 수신한 FileMessage에 대해 getURL() 메소드를 호출했을 때 얻는 URL이 다르다.

다음은 urlOnly 속성값에 따른 getURL() 값의 의미에 대한 설명이다.

urlOnly	getURL()
true	JEUS MQ 서버에 위치한 파일의 URL이다. 이 URL을 사용하여 HTTP, FTP와 같은 별도의 프로토콜을 통해 파일 수신이 가능하다.
false	JEUS MQ 클라이언트 라이브러리가 파일의 내용까지 전송받아 로컬에 저장한 임시 파일의 URL이다. 자세한 내용은 <a href="#">임시 파일 저장 경로</a> 를 참고한다.

FileMessage를 송신할 때 MessageProducer.send() 메소드는 항상(세션의 승인 모드(Acknowledge Mode)를 NONE\_ACKNOWLEDGE로 사용하는 경우에도) 서버에 파일 내용이 모두 전송된 뒤에 리턴한다.



FileMessage에 포함된 파일은 기본적으로 4KB 단위로 나누어 전송된다. 이 블록 사이즈는 JEUS MQ 클라이언트를 실행할 때 "-Djeus.jms.file.blocksize=<integer value>"와 같이 시스템 프로퍼티를 설정해서 변경할 수 있다.

## 임시 파일 저장 경로

메시지 수신자가 FileMessage를 수신할 때 파일을 같이 전송받는 경우 전송받은 파일은 임시 파일 경로에 저장된다. 임시 파일 저장 경로는 다음 조건을 검사해서 결정한다.

- JEUS에 deploy되어 있는 애플리케이션인 경우

```
SERVER_HOME/.workspace/client/
```

- jeus.jms.client.workdir 시스템 프로퍼티가 설정되어 있는 경우

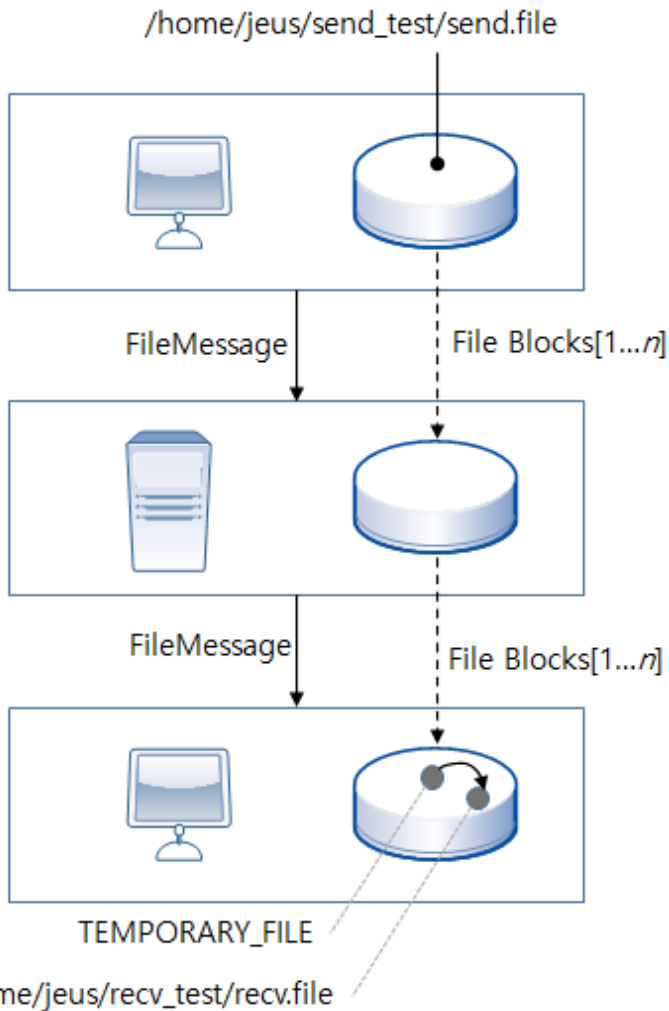
```
시스템 프로퍼티로 지정한 경로
```

- 그 밖의 경우

USER\_HOME/.jeusmq\_client\_work/

## 파일 메시지 전송 예

FileMessage를 전송하는 간단한 예를 통해 FileMessage API를 사용하는 과정을 설명한다.



FileMessage 전송 예

다음의 Java 코드는 `/home/jeus/send_test/send.file`이라는 파일을 FileMessage를 사용하여 송신하는 예제이다.

FileMessage 송신

```
...
jeus.jms.JeusSession session = (jeus.jms.JeusSession)
    connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
MessageProducer producer = session.createProducer(destination);

jeus.jms.FileMessage message = session.createFileMessage();
File file = new File("/home/jeus/send_test/send.file");
message.setURL(file.toURI().toURL());
```

```
producer.send(message);
```

```
...
```

다음은 수신한 파일의 URL로부터 InputStream을 얻어 파일 내용을 "/home/jeus/recv\_test/recv.file"이라는 이름의 파일에 쓰는 예제이다.

FileMessage 수신

```
...
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
MessageConsumer consumer = session.createConsumer(destination);
Message message = consumer.receive();

if (message instanceof jeus.jms.FileMessage) {
    URL url = ((jeus.jms.FileMessage) message).getURL();
    if (url != null) {
        InputStream inputStream = url.openStream();
        BufferedInputStream bufInputStream = new BufferedInputStream(inputStream);

        File outFile = new File("/home/jeus/recv_test/recv.file");
        FileOutputStream fileOutputStream = new FileOutputStream(outFile);
        BufferedOutputStream bufOutputStream = new BufferedOutputStream(fileOutputStream);

        int buf;
        while ((buf = bufInputStream.read()) != -1) {
            bufOutputStream.write(buf);
        }
        bufOutputStream.close();
        bufInputStream.close();
    }
}
...
```

## 2.5. 트랜잭션

본 절에서는 JEUS MQ가 지원하는 로컬 트랜잭션, 분산(XA) 트랜잭션에서 각각의 특성과 트랜잭션 적용 범위, 그리고 트랜잭션 처리에 있어서 클라이언트 개발자가 알아야 할 내용을 설명한다.

JMS 트랜잭션은 하나의 세션 내에서 메시지를 송신하고 수신하는 작업을 포함한다. JMS 스펙은 세션 내에서 시작, 완료되는 로컬 트랜잭션과 하나 이상의 JMS 세션 및 EJB나 JDBC 등의 기타 트랜잭션 리소스의 작업을 포함하는 분산 트랜잭션을 정의하고 있다.

트랜잭션에 참여하는 세션을 통해서 송신한 메시지는 실제 서버에 도착한 것으로 간주되지 않기 때문에 동일한 세션에서 생성한 메시지 수신자에 전달되지 않는다. 또한 동일한 세션을 이용해서 생성된 큐 브라우저를 통해서도 확인할 수 없다.

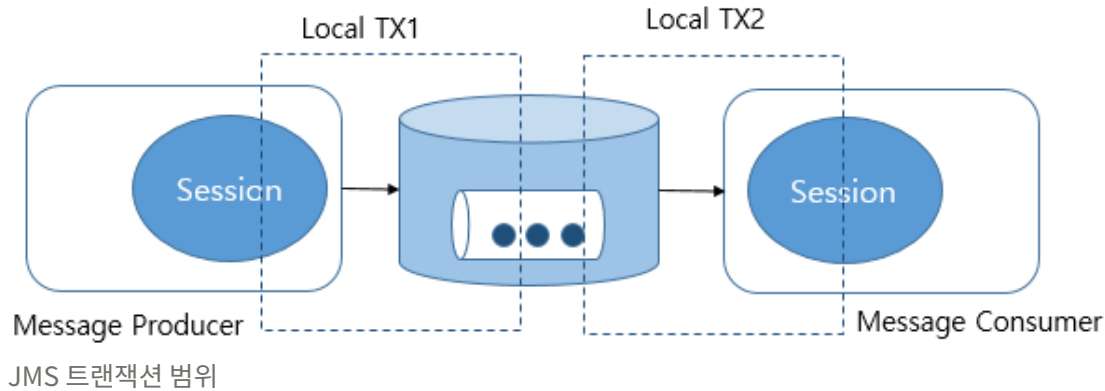
### 2.5.1. 로컬 트랜잭션

로컬 트랜잭션은 Connection.createSession(boolean transacted, int acknowledgeMode) 메소드에

transacted 파라미터 값을 true로 설정하고 생성한 세션에 의해 수행되며 이전에 commit이나 rollback이 수행된 시점 또는 처음 세션이 생성된 시점 이후로 해당 세션을 통해 이루어진 모든 메시징 작업들을 포함한다. 즉, 모든 작업은 어느 하나의 트랜잭션에 속하게 된다는 것을 의미한다.

하나의 로컬 트랜잭션 내에서 여러 개의 세션을 처리하는 것은 불가능하지만 하나 이상의 메시지 송신자를 생성하여 다수의 Destination에 메시지를 송신하는 것은 가능하다. 마찬가지로 다수의 Destination으로부터 메시지를 수신할 수 있다.

다음은 로컬 트랜잭션에 참여하는 작업들과 그 범위를 나타낸 것이다.



로컬 트랜잭션의 완료는 세션의 commit(), rollback() API를 사용해서 이루어진다. 트랜잭션 처리된 세션은 항상 특정 트랜잭션에 참가하고 있으므로 명시적으로 트랜잭션을 시작하는 API는 존재하지 않는다.

로컬 트랜잭션은 분산 트랜잭션과 달리 트랜잭션의 commit과 rollback이 클라이언트에 의해 직접 일어날 수 있으므로 비동기적인 메시지의 처리를 트랜잭션 내에서 하는 것이 가능하다.



JMS 스펙에서는 로컬 트랜잭션이 Jakarta EE Web이나 EJB Container 위에서 동작하는 클라이언트에서 사용되는 것을 제한하고 있다. 따라서, 이러한 클라이언트에서는 트랜잭션 처리된 세션을 생성하는 것이 제한되어있다.

## 2.5.2. 분산 트랜잭션

JMS 스펙은 XASession이 제공하는 XAResource를 트랜잭션에 등록함으로써 XA 트랜잭션에 참여할 수 있다. 이는 JMS의 부가적인 스펙으로서 실제 구현 방식은 각 벤더에 따라 다르다.

JEUS MQ 클라이언트에서 XASession을 사용할 때에는 다음 사항에 주의해야 한다.

- XASession에 대해 Session.getAcknowledgeMode()를 호출하면 이 XASession이 글로벌 트랜잭션에 참여하고 있는 경우 Session.SESSION\_TRANSACTED를 리턴하고, 그렇지 않은 경우 Session.AUTO\_ACKNOWLEDGE를 리턴한다.
- 글로벌 트랜잭션에 참여하고 있는 XASession에 대해 Session.commit()이나 Session.rollback()을 호출하면 TransactionInProgressException이나 IllegalStateException이 발생한다.

### 분산 트랜잭션의 전파

JEUS MQ 클라이언트 라이브러리는 XASession의 생성 시점과 무관하게 JMS API를 이용하는 스레드의 글로벌

트랜잭션에 XAResource를 등록시킨다. 이렇게 클라이언트 스레드의 트랜잭션이 전파되기 위해서는 명시적인 API 호출이 필요하다. 따라서 JEUS MQ의 분산 트랜잭션 참가는 **동기적 형태의 API 호출을 통해서만** 이루어진다. 메시지를 송신하거나 동기적으로 수신하는 경우가 이에 해당하며 MessageListener를 통해 비동기적인 메시지 수신을 분산 트랜잭션에 포함시켜서는 안 된다.



분산 트랜잭션 내에서 비동기적으로 수신한 메시지를 처리하고 싶다면 MDB를 사용할 수 있다. 자세한 내용은 JEUS EJB 안내서의 "Message Driven Bean(MDB)"을 참고한다.

## 분산 트랜잭션의 복구

JEUS MQ 서버는 트랜잭션 내에서 진행 중인 세션의 작업들을 스토리지에 보관하며 불의의 사고로 인한 서버의 재기동에도 해당 트랜잭션의 작업을 복구할 수 있다. 트랜잭션 매니저는 XASession에서 얻어진 XAResource를 통해서 JEUS MQ 내에 진행 중인 트랜잭션 ID들을 얻을 수 있으며, 이를 이용하여 해당 트랜잭션을 commit하거나 rollback할 수 있다.

트랜잭션 중에 장애가 발생할 경우 좀 더 빨리 장애를 극복하기 위해서는 JEUS MQ 장애 극복 기능을 사용할 것을 권장한다. 자세한 내용은 [JEUS MQ 장애 극복](#)을 참고한다.

JEUS MQ 서버는 In-Doubt 상태가 아닌 트랜잭션의 작업은 기동할 때 자동적으로 rollback 처리를 한다.

## 3. JEUS MQ 서버 설정

본 장에서는 JEUS MQ 서버를 실행하기 위한 JMS 엔진과 리소스 설정 방법에 대해서 설명한다.

### 3.1. 개요

JEUS MQ 서버는 JMS 엔진으로 표현되며, 서버 당 하나의 JMS 엔진이 실행될 수 있다. JEUS의 각 구성 요소에 대한 설명은 JEUS Server 안내서의 "소개"를 참고한다.

#### 3.1.1. 디렉터리 구조

다음은 JEUS 및 JEUS MQ 서버를 설정하고 관리하기 위한 파일들의 위치를 보여준다. JEUS\_HOME은 JEUS가 설치된 경로를 나타낸다.

```
{JEUS_HOME}
|--bin
|   |--[01]jeusadmin
|--logs

* Legend
- [01]: binary or executable file
- [X] : XML document
- [J] : JAR file
- [T] : Text file
- [C] : Class file
- [V] : java source file
- [DD] : deployment descriptor
```

#### jeusadmin

JEUS Manager를 관리하기 위한 콘솔 툴이다. JEUS Manager와 서버 제어를 포함해 JEUS 전반에 대한 관리 기능을 제공한다. 자세한 사용법은 JEUS Reference 안내서의 "jeusadmin"을 참고한다.

## 3.2. JMS 리소스 설정

Jms Resource은 다음의 2개 요소로 구성된다.

#### • Destination

JEUS MQ 서버 내의 Destination에 관한 설정이다. Queue나 Topic과 같은 Destination 역시 JEUS MQ 서버가 기동할 때 도메인 설정을 읽어 JEUS JNDI 서비스에 등록한다. 자세한 설정 방법은 [Destination 설정](#)을 참고한다.

#### • Durable Subscription

JEUS MQ 서버 내의 Durable Subscription에 관한 설정이다. 자세한 설정 방법은 [Durable Subscription](#)

설정을 참고한다.

### 3.2.1. Destination 설정

Queue나 Topic과 같은 Destination 역시 JEUS MQ 서버가 기동할 때 도메인 설정을 읽어 JEUS JNDI 서비스에 등록한다.

#### 기본 설정

다음은 domain.xml에서 Destination을 기본적으로 설정하는 예이다. 설정 값은 <destination> 태그 안에 구성한다.

domain.xml

```
<domain>
  ...
  <jms-resource>
    <destination>
      <type>queue</type>
      <name>ExamplesQueue</name>
      <export-name>jms/ExamplesQueue</export-name>
    </destination>

    <destination>
      <type>topic</type>
      <name>ExamplesTopic</name>
      <export-name>jms/ExamplesTopic</export-name>
    </destination>
  </jms-resource>
</domain>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<type>	Destination의 타입이다. queue와 topic 중 하나를 지정할 수 있다.
<export-name>	Destination은 Connection Factory와 마찬가지로 <export-name>으로 JNDI Lookup 할 수 있다. <export-name>을 지정하지 않으면 <name> 값이 사용된다.

#### 메시지 흐름 제어 설정

메시지 수신자가 Destination에 Message Listener를 등록한 경우 JEUS MQ 서버는 Destination에 메시지가 도착하면 해당 클라이언트에게 메시지를 전송한다. 그런데 서버가 메시지를 전송하는 속도보다 클라이언트에서 메시지를 처리하는 속도가 느리면 클라이언트 메모리에 메시지가 계속해서 쌓이게 되고, 결국 클라이언트 JVM에서 OutOfMemoryError가 발생할 수 있다. 이와 같은 상황을 방지하기 위해 클라이언트 측에 아직 처리되지 않고 쌓여 있는 메시지의 최대 개수를 설정하여 이 개수를 초과하면 서버가 클라이언트로 잠시 동안 메시지를 전송하지 않도록 할 수 있다.

다음은 domain.xml에서 Destination별로 메시지 흐름 제어 설정을 정의한 예이다. 설정 값은 <destination> 태그



안에 구성한다.

domain.xml

```
<domain>
...
  <jms-resource>
    <destination>
      <type>queue</type>
      <name>ExamplesQueue</name>
      <export-name>jms/ExamplesQueue</export-name>
      <max-pending-limit>1000</max-pending-limit>
      <resume-dispatch-factor>0.5</resume-dispatch-factor>
    </destination>
  </jms-resource>
</domain>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<max-pending-limit>	메시지의 최대 개수이다. (기본값: 8192)
<resume-dispatch-factor>	<p>&lt;max-pending-limit&gt; 설정에 따라 JEUS MQ 서버가 클라이언트로 메시지 전송을 멈춘 경우 클라이언트에 쌓인 메시지 개수가 &lt;max-pending-limit&gt; 값의 어느 비율까지 줄어들었을 때 메시지 전송을 재개할지 지정한다. (기본값: 0.4)</p> <p>위의 예시와 같이 설정한 경우에는 클라이언트에 쌓인 메시지가 1,000개일 때 서버가 메시지 전송을 멈추고, 메시지 개수가 500개로 줄어들면 메시지 전송을 재개한다.</p>

### 3.2.2. Durable Subscription 설정

일반적으로 Durable Subscriber는 클라이언트가 생성하기 때문에 생성 이전에 Topic에 도착한 메시지는 클라이언트에게 전달되지 않는다. 이런 상황을 방지하기 위해 JEUS MQ는 Durable Subscription을 설정하여 서버가 실행될 때 미리 Durable Subscription을 등록해두었다가 조건에 맞는 클라이언트가 접속했을 때 Durable Subscriber로 메시지를 전달하는 기능을 제공한다. Session.createDurableSubscriber() 메소드를 호출할 때 클라이언트 ID와 Subscription 이름 및 Topic이 필요한 것처럼 domain.xml에서 Durable Subscription을 설정할 때도 동일한 항목을 설정해야 한다. 추가로 Message Selector도 설정할 수 있다.

domain.xml

```
<domain>
...
  <jms-resource>
    <durable-subscription>
      <client-id>client_id1</client-id>
      <name>subscription1</name>
      <destination-name>ExamplesTopic</destination-name>
    </durable-subscription>
  </jms-resource>
```

## 3.3. JMS Quota 설정

### 3.3.1. Quota 설정

Destination에 클라이언트에게 전달되지 않은 메시지가 계속해서 늘어나면 JVM이 OutOfMemoryError를 내면서 서버가 종료될 수 있다. 이를 방지하기 위해 JEUS MQ는 Quota 설정을 통해 JMS Destination별 메모리 관리를 위한 Quota를 설정 할 수 있는 방법을 제공한다.

Destination이 많은 메모리를 사용하고 있지 않을 때 JEUS MQ 서버는 메시지 내용에 대한 Strong Reference를 가지고 있다. JMS Destination 내에 '**Quota**'를 지정하면 메모리 사용량에 따라 JEUS MQ 서버는 다음과 같이 동작한다.

- Destination이 사용하고 있는 메모리가 '**Quota**' 설정값 이상이면 클라이언트의 메시지 송신 시도는 실패하고 JMSException이 발생한다.
- Destination이 사용하고 있는 메모리가 '**Quota**' 설정값의 75%이상이면 저장소에 저장한 메시지의 내용은 메모리에서 관리하지 않는다.

Destination에 도착한 메시지의 내용이 메모리에서 삭제되어도 메시지는 유실되지 않는다. 메시지는 저장소에 저장되어 있기 때문에 JEUS MQ 서버는 필요할 때 메시지 내용을 저장소로부터 읽어서 처리한다. 이 경우 처리 속도는 메시지 내용이 메모리에 있을 때보다 느릴 수 밖에 없다.



JEUS MQ 서버는 저장소에 저장하지 않은 메시지 내용을 항상 메모리에 가지고 있으므로, '**Quota**'에 의한 설정은 저장소가 설정되어 있고 메시지를 DeliveryMode.PERSISTENT로 지정한 경우에만 영향을 미친다.

Quota를 설정한 후에 Destination에서 Quota 기능을 사용하도록 설정할 수 있다. 자세한 내용은 [Destination 설정](#)을 참고한다.

## 3.4. JMS 엔진 설정

domain.xml 파일을 작성하는 것으로 JMS 엔진 설정을 할 수 있다.

### 3.4.1. 기본 정보 설정

JEUS MQ 서버는 <jms-engine>에서 설정할 수 있다. JMS 엔진은 해당 서버에서 JMS 서버를 사용하기 위한 환경을 제공한다. 서버가 부팅될 때 실행되며, 하나의 서버에서는 하나의 JMS 엔진만 지원한다.

JMS 엔진은 Engine Roll 등에 해당하는 기본 선택사항 및 JEUS MQ 서버의 Failover, Quota와 JMS 엔진의 고급 선택사항까지 설정할 수 있다. 상단의 Failover 설정에서는 서버나 네트워크에 장애가 발생하는 경우 이를 복구하는 정보를 설정한다. Failover 설정에 대한 자세한 내용은 [JEUS MQ 장애 극복](#)을 참고한다.

## • 엔진 메모리 관리

JMS 엔진 기본 선택사항의 '**Max Bytes**', '**Max Messages**' 항목은 JMS 엔진의 메모리 관리를 위한 Quota 정보를 설정한다. JEUS MQ 서버는 서버 내 모든 Destination의 메모리를 통괄하는 JMS 엔진의 Quota 설정과 각 Destination의 Quota 설정 정보를 사용하여 메모리를 관리한다. 메모리 관리 정책에 대한 자세한 내용은 [JMS Quota 설정](#)을 참고한다.

다음은 엔진 메모리와 관련된 부분을 서버에 설정한 예시이다.

domain.xml

```
<domain>
...
  <jms-engine>
    <max-byte>128M</max-byte>
    <max-message>0</max-message>
  </jms-engine>
</domain>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<max-byte>	JMS Engine에서 사용할 수 있는 최대 메모리 크기를 설정한다. 숫자 뒤에 'K'(KiloBytes), 'M'(MegaBytes), 'G'(GigaBytes)를 붙여 설정할 수 있다. (기본값: 128MBytes)
<max-message>	JMS Engine에서 사용할 수 있는 최대 메시지 수를 설정한다. 설정하지 않으면 메시지 수를 제한하지 않는다.

## • 고급 선택사항 설정

JMS 엔진 고급 선택사항에서는 JEUS MQ 서버가 사용하는 Thread에 대한 정보를 설정한다.

JMS 엔진의 메시지 처리 방식은 크게 두 가지로 나뉜다. 먼저, 일반적인 메시지는 순서 보장과 동시처리에 따른 경쟁을 피하기 위해 Event manager에 의해 Single Thread로 처리된다. 그외의 특수 기능, 클러스터간 메시지 전송, 트랜잭션 제어, JMS 엔진 제어를 위한 메시지 등은 각 형태에 따라 Thread pool의 Thread로 처리하고 있다.

다음은 Thread에 대한 정보를 서버에 설정한 예시이다.

domain.xml

```
<domain>
...
  <jms-engine>
    <thread-pool>
      <min>10</min>
      <max>20</max>
      <keep-alive-time>300</keep-alive-time>
    </thread-pool>
  </jms-engine>
</domain>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<min>	Thread pool의 최소 크기를 지정한다.
<max>	Thread pool의 최대 크기를 지정한다.
<keep-alive-time>	min 설정 개수를 초과하는 Thread들 중에서 여기에 지정된 시간 동안 사용되지 않은 것은 소멸된다. (단위: 초, 기본값: 300)

### 3.4.2. 서비스 채널 설정

JEUS MQ 서버는 서비스 채널(Service Channel)을 통해서 클라이언트와 통신한다. 하나의 JEUS MQ 서버에는 최소한 하나의 서비스 채널이 있어야 하며 각 서비스 채널마다 서비스 URL 등의 네트워크 설정을 다르게 지정할 수 있다.

서비스 채널이 사용하는 리스너의 이름은 <service-config> 하위의 <listener-name>에서 설정한다. 리스너 설정에 대한 자세한 설명은 JEUS Server 안내서의 "Listener 설정"을 참고한다.

다음은 서비스 채널을 서버에 설정한 예시이다.

domain.xml

```
<domain>
...
  <jms-engine>
    <service-config>
      <name>default</name>
      <listener-name>jms</listener-name>
      <client-limit>1000</client-limit>
      <client-keepalive-timeout>20</client-keepalive-timeout>
    </service-config>
  </jms-engine>
</domain>
```

다음은 설정 태그에 대한 설명이다. 메시징 서비스를 제공하기 위한 서비스 채널에 대한 설정으로 최소한 하나 이상 설정되어야 한다.

태그	설명
<name>	서비스 채널의 이름을 설정한다. Connection Factory에 채널 정보를 지정하기 위해 설정한다.
<listener-name>	서비스 채널의 Listener를 지정한다. 서버에 이미 존재하는 설정들에 있는 이름을 설정한다. 설정하지 않으면 Base-listener가 선택된다.
<client-limit>	서비스 채널이 허용하는 최대 클라이언트 수를 지정한다.

태그	설명
<client-keepalive-timeout>	클라이언트와의 연결이 비정상 종료되었을 경우 재접속을 기다리는 시간이다. 설정한 시간이 지나면 해당 클라이언트의 리소스는 모두 서버에 반환된다. 설정된 시간 내에는 해당 클라이언트의 clientID 값이 유지되므로 네트워크 상태가 불량한 경우에만 설정한다. 초 단위로 설정하며 0 이하의 값을 지정하면 기다리지 않고 즉시 리소스를 반환한다.

### 3.4.3. Connection Factory 설정

Connection Factory는 JMS 관리 객체로 클라이언트가 JMS 서버에 접속하는 데 필요한 연결 설정 정보와 클라이언트를 위한 기본 정보들을 가지고 있다. JEUS MQ 서버가 기동될 때 생성되어 JEUS JNDI 서비스에 등록된다.

다음은 Connection Factory를 서버에 설정한 예시이다.

domain.xml

```
<domain>
  ...
  <jms-engine>
    <connection-factory>
      <type>nonxa</type>
      <name>ConnectionFactory</name>
      <export-name>jms/ConnectionFactory</export-name>
    </connection-factory>
  </jms-engine>
</domain>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<type>	해당 Connection Factory의 종류를 설정한다.
<name>	JMS 시스템 내에서 관리의 목적으로 사용되는 Connection Factory의 이름이다.
<export-name>	해당 Connection Factory가 Naming Server에 바인딩되는 이름을 설정한다. 설정하지 않으면 Name 속성이 그대로 사용된다.

### 3.4.4. Persistence Store 설정

Persistence Store는 서버가 재기동될 때 메시지나 Subscription, 트랜잭션들의 이전 상태를 복구하기 위해 필요하다. Persistence Store를 설정하지 않으면 클라이언트에서 DeliveryMode.PERSISTENT 방식으로 메시지를 송신한 경우에 서버에서 장애가 발생했을 때 메시지 전달을 보장할 수 없다. JEUS MQ에서 제공하는 Persistence Store의 종류에는 저널 로그와 데이터베이스가 있다.

Persistence Store 설정은 <jms-engine> 하위의 <persistence-store>에서 할 수 있다.

다음은 Persistence Store를 서버에 설정한 예시이다.

```

<domain>
  ...
  <jms-engine>
    <persistence-store>
      <journal>
        <base-dir>/home/example/store/jeusmq</base-dir>
        <!--<initial-log-file-count>5</initial-log-file-count>-->
        <!--<max-log-file-count>10</max-log-file-count>-->
        <!--<log-file-size>128M</log-file-size>-->
        <!--<property>-->
          <!--<key>jeus.store.journal.overflow-factor</key>-->
          <!--<value>0.3</value>-->
        <!--</property>-->
      </journal>
    <!--<jdbc>-->
      <!--<data-source>datasource1</data-source>-->
      <!--<vendor>oracle</vendor>-->
      <!--<destination-table>TEST_DEST</destination-table>-->
      <!--<durable-subscriber-table>TEST_DSUB</durable-subscriber-table>-->
      <!--<message-table>TEST_MSG</message-table>-->
      <!--<subscription-message-table>TEST_SMSG</subscription-message-table>-->
      <!--<transaction-table>TEST_TRAN</transaction-table>-->
    <!--</jdbc>-->
  </persistence-store>
</jms-engine>
</domain>

```

다음은 **<persistence-store>**의 Store 설정과 관련된 하위 태그에 대한 설명이다.

- **<journal>**

저널 로그 방식을 사용한다.

태그	설명
<base-dir>	Store를 생성할 디렉터리 이름을 설정한다. 이 디렉터리 이름은 각 Store별로 유일해야 한다.
<initial-log-file-count>	Journal Store를 생성할 때 초기에 생성할 로그 파일의 개수를 설정한다.
<max-log-file-count>	최대로 생성할 로그 파일의 개수를 설정한다.
<log-file-size>	로그 파일의 크기를 지정한다. Integer 타입의 값이나 숫자 뒤에 'K'(KiloBytes), 'M'(MegaBytes), 'G'(GigaBytes)를 붙여 설정할 수 있다.

- **<jdbc>**

데이터베이스 방식을 사용한다. 현재 호환 가능한 외부 데이터베이스는 'Oracle Database 9i 이상(Enterprise Edition)', 'Tibero 3 SP2 이상', 'Altibase 4.x(5 이상은 지원하지 않음)'이다.

**'Data Source'** 항목에 설정된 값은 Persistence Store로 사용할 데이터소스의 JNDI 이름이다. JEUS에 데이터소스를 추가하는 방법은 "DB Connection Pool과 JDBC"를 참고한다.

위와 같이 설정하는 경우 해당 데이터베이스 내의 테이블 이름을 변경할 수 있다. 이 태그들을 사용하면 운용

중인 서버에서 테스트를 수행하는 경우 별도의 Persistence Store를 설치하지 않고도 사용하는 테이블의 이름만 바꿔서 JEUS MQ 서버를 테스트하는 것이 가능하다. 테이블의 이름을 설정하지 않는 경우 기본적으로 `<SERVER-NAME>_DEST`, `<SERVER-NAME>_DSUB`, `<SERVER-NAME>_MESG`, `<SERVER-NAME>_SMSG`, `<SERVER-NAME>_TRAN`이라는 이름의 테이블을 사용하게 된다. JDBC를 설정했을 때 생성되는 테이블들의 각 컬럼에 대한 설명은 [JDBC Persistence Store 컬럼](#)을 참고한다.

태그	설명
<code>&lt;data-source&gt;</code>	DB의 데이터소스를 설정한다.
<code>&lt;destination-table&gt;</code>	Destination 테이블 이름을 변경한다. 단, 영문 소문자는 허용되지 않는다.

### 3.4.5. Message Sort 설정

Message Sort 기능에 대한 설정은 [JEUS MQ Message Sort](#)의 설명을 참고한다.

## 3.5. 서버 관리 및 모니터링

JEUS MQ 서버 관리는 다음과 같은 작업을 통해 메시지 유실 없이 지속적인 서비스가 가능하도록 하는 것이다.

- 운용 중인 JEUS MQ 서버의 리소스 관리 및 모니터링

다음의 리소스들이 관리 및 모니터링의 대상이 된다.

- Connection Factory, Destination과 같은 JMS 관리 객체
- Destination과 Durable Subscription의 메시지
- 클라이언트에 할당된 커넥션, 세션, 메시지 송/수신자
- JEUS MQ 서버가 차지하는 메모리 영역
- JEUS MQ 서버가 사용하는 Persistent Storage
- 장애 상황에 대처하고 장애로부터 JEUS MQ 서버를 복구

JEUS는 콘솔 툴인 jeusadmin을 제공해서 JEUS MQ 서버 관리 및 모니터링하는 기능을 제공한다. 본 절에서는 각 툴을 사용해서 JEUS MQ 서버를 관리 및 모니터링하는 방법에 대해 설명한다.

### 3.5.1. 서버 관리

다음은 콘솔 툴(jeusadmin)을 사용해서 서버를 관리하는 방법에 대한 설명이다.

#### 콘솔 툴 사용

콘솔 툴은 콘솔에서 여러 가지 명령어를 실행해서 서버 관리작업을 수행할 수 있는 툴로 다음의 경로에 위치한다.

```
JEUS_HOME/bin/
```

다음은 서버 관리를 위해서 제공되는 명령어 목록이다.

- **Connection Factory 관련**

명령어	설명
add-jms-connection-factory	Connection Factory를 추가한다.
remove-jms-connection-factory	Connection Factory를 제거한다.

- **Destination 관련**

명령어	설명
add-jms-destination	Destination을 추가한다.
remove-jms-destination	Destination을 제거한다.

- **Durable Subscription 관련**

명령어	설명
add-jms-durable-subscription	Durable Subscription을 추가한다.
remove-jms-durable-subscription	Durable Subscription을 제거한다.



명령어에 대한 자세한 설명과 사용법, 사용 예는 JEUS Reference 안내서의 "JMS 엔진 관련 명령어"를 참고한다.

## 3.5.2. 서버 모니터링

본 절에서는 콘솔 툴을 사용해서 서버 모니터링을 하는 방법에 대해 설명한다.

### 콘솔 툴 사용

콘솔 툴은 콘솔에서 여러 가지 명령어를 실행해서 서버 관리작업을 수행할 수 있는 툴로 다음의 경로에 위치한다.

```
JEUS_HOME/bin/
```

다음은 서버 모니터링을 위해서 제공되는 명령어 목록이다.

- **Connection Factory 관련**

명령어	설명
list-jms-connection-factories	Connection Factory 목록을 조회하거나 지정된 Connection Factory의 정보를 출력한다.

- **Destination 관련**



명령어	설명
list-jms-destinations	Destination 목록을 조회하거나 지정된 Destination 정보를 출력한다.
control-jms-destination	지정된 Destination의 상태를 제어한다.

#### • 메시지 관련

명령어	설명
list-jms-messages	지정된 Destination 내의 메시지들의 정보를 조회한다.
view-jms-message	지정된 메시지의 상세한 정보를 조회한다.
move-jms-messages	지정된 메시지들을 서버나 클러스터 내의 다른 Destination으로 이동한다.
delete-jms-messages	지정된 메시지들을 Destination에서 삭제한다.
export-jms-messages	지정된 메시지들을 XML 파일 형태로 내보낸다.
import-jms-messages	내보내진 XML 파일의 메시지들을 지정된 Destination으로 가져온다.

#### • Durable Subscription 관련

명령어	설명
list-jms-durable-subscriptions	Durable Subscription 목록을 조회하거나 지정된 Durable Subscription 정보를 조회한다.

#### • 클라이언트 관련

명령어	설명
list-jms-clients	클라이언트 목록을 조회하고 정보를 출력한다.
ban-jms-client	클라이언트로부터의 연결을 강제로 끊는다.

#### • 트랜잭션 관련

명령어	설명
list-jms-pending-transactions	pending 트랜잭션 목록을 출력한다.
commit-jms-pending-transaction	지정된 pending 트랜잭션을 강제로 commit한다.



명령어에 대한 자세한 설명과 사용법, 사용 예는 JEUS Reference 안내서의 "JMS 엔진 관련 명령어"를 참고한다.

## 4. JEUS MQ 클러스터링

본 장에서는 JEUS MQ 서버의 부하를 줄이고 원활한 서비스를 제공하기 위해서 여러 대의 서버를 하나로 묶어서 사용하는 클러스터링에 대해서 설명한다.

### 4.1. 개요

한 대의 JEUS MQ 서버를 사용해서 서비스를 제공할 경우 이를 이용하는 클라이언트의 수가 너무 많거나 또는 서버에 보관되고 있는 메시지의 양이 너무 많으면 네트워크에 부하가 걸리거나 서버의 메모리 사용량이 증가한다. 이는 전체적인 성능 저하나 서버 다운의 원인이 될 수 있다. 이런 경우 네트워크 또는 메모리 부하를 분산하고 전체적인 성능을 유지시키기 위해서 JEUS MQ 서버를 증설하고 전체 MQ 서버들을 한 대의 서버처럼 동작하게 하는 것을 JEUS MQ 클러스터링이라고 한다. 이때 JEUS MQ 클라이언트는 자신이 접속하려는 JEUS MQ 서버가 클러스터링 되어 있는지 여부와 상관없이 단일 서버를 이용할 때와 동일한 방법으로 접근할 수 있다.

또한, JEUS MQ 클러스터링은 JEUS MQ 장애 극복의 개념을 포함한 기능으로 제공하고 있다. 자세한 내용은 [JEUS MQ 장애 극복](#)을 참고한다.

### 4.2. 클러스터링 종류

클러스터링은 Connection Factory 클러스터링과 Destination 클러스터링의 두 종류로 나뉘어진다.

#### 4.2.1. Connection Factory 클러스터링

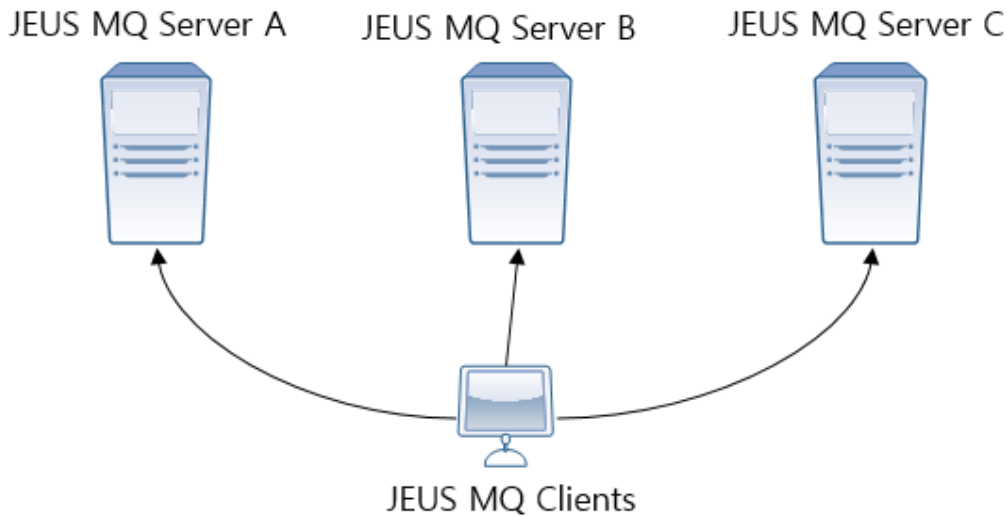
두 대 이상의 JEUS MQ 서버를 구성했다면 클라이언트로부터 이들 서버에 접근하는 연결들을 분산시킬 필요가 있다. 클라이언트로부터 서버로의 연결은 Connection Factory를 통해서 이루어지므로 Connection Factory에서 이런 분산 기능을 지원해야 한다. 이를 **Connection Factory 클러스터링**이라고 한다.

Connection Factory로부터 Connection을 생성할 때 매 연결마다 서로 다른 서버로 연결하게 되며 매번 다른 서버를 선택하기 위한 정책(Policy)을 설정할 수 있다.



하나의 JEUS MQ 클라이언트는 클러스터 내의 한 서버로 연결되며 클라이언트는 어느 서버로 연결되는지 고려할 필요가 없다.

다음은 Connection Factory 클러스터링을 간략하게 나타낸 그림이다.

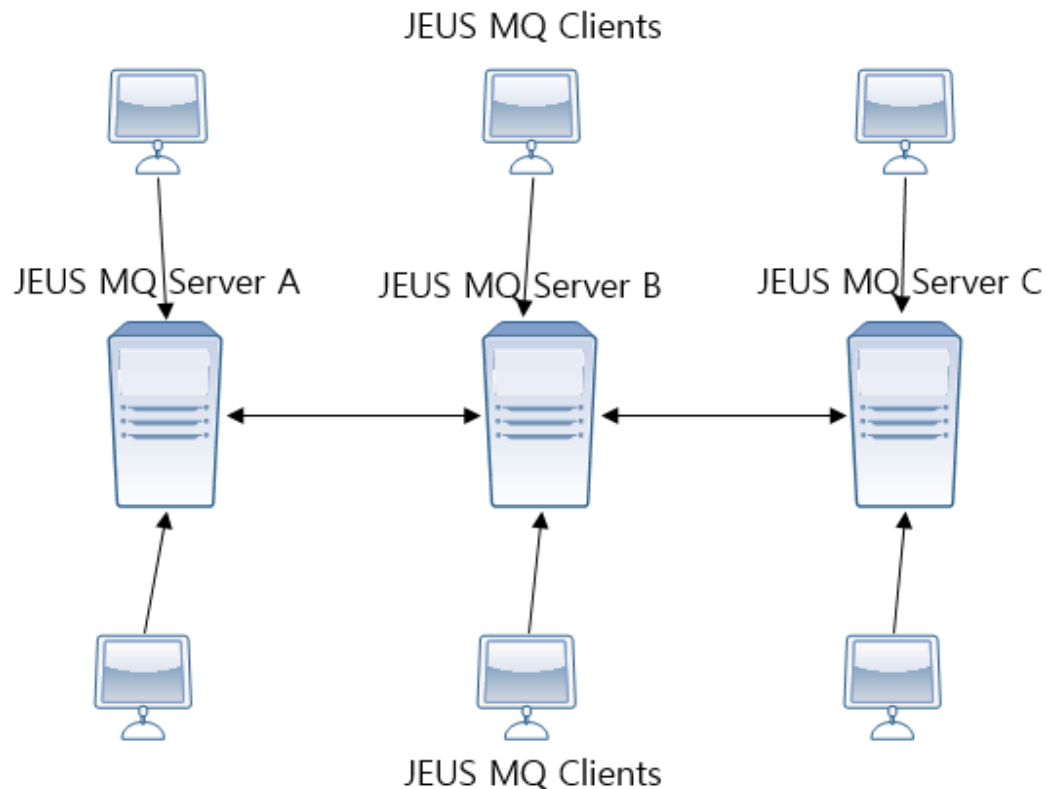


Connection Factory 클러스터링

#### 4.2.2. Destination 클러스터링

Connection Factory 클러스터링에 의해서 클라이언트 연결이 적절하게 분산되더라도 클라이언트의 처리 속도 차이나 네트워크 환경에 따라서 클라이언트가 연결된 JEUS MQ 서버의 메시지가 모두 소모되어서 더 이상 받을 수 없는 경우가 있다. 또한 서버에 메시지가 쌓이는 속도에 비해서 소모되는 양이 적다면 메시지가 계속해서 쌓이는 현상이 발생할 수 있다. 이러한 상황에서는 남은 메시지를 메시지가 부족한 JEUS MQ 서버로 이동시켜서 서비스가 지속적으로 이루어지도록 해야한다. 이런 기능을 **Destination 클러스터링**이라고 한다.

다음은 Destination 클러스터링을 나타낸 그림이다.



Destination 클러스터링

## 4.3. 클러스터링 사용

본 절에서는 JEUS MQ 클러스터링을 사용하기 위해서 필요한 설정들에 대해서 설명한다.

### 4.3.1. 서버 설정

JEUS MQ 클러스터링의 전체적인 구성은 JEUS 서버의 클러스터링에 따른다. JEUS 서버 클러스터링에 관한 설명은 JEUS Domain 안내서의 "JEUS 클러스터링"을 참고한다.

#### JEUS MQ 클러스터링 설정 주의 사항

JEUS MQ 클러스터링에 참여하는 JEUS MQ 서버들의 Destination, Durable Subscription 설정들은 domain.xml의 <cluster>에서 설정하여 그에 포함된 각 서버들에 동일하게 적용되며, 서버마다 주소가 다르기 때문에 별도로 설정해야 하는 Connection Factory는 domain.xml의 <server>에서 설정한다. 단, 이 경우도 이름은 동일하게 설정해야 한다.

클러스터의 Destination이나 Durable Subscription은 domain.xml의 <cluster> 설정 후 하위의 <jms-resource>에서 설정할 수 있다. 각각의 Destination이나 Durable Subscription을 설정하는 방법은 서버에 설정하는 것과 동일하므로 [JMS 리소스 설정](#)의 설명을 참고한다.



1. 클러스터에 서버를 추가하는 방법에 대해서는 JEUS Domain 안내서의 "클러스터에 서버 추가"를 참고한다.
2. 클러스터 내의 Destination을 설정하는 경우 서버에 설정되었던 Destination이나 Durable Subscription의 설정들은 무시되나 명확한 동작을 위해 중복된 이름을 사용하지 않도록 한다.

### 4.3.2. 클라이언트 설정

본 절에서는 JEUS MQ 클러스터링을 사용하는 방법에 대해서 설명한다. 클라이언트에서 JEUS MQ 클러스터링을 이용하기 위해서 제일 먼저 설정하는 것이 Connection Factory이기 때문에 주로 이에 관해서 설명한다.

JEUS MQ의 Connection Factory 클러스터링은 클라이언트가 Connection Factory를 얻어오는 방법에 따라서 다음과 같이 2가지로 구분할 수 있다.

#### • 클러스터링된 Connection Factory를 사용하는 방법

JEUS MQ 서버의 클러스터링 여부와 상관없이 [Connection Factory](#)에서 설명한 것과 같은 방법으로 사용할 수 있다. 단, 사용하려는 Connection Factory가 서버 설정절의 설명처럼 클러스터링된 서버간에 같은 이름을 사용해야 한다. 한번 얻어온 Connection Factory를 재활용해서 커넥션만 다시 생성하여 사용하며, 여러 JEUS MQ서버들 중에서 접속할 곳을 선택하는 정책은 서버의 Connection Factory 설정에서 설정할 수 있는데, 현재는 Round-robin방식과 Random방식을 지원하고 있다.

#### • JEUS MQ 전용 API를 사용하여 클러스터링된 Connection Factory를 생성하는 방법

JEUS MQ 전용 API를 사용하여 Connection Factory를 직접 생성하는 경우에는 JEUS MQ 서버가

클러스터링되어 있다면 [Connection Factory](#)에서 설명한 방법에 외에 추가적인 작업이 필요하다.

다음과 같이 `JeusConnectionFactoryCreator.addServerAddress()` 메소드를 이용하여 클러스터링에 참여하고 있는 모든 서버들의 정보를 추가해야 한다.

```
jeus.jms.client.util.JeusConnectionFactoryCreator connectionFactoryCreator =
new jeus.jms.client.util.JeusConnectionFactoryCreator();

connectionFactoryCreator.setFactoryName("ConnectionFactory");
connectionFactoryCreator.addServerAddress("192.168.1.2", 9741, <service-name>);
connectionFactoryCreator.addServerAddress("192.168.1.3", 9741, <service-name>);
connectionFactoryCreator.addServerAddress("192.168.1.4", 9741, <service-name>);

ConnectionFactory connectionFactory = connectionFactoryCreator.createConnectionFactory();
```

## 4.4. 예제

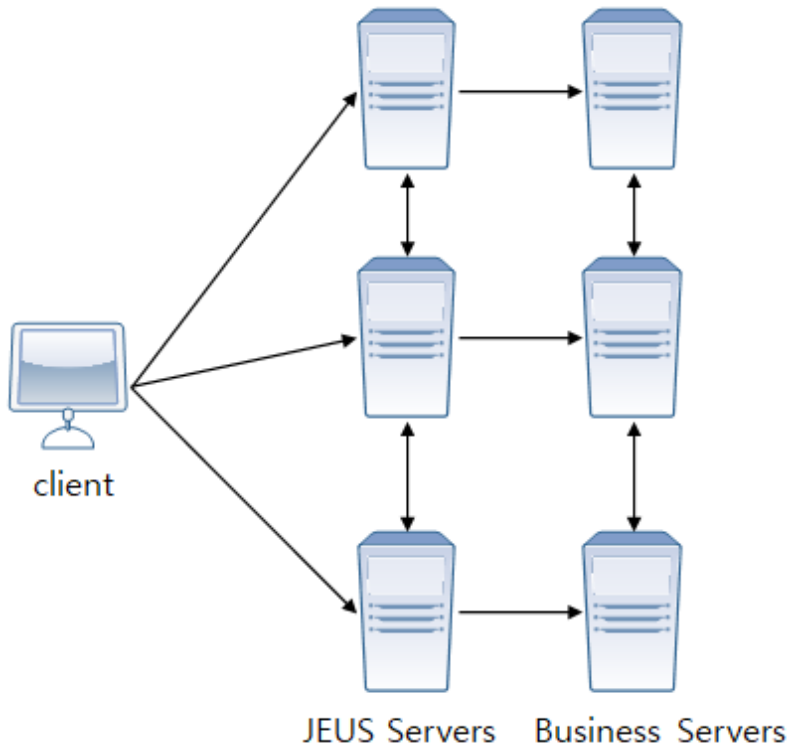
본 절에서는 JEUS MQ 클러스터링의 사용 예에 대해 일반적인 경우와 잘못 사용된 경우를 예제를 사용해서 설명한다.

### 4.4.1. 일반적인 사용 예제

물품의 주문을 JEUS MQ를 통해서 처리하는 쇼핑몰을 가정해보자.

웹 페이지에서 생성된 주문은 메시지로 작성되어 클러스터링으로 구성된 JEUS MQ의 큐에 보관되고, 실제로 주문을 처리하는 비즈니스 서버에서 이 메시지를 가져간다. 그리고 이 JEUS MQ들은 물론 클러스터링으로 구성한다.

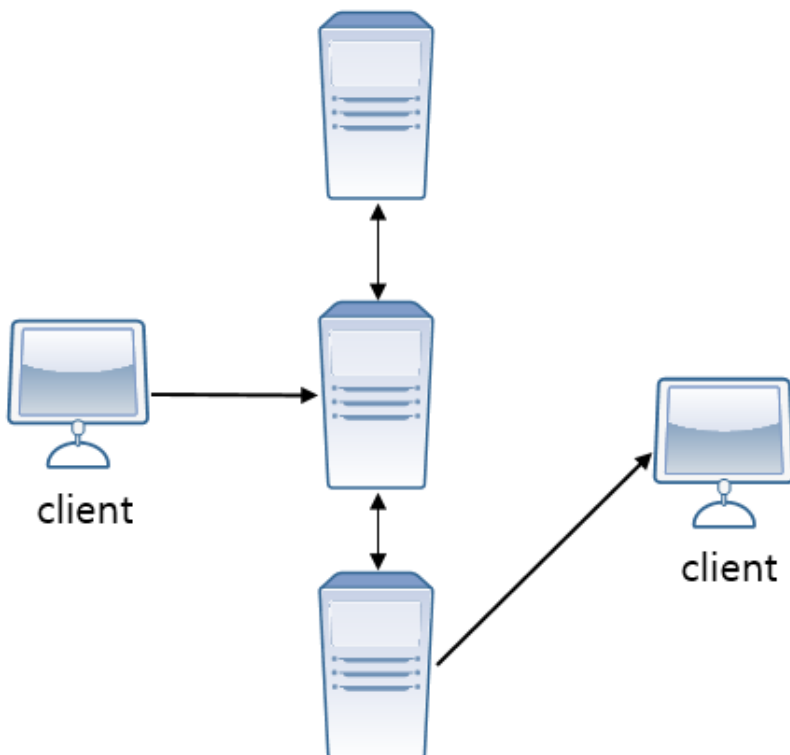
비즈니스 서버는 각각 하나씩 대응하는 JEUS MQ 서버에서 메시지를 가져가고 주문 메시지는 각 JEUS MQ 서버들에 골고루 쌓인다면 이 구조는 JEUS MQ 서버를 이상적으로 구성한 예가 된다. 만약 특정 비즈니스 서버의 처리 속도가 빨라서 그에 대응하는 JEUS MQ 서버의 큐가 다른 곳보다 먼저 비워지면, 해당 JEUS MQ 서버는 Destination 클러스터링을 통해서 다른 JEUS MQ 서버로부터 메시지들을 받아와서 해당 비즈니스 서버에 메시지를 제공할 것이다.



JEUS MQ 클러스터링 구성의 좋은 예

#### 4.4.2. 잘못된 사용 예제

JEUS MQ 클러스터링의 구성 방법은 매우 다양하지만, 다음과 같은 구조로 사용하는 것은 오히려 효율을 떨어뜨릴 수 있다.



JEUS MQ 클러스터링 구성의 나쁜 예

JEUS MQ 클러스터링을 구성하긴 했으나 사용하는 JMS 클라이언트가 한 번 연결한 커넥션을 계속 재활용하면

커넥션의 부하 분산이 제대로 동작하지 않고 JEUS MQ 서버 간에 불필요한 메시지 이동이 일어나게 되어 효율을 크게 저하시킬 수 있다.

## 5. JEUS MQ 장애 극복

본 장에서는 JEUS MQ에서 서버나 네트워크에 장애가 발생해서 더 이상 메시징 서비스를 할 수 없을 때 JMS 클라이언트가 다시 연결을 맺고 클라이언트 상태를 복구하는 방법에 대해서 설명한다. 또한 JMS 클라이언트의 복구를 위해서 필요한 서버 구성과 서버 장애 극복에 대해서 설명한다.

### 5.1. 개요

JEUS MQ에서의 장애 극복이란 장애가 발생했을 때 클라이언트 애플리케이션이 자동으로 연결을 다시 맺고 클라이언트의 모든 상태를 장애가 발생하기 이전으로 복구하는 것을 의미한다.

장애가 발생하는 원인은 다음과 같이 크게 2가지로 나눌 수 있다.

- 네트워크 장애

네트워크 장애는 JEUS MQ 서버와 클라이언트 사이의 네트워크에 문제가 발생해서 더 이상 통신을 할 수 없는 상태를 의미한다.

네트워크 장애는 일시적인 장애일 수도 있고 서버가 다운되거나 네트워크가 완전히 사용 불가능할 경우도 포함된다. 네트워크 장애가 발생하면 JEUS MQ 클라이언트들은 네트워크에 장애가 발생한 서버나 해당 서버와 클러스터링 관계에 있는 다른 서버에 재연결을 시도하고 재연결이 완료되면 클라이언트의 상태를 자동 복구하여 서비스를 계속하게 된다.

- 서버 장애

서버 장애는 네트워크 장애를 제외하고 서버에 발생하는 모든 장애를 포함한다. 일반적으로 메시징 데이터를 저장하는 디스크나 데이터베이스 작업을 수행할 때 오류가 발생하거나 메모리 부족 등이 이에 해당한다. 현재 서비스 중인 서버에 장애가 발생하면 대기 중이던 백업 서버는 이전에 사용하던 데이터들을 자동으로 복구하고 서비스를 계속 이어서 하게 된다.

이러한 장애를 극복하기 위해서는 JEUS MQ 서버들 간에 네트워크를 구성하고 JEUS MQ 클라이언트에 필요한 설정들을 해야 한다. 그리고 클라이언트들은 장애 극복을 위해서 JEUS MQ에서 제공하는 Client API를 호출하여 장애 극복에 필요한 여러 속성들을 직접 설정할 수 있다.

### 5.2. 서버 장애 극복

본 절에서는 JEUS MQ의 장애 극복을 이용하기 위해서 필요한 서버들 간의 네트워크 구성과 기타 설정들에 대해서 설명한다.

#### 5.2.1. 네트워크 구성

JEUS MQ 장애 극복을 위해서는 한 대 이상의 Active 서버가 필요하고, 그 서버들은 클러스터링되어 있어야 한다. Standby 서버는 선택 사항이며, 장애가 발생할 때 전체 처리 가능 용량에 여유를 두기 위해서 설정한다. JEUS 클러스터링 설정은 JEUS Domain 안내서의 "JEUS 클러스터링"을 참고한다.



- Active 서버

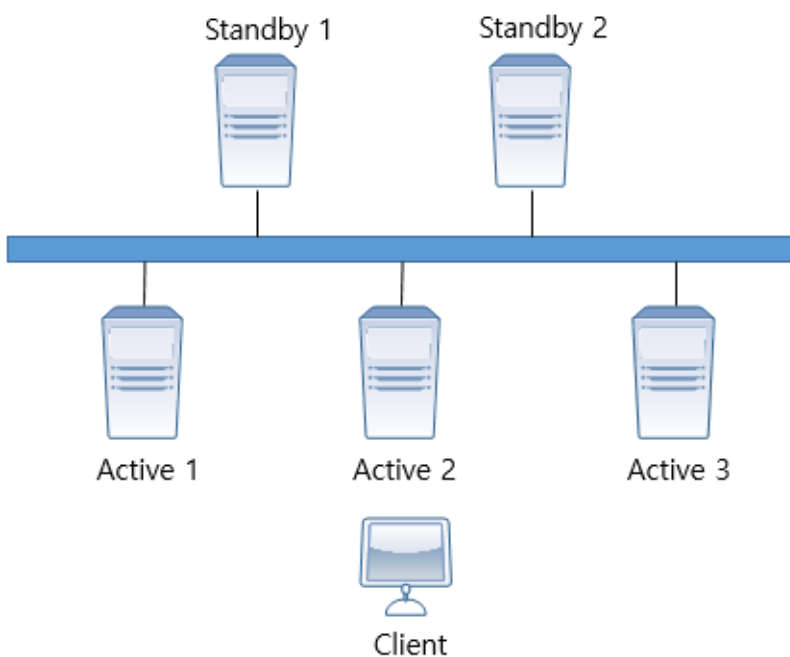
장애가 없을 때 클라이언트의 요청을 받아서 처리하는 메인 서버이다.

- Standby 서버

Active 서버에 장애가 발생하면 서버가 수행하던 서비스들을 이어받아서 수행하는 백업 서버이다.

JEUS MQ 클러스터링과 JEUS MQ 장애 극복은 통합된 기능으로 JEUS MQ 클러스터링을 설정하면 JEUS MQ 장애 극복 기능도 함께 동작한다. 이는 예전의 Active 서버와 Standby 서버를 쌍으로 구성해야 했던 점을 개선하여 자유롭고 다양한 구성을 가능하게 하고 Active 서버의 수와 Standby 서버의 수 역시 자유롭게 설정할 수 있다. 일반적으로 다수의 Active 서버에 소수의 Standby 서버를 두는 구성을 하거나 Standby 서버 없이 Active 서버만으로 구성한다.

다음은 장애 극복을 위한 MQ 서버들 간의 네트워크 구성이다.



3대의 Active 서버와 2대의 Standby 서버를 이용한 JEUS MQ 클러스터링 구성

위와 같은 구성에서 Active 서버들 중 하나에 장애가 발생하면 우선 아무런 서비스를 하고 있지 않은 Standby 서버들 중 하나가 그 역할을 이어받아서 서비스한다. 서비스를 수행하고 있는 Active 서버들과 Standby 서버들 중 하나에 추가적으로 장애가 발생하면 마찬가지로 서비스를 하고 있지 않은 Standby 서버들 중 하나가 이어받게 되며, 더 이상 서비스를 수행하고 있지 않은 서버가 없다면 서비스를 수행하고 있는 Active 서버들 중의 하나가 이어받아서 한 서버가 두 개 이상의 서버 역할을 하게 된다. 이는 해당 구성에서 마지막 하나의 서버만 남을 때까지 계속되며 마지막 남은 서버에도 장애가 발생하면 JEUS MQ 클러스터링 서비스는 더 이상 동작하지 않는다.

## Active 서버와 Standby 서버 구성

다음은 Active 서버와 Standby 서버의 장애 극복 설정을 하는 예제이다.

- Active 서버 설정

**Active 서버의 장애 극복 설정**

domain.xml

```
<domain>
...
  <jms-engine>
    <engine-roll>Active</engine-roll>
    <failover-check-timeout>5</failover-check-timeout>
    <failover-check-count>0</failover-check-count>
  </jms-engine>
</domain>
```

다음은 설정 태그에 대한 설명이다.

태그	설명
<engine-roll>	JMS Engine의 역할을 설정한다. (기본값: Active) <ul style="list-style-type: none"><li>◦ Active: 정상시에 기동되어 서비스를 하는 역할</li><li>◦ Standby: Active의 장애 시에 기동되어 서비스를 이어받는 역할</li></ul>
<failover-check-timeout>	장애 감지 후 Failover를 실행하기 전에 대상이 되는 JMS Engine의 생존 여부를 다시 확인하는 시간을 설정한다. 이 값은 한 번의 시도에 소요되는 시간이다. (단위: 초, 기본값: 5)
<failover-check-count>	장애 감지 후 Failover를 실행하기 전에 대상이 되는 JMS Engine의 생존 여부를 다시 확인하는 최대 횟수를 설정한다. (기본값: 0)  지정된 횟수만큼 시도했으나 생존 여부를 확인할 수 없으면 장애로 판단하고 Failover를 시작한다. 0으로 설정한 경우 장애를 감지한 즉시 Failover를 시작한다.

#### • Standby 서버 설정

Standby 서버 설정은 **Active 서버의 장애 극복 설정**에서 <engine-roll> 태그에 'Standby'로 설정한다.

## 5.2.2. Connection Factory 설정

Connection Factory 설정은 JEUS MQ에 장애가 발생했을 때 클라이언트가 다른 Active 서버나 Standby 서버로 시도하는 재연결과 관련된 설정이다.

다음은 Connection Factory 설정의 예로 설정 정보는 <connection-factory> 태그 아래 구성한다.

**Connection Factory** 설정 : <domain.xml>

```
<domain>
...
  <jms-engine>
    <connection-factory>
      <type>queue</type>
      <name>qcf</name>
      <service>default</service>
      <reconnect-enabled>true</reconnect-enabled>
    </connection-factory>
  </jms-engine>
</domain>
```

```

    <reconnect-period>0</reconnect-period>
    <reconnect-interval>5000</reconnect-interval>
  </connection-factory>
</jms-engine>
</domain>

```

다음은 설정 태그에 대한 설명이다.

태그	설명
<reconnect-enabled>	장애가 발생 시 재연결 여부를 설정한다. (기본값: false)  장애가 발생 시 재연결을 통해 클라이언트를 복구하려면 이 값을 true로 변경해야 하며, true로 설정 시 JEUS MQ 클라이언트는 Active 서버와 Standby 서버에 반복적으로 재연결을 시도한다.
<reconnect-period>	재연결을 시도할 시간을 설정한다. 기본값으로 설정하면 무한대로 시도한다. (기본값: 0)
<reconnect-interval>	재연결 시도 간의 대기 시간을 설정한다. (기본값: 5)

### 5.2.3. Persistence Store 설정

Persistence Store는 DeliveryMode가 PERSISTENT일 때 메시지를 저장하는 역할을 한다.

장애가 발생했을 때 다른 Active 서버나 Standby 서버가 이 Persistence Store에 저장된 메시지를 복구하기 때문에 메시지의 유실 없이 서비스를 계속할 수 있다. 따라서, JEUS MQ 서버의 장애 극복의 가장 핵심적인 리소스이다.

JEUS MQ 장애 극복에서 Persistence Store를 설정하기 위해서는 가능하면 Persistence Store가 각 Active 서버나 Standby 서버에서 모두 접근 가능한 곳에 위치해야 한다.

- 저널 로그 Persistence Store

저널 로그를 Persistence Store로 사용할 경우에는 저널 로그 베이스 디렉터리(저널 로그 설정의 'Base Dir'의 값)는 Active 서버와 Standby 서버가 모두 접근 가능한 곳에 위치해야 한다. 이를 위해서 SAN과 같은 디스크 공유 하드웨어를 구성하고 그 위에 저널 로그의 베이스 디렉터리를 생성해야 한다.

- JDBC Persistence Store

JDBC를 Persistence Store로 사용할 경우에는 단순히 domain.xml의 <jms-engine><persistence-store><jdbc> 하위에 데이터소스를 설정하면 된다. 다만, 데이터베이스도 장애가 발생하여 서비스를 할 수 없는 경우가 있기 때문에 Tibero의 TAC이나 OracleDB의 RAC같은 클러스터링 기술을 사용하여 데이터베이스도 장애 극복을 위한 구성을 해야 한다.



만일 일부 서버 간에 Persistence Store에 서로 접근할 수 없는 경우는 최대한 접근 가능한 서버를 찾아서 장애 극복을 시도한다.

## 5.2.4. 자동 복원(Fail-Back)

Active 서버의 장애로 인해 다른 Active 서버나 Standby 서버의 장애 극복 기능이 활성화되었을 때 서버 관리자는 Active 서버에 발생한 장애가 무엇인지를 파악하고 가능한 빨리 장애 요인을 해결하여 Active 서버를 다시 기동해야 한다.

장애가 있었던 Active 서버가 다시 기동되면 해당 서버의 서비스를 대신하고 있던 서버로부터 데이터를 마이그레이션하고 접속해있는 클라이언트들을 되살아난 Active 서버로 옮겨야 한다. 이런 작업을 복원(Fail-Back)이라 한다. 이 작업은 항상 자동으로 이루어진다.

## 5.3. 클라이언트 장애 극복

서버나 네트워크에 장애가 발생하여 JEUS MQ 클라이언트와 서버 간의 연결이 끊어지면 클라이언트는 클러스터 내의 Active 서버와 Standby 서버들을 교대로 재연결을 시도한다. 재연결이 성공하면 클라이언트는 연결이 끊어지기 이전 상태로 복구를 시도한다. 이런 클라이언트 장애 극복 과정은 클라이언트 애플리케이션에 대한 수정 없이 JEUS MQ 설정을 통해서 자동으로 수행된다.

본 절에서는 클라이언트의 장애 극복에 대해서 자세히 알아보고 그에 따른 제약 사항과 메시지 유실 없이 장애를 극복하기 위한 방법에 대해서 설명한다.

### 5.3.1. 재연결

JEUS MQ 클라이언트와 서버의 연결이 끊어질 경우 재연결 여부를 결정하는 '**Reconnect Enabled**' 항목과 관련 설정 항목들은 해당 Connection Factory를 이용하여 생성하는 모든 커넥션에 적용된다. 이에 대한 자세한 내용은 [Connection Factory 설정](#)을 참고한다.

만약 특정 커넥션에 대해서 이 설정을 변경하려면 다음의 예제와 같이 JEUS MQ Client API인 "jeus.jms.client.facility.connection.JeusConnection" 클래스를 이용하여 설정을 변경할 수 있다.

```
...
import jeus.jms.client.facility.connection.JeusConnection;
...
Context ctx = new InitialContext();
ConnectionFactory factory = ctx.lookup("connection-factory");
JeusConnection connection = (JeusConnection)factory.createConnection("jeus", "jeus");
connection.setReconnectEnabled(true);
connection.setReconnectInterval(1000); // 1초
connection.setReconnectPeriod(3600000); // 1시간
...
```

서버의 Connection Factory 설정을 통해 '**Reconnect Enabled**'의 값을 true로 설정하였다면 모든 재연결 과정은 클라이언트 애플리케이션에 자동으로 수행되어 클라이언트 코드는 수정할 필요가 없다.

### 5.3.2. Connection Factory 재사용

JEUS MQ 클러스터 내의 Active 서버와 Standby 서버들은 동일한 이름의 Connection Factory를 가지고 있기

때문에 한 번 JNDI Lookup을 통해서 얻은 Connection Factory는 서버나 네트워크의 장애가 발생한 경우에 다시 Lookup할 필요없이 재사용할 수 있다.

### 5.3.3. Destination 재사용

Connection Factory와 마찬가지로 JEUS MQ 클러스터 내의 Active 서버와 Standby 서버들은 동일한 이름의 Destination을 가지고 있다. 따라서 한 번 JNDI Lookup을 통해서 얻은 Destination은 서버나 네트워크에 장애가 발생해도 다시 JNDI를 Lookup할 필요없이 재사용할 수 있다.

장애 발생 이전에 Destination에 저장되었던 메시지들은 서버 장애가 발생한 경우 모두 자동 복구되므로 클라이언트는 해당 Destination을 통해서 메시징 작업을 계속할 수 있다.

### 5.3.4. 응답 대기시간

JEUS MQ의 클라이언트에서 보내는 모든 요청들은 'Request Blocking Time'이라는 응답이 오기까지 기다리는 요청 응답시간을 갖는다(기본값: 200000, 단위: ms). 이 시간은 JEUS MQ 서버 domain.xml의 **Connection Factory** 설정 중 **<request-blocking-time>** 에서 설정할 수 있다.

서버뿐만 아니라 각 커넥션별로 다른 설정을 하는 경우 JEUS MQ Client API인 "jeus.jms.client.facility.connection.JeusConnection" 클래스를 이용하여 설정을 변경할 수 있다.

```
...
import jeus.jms.client.facility.connection.JeusConnection;
...
Context ctx = new InitialContext();
ConnectionFactory factory = ctx.lookup("connection-factory");
JeusConnection connection = (JeusConnection)factory.createConnection("jeus", "jeus");
connection.setRequestBlockingTime(300000); // 5분
...
```

RequestBlockingTime은 세션 트랜잭션이나 XA의 경우에 트랜잭션 타임아웃의 기본값으로도 사용된다.

### 5.3.5. 커넥션 복구

커넥션 복구가 설정되지 않은 JEUS MQ의 커넥션들은 기본적으로 물리적 연결(Socket)을 공유한다. 그러나 domain.xml의 Connection Factory 설정에서 **<reconnect-enabled>**의 값이 true로 설정된 경우에는 장애 극복을 위해 각각의 커넥션과 물리적 연결이 1대 1 관계가 된다.



물리적 연결과 커넥션이 1대 1 관계가 되면 매번 연결을 맺을 때 마다 새로운 물리적 연결을 생성하게 되므로 성능이 다소 저하될 수 있다. 이 문제는 매번 커넥션을 생성하지 않고 재사용하도록 클라이언트 애플리케이션을 작성함으로써 해결이 가능하다.

연결이 복구될 때 커넥션이 재연결될 뿐만 아니라 커넥션의 상태도 모두 복구된다.

- 시작(start) 상태

메시지 수신을 위해서 Connection.start()를 호출한 경우 발생한 장애가 복구되면 시작 상태가 복구되어 메시지 수신을 계속할 수 있다.

- 정지(stop) 상태

메시지 수신을 중지하기 위해서 Connection.stop()을 호출한 경우 장애가 복구되면 정지 상태가 복구되어 더 이상 메시지 수신을 하지 않는다.

커넥션 상태뿐만 아니라 커넥션 이하의 세션과 커넥션 메시지 수신자(ConnectionConsumer)들도 모두 복구된다.

- 세션(Session) 복구

커넥션을 통해서 생성된 세션은 장애 발생 이전에 Session.close()가 호출되지 않은 경우에 커넥션 장애가 복구되면 같이 복구된다. 자세한 내용은 [세션 복구](#)를 참고한다.

- 커넥션 메시지 수신자(ConnectionConsumer) 복구

커넥션을 통해서 생성된 커넥션 메시지 수신자는 장애 발생 이전에 ConnectionConsumer.close()가 호출되지 않았다면 커넥션이 복구될 때 같이 복구되며 커넥션이 시작 상태인 경우 메시지 수신을 계속하게 된다. 단, 장애 이전에 수신했던 메시지들은 모두 서버로 되돌려 보내지고 새로 받게 되므로 이후 수신된 메시지들의 Message.getJMSRedelivered()가 true가 될 수 있다.

이 외에도 세션이나 커넥션 메시지 수신자를 생성하는 메소드들은 장애가 복구되면 요청을 재전송하여 응답이 오기까지 기다리게 된다. Connection.close가 호출되는 경우에는 응답이 오는 것에 상관없이 발생한 장애를 복구하지 않는다.

## 5.3.6. 세션 복구

세션은 Session.close()가 호출되지 않은 경우라면 커넥션이 복구될 때 같이 복구된다. 또한 세션 이하의 메시지 수신자(MessageConsumer)나 메시지 송신자(MessageProducer)들도 세션이 복구될 때 모두 복구된다.

세션에는 여러 가지 객체를 생성하는 메소드들을 가지고 있는데 각 메소드는 발생한 장애를 복구할 때 다음과 같이 동작한다.

- 메시지 생성 메소드

메시지를 생성하는 메소드들은 장애 여부와 상관없이 항상 바로 생성된다.

```
createBytesMessage()
createMapMessage()
createMessage()
createObjectMessage()
createObjectMessage(Serializable object)
createStreamMessage()
createTextMessage()
createTextMessage(String text)
```

- 큐 브라우저 생성 메소드

큐 브라우저를 생성하는 메소드는 장애가 복구된 후에도 요청을 완료한다. 만약 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.

```
createBrowser(Queue queue)
createBrowser(Queue queue,String messageSelector)
```

#### • Destination 생성 메소드

Destination을 생성하는 메소드들은 장애가 복구된 후에도 생성 요청을 완료한다. 만약 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.

```
createQueue(String queueName)
createTopic(String topicName)
```

#### • 임시 Destination 생성 메소드

임시 Destination을 생성하는 메소드들은 장애와 상관없이 생성 요청을 완료한다.

```
createTemporaryQueue()
createTemporaryTopic()
```

#### • 메시지 수신자 생성 메소드

메시지 수신자를 생성하는 메소드들은 장애가 복구된 후에도 생성 요청을 완료한다. 만약 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.

```
createConsumer(Destination destination)
createConsumer(Destination destination, java.lang.String messageSelector)
createConsumer(Destination destination, java.lang.String messageSelector,boolean NoLocal)
```

#### • Durable 메시지 수신자 생성 메소드

Durable 메시지 수신자를 생성하는 메소드들은 장애가 복구된 후에도 생성 요청을 완료한다. 만약 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.

```
createDurableSubscriber(Topic topic, String name)
createDurableSubscriber(Topic topic,String name, String messageSelector,boolean noLocal)
```

#### • 메시지 송신자 생성 메소드

메시지 송신자를 생성하는 메소드는 장애가 복구된 후에도 생성 요청을 완료한다. 만약 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.

```
createProducer(Destination destination)
```

세션에 장애가 발생하면 세션 트랜잭션은 다음과 같은 영향을 받는다.

구분	동작
commit()	<p>트랜잭션 세션에서 생성한 메시지 수신자와 메시지 송신자를 이용하여 메시지를 수신하거나 전송하는 중에 장애가 발생한 경우 이후 첫 번째로 호출되는 commit은 "jakarta.jms.TransactionRolledBackException"을 발생하며 rollback된다. commit 시점에 commit할 메시지가 없다면 이 예외는 발생하지 않는다. 장애에 의해 commit이 실패된 후 호출되는 commit은 모두 정상적으로 동작한다.</p> <p>만약 commit 중에 장애가 발생했는데 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다. 이 경우 commit 여부는 알 수 없으며 관리 도구를 통해서 commit 여부를 확인해야 한다.</p>
rollback()	<p>rollback은 장애 극복 후에도 rollback 요청을 완료한다.</p> <p>만약 rollback 중에 장애가 발생했는데 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다. rollback의 경우 JMSEException이 발생해도 rollback되는 것을 보장할 수 있다.</p>

Session.recover() 메소드의 경우 장애가 복구된 후에도 recover 요청을 완료한다. 만약 recover 호출 후에 장애가 발생했는데 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.

세션의 ACKNOWLEDGE 모드를 Session.CLIENT\_ACKNOWLEDGE로 설정한 경우 Message.acknowledge()를 통해서 세션 내에 존재하는 acknowledge가 안 된 메시지들에 대해서 acknowledge를 호출할 수 있다. 만약 acknowledge 중에 장애가 발생하면 각 메시지에 대해서 ExceptionListener를 통해서 jeus.jms.common.message.MessageAcknowledgeException을 통보받게 된다. 이 예외는 메시지 acknowledge 중에 장애가 발생하여 메시지가 재전송(Redelivered)될 수 있음을 알려준다.



MessageAcknowledgeException.getErrorCode()를 호출하면 실패한 메시지의 MessageID를 얻을 수 있다.

### 5.3.7. 메시지 전송 복구

본 절에서는 메시지 전송자를 통해서 메시지를 전송할 때 장애가 발생할 경우에 대해서 설명한다.

메시지 송신자의 send 메소드는 메시지가 서버에 전달되어 응답되기까지 blocking된다. 만약, 도중에 장애가 발생할 경우 다음과 같은 상황이 발생한다.

- **send를 호출했지만 메시지가 아직 전송되지 않은 경우**

이 경우 장애가 복구된 후에 메시지는 서버로 전송되고 정상 처리된다. 만약 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.

- **send를 호출하고 메시지가 서버에서 처리된 후에 네트워크에 장애가 발생한 경우**

이 경우 장애가 복구되어 다시 서버에 접속하면 응답 메시지를 받아 정상적으로 처리된다. 만약 RequestBlockingTime이 지난 후에도 장애가 복구되지 않는다면 JMSEException이 발생한다.



- **send를 호출하고 메시지가 서버에서 처리된 후에 서버에서 장애가 발생하였고 그 장애가 복구된 경우**

이 경우 장애가 복구되어 다시 서버에 접속해도 메시지 전송 여부를 알 수 없으므로 RequestBlockingTime만큼 기다리다가 ExceptionListener를 통해서 "jeus.jms.common.message.MessageSendException"을 통보한다.

- **send를 호출하고 메시지가 서버에서 처리되기 전에 네트워크나 서버에 장애가 발생한 경우**

이 경우에도 위의 경우와 마찬가지로 복구된 서버에 다시 접속해도 메시지 전송 여부를 알 수 없으므로 RequestBlockingTime만큼 기다리다가 ExceptionListener를 통해서 "jeus.jms.common.message.MessageSendException"을 통보한다.



MessageSendException.getErrorCode()를 호출하면 실패한 메시지의 MessageID를 얻을 수 있다.

### 5.3.8. 메시지 수신 복구

메시지 수신은 크게 동기식과 비동기식 방법으로 나뉘지며 장애가 복구된 경우 동작 방식이 조금씩 다르다. 우선 동기식 메시지 수신하는 경우 장애 복구에 대해서 설명한다.

#### 동기식 메시지 수신 복구

메시지 수신자에는 MessageConsumer.receive()와 MessageConsumer.receive(long timeout), MessageConsumer.receiveNoWait() 3가지 동기식 메시지 수신 메소드가 있다.

각 메소드 호출할 때 장애가 발생하면 다음과 같이 동작한다.

구분	동작
receive()	<p>receive() 메소드의 경우 원래 메시지를 수신할 때까지 blocking되지만, 장애가 발생하면 언제 복구될 지 알 수 없으므로 대기시간을 RequestBlockingTime으로 변경한다. 만약 시간 내에 장애가 복구되면 메시지 수신을 요청하는 메시지를 다시 보내서 메시지를 수신하고 그렇지 않으면 JMSException이 발생한다.</p> <p>Session.AUTO_ACKNOWLEDGE로 설정한 경우에는 메시지를 수신하고 클라이언트에 메시지를 넘겨주기 전에 acknowledge를 서버에 전송한다. 이때 장애가 발생하면 acknowledge 실패한 메시지에 대해서 ExceptionListener를 통해 jeus.jms.common.message.MessageAcknowledgeException을 통보한다. 이 Exception은 메시지 acknowledge 중에 장애가 발생해서 메시지가 재전송(Redelivered)될 수 있음을 알려준다.</p>

구분	동작
receive(long timeout)	<p>receive(long timeout) 메소드는 원래 메시지를 수신할 때까지 blocking되지만, 장애가 발생하면 언제 복구될지 알 수 없으므로 타임아웃이 RequestBlockingTime보다 큰 경우 제한 시간을 RequestBlockingTime으로 변경한다. 만약, 제한 시간 내에 장애가 복구되면 메시지 수신을 요청하는 메시지를 다시 보내어 메시지를 수신하고 그렇지 않으면 JMSException이 발생한다.</p> <p>Session.AUTO_ACKNOWLEDGE로 설정한 경우 메시지를 수신하고 클라이언트에 메시지를 넘겨주기 전에 acknowledge를 서버에 전송한다. 이때 장애가 발생하면 acknowledge 실패한 메시지에 대해서 ExceptionListener를 통해서 jeus.jms.common.message.MessageAcknowledgeException을 통보한다. 이 Exception은 메시지 acknowledge 중에 장애가 발생하여 메시지가 재전송(Redelivered)될 수 있음을 알려준다.</p>
receiveNoWait()	<p>receiveNoWait() 메소드는 원래 메시지를 수신하지 않아도 blocking되지 않는다. 따라서 장애가 발생해도 바로 반환된다.</p>

## 비동기식 메시지 수신 복구

비동기식 메시지 수신 복구 방법으로 수신한 메시지들은 MessageListener.onMessage 중이거나 MessageListener.onMessage에서 처리되어 acknowledge 중이거나 미리 가져오기를 통해 클라이언트 큐에 쌓여 있는 메시지들로 나눌 수 있다.

각각의 메시지들에 대해서 JEUS MQ는 다음과 같이 장애를 복구한다.

- onMessage 중에 장애가 발생하면 장애가 복구된 후에 acknowledge가 전송되므로 정상적으로 처리된다.
- onMessage가 완료된 후 acknowledge가 전송되는 중에 장애가 발생하면 acknowledge 실패한 메시지에 대해서 ExceptionListener를 통해서 jeus.jms.common.message.MessageAcknowledgeException을 통보한다. 이 Exception은 메시지 acknowledge 중에 장애가 발생하여 메시지가 재전송(Redelivered)될 수 있음을 알려준다.
- 미리 가져오기로 클라이언트 큐에 쌓여 있는 메시지들은 장애 극복 후에 모두 서버로 되돌려 보내지며 나중에 다시 클라이언트로 전송된다. 이 메시지들의 Message.getJMSRedelivered()의 값은 true가 될 수 있다.



MessageAcknowledgeException.getErrorCode()를 호출하면 실패한 메시지의 MessageID를 얻을 수 있다.

### 5.3.9. 메시지 유실 방지와 트랜잭션

JEUS MQ의 장애 극복이 클라이언트 애플리케이션에 명확하게 자동으로 처리된다. 그러나 메시지 송수신 과정에서 메시지를 유실할 위험이 있으므로 ExceptionListener를 통해서 이런 메시지들에 대해서 별도로 처리해야 하는 문제점이 있다.

엔터프라이즈 메시징 애플리케이션에서 메시지 유실은 치명적인 문제를 일으킬 수 있다. 메시지 유실을 방지하면서 완벽하게 장애를 극복할 수 있는 유일한 방법은 트랜잭션을 사용하는 것이다.

따라서 다음에 제시하는 방법을 통해서 애플리케이션을 작성할 것을 적극 권장한다.

- Jakarta EE 환경에서는 반드시 트랜잭션 내에서 메시지를 송수신한다.
- 서블릿의 경우 UserTransaction을 JNDI로부터 Lookup하여 UserTransaction 내에서 메시지를 송수신한다.
- EJB의 경우 EJB 메소드의 트랜잭션 속성(TransactionAttribute)을 "Required"나 "RequireNew"로 설정해서 항상 트랜잭션 내에서 메시지를 송수신한다.

일반 Java 클라이언트에서는 세션을 생성할 때 Connection.createSession(true, Session.SESSION\_TRANSACTED)를 호출하여 생성한다. 이렇게 생성한 세션은 commit()이나 rollback() 호출을 통해서 트랜잭션 내에서 메시지를 송수신할 수 있다.

## 6. JEUS MQ 특수 기능

본 장에서는 JEUS MQ의 특수 기능인 Message Bridge, Message Sort, Global Order, Message Group, Message Management, Topic Multicast 기능에 대해 설명한다.

### 6.1. JEUS MQ Message Bridge

Message Bridge는 두 개의 서로 다른 MQ를 연결해주는 기능으로 연결할 수 있는 서로 다른 MQ는 다음의 경우를 포함한다.

- 동일한 MQ 서비스의 다른 버전(서로 호환되지 않는 다른 버전의 JEUS MQ 간의 연결)
- 각각 다른 MQ 서비스(WebLogic 등 타 벤더의 MQ와 JEUS MQ와의 연결)



연결할 수 있는 MQ 서비스에는 제한이 없기 때문에 JEUS MQ 서비스의 동일한 버전 간에도 Message Bridge를 설정할 수 있지만 그러한 설정을 하는 것은 추천하지 않는다.

JEUS MQ 서비스의 버전이 동일할 경우에는 클라이언트를 직접 원격 Destination에 붙여서 사용하는 것이 더 안전하고, 효율적이다. 굳이 동일한 버전의 JEUS MQ 서비스 간의 Bridge를 설정해야 한다면 두 JEUS MQ간의 일반적인 메시지 서비스와 구분을 위해 Bridge가 설정된 서버의 실행 스크립트에 다음의 옵션을 추가해야 한다.

```
-Djeus.jms.client.use-single-server-entry=false
```

#### 6.1.1. 서버 설정

하나의 Message Bridge를 사용하려면 2가지 설정이 필요하다.

- Message Bridge의 양끝에서 각 MQ 서비스의 연결에 대한 설정을 나타내는 Bridge Connection을 2개 설정
- 두 Bridge Connection을 연결하는 실제 Bridge를 나타내는 Bridge Entry 설정

Message Bridge는 JEUS의 MQ와는 별도로 동작하기 때문에 domain.xml의 <resources> 아래에 설정한다. 다음은 JEUS 6의 MQ로 부터 Weblogic 10.3의 MQ로의 Bridge를 설정한 예제이다.

domain.xml

```
<domain>
...
  <resource>
    <message-bridge>
      <bridge-connections>
        <connection>
          <name>jeus6</name>
          <classpath>file:///home/seunghoon/workspace/jeus6/lib/client/
clientcontainer.jar</classpath>
          <jndi-provider-url>127.0.0.1:19736</jndi-provider-url>
          <jndi-initial-context-factory>jeus.jndi.JNSContextFactory
```

```

        </jndi-initial-context-factory>
        <connection-factory>XAQueueConnectionFactory</connection-factory>
        <xa-support>>false</xa-support>
    </connection>
    <connection>
        <name>weblogic103</name>
        <classpath>
            file:///home/seunghoon/bea/wlserver_10.3/server/lib/wljmsclient.jar
        </classpath>
        <jndi-provider-url>t3://localhost:7001 </jndi-provider-url>
        <jndi-initial-context-factory>
            weblogic.jndi.WLInitialContextFactory</jndi-initial-context-factory>
        <connection-factory>QueueConnectionFactory</connection-factory>
        <xa-support>>false</xa-support>
    </connection>
</bridge-connections>
<bridges>
    <bridge>
        <name>bridge1</name>
        <source>
            <connection-name>jeus6</connection-name>
            <destination>ExamplesQueue</destination>
            <type>queue</type>
        </source>
        <target>
            <connection-name>weblogic103</connection-name>
            <destination>ExamplesQueue</destination>
            <type>queue</type>
        </target>
    </bridge>
</bridges>
</message-bridge>
</resource>
</domain>

```

다음은 설정 태그에 대한 설명이다. <message-bridge> 아래에 모든 설정을 할 수 있다.

- <bridge-connection>

연결할 특정 MQ를 나타내는 설정이다.

태그	설명
<name>	해당 MQ를 대표할 이름을 설정한다.
<classpath>	해당 MQ가 제공하는 JMS 클라이언트 라이브러리의 경로를 지정한다.
<jndi-provider-url>	해당 MQ의 Destination이나 ConnectionFactory가 등록된 Naming Service에 접근할 때 필요한 Provider URL을 설정한다.
<jndi-initial-context-factory>	해당 MQ의 Destination이나 ConnectionFactory가 등록된 Naming Service에 접근할 때 필요한 Initial Context Factory의 fully qualified name을 설정한다.
<connection-factory>	해당 MQ에 접근할 때 사용할 ConnectionFactory의 이름을 설정한다.

태그	설명
<xa-support>	해당 MQ가 XA를 지원하는지의 여부를 설정한다. 만일 이 값을 true로 설정해두어도 실제 ConnnectionFactory가 지원하지 않을 경우에는 XA를 사용할 수 없다.

- <bridges><bridge>

<bridge-connection>에서 설정한 MQ들간의 연결(Bridge)을 나타내는 설정이다. 이때 하위 <bridge> 태그는 하나의 Bridge를 의미한다.

태그	설명
<name>	해당 Bridge의 이름을 나타낸다.
<source>	해당 Bridge가 메시지를 받아올 MQ와 그 MQ의 특정 Destination을 나타낸다. <ul style="list-style-type: none"> <li>◦ &lt;connection-name&gt;: &lt;bridge-connection&gt;에 설정한 이름들 중 하나를 설정한다.</li> <li>◦ &lt;destination&gt;: 해당 MQ의 특정 Destination을 설정한다.</li> <li>◦ &lt;type&gt;: 해당 Destination의 타입을 설정한다. queue와 topic 중 하나를 지정할 수 있다.</li> </ul>
<target>	해당 Bridge가 받아온 메시지를 전달할 MQ와 그 MQ의 특정 Destination을 나타낸다. 설정 방법은 <source>와 동일하다.

## 6.2. JEUS MQ Message Sort

어떤 큐에 메시지 송신자만 있고 수신자가 없거나 또는 수신자가 있지만 수신하는 속도가 상대적으로 느릴 때 해당 큐에는 메시지가 쌓일 수 있다. 이런 경우에 쌓이는 메시지를 사용자가 지정한 키값의 순서대로 정렬해주는 것이 **Message Sort 기능**이다. 큐와 마찬가지로 Durable Subscription의 경우에도 메시지가 쌓일 수 있는데 이 경우에도 Message Sort 기능을 이용할 수 있다. 지정한 키값에는 JMS의 기본적인 프로퍼티값 또는 사용자가 지정한 프로퍼티(User Property) 값들이 포함된다.

Message Sort 설정은 서버 설정 방법과 클라이언트 설정 방법이 다르다. Message Sort와 관련된 서버 설정은 domain.xml을 사용하고, 클라이언트 설정은 정렬 대상 메시지에 프로퍼티 값으로 설정한다.

### 6.2.1. 서버 설정

Message Sort 기능을 사용하려면 기준을 설정해야 한다. 이후에 Destination이나 Durable Subscription에서 위의 기준으로 만들어진 Message Sort 기능을 사용하도록 설정할 수 있다. 자세한 내용은 [Destination 설정](#)과 [Durable Subscription 설정](#)을 참고한다.

다음은 메시지 정렬 기능을 사용하기 위해서 서버에 설정한 예시이다.

```

<domain>
  ...
  <jms-engine>
    <message-sort>
      <name>example</name>
      <key>TEST</key>
      <type>Integer</type>
      <direction>ascending</direction>
    </message-sort>
  </jms-engine>

  <jms-resource>
    <destination>
      <type>queue</type>
      <name>ExamplesQueue</name>
      <message-sort>example</message-sort>
    </destination>
  </jms-resource>
</domain>

```

**<message-sort>** 태그는 정렬의 키값이 될 프로퍼티 및 프로퍼티의 타입, 정렬의 방향 등을 설정하고, Destination이나 Durable Subscriber에 이 <message-sort>의 이름을 추가하면 해당 설정이 적용된다.

다음은 <message-sort>의 설정 태그에 대한 설명이다.

태그	설명
<name>	<message-sort> 설정을 나타낼 이름을 정의한다. 이 이름을 큐나 Durable Subscriber의 <message-sort> 설정에 추가하면 이 설정이 적용된다.
<key>	정렬의 기준이 될 키의 이름을 정의한다.
<type>	키값의 타입을 설정한다. Boolean, Byte, Float, Integer, Double, String을 설정할 수 있다. (기본값: String)
<direction>	정렬의 방향을 지정한다. ascending과 descending 중에서 설정한다.

지정한 키값에 해당하지 않는 메시지들은 정렬의 대상이 되지 않는다. 이 경우에는 최대한 원래의 순서가 보장된다. 즉, 키값에 해당하는 메시지와 그렇지 않은 메시지들이 혼합되어서 큐에 쌓일 경우 키값에 해당하는 메시지들은 설정에 따라 정렬이 되고 그렇지 않은 메시지들은 메시지들 간의 원래의 순서가 보장된다.

## 6.2.2. 클라이언트 설정

클라이언트에서는 대상이 될 메시지에 정렬의 기준이 될 키값을 프로퍼티로 설정한다. 서버의 Message Sort 설정에서 Message Sort 키를 "TEST\_KEY"로 설정했다는 가정을 하고 다음의 예제와 같이 설정하면 메시지들이 정렬된다.

```

Message msg = session.createTextMessage("Test");
msg.setIntProperty("TEST_KEY", 1);

```

## 6.3. JEUS MQ Global Order

Global Order는 Destination에 쌓인 메시지가 반드시 하나씩 클라이언트로 전달되도록 제한하여 엄격한 처리 순서를 보장하는 기능이다. 일반적인 경우에는 메시지가 여러 클라이언트에서 병렬로 처리될 가능성이 있고, 이 경우에 각각의 처리 순서는 제한할 방법이 없다. 즉, 보통의 경우 병렬로 여러 메시지 수신자를 붙이면 메시지의 처리 순서가 메시지를 송신한 순서와는 상관없이 처리된다. 이런 동작이 스펙 측면에서는 전혀 문제가 없지만, 이를 보완하여 처리 순서를 보장해야 할 필요가 있을 때 사용되는 기능이다.

### 6.3.1. 클라이언트 설정

Global Order 기능은 서버에 별다른 설정 없이 클라이언트에서 전용 API를 호출하여 사용할 수 있다.



Global Order는 메시지 수신자가 하나일 경우에는 기존과 동작의 차이가 거의 없다. 따라서 Global Order 기능을 사용할 때에는 Destination에 여러 메시지 수신자가 연결되도록 설정해야 한다.

메시지 송신자에서 Producer를 생성한 후 JeusMessageProducer로 casting하여 전용 API를 호출한다.

```
JeusMessageProducer producer = (JeusMessageProducer) session.createProducer(queue);  
// Global Order 지정 및 이름 할당  
producer.startGlobalOrder("GLOBAL-ORDER-NAME");  
// 이름을 지정하지 않는 경우  
//producer.startGlobalOrder();
```

GLOBAL-ORDER-NAME은 각 Global Order를 구별해주는 이름으로 할당하지 않는 경우 임의의 이름이 할당된다. 이후 메시지 송신은 기존과 같다. Global Order를 구별해주는 이름은 여러 클라이언트에 걸쳐서 공유하여 사용할 수 있다. 이 기능을 클러스터링과 함께 사용하면 클러스터 전체에 대해서 처리 순서를 최대한 보장한다.

## 6.4. JEUS MQ Message Group

Message Group은 특정한 목적을 가진 그룹을 설정해서 Destination에 해당 그룹에 속한 메시지가 모두 모였을 때 그 그룹의 메시지를 한 메시지 수신자에게 모두 전달하는 기능이다. 예를 들어 총 10 개짜리 크기의 그룹을 설정하면 해당 그룹에 10개의 메시지가 모두 모이기 전까지 그 메시지들은 묶인 상태로 서비스되지 않고, 10번째의 메시지가 도착하는 순간 한 메시지 수신자에게 묶음 메시지가 전달된다.

트랜잭션의 개념과 유사하며, 비슷한 용도로 사용될 수 있다. Message Group은 클러스터링과 병행하여 사용할 수 있으며, 이러한 경우에도 한 메시지 수신자에게 묶음 메시지를 전달할 수 있도록 처리해 준다.

### 6.4.1. 서버 설정

다음은 메시지 그룹 기능을 사용하기 위해서 서버에 설정한 예시이다.



```

<domain>
  ...
  <jms-resource>
    <destination>
      <type>queue</type>
      <name>ExamplesQueue</name>
      <message-group>
        <message-handling>Pass</message-handling>
        <expiration-time>-1</expiration-time>
      </message-group>
    </destination>
  </jms-resource>
</domain>

```

**<message-group>** 태그는 Destination에 적용할 메시지 그룹 설정을 정의한다.

다음은 <message-group>의 설정 태그에 대한 설명이다.

태그	설명
<message-handling>	Destination이 메시지 그룹을 어떻게 다룰 것인지 정의한다. Pass와 Gather를 설정할 수 있다. Pass는 일반 메시지와 동일하게 취급한다. Gather는 메시지 그룹을 완성시켜서 하나의 메시지로 전달한다.
<expiration-time>	Destination에 완성되지 않은 메시지 그룹이 최대 얼마 동안 존재할 수 있는지 초단위로 정의한다. 기본값은 -1로, 완성될 때까지 없어지지 않는다.



**'Expiration Time'**을 설정하지 않고 사용하면 완성되지 않은 Message Group은 영원히 서버에서 사라지지 않고 보관된다. 메모리의 낭비가 발생할 수 있으므로 세심한 주의가 필요하다.

## 6.4.2. 클라이언트 설정

클라이언트 설정은 메시지 송신자와 메시지 수신자 각각의 정보를 설정해야 한다.

- 메시지 송신자 설정

메시지의 User Property를 설정하는 방법을 통해서 기능을 활성화한다.

```

Message msg = session.createMessage();
// Message Group 이름을 설정한다. Group을 대표하는 이름이다.
msg.setStringProperty("JMS_JEUS_MSG_GROUP_NAME", "MESSAGE-GROUP-NAME");

// 해당 Group 내에서의 순서를 나타낸다. 나중에 클라이언트로 전달될 때의 순서이기도 하다.
msg.setIntProperty("JMS_JEUS_MSG_GROUP_NUMBERING", 1);
producer.send(msg);

msg = session.createMessage();
msg.setStringProperty("JMS_JEUS_MSG_GROUP_NAME", "MESSAGE-GROUP-NAME");

```

```
msg.setIntProperty("JMS_JEUS_MSG_GROUP_NUMBERING", 2);
producer.send(msg);
. . .
msg = session.createMessage();
msg.setStringProperty("JMS_JEUS_MSG_GROUP_NAME", "MESSAGE-GROUP-NAME");
msg.setIntProperty("JMS_JEUS_MSG_GROUP_NUMBERING", 10);

// 해당 메시지가 해당 Group의 마지막 메시지임을 나타낸다.
msg.setBooleanProperty("JMS_JEUS_MSG_GROUP_END", true);
producer.send(msg);
```

- 메시지 수신자 설정

완성된 Message Group을 받으면 모든 메시지가 하나로 묶여 목록으로 이루어진 ObjectMessage로 받게 된다.

```
// message group으로 받는 메시지들은 하나의 ObjectMessage이고, 이 ObjectMessage는 Message들의
// List이다.
ObjectMessage result = (ObjectMessage) receiver.receive(TIME_OUT);
List list = (List) result.getObject();

int cnt = 1;
// 다음과 같이 목록에서 하나씩 꺼내와 사용할 수 있다.
for(Object obj : list) {
    // 메시지 송신자에서 설정한 순서대로 메시지를 처리한다.
    TextMessage msg = (TextMessage) obj;
    . . .
}
```

## 6.5. JEUS MQ Message Management 기능

Message Management 기능은 JEUS MQ를 이용해서 메시징 서비스를 사용하는 도중 Destination 내에 들어온 메시지들을 확인, 이동, 삭제 등 메시지 관리를 위해 제공되는 기능이다.

Message Management 기능은 다음과 같이 3가지로 구분된다.

- [메시지 모니터링](#)

Destination 내의 메시지를 모니터링한다.

- [메시지 제어](#)

메시지의 이동, 삭제, 내보내기, 가져오기를 통해 메시지를 제어한다.

- [Destination 모니터링](#)

Destination의 상세 정보를 모니터링한다.

- [Destination 제어](#)

원활한 메시지 모니터링과 제어 작업을 위해 Destination을 제어한다.

이때 메시지 모니터링과 메시지 제어 기능은 현재 서비스 중인 Queue나 Durable subscription에 남아 있는 메시지를 그 대상으로 하고, Destination 제어 기능은 Queue와 Topic 모두를 그 대상으로 한다.

### 6.5.1. 메시지 모니터링

Queue 또는 Durable subscription 내에 들어온 메시지를 모니터링하고 각 메시지 정보를 조회한다.

#### 메시지 목록 조회

Queue 또는 Durable subscription 내에 남아있는 모든 메시지들의 목록을 조회하는 기능이다. 각 메시지들의 ID, 타입, 생성 시각 등의 간단한 정보를 조회하고, 조회된 목록에서 메시지를 선택해서 상세 정보 조회, 삭제, 이동, 내보내기 등의 다른 기능을 수행할 수 있다.



JMS는 메시지가 매우 빠른 속도로 생산되고 소비되기 때문에 Destination의 소비 일시중지 기능을 사용하지 않고 메시지 목록을 조회할 경우 현재 목록에는 조회된 메시지가 이미 소비되어 존재하지 않을 수 있다.

메시지 목록 조회 과정은 다음과 같다.

1. 조회할 메시지의 위치에 따라 jeusadmin의 명령어가 다음과 같이 달라진다.

- Queue 내의 메시지

Destination 목록 조회와 관련된 명령어는 JEUS Reference 안내서의 "Destination 목록 조회"를 참고한다.

- Durable subscription 내의 메시지

Durable Subscription 목록 조회와 관련된 명령어는 JEUS Reference 안내서의 "Durable Subscription 목록 조회"를 참고한다.

2. Queue 또는 Durable subscriber 내의 메시지 목록을 조회할 수 있는 기능도 있다.

메시지 목록 조회 기능을 이용할 때에는 메시지 ID나 타입, 생성 시각으로 필터링할 수도 있고, JMS 스펙에서 지정하는 Message Selector에 맞춰 특정 메시지만 선택해서 조회할 수도 있다. Queue 또는 Durable subscription 내의 모든 메시지를 조회하려면 각 파라미터를 설정하지 않은 상태로 명령어를 입력한다.

3. 메시지 조회 목록에서 원하는 메시지들을 삭제, 이동, 내보내기 등의 기능을 사용할 수 있으며, 각 메시지의 상세 내용을 확인할 수도 있다.

#### 메시지 상세 정보 조회

메시지 상세 정보 조회는 Queue 또는 Durable subscription으로 들어온 메시지의 상세 정보를 조회하는 기능이다. JMS 스펙에서 지정하는 헤더들의 모든 내용을 확인할 수 있으며 메시지의 정보와 설정된 속성들을 확인할 수 있다.



JMS는 메시지가 매우 빠른 속도로 생산되고 소비되기 때문에 Destination의 소비 일시중지

기능을 사용하지 않고, 메시지 상세 정보 조회 기능을 사용할 경우 현재 목록에는 조회된 메시지가 이미 소비되어 존재하지 않을 수 있다.

메시지 상세 정보 조회와 관련된 명령어는 JEUS Reference 안내서의 "메시지 상세 정보"를 참고한다.

## 6.5.2. 메시지 제어

JeusAdmin에서는 서버에 들어온 메시지에 대해 이동, 삭제, 내보내기, 가져오기와 같은 제어 기능을 제공한다.

### 메시지 이동

서버 또는 클러스터 내의 다른 Destination으로 메시지를 이동시킬 수 있다. 잘못된 Destination으로 메시지를 보내는 등의 실수 상황에 대처할 수 있는 유용한 기능이다.



1. JMS는 메시지가 매우 빠른 속도로 생산되고 소비되기 때문에 원본 메시지가 존재하는 Destination의 소비 일시중지 기능을 사용하지 않을 경우 이동시킬 메시지가 이미 해당 Destination에 존재하지 않을 수 있다.
2. 메시지 이동은 메시지를 그대로 다른 Destination에 옮기는 기능이다. 만약 대상 Destination에 이동시킬 메시지와 같은 ID를 가진 메시지가 이미 존재한다면 이동할 메시지가 이전 메시지를 덮어쓴다. 대상 Destination에서 같은 ID를 가진 메시지가 존재해서 덮어쓰려 할 때 해당 메시지가 이미 클라이언트로 전달되었다면 서버와 클라이언트 간에 메시지가 서로 맞지 않는 상황이 발생할 수 있다. 그러므로 메시지를 이동시킬 때에는 두 Destination이 모두 소비 일시중지 상태인 경우에 사용할 것을 권장한다.

메시지 이동과 관련된 명령어는 JEUS Reference 안내서의 "메시지 이동"을 참고한다.

'All' 옵션을 넣으면 현재 대상이 되는 Queue 또는 Durable subscriber의 모든 메시지를 대상 Destination으로 이동시킨다.

### 메시지 삭제

메시지 삭제는 서버 내의 메시지가 더 이상 필요하지 않게 된 경우에 사용하는 기능이다.



JMS는 메시지가 매우 빠른 속도로 생산되고 소비되기 때문에 Destination의 소비 일시중지 기능을 사용하지 않으면 메시지 목록 조회 화면에서 확인이 된 메시지로도 실제로는 이미 소비되어 삭제할 수 없는 경우가 발생할 수 있다. 따라서 정확한 메시지 삭제를 위해서는 해당 Destination이 소비 일시중지 상태에서 사용할 것을 권장한다.

메시지 삭제와 관련된 명령어는 JEUS Reference 안내서의 "메시지 삭제"를 참고한다.

현재 Destination 내의 모든 메시지를 삭제하려면 'All' 옵션을 넣는다.

## 메시지 내보내기

Queue 또는 Durable subscriber 내의 메시지를 다른 서버나 클러스터로 옮길 수 있는 형태로 내보내는 기능이다.



JMS는 메시지가 매우 빠른 속도로 생산되고 소비되기 때문에 목록에는 조회되어도 메시지가 이미 소비되었을 수도 있다. 그러므로 원하는 메시지들을 제대로 내보내기 위해서는 Destination이 소비 일시중지 상태일 때 사용해야 한다.

메시지 내보내기와 관련된 명령어는 JEUS Reference 안내서의 "메시지 Export"를 참고한다.

만약 '**All**' 옵션을 넣으면 해당 Destination이나 Durable subscription 내의 모든 메시지를 내보낼 수 있다. 결과적으로 메시지 내보내기가 수행되며 브라우저를 통해 "messages.xml"이라는 파일을 다운로드 받는다.

## 메시지 가져오기

메시지 내보내기 기능을 수행하면 생성되는 XML 파일을 이용해서 서버 내의 특정 Destination으로 메시지를 가져올 수 있다. 메시지 이동과는 달리 전혀 새로운 메시지로 취급되며, 메시지가 Destination으로 들어오는 순간 새로운 Message ID가 발급된다.



메시지 가져오기에서 '**Overwrite**' 항목을 체크해서 기존 메시지를 대체할 때 메시지가 이미 클라이언트로 전달되었을 가능성이 있다. 이 경우에는 가져오기는 성공하지만 덮어쓰기는 제대로 수행되지 않는다. 따라서 이 기능을 사용할 때에는 해당 Destination이 소비 일시중지 상태일 때 사용할 것을 권장한다.

메시지 가져오기와 관련된 명령어는 JEUS Reference 안내서의 "메시지 Import"를 참고한다.

참고로 '**Overwrite**' 옵션을 넣으면 이전에 있던 메시지의 ID를 그대로 사용하고, 만약 대상이 되는 Destination 내에 같은 ID를 가진 메시지가 이미 존재한다면 그 메시지를 덮어쓴다.

## 6.5.3. Destination 모니터링

JeusAdmin에서는 다음과 같이 Destination별 상세 정보와 통계를 모니터링할 수 있다.

항목	설명
Processed Messages	해당 Destination에서 처리한 메시지이다.
Remaining Messages (current)	해당 Destination에 현재 남아있는 메시지이다.
Remaining Messages (high mark)	해당 Destination에 남아있는 메시지의 최댓값이다.
Pending Messages	해당 Destination에 들어오고 아직 Receiver에게 전달되지 않은 메시지이다. Destination이 Queue일 때만 조회할 수 있다.
Dispatched Messages	해당 Destination에서 Receiver에게 전달 된 후 아직 응답이 돌아 오지 않은 메시지이다. Destination이 Queue일 때만 조회할 수 있다.

항목	설명
Delivered Messages	해당 Destination에 들어 온 메시지 중 Receiver로 전달이 끝난 메시지이다.
Expired Messages	해당 Destination에 들어 온 메시지 중 만료된 메시지이다.
Moved Messages	해당 Destination에서 다른 Destination으로 옮겨간 메시지이다. Destination이 Queue일 때만 조회할 수 있다.
Removed Messages	해당 Destination으로 들어온 메시지 중 지워진 메시지이다. Destination이 Queue일 때만 조회할 수 있다.
Poisoned Messages	해당 Destination으로 들어온 메시지 중 잘못 처리된 메시지이다.
Memory Usage (current)	해당 Destination에서 현재 사용중인 메모리 용량이다.
Memory Usage (high mark)	해당 Destination에서 사용한 메모리의 최댓값이다.

## 6.5.4. Destination 제어

JeusAdmin에서는 Destination별로 생산 및 소비 서비스를 일시중지하거나 재개할 수 있다.

### Destination의 생산 일시중지 및 재개

해당 Destination으로 메시지를 생산하는 것을 일시중지하거나 재개하는 기능이다. Destination이 생산 일시중지 상태가 되면 해당 Destination으로 메시지를 생산하는 것이 불가능해진다.

Destination의 생산 일시중지 및 재개와 관련된 명령어는 JEUS Reference 안내서의 "Destination의 상태 제어"를 참고한다.



메시지 생산이 일시중지된 상태에서 Destination으로 메시지를 보낼 경우 일정 시간 기다리다가 `ExceptionListener`를 통해 `jeus.jms.common.destination.InvalidDestinationStateException`을 통보한다.

### Destination의 소비 일시중지 및 재개

해당 Destination에서 메시지 소비를 일시중지하거나 재개하는 기능이다. Destination의 소비가 일시중지 상태가 되면 해당 Destination에서 메시지를 소비하는 것이 불가능해진다.

Destination의 소비 일시중지 및 재개와 관련된 명령어는 JEUS Reference 안내서의 "Destination의 상태 제어"를 참고한다.



메시지 소비가 일시중지된 상태에서 Destination에서 메시지를 소비하려할 경우 해당 Destination에 메시지가 존재하지 않을 때와 동일하게 동작한다.

## 6.6. JEUS MQ Topic Multicast

Topic Multicast는 Topic으로 들어온 메시지를 Subscriber로 보낼 때 UDP방식을 이용한 IP Multicast로 전송하는 기능이다. Multicast 방식으로 메시지를 전송하기 때문에 pub-sub 구조인 Topic을 사용할 때 성능상의 큰 이점이 있으나, UDP로 전송하는 만큼 전송의 신뢰도는 떨어질 수 있다.

Topic Multicast 기능은 IP Multicast를 사용할 수 있는 환경에서만 사용이 가능하다. IP Multicast를 사용할 수 있는 환경인지 테스트하는 방법은 JEUS Domain 안내서의 "Virtual Multicast를 사용하는 도메인 생성"을 참고한다.

### 6.6.1. 서버 설정

IP Multicast를 통해 Topic 메시지를 주고받기 위해서는 두 가지 설정이 필요하다. 먼저 서버와 클라이언트가 같은 Multicast 주소로 연결하도록 설정해야 하며, 서버에 연결된 클라이언트가 메시지를 받는 방식을 Topic Multicast로 지정하도록 설정해야 한다.

Multicast 주소는 Destination 설정에 설정하도록 되어 있다. Topic인 Destination 설정에 Multicast 주소를 포함시켜 해당 Destination을 사용하는 클라이언트에서 별도의 클라이언트 설정 없이 서버와 동일한 Multicast 주소를 통해 메시지를 받을 수 있게 되어 있다.

클라이언트가 메시지를 받는 방식을 Topic Multicast로 지정하는 설정은 ConnectionFactory에 설정하고 있다. Topic Multicast 기능을 사용하려면 Topic Multicast가 설정된 ConnectionFactory를 사용한다. 메시지를 받아오려면 Destination에 Topic Multicast 설정이 되어 있지 않다면 TCP 방식(기존 방식)으로 메시지를 받아오게 된다. 마찬가지로 Topic Multicast가 설정된 Destination에서 메시지를 받아오더라도 ConnectionFactory에 Topic Multicast 설정이 되어있지 않다면, TCP 방식으로 메시지를 받아온다. Durable subscription을 사용하는 경우 일반 Topic 메시지보다 높은 수준의 신뢰도를 요구하며, 1개의 Subscription이 동시에 1개의 클라이언트에만 메시지를 전달하기 때문에 Topic Multicast 기능을 지원하지 않는다.

## 6.7. 신뢰도 높은 메시지 송신

JEUS MQ를 이용해 메시지를 전달할 때 Persistent Store를 사용하면, 서버의 장애가 발생 했을 때에도 높은 신뢰도의 메시지 전달을 보장할 수 있다. 하지만 이것은 JEUS MQ 서버와 메시지 수신자간의 메시지 전달만을 보장하고 있으며, 메시지 송신자가 서버로 전달할 때 서버 혹은 클라이언트의 장애가 발생했을 때의 메시지 전달은 보장하지 못하고 있다.

이를 보완하기 위해서 JEUS MQ에서는 Persistence Store와 유사한 기능을 하는 JEUS의 Local Persistent Queue(이하 LPQ)를 이용하여 메시지 송신자와 JEUS MQ 서버 간의 신뢰도를 높이는 기능을 제공하고 있다. Local Persistent Queue는 데이터를 로컬의 저장소에 저장한 후, 저장된 데이터는 반드시 주어진 작업대로 처리되는 것을 보장해주는 서비스이다. JEUS MQ에서는 LPQ를 이용하여 전송하고자 하는 메시지를 전송하기 전에 저장해놓고, 메시지가 서버에 전송될 때까지 전송을 시도하여 메시지 송신자에서 서버로의 전송에 대한 신뢰도를 높이고 있다.

LPQ를 이용한 신뢰도 높은 메시지 송신은 다음과 같은 특징이 있다.

- 서버 상태에 상관없이 메시지 전송에 성공할 때까지 메시지 재전송
- 클라이언트의 상태 이상으로 전송 실패한 메시지 복구
- 비동기 메시지 전송



JMS 클라이언트가 메시지를 보낼 때 해당 클라이언트가 Jakarta EE 애플리케이션이고, 메시지를 받는 JMS 서버가 같은 해당 클라이언트가 deploy된 서버라면, 성능 향상을 위해 네트워크를 통하지 않고 메시지를 전달하기 때문에 LPQ에 관한 설정은 무시된다.

### 6.7.1. LPQ 활성화

LPQ를 활용하여 신뢰도 높은 메시지 송신 기능을 사용하려면 먼저 LPQ를 활성화시켜야 한다. LPQ를 활성화시키는 과정은 메시지들을 저장할 저장소 생성, 메시지 처리를 위한 큐와 그리고 이들을 관리하기 위한 관리 객체 생성으로 이뤄져 있다.



LPQ를 사용하려는 stand alone 클라이언트는 실행할 때 LPQ에 관한 라이브러리와 저장소에 관한 라이브러리가 추가로 필요하다. 이들은 각각 jeus-lpq-spi.jar, jeus-lpq.jar, jms-extensions.jar와 jeus-store.jar이며, JEUS\_HOME/lib/system 폴더에 존재한다.

LPQ를 활성화시키는 방법은 크게 3가지로 나뉜다.

- JVM 옵션을 통한 활성화

JEUS MQ 클라이언트를 실행시킬 때에 다음과 같은 JVM 옵션을 적용하여 LPQ를 활성화시킬 수 있다. 이 옵션을 사용하면 클라이언트 내에서 JMS가 Connection을 생성할 때 LPQ가 활성화된다.

```
-Djeus.jms.client.send-by-lpq-only=true
```

- LPQ 설정 파일을 통한 활성화

JEUS MQ 클라이언트를 실행시킬 때 LPQ를 위한 설정 파일이 지정된 위치에 있다면, 해당 파일을 읽어 그 설정대로 LPQ를 활성화한다. LPQ 설정 파일의 위치와 설정 방법에 관한 더 자세한 설명은 [LPQ 설정](#)을 참고한다.

- JEUS 전용 API를 통한 활성화

클라이언트 소스에서 JEUS 전용 세션 객체인 JeusSession으로부터 다음 API를 사용하여 LPQ를 활성화시킬 수 있다.

```
public void startLPQ();
```



1. JVM 옵션을 통한 경우나 JEUS 전용 API를 통한 경우에도 설정 파일이나 시스템 프로퍼티를 통해 LPQ의 동작을 설정할 수 있다. 설정되지 않은 항목에 대해서는 기본값이 적용된다. LPQ 설정에 관한 더 자세한 설명은 [LPQ 설정](#)을 참고한다.
2. JEUS 전용 API를 통한 경우가 아니라면 LPQ가 활성화되는 시점이 명시되지 않는데, 이때는 해당 JVM에서 최초의 Connection이 생성될 때 LPQ가 활성화된다. 또한, LPQ가 동작을 멈추는 시점은 마지막 Connection이 close될 때이다. 단, 이때 LPQ 내부에 들어온 메시지가 있다면 이들이 처리될 때까지 기다리게 된다.



## 6.7.2. LPQ 사용 설정

LPQ가 활성화 되었다면 메시지를 보낼 때 이를 LPQ를 통해 전송되도록 하여 메시지 송신의 신뢰도를 높일 수 있다. 이때 메시지가 LPQ를 통해 전송되도록 설정하는 단위는 다음과 같이 네가지로 나눌 수 있다.

### • JVM 단위 설정

위 LPQ 활성화 방법 중 JVM 옵션을 통해 활성화한 경우, 해당 JVM에서 보내는 모든 메시지는 LPQ를 통해 전송된다.

### • ConnectionFactory 단위 설정

JVM 옵션에 아래와 같이 LPQ를 사용할 Connection Factory 이름들을 추가하면, 해당 Connection Factory를 이용해 만들어진 Connection에서 전송되는 모든 메시지는 LPQ를 통해 보내게 된다. 각 Connection Factory의 이름은 콤마(,)로 구분한다.

```
jeus.jms.client.connection-factory-for-lpq=<ConnectionFactoryName1,ConnectionFactoryName2,...>
```



JMS 스펙 상 Connection Factory 이름에 대한 제한점은 없지만, 구분자인 콤마(,)가 들어가 있다면 위 옵션을 정상적으로 적용할 수 없어 이를 통해서는 LPQ 기능을 사용할 수 없다.

### • Session 단위 설정

Session별로 LPQ를 통해 보낼지를 지정하는 방법으로 JEUS 전용 메시지 송신자 객체의 다음 API를 사용하면, 해당 Session으로 보내는 모든 메시지를 LPQ를 통해 보낼지를 설정할 수 있다.

```
public void setLPQOnly(boolean lpqOnly);
```

### • 메시지 단위 설정

메시지별로 LPQ를 통해 보낼지를 지정하는 방법으로 JEUS 전용 메시지 송신자 객체의 다음 API들을 사용하거나 메시지의 User Property를 설정하는 방법이 있다.

```
// JEUS 전용 API
public void sendWithLPQ(Message message);
public void sendWithLPQ(Message message, int deliveryMode, int priority, timeToLive);
public void sendWithLPQ(Destination destination, Message message);
public void sendWithLPQ(Destination destination, Message message, int deliveryMode, int priority, timeToLive);

// JMS 메시지의 User Property
Message msg = session.createMessage();
msg.setBooleanProperty("JMS_JEUS_USE_LPQ", true);
```



JEUS 전용 API를 사용하여 LPQ를 활성화하는 경우는 해당 세션에서만 LPQ를 사용할 수 있기 때문에 그 이외의 세션을 통해 보내는 메시지에는 LPQ 설정을 하더라도 무시된다.

### 6.7.3. LPQ 리스너 설정

메시지 전송에 LPQ를 사용하면 메시지가 비동기적으로 처리되어 실제 메시지가 전송되는 것을 클라이언트에서 알 수 없다. 때문에 메시지 전송의 결과와 정확한 시점을 알 수 있도록 LPQ에서는 리스너를 제공하고 있다.

리스너를 통해 메시지 전송의 결과를 받기 위해 interface로 제공되는 리스너인 `jeus.jms.LPQMessageListener`는 다음과 같다.

```
package jeus.jms;

public interface LPQForwardListener {
    /**
     * 메시지의 전송이 완료되었을 때 이벤트
     * @param message 전송된 메시지
     */
    public void onComplete(Message message);

    /**
     * 메시지의 전송 도중 Exception이 발생했을 때 이벤트
     * @param message 전송하려던 메시지
     * @param e 발생한 Exception
     */
    public void onException(Message message, Exception e);

    /**
     * 메시지의 전송이 실패하였을 때 이벤트
     *
     * @param message 전송이 실패한 메시지
     * @param cause 전송이 실패한 원인
     */
    public void onFailure(Message message, Throwable cause);
}
```

구현된 `LPQMessageListener`를 JMS 세션의 JEUS 전용 객체인 `JeusSession`에 다음 API를 사용하여 등록하면 메시지 전송의 결과를 제공받을 수 있다.

```
public void setLPQMessageListener(jeus.jms.LPQMessageListener lpqMessageListener);
```

또한 JEUS 전용 객체를 사용하여 LPQ를 활성화하는 경우 `JeusSession`의 다음 API를 사용하여 LPQ 활성화와 리스너 등록을 동시에 할 수도 있다.

```
public void startLPQ(jeus.jms.LPQMessageListener lpqMessageListener);
```

## 6.7.4. LPQ 설정

LPQ는 실패한 전송에 대한 처리, 연결이 끊어졌을 때의 동작, 저장소의 위치 또는 크기에 대한 설정들이 필요하다.

### LPQ 설정 항목

다음은 LPQ 설정 항목에 대한 설명이다.

- 공통 항목

항목	타입	설명
jeus.lpq.name	String	사용하는 LPQ의 이름을 설정한다. (기본값: JEUS_LPQ)
jeus.lpq.max-message-count	int	LPQ가 한 번에 처리할 수 있는 최대 메시지 수를 설정한다. (기본값: 819200)
jeus.lpq.time-to-live	long	LPQ 내의 메시지들이 최대 얼마동안 존재할 수 있는지를 설정한다. (기본값: 43200000ms (12시간))

- 전달 관련 항목

항목	타입	설명
jeus.lpq.retry-limit	int	전달에 실패한 메시지의 재전송을 시도하는 횟수를 설정한다. 0 이하의 값으로 설정된 경우 무한히 재시도한다. (기본값: -1)
jeus.lpq.retry-interval	long	메시지를 재전송할 때 매번 전송 시도 간 간격을 설정한다. (기본값: 1000ms)
jeus.lpq.retry-interval-increment	long	메시지를 재전송할 때 매번 전송 시도 간 간격의 증가량을 설정한다. (기본값: 0ms)

- 재연결 관련 항목

항목	타입	설명
jeus.lpq.reconnect-retry-interval	long	연결이 끊어졌을 때 재연결 시도간의 간격을 설정한다. (기본값: 5000ms)

- 저장소 관련 항목

항목	타입	설명
jeus.lpq.store.store-mode	int	사용할 저장소의 형태를 설정한다. 1은 journal store를 사용하고 2는 메모리의 저장소를 사용한다. (기본값: 1)

항목	타입	설명
jeus.lpq.store.journal.store-base-dir	String	Store를 생성할 디렉터리 이름을 설정한다. 이 디렉터리 이름은 각 LPQ 설정별로 유일해야 하며 동시의 2개 이상의 접근은 허용하지 않는다.  상대 경로로 설정되는 경우 설정 파일이 발견된 위치나 JVM이 실행된 위치의 하위로 설정된다.  (기본값: JEUS_LPQ_STORE)
jeus.lpq.store.journal.initial-log-file-count	int	Journal Store를 생성할 때 초기에 생성할 로그 파일들의 개수를 설정한다. (기본값: 2)
jeus.lpq.store.journal.max-log-file-count	int	최대로 생성할 로그 파일들의 개수를 설정한다.  (기본값: 10)
jeus.lpq.store.journal.log-file-size	String	로그 파일의 크기를 지정한다. (기본값: 64M)  Integer 타입의 값이나 숫자 뒤에 다음의 문자를 붙여 설정할 수 있다.  <ul style="list-style-type: none"> <li>• 'K'(KiloBytes)</li> <li>• 'M'(MegaBytes)</li> <li>• 'G'(GigaBytes)</li> </ul>

## LPQ 설정 방법

다음은 LPQ 설정 방법에 대한 설명이다. 설정 방법 간의 우선순위는 런타임 프로퍼티 설정 > 설정 파일 > JVM 옵션 순이다.

### • 설정 파일

LPQ 설정 파일이 지정된 위치에 존재한다면 LPQ가 활성화될 때 그 파일을 읽어 활성화되는 LPQ에 해당 설정을 적용시킨다. 지정된 위치는 해당 application이 설치된 위치인 DEPLOYED\_HOME/myApp/WEB-INF/, DEPLOYED\_HOME/myApp/META-INF/, DEPLOYED\_HOME/myApp/으로 상기한 순서대로 설정 파일을 찾는다. 설정 파일의 세부 경로의 기본값은 'jeus-lpq.properties'으로 다음 옵션을 통해 변경할 수 있다.

```
-Djeus.jms.client.lpq-configuration-path=jeus-lpq.properties
```

다음은 LPQ 설정 파일의 예이다. 이 예제 파일은 JEUS\_HOME/templates/lpq/에 jeus-lpq.properties라는 이름으로 제공되고 있다.

```
#JEUS Local-Persistent-Queue Configuration
#[commons]
jeus.lpq.name=JEUS_LPQ
jeus.lpq.max-message-count=819200
jeus.lpq.time-to-live=43200000
#[forward]
```

```
jeus.lpq.retry-limit=-1
jeus.lpq.retry-interval=1000
jeus.lpq.retry-interval-increment=0

#[reconnect]
jeus.lpq.reconnect-retry-interval=5000

#[store]
jeus.lpq.store.store-mode=1

#[journal-store]
jeus.lpq.store.journal.store-base-dir=JEUS_LPQ
jeus.lpq.store.journal.max-log-file-count=10
jeus.lpq.store.journal.initial-log-file-count=2
jeus.lpq.store.journal.log-file-size=64M
```

## • 시스템 프로퍼티

위 설정 항목들은 각 항목명을 key로 하여 시스템 프로퍼티로도 설정할 수 있다. 예를 들어 LPQ의 이름을 설정한다면 JEUS MQ 클라이언트를 실행할 때 다음과 같이 JVM 옵션을 설정할 수 있다.

```
-Djeus.lpq.name=<jeus lpq name>
```

다음과 같이 클라이언트 소스에 코드를 추가하여 런타임에 설정을 변경할 수도 있다.

```
System.setProperty("jeus.lpq.name", <jeus lpq name>);
```

# Appendix A: Journal Store 추가 속성

Journal Store에 추가적으로 설정할 수 있는 속성들에 대해서 설명한다. 이 속성들은 고급 속성들로 플랫폼 사이에 발생할 수 있는 성능 문제나 기능의 상이함을 해결할 때 사용할 수 있다.

- jeus.store.journal.control-file-name

설명	컨트롤 파일(control file)을 다른 이름으로 지정할 때 사용된다. 특별한 경우를 제외하고는 이 설정을 사용할 일은 없다.
기본값	control.dat

- jeus.store.journal.log-file-mode

설명	로그 파일을 열 때 사용하는 파일 모드를 지정한다. 'rw', 'rws', 'rwd' 등이 설정될 수 있으며 'rw'로 설정될 경우 강제로 file force를 실행한다.
기본값	rwd

- jeus.store.journal.max-move-count

설명	로그 파일이 가득차면 필요한 공간 확보를 위해 Overflow 처리를 통해서 레코드들의 위치를 옮긴다. 오랫동안 사용되지 않는 레코드들이 로그 파일에 계속 남게 되면 성능이 저하되므로 제2의 저장소로 옮긴다. 이때 레코드를 옮기는 최대 횟수를 지정한다.
기본값	1

- jeus.store.journal.overflow-factor

설명	현재 사용 중인 로그 파일에 이 값보다 작거나 같은 비율의 빈 공간이 남을 경우 Overflow를 검사한다.
기본값	0.5

- jeus.store.journal.min-buffer-size

설명	레코드들을 쓸 때 한 번에 batch write를 하기 위한 최소 버퍼 사이즈이다.
기본값	4KB

- jeus.store.journal.max-buffer-size

설명	레코드들을 쓸 때 한 번에 batch write를 하기 위한 최대 버퍼 사이즈이다.
기본값	4MB

- jeus.store.journal.use-direct-buffer

<b>설명</b>	FileChannel을 통해서 write할 때 DirectByteBuffer를 사용할지 여부를 설정한다.
<b>기본값</b>	false

- jeus.store.journal.max-waiting-thread-count

<b>설명</b>	버퍼의 사용을 대기하는 최대 스레드 개수이다.
<b>기본값</b>	32

# Appendix B: JDBC Persistence Store 컬럼

JEUS MQ에서 Persistence store를 JDBC로 설정했을때 생성되는 테이블의 컬럼에 대해 설명한다.

## B.1. Destination Table

Destination 정보를 저장할 테이블이다.

항목	타입	설명
DT_ID	BIGINT	Destination의 ID
DT_NAME	VARCHAR(255)	Destination의 이름
DT_QUEUE	BIT	Destination이 Queue인지 Topic인지 구분 <ul style="list-style-type: none"><li>◦ Queue : true</li><li>◦ Topic : false</li></ul>
DT_VALID	BIT	Destination이 유효한 상태인지 구분
DT_LVID	BIGINT	Destination의 현재 버전
DT_DYNAMIC	BIT	Destination이 동적으로 생성되었는지 구분
DT_OBJECT	BLOB	Destination의 바이너리 데이터

## B.2. Durable Subscription Table

Durable Subscription 정보를 저장할 테이블이다.

항목	타입	설명
DS_ID	BIGINT	Durable subscription의 ID
DS_CLIENT_ID	VARCHAR(255)	Durable subscription에 지정된 Client ID
DS_NAME	VARCHAR(255)	Durable subscription의 이름
DS_SELECTOR	VARCHAR(255)	Durable subscription에 지정된 메시지 selector
DS_VALID	BIT	Durable subscription이 유효한 상태인지 구분
DS_LVID	BIGINT	Durable subscription의 현재 버전
DT_ID	BIGINT	Durable subscription과 연결된 Topic의 ID
DT_LVID	BIGINT	Durable subscription과 연결된 Topic의 버전

## B.3. Message Table

Message 정보를 저장할 테이블이다.



항목	타입	설명
MG_ID	BIGINT	메시지의 ID
MG_TYPE	TINYINT	메시지의 타입
MG_LENGTH	INTEGER	메시지의 길이
MG_OBJECT	TINYINT[]	메시지의 바이너리 데이터
MG_STATUS	SMALLINT	메시지의 상태
MG_GLOBAL_ORDER_CLOCK	SMALLINT	메시지에 설정된 Global Order의 시각
MG_PERSISTENT	TINYINT[]	메시지가 Persistent로 설정되어 있는지 구분
DT_ID	BIGINT	메시지가 들어있는 Destination의 ID
DT_LVID	BIT	메시지가 들어있는 Destination의 버전
MG_HEADER_LENGTH	INTEGER	메시지 헤더의 길이
MG_HEADER_OBJECT	TINYINT[]	메시지 헤더의 바이너리 데이터

## B.4. MetaInfo Table

JEUS MQ의 Persistence store에 대한 정보를 저장할 테이블이다.

항목	타입	설명
SERVER_NAME	VARCHAR(255)	JEUS MQ의 서버 이름
VERSION	BIGINT	JEUS MQ의 버전 정보

## B.5. Subscription Message Table

Durable Subscription의 메시지 정보를 저장할 테이블이다.

항목	타입	설명
DM_ID	BIGINT	Durable subscription 메시지의 ID
DM_STATUS	SMALLINT	Durable subscription 메시지의 상태
DM_LVID	BIGINT	Durable subscription 메시지의 현재 버전
MG_ID	BIGINT	실제 메시지의 ID
DS_ID	BIGINT	Durable subscription 메시지가 들어있는 Durable subscription ID

## B.6. Transaction Table

트랜잭션 정보를 저장할 테이블이다.

항목	타입	설명
TR_ID	BIGINT	트랜잭션의 ID
TR_STATUS	TINYINT	트랜잭션의 상태
TR_OBJECT	TINYINT[]	트랜잭션의 바이너리 데이터