

Server 안내서

JEUS 9

TMAXSOFT

저작권 공지

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

회사 정보

(주)티맥스소프트

주소 : 경기도 성남시 분당구 황새울로258번길 29, 티맥스수내타워 8-9층

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	3.1.2	2025-01-03	-
JEUS 9	3.1.1	2024-09-27	-

목차

1. 소개	1
1.1. 구성 요소	1
1.1.1. Master Server(Master)	2
1.1.2. Managed Server(MS)	3
1.2. 클래스 로더의 구조	8
1.3. 서버 디렉터리 구조	9
1.4. Launcher	11
2. JEUS 설정	12
2.1. 개요	12
2.2. 서버 추가	12
2.2.1. 콘솔 툴 사용	12
2.3. 서버 설정	13
2.3.1. 환경변수 설정	14
2.3.2. 기본 설정	15
2.3.2.1. 기본 설정 동적으로 변경	15
2.3.2.2. Action On Resource Leak 설정	16
2.3.2.3. Jvm Config 설정	18
2.3.2.4. 클래스 패스 설정	20
2.3.3. Listener 설정	20
2.3.4. Thread Pool 설정	21
2.3.5. Lifecycle Invocation 설정	23
2.3.6. Resource Reference 설정	25
2.4. Server Template 설정	25
3. JEUS 서버 제어 및 모니터링	27
3.1. 서버 제어 및 모니터링	27
3.1.1. Managed Server의 Lifecycle	27
3.1.2. Managed Server 시작	28
3.1.3. Managed Server 종료	30
3.1.4. Managed Server 일시 정지	31
3.1.5. Managed Server 복귀	32
3.2. 서버 엔진 설정	33
3.2.1. 엔진 사용 여부 설정	33
3.2.2. 엔진 초기화 시점 설정	34
3.3. Thread 모니터링 및 제어	35
3.3.1. Thread 모니터링	35
3.3.2. Thread 제어	39
3.4. 메모리 모니터링 및 제어	41
3.4.1. 메모리 모니터링	41
3.4.2. 메모리 사용량에 따른 제어	42

4. JNDI Naming Server	43
4.1. 개요	43
4.2. 기본 개념과 구조	43
4.2.1. 기본 개념	43
4.2.2. 바인딩된 객체 확인	44
4.2.3. JNDI Naming Server 아키텍처	44
4.2.4. JNDI 클러스터링	45
4.3. JNDI Naming Server 설정	47
4.3.1. JNSServer 설정	47
4.3.2. JNSClient 설정	48
4.4. 클러스터링 환경에서 JNDI	49
4.5. JNDI 프로그래밍	50
4.5.1. JEUS 환경설정	50
4.5.2. InitialContext를 위한 프로퍼티 설정	50
4.5.3. Context를 사용한 Named Object의 Lookup	52
4.5.4. Named Object 사용	53
4.5.5. Context 닫기	53
4.5.6. 클러스터링 Context 생성	53
4.5.7. 원격으로 Lookup 실행	54
5. External Resource	56
5.1. 리소스 종류	56
5.2. 리소스 설정	57
5.2.1. 데이터소스 설정	57
5.2.2. 메일 소스 설정	57
5.2.3. URL 소스 설정	58
5.2.4. Custom Resource 설정	58
5.2.5. External Source 설정	61
5.2.6. External Resource 설정	63
6. DB Connection Pool과 JDBC	66
6.1. 개요	66
6.2. 데이터소스와 JDBC Connection Pooling	66
6.2.1. JDBC 드라이버	67
6.2.2. JDBC Connection Pool	67
6.2.3. 데이터소스	68
6.2.4. 클러스터 데이터소스	69
6.2.4.1. 장애복구 기능	69
6.2.4.2. DataSource Affinity 기능	70
6.2.4.3. ONS와 결합된 클러스터 데이터소스	71
6.3. 데이터소스 및 Connection Pool 관리	72
6.4. 데이터소스 설정	74
6.4.1. Connection Pool 설정	77

6.5. 클러스터 데이터소스 설정	83
6.5.1. 클러스터 데이터소스 설정	83
6.5.2. 클러스터 데이터소스에 속한 컴포넌트 데이터소스 설정	86
6.6. 데이터소스 관련 설정 동적 변경	87
6.6.1. 데이터소스 추가	87
6.6.2. 서버에 데이터소스 등록	88
6.6.3. 서버로부터 데이터소스 제거	89
6.6.4. 클러스터에 데이터소스 등록	90
6.6.5. 클러스터로부터 데이터소스 제거	91
6.6.6. 클러스터에 서버 추가	93
6.6.7. 클러스터로부터 서버 삭제	94
6.6.8. 클러스터 삭제	95
6.6.9. 데이터소스 삭제	97
6.6.10. 데이터소스 설정 변경	98
6.6.11. 데이터소스 설정 확인	100
6.7. 클러스터 데이터소스 설정 동적 변경	102
6.7.1. 클러스터 데이터소스 추가	102
6.7.2. 서버에 클러스터 데이터소스 등록	103
6.7.3. 서버로부터 클러스터 데이터소스 제거	104
6.7.4. 클러스터에 클러스터 데이터소스 등록	106
6.7.5. 클러스터로부터 클러스터 데이터소스 제거	107
6.7.6. 클러스터에 서버 추가	108
6.7.7. 클러스터로부터 서버 삭제	108
6.7.8. 클러스터 삭제	108
6.7.9. 클러스터 데이터소스 삭제	108
6.7.10. 클러스터 데이터소스 설정 변경	110
6.7.11. 클러스터 데이터소스 설정 확인	111
6.8. JDBC Connection Pool 모니터링	112
6.8.1. JDBC Connection Pool 목록 확인	112
6.8.2. 특정 JDBC Connection Pool의 상세 정보 확인	114
6.9. JDBC Connection Pool 제어	115
6.9.1. Connection Pool 생성	116
6.9.2. Connection Pool 비활성화	116
6.9.3. Connection Pool 활성화	117
6.9.4. Connection Pool의 Connection 교체	118
6.9.5. Connection Pool의 Connection 개수 최소화	119
6.9.6. Connection Pool의 Connection 연결 반환(Return Connection)	120
6.9.7. Connection Pool의 Connection 강제 끊기(Destroy Connection)	121
6.10. JEUS JDBC 프로그래밍	121
6.10.1. 데이터소스로부터 Connection 얻기	122
6.10.2. 트랜잭션 프로그래밍 규칙	122

6.10.3. JDBC 드라이버의 Connection 구현체 인스턴스 얻기	122
6.10.4. Standalone 클라이언트에서의 Connection Pool	123
6.11. 글로벌 트랜잭션(XA)과 Connection Sharing	123
6.12. 여러 사용자에게 대한 Connection Pooling 서비스 지원	123
7. 트랜잭션 매니저	125
7.1. 개요	125
7.1.1. 애플리케이션	126
7.1.2. JEUS 트랜잭션 매니저	126
7.1.3. 리소스 매니저	127
7.2. 서버 트랜잭션 매니저 설정	128
7.2.1. Worker Thread Pool 설정	128
7.2.2. 타임아웃 설정	130
7.2.3. Root Coordinator와 Sub Coordinator 관련 설정	131
7.2.4. 트랜잭션 Join 설정	132
7.3. 클라이언트 트랜잭션 매니저 설정	132
7.3.1. 트랜잭션 매니저 사용 유무 설정	132
7.3.2. 트랜잭션 매니저 타입 설정	133
7.3.3. 트랜잭션 매니저의 TCP/IP Port 설정	133
7.3.4. Worker Thread Pool 설정	133
7.3.5. 타임아웃 설정	133
7.4. 트랜잭션 애플리케이션 작성	135
7.4.1. 로컬 트랜잭션(Local Transaction)	135
7.4.2. 클라이언트 관리 트랜잭션(Client Managed Transaction)	137
7.4.3. Bean 관리 트랜잭션(Bean Managed Transaction)	139
7.4.4. 컨테이너 관리 트랜잭션(Container Managed Transaction)	141
7.4.5. 트랜잭션 매니저 사용	143
7.5. 트랜잭션 복구	143
7.5.1. 트랜잭션 복구 과정	143
7.5.2. 복구 관련 로그 파일	145
7.5.3. 복구 관련 설정	146
7.5.4. 리소스 매니저 장애	146
7.6. 트랜잭션 프로파일 기능	146
7.7. IP 대역이 다른 서버 간에 트랜잭션 통신 문제	149
8. Logging	150
8.1. 개요	150
8.2. JEUS 로거 기본 구조	152
8.2.1. 개요	152
8.2.2. Launcher 로거	154
8.2.3. 서버 로거	155
8.2.4. 액세스 로거	157
8.2.5. 사용자 로거	158

8.2.6. 로거 리스트	158
8.2.7. 로그 메시지 모듈 이름	162
8.3. 로깅 설정	163
8.3.1. 로거 정보 확인	163
8.3.2. 동적으로 로거 설정	165
8.3.3. 표준 출력과 표준 에러를 로그 형식으로 출력 설정	166
8.3.4. 로거 설정	168
8.3.5. 로그 파일 로테이션 설정	174
8.3.6. 프로퍼티 설정	174
Appendix A: JEUS에서 사용하는 Port	176
A.1. 서버 Port	176
Appendix B: JDBC 데이터소스 구성 예제	177
B.1. 개요	177
B.2. Oracle Thin(Type4) 구성 예제	177
B.2.1. Oracle Thin Connection Pool 데이터소스	177
B.2.2. Oracle Thin XA 데이터소스	178
B.2.3. java.util.Properties 설정 예제 with Oracle ASO	179
B.3. Oracle OCI (Type2) 구성 예제	180
B.3.1. Oracle OCI Connection Pool 데이터소스	180
B.4. DB2 구성 예제	181
B.4.1. DB2 Type4(JCC) Connection Pool 데이터소스	181
B.4.2. DB2 Type4(JCC) XA 데이터소스	181
B.4.3. DB2 Type2(JCC) XA 데이터소스	182
B.5. Sybase 구성 예제	183
B.5.1. Sybase jConnect 5.x Connection Pool 데이터소스	183
B.5.2. Sybase jConnect 6.x XA 데이터소스	183
B.6. Microsoft SQL Server 구성 예제	184
B.6.1. Microsoft SQL Server 2005 Connection Pool 데이터소스	184
B.7. Informix 구성 예제	185
B.7.1. Informix Connection Pool 데이터소스	185
B.8. Tibero 구성 예제	186
B.8.1. Tibero Connection Pool 데이터소스	186
B.9. MySQL 5.x 구성 예제	187
B.9.1. MySQL Connector/J Connection Pool 데이터소스	187

1. 소개

본 장에서는 JEUS 서버의 구성 요소 및 그 구성 요소가 제공하는 서비스에 대해서 설명한다.

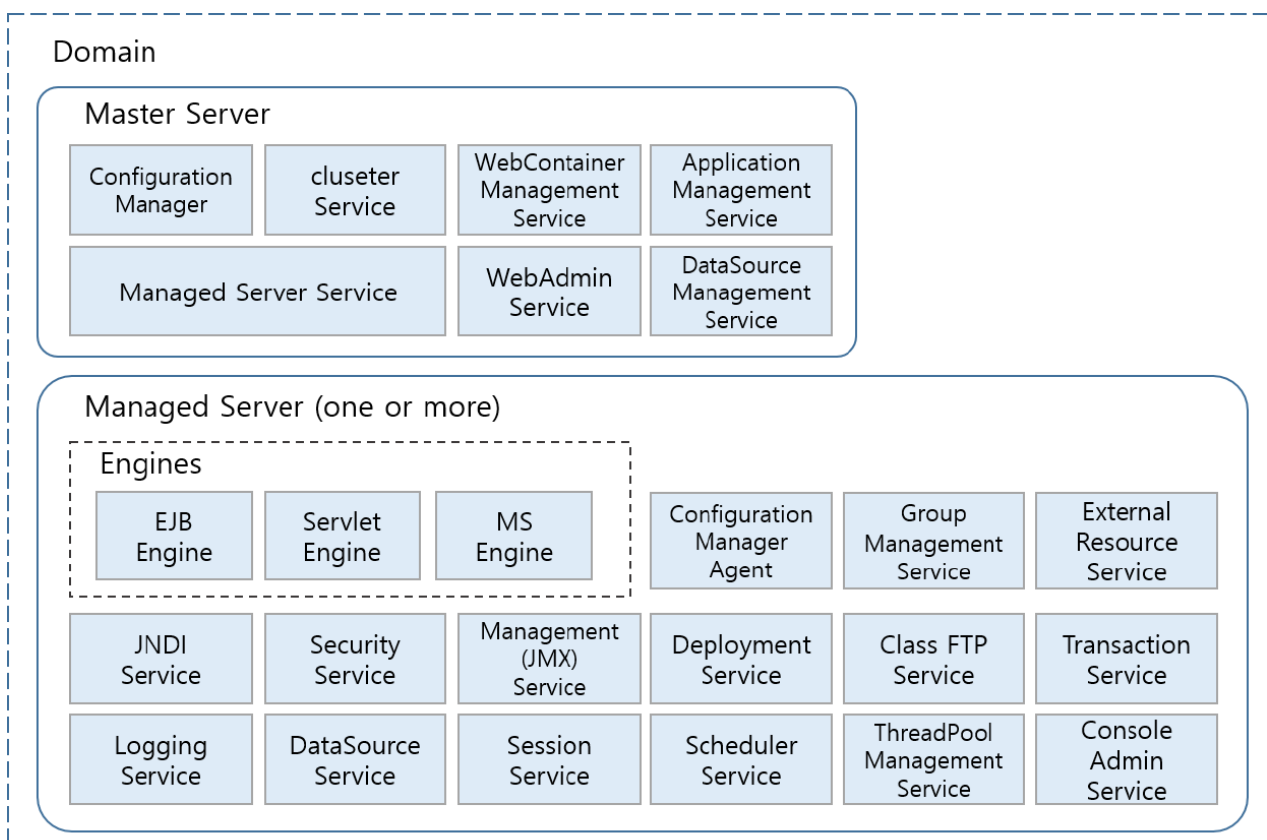
1.1. 구성 요소

JEUS 서버는 여러 개의 서버를 같은 도메인(Domain)으로 묶어 Master Server를 통해 중앙에서 관리할 수 있다.



서버 기동 방법과 도메인에 대한 자세한 내용은 "JEUS Domain 안내서"를 참고한다.

다음은 JEUS의 주요 구성요소이다.



JEUS의 구성 요소

• 도메인(Domain)

도메인은 서버와 클러스터를 관리하는 그룹을 의미한다. 도메인은 Master Server(이하 Master)와 Managed Server(이하 MS)로 구성된다. 도메인에 대한 자세한 내용은 "JEUS Domain 안내서"를 참고한다.

• Master Server(MASTER)

MASTER는 도메인에서 하나만 존재하는 서버이다. 도메인 전체의 설정과 애플리케이션을 관리한다. 또한, 도메인에 속한 여러 Managed Server(MS)를 관리하고 제어한다.

다음은 MASTER의 주요 서비스이다.

- 동적 설정 반영 서비스
- 도메인 애플리케이션 관리 서비스
- 도메인 데이터소스 관리 서비스
- 클러스터 관리 서비스

• Managed Server(MS)

MS는 도메인에서 하나 이상 존재할 수 있는 구성 요소이다. 사용자가 deploy하는 애플리케이션을 서비스하고, 이러한 애플리케이션이 필요로 하는 서비스를 제공한다.

다음은 MS의 주요 서비스이다.

- EJB, WEB, JMS 엔진 서비스
- JNDI 서비스
- Management 서비스
- 보안 서비스
- HTTP 세션 클러스터링 서비스
- 트랜잭션 서비스
- 데이터베이스(이하 DB) 또는 엔터프라이즈 정보 시스템(EIS : Enterprise Information System) 연결 서비스
- Scheduler 서비스 등의 다양한 서비스



도메인 구성에 대한 자세한 내용은 JEUS Domain 안내서의 "도메인 구성"을 참고한다.

1.1.1. Master Server(Master)

Master Server(MASTER)는 도메인을 관리하는 서버이다. 도메인에서 오직 하나만 존재한다. MASTER의 역할은 도메인 설정을 관리하고, 도메인에서 서비스되는 애플리케이션을 관리 및 제어한다. 또한 도메인에 속하는 MS를 관리한다.



Master 개념에 대한 자세한 설명은 JEUS Domain 안내서의 "Master Server(MASTER)"를 참고한다.

다음은 MASTER에서만 사용할 수 있는 주요 서비스이다. 도메인 단위로 관리되어야 하는 서비스들이기 때문에 MASTER에서만 실행된다. 사용자가 아래의 서비스들을 사용하기 위해서는 MASTER와 연결된 콘솔 툴 (jeusadmin)을 이용해야 한다.

• 동적 설정 반영 서비스

MASTER에서는 동적 설정 반영 서비스를 통해 도메인의 설정이 도메인에 존재하는 모든 서버에서 동일하게 유지될 수 있도록 관리한다. 도메인의 설정을 변경하는 작업은 모두 MASTER를 통해서만 가능하다.

콘솔 툴(jeusadmin)을 통해서 동적 설정 변경 명령을 내릴 수 있다. 이때 동적 반영이 가능한 항목은 운영 중인 서버를 재기동하지 않고도 변경한 설정을 서버에 적용할 수 있고, 동적 반영이 불가능한 항목은 운영 중인 서버를 재기동해야 적용된다. 이것이 동적 설정 반영 서비스의 역할이다.

도메인에서 설정 변경을 관장하는 서비스에 대한 자세한 내용은 JEUS Domain 안내서의 "도메인 설정변경"을 참고한다.

• 도메인 애플리케이션 관리 서비스

MASTER에서는 애플리케이션 관리 서비스를 통해 도메인에서 서비스될 모든 애플리케이션을 관리한다. MS에서 애플리케이션을 서비스하기 위해서는 먼저 도메인에 애플리케이션을 등록해야 한다. 그 후에 이 애플리케이션을 서비스할 대상을 Target으로 설정하고 MASTER로 애플리케이션 **deploy** 명령을 실행한다. 도메인 애플리케이션 관리 서비스에서는 MASTER에서의 애플리케이션 상태와 MS에서의 애플리케이션 상태가 동일하게 유지될 수 있도록 한다. 또한 MASTER에서 애플리케이션 파일이 변경되었다면 MS에서도 파일을 재전송받아 동기화될 수 있도록 한다.

도메인에서 애플리케이션의 관리 방법에 대한 자세한 내용은 JEUS Applications & Deployment 안내서의 "도메인 환경에서 애플리케이션 관리"를 참고한다.

• 도메인 데이터소스 관리 서비스

MASTER에서는 데이터소스 관리 서비스를 통해 도메인에 등록된 모든 데이터소스를 관리한다. MS에서 데이터소스를 사용하기 위해서는 도메인에 등록된 데이터소스를 지정해야 한다. 도메인에서 데이터소스를 등록하고 관리하는 방법에 대한 자세한 내용은 [DB Connection Pool](#)과 [JDBC](#)를 참고한다.

• 클러스터 관리 서비스

MASTER에서는 클러스터 관리 서비스를 통해 도메인에 존재하는 클러스터를 관리한다. 클러스터는 동일한 서비스를 수행하는 여러 개의 서버들의 집합으로 서비스 부하 분산(Load Balancing)과 장애 상황에서의 Failover 기능을 제공한다. 클러스터에 대한 자세한 내용은 JEUS Domain 안내서의 "JEUS 클러스터링"을 참고한다.

• Managed Server(MS) 서비스

MASTER도 MS와 같은 애플리케이션 서비스를 할 수 있기 때문에 MS에서 실행되는 모든 서비스들을 수행할 수 있다. 규모가 작은 운영계이거나 개발계의 경우처럼 하나의 서버만 띄워도 서비스가 가능한 상황에서는 MASTER에서도 MS와 같은 애플리케이션 서비스를 할 수도 있다. MS의 서비스에 대한 자세한 설명은 [Managed Server\(MS\)](#)를 참고한다.



MASTER에서 MS의 역할을 할 수는 있지만 이를 권장하지는 않는다. 개발계나 규모가 아주 작은 운영계의 경우에는 도메인에 MASTER 하나만 존재하더라도 서비스에 무리가 없을 수 있지만, 대부분의 경우 MASTER는 관리의 역할만 할 수 있도록 두고 애플리케이션 서비스를 하는 MS를 별도로 두는 것이 일반적이다.

1.1.2. Managed Server(MS)

Managed Server(MS)는 실제 애플리케이션을 서비스하기 위한 엔진들과 여러 서비스들을 관장하는 서버 인스턴스를 의미한다. MS는 도메인에 여러 개 존재할 수 있다. MS의 주요 역할은 사용자가 deploy하는

애플리케이션을 서비스하고, 애플리케이션이 필요로 하는 리소스나 서비스를 제공하는 것이다.

다음은 MS의 주요 서비스에 대한 설명이다.

- **엔진 서비스**

MS의 대표적인 서비스로 Servlet, EJB 컴포넌트를 관리하는 엔진(Engine)과 Jakarta™ Messaging (JMS)를 제공하는 엔진 서비스가 있다. 엔진 서비스에 대한 자세한 내용은 [엔진\(Engine\) 서비스](#)를 참고한다.

- **JNDI 서비스**

JNDI 서비스는 Jakarta EE에서 정의한 JNDI 표준의 매커니즘을 제공하는 것으로 JEUS에 등록된 다양한 객체들을 정해진 이름으로 찾아서 사용할 수 있는 방법을 제공한다.

JEUS 서버에서는 이러한 서비스를 제공하는 개체를 JNDI Naming Server라고 부른다. 클러스터링 환경에서는 각각의 MS에서 동작하는 JNDI Naming Server 간에 서로 정보를 교환하면서 마치 하나의 JNDI Naming Server가 있는 것처럼 동작한다. JNDI 서비스에 관한 자세한 내용은 [JNDI Naming Server](#)를 참고한다.

- **Management 서비스**

Management 서비스는 Java Management Extensions(JMX)를 통해 서비스, 컴포넌트, 애플리케이션을 관리 및 모니터링하는 서비스이다.

JMX에 기술된 JMX Remote API Connector(RMI Connector/JMXMP Connector), HTML 어댑터, SNMP 어댑터와 같은 관리 객체를 JEUS Manager에 설정하고, 이를 통해 JEUS의 관리 및 모니터링 정보에 접속하는 방법을 제공한다. Management 서비스의 설정 및 각 관리 객체에 대한 자세한 내용은 "JEUS JMX 안내서"를 참고한다.

- **보안(Security) 서비스**

보안 서비스는 애플리케이션 및 내부 컴포넌트의 보안 요청에 대한 응답 및 각종 인증 작업을 수행한다.

보안 서비스에 관련된 자세한 내용은 "JEUS Security 안내서"를 참고한다.

- **HTTP 세션 클러스터링 서비스**

HTTP 세션 클러스터링 서비스는 서블릿 엔진들 사이에 HTTP 세션이 유지될 수 있도록 하는 서비스이다.

JEUS는 분산식 HTTP 세션 클러스터링을 지원한다. 해당 서비스로 인해 서블릿 엔진에 대한 장애에도 HTTP 세션은 유지된다. 분산식 HTTP 세션 클러스터링은 서버를 운영할 때 HTTP 세션에 대한 메모리의 부담이 적은 효율적인 분산 방식이다. JEUS에서 제공하는 HTTP 세션 클러스터링 서비스에 관한 자세한 설명은 JEUS 세션 관리 안내서의 "분산 세션 서버"를 참고한다.

위에 설명한 서비스는 클러스터링에 참여한 경우 자동으로 제공되는 서비스이다. 클러스터링에 참여하지 않은 경우에 도메인에 존재하는 모든 서버에서 세션의 유지를 제공하는 제한된 서비스가 있다. 하지만 이 서비스에는 표현된대로 제한사항이 존재한다. 자세한 설명은 JEUS 세션 관리 안내서의 "도메인 와이드 세션 클러스터 모드"를 참고한다.

- **Class FTP 서비스**

EJB 2.x의 경우 원격 클라이언트에서 EJB를 호출하기 위해서는 기본적으로 RMI Stub Class를 필요로 하게 된다. 이때 원격 클라이언트에 RMI Stub Class를 미리 패키징하지 않더라도 Class FTP 서비스를 통해서 필요한

클래스들을 전송받아서 사용할 수 있다.

만약 Class FTP 서비스를 사용하지 않는다면 RMI Stub Class는 원격 클라이언트의 클래스 패스(classpath)에 있어야 한다. 이에 대한 예제는 JEUS EJB 안내서의 "EJB 클라이언트"를 참고한다.



실제로 클래스 파일을 전송하는 프로토콜은 FTP 프로토콜이 아니라 HTTP 프로토콜을 사용한다.

EJB 3.x는 Dynamic Proxy 방식을 사용하기 때문에 이 서비스를 사용하지 않는다. EJB 2.x도 기본적으로는 Dynamic Proxy 방식을 사용하고 설정을 통해 RMI Stub 방식을 사용할 수 있다.

• Scheduler 서비스

Scheduler는 사용자가 미리 지정한 시간에 특정 작업들을 실행시킨다. Scheduler 서비스에 대한 자세한 정보는 "JEUS Scheduler 안내서"를 참고한다.

• Logging 서비스

서버에서 발생하는 이벤트와 에러를 파일로 기록한다. JEUS에서 Java Logging Technology에서 지원하는 로거의 레벨이나 핸들러 등을 설정할 수 있다. Logging 서비스에 관한 자세한 내용은 [Logging](#)을 참고한다.

• DB 연결 서비스

서버에서는 애플리케이션이 JDBC Connection Pool을 통해서 DB에 접근할 수 있도록 지원한다. 이에 관한 자세한 내용은 [DB Connection Pool](#)과 [JDBC](#)를 참고한다.

• 트랜잭션 서비스

서버에서는 트랜잭션 매니저를 통해서 애플리케이션이 트랜잭션을 사용할 수 있도록 지원한다. 트랜잭션 매니저에 관한 설명은 [트랜잭션 매니저](#)를 참고한다.

• 외부 리소스(External Resources)

서버에서는 애플리케이션에 여러 가지 종류의 외부 리소스에 대한 연결을 제공한다.

외부 리소스	설명
Data Source	DB와 연결한다. (JDBC 표준에서 정의한 데이터소스)
Mail Source	메일 서버와 연결한다.
URL Source	URL 소스와 연결한다.
Message Bridge	JMS Destination 사이에 사용되는 Bridge이다.
Custom Resource	JavaBean 형태의 Custom Resource를 JNDI 저장소에 등록한다.
External Source	TP Monitor(Tmax), IBM MQ 등의 엔터프라이즈 정보 시스템(EIS)과의 연결한다(JCA 리소스 어댑터를 deploy하는 방식과는 구분된다).
JAXR Source	XML 레지스트리 소스이다.

외부 리소스에 대한 연결 정보는 사용자가 직접 추가할 수 있으며 JNDI Naming Server에 등록되어 애플리케이션에서 JNDI Lookup Operation을 통해 이용할 수 있다. 외부 리소스는 도메인에 설정되지만 도메인에 속한 모든 서버에서 이 서비스를 이용하게 되므로 서버의 서비스로 분류된다.

클러스터링 환경일 경우에는 JNDI Naming Server를 통해서 클러스터에 속한 모든 서버가 같은 리소스 정보를 공유해서 사용하게 된다. 리소스 설정에 관련된 내용은 [External Resource](#)를 참고한다.

• 엔터프라이즈 정보 시스템(EIS) 연결 서비스

애플리케이션이 deploy된 JCA 리소스 어댑터나 외부 리소스 등록 정보를 통해서 EIS에 접근할 수 있다. JCA 리소스 어댑터에 관한 자세한 내용은 "JEUS Jakarta Connectors 안내서"를 참고한다.

엔진(Engine) 서비스

엔진은 Jakarta EE에서 정의한 EJB 컨테이너, 웹 컨테이너와 매핑되는 개념으로 사용자가 deploy한 컴포넌트를 관리하고 서비스하는 역할을 한다. 엔진은 서버에 포함되는 서비스이며, 사용자가 설정하지 않아도 서버가 부팅할 때 항상 기본 설정으로 실행된다.

서버가 운영 중에 콘솔 툴을 통해 엔진 설정을 변경할 수 있다. 각 엔진에서 제공하는 기본 설정과 동적 변경 가능한 설정에 대해서는 각 엔진의 매뉴얼을 참고한다.

서버에서 제공하는 서비스 엔진은 다음과 같이 3가지 종류가 있다.

엔진	설명
EJB 엔진	EJB 컴포넌트를 관리하고 실행하는 EJB 컨테이너 역할을 한다. 이에 대한 자세한 사항은 "JEUS EJB 안내서"를 참고한다.
웹 엔진 (또는 서블릿 엔진)	웹 컨테이너라고 할 수 있으며, 웹 클라이언트 또는 웹 서버로부터 요청을 받아서 사용자가 deploy한 Servlet(또는 JSP, JSF)을 통해 동적인 웹 콘텐츠를 생성한다. 이에 대한 자세한 사항은 "JEUS Web Engine 안내서"를 참고한다.
JMS 엔진	Jakarta Messaging(JMS)을 제공한다. 이에 대한 자세한 사항은 "JEUS MQ 안내서"를 참고한다.



JEUS 6까지는 엔진 컨테이너에 엔진 설정이 없으면 엔진이 실행되지 않아서 애플리케이션이 서비스될 수 없었다. 웹 엔진을 설정하지 않은 경우에 웹 애플리케이션을 서비스하려면 엔진을 추가하고 JEUS를 재기동해야만 했다. 하지만 JEUS 7부터는 MS를 부팅하면 기본 설정으로 엔진이 실행되어 있는 상태이기 때문에 사용자가 설정하지 않았다고 하더라도 어떤 타입의 애플리케이션이든 deploy하여 서비스 가능하다.

Managed Server의 INDEPENDENT 상태

INDEPENDENT 모드는 MS가 MASTER의 관리 없이 부팅되어 운영되고 있는 모드를 의미한다. 즉, 부팅할 때 MASTER의 URL 정보가 없거나 MASTER가 장애상황인 경우에 발생한다.

MS를 시작시킬 때는 항상 MASTER의 URL 정보를 옵션으로 주어야 한다. Launcher는 서버를 실행할 때 설정한 URL 정보로 MASTER의 URL을 알 수 있기 때문에 이 정보가 없는 경우에는 MASTER로부터 설정을 받아올 수가

없다.

MASTER URL을 주지 않은 경우 부팅하려고 하는 서버의 머신에 캐시되어 있는 설정 파일들이 있다면 서버는 해당 설정 파일을 기반으로 하여 INDEPENDENT 상태로 기동될 것이다. 하지만 서버의 머신에 설정 파일이 존재하지 않는다면 서버는 부팅될 수 없다.

서버의 머신에 설정 파일이 존재한다고 하더라도 이전에 사용하던 파일이기 때문에 Master와 동기화가 맞지 않을 가능성이 높다. 따라서 항상 서버를 부팅할 때는 Master의 URL을 주어야 한다.

MASTER URL을 주지 않아도 되는 경우는 Master와 MS가 같은 머신에 존재하여 도메인 설정 파일을 전송받지 않아도 되는 경우 밖에 없다. MASTER가 장애상황이 아닌데 URL을 주지 않은 경우에는 서버는 INDEPENDENT 상태라고 인식하겠지만, MS에 그룹 관리 서비스가 시작되고 난 후에는 MASTER와 통신이 되어 다시 DEPENDENT 모드로 복구될 것이다.

INDEPENDENT 상태는 URL을 주지 않은 상황보다 MASTER가 장애상황인 경우에 많이 발생할 것이다. MS를 기동할 때 MASTER가 장애상황이었다면 MS의 Launcher는 설정을 받아오는 작업도 실패할 것이다. MASTER가 fallback되어 INDEPENDENT 상태로 부팅된 MS가 DEPENDENT 모드로 복구되면 MASTER로부터 설정 동기화를 하고 Deploy에 실패한 애플리케이션을 다시 deploy한다.



MASTER가 복구되어 설정 동기화를 한다고 하더라도, 동적으로 반영되지 않는 설정이 변경된 경우나 이미 deploy되어 서비스되고 있는 애플리케이션이 변경되었다면 MS를 재기동해야 한다.

다음은 MS가 INDEPENDENT 상태로 부팅했을 때의 로그이다.

```
JEUS_HOME/bin$ ./startManagedServer -domain jeus_domain -server server1 -u admin -p admin -masterurl
localhost:9736
*****
- JEUS Home           : /home/jeus
- Java Vendor         : Sun
- Added Java Option :
*****
===== JEUS LICENSE INFORMATION =====
== VERSION : JEUS 9 (9.0.0.0-b15)
== EDITION: Enterprise (Trial License)
== NOTICE: This license restricts the number of allowed clients.
== Max. Number of Clients: 5
=====
[2024.09.25 16:12:50][0] [launcher-1] [Launcher-0052] Receiving the configuration failed. Attempting
to start as INDEPENDENT.
<<__Exception__>>
java.io.IOException: Connection failed. host:localhost, port:9736, virtual id:FileTransfer
at jeus.net.SocketProxy.getConnection(SocketProxy.java:69)
at jeus.net.SocketProxy.getConnection(SocketProxy.java:25)
at jeus.server.filetransfer.ConfigurationSynchronizer.connect(ConfigurationSynchronizer.java:116)
at jeus.server.filetransfer.ConfigurationSynchronizer.connect(ConfigurationSynchronizer.java:99)
at
jeus.server.filetransfer.ConfigurationSynchronizer.checkConnection(ConfigurationSynchronizer.java:141
)
at
jeus.server.filetransfer.ConfigurationSynchronizer.downloadConfigFileFromMasterServer(ConfigurationSy
nchronizer.java:181)
```

```

at
jeus.launcher.ManagedServerLauncher.receiveConfigurationFromMasterServer(ManagedServerLauncher.java:2
45)
at jeus.launcher.ManagedServerLauncher.updateXmIs(ManagedServerLauncher.java:133)
at jeus.launcher.ManagedServerLauncher.pullLatestDomainType(ManagedServerLauncher.java:67)
at jeus.launcher.ManagedServerLauncher.initDescriptor(ManagedServerLauncher.java:53)
at jeus.launcher.Launcher.start(Launcher.java:146)
at jeus.launcher.ManagedServerLauncher.start(ManagedServerLauncher.java:73)
at jeus.launcher.ManagedServerLauncher.main(ManagedServerLauncher.java:45)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at jeus.server.Bootstrapper.callMainMethod(Bootstrapper.java:576)
at jeus.server.Bootstrapper.callMain(Bootstrapper.java:583)
at jeus.server.Bootstrapper.main(Bootstrapper.java:151)
at jeus.server.ManagedServerLauncherBootstrapper.main(ManagedServerLauncherBootstrapper.java:10)
<<__Exception__>>
[2024.09.25 16:12:50][1] [launcher-1] [Config-0157] SecurityDomainsConfigServiceProvider is
jeus.service.descriptor.SecurityDomainsDescriptorFile.

...
[2024.09.25 16:12:53][0] [server1-1] [SERVER-0249] Successfully started the server INDEPENDENTLY
[2024.09.25 16:12:53][2] [server1-1] [SERVER-0248] The JEUS server is RUNNING.
[2024.09.25 16:12:53][2] [server1-1] [SERVER-0401] The elapsed time to start: 4079ms.
[2024.09.25 16:12:53][2] [launcher-21] [Launcher-0034] The server[server1] initialization completed
successfully[pid : 15332].
[2024.09.25 16:12:53][0] [launcher-1] [Launcher-0040] Successfully started the server. The server
state is now RUNNING.
JEUS_HOME/bin$

```

그 후에 MASTER가 정상상황이 되었다면 MS는 다시 설정을 동기화하고, deploy에 실패한 애플리케이션이 있었다면 다시 deploy를 수행할 것이다.

```

[2024.09.25 16:23:20][2] [server1-55] [Domain-0101] JEUS Master Server recovered. server1 is
communicating with the JEUS Master server.
[2024.09.25 16:23:20][2] [server1-55] [SERVER-0201] Successfully connected to the JEUS Master
Server(192.168.14.63:9736).
[2024.09.25 16:23:20][2] [server1-55] [SERVER-0308] Resynchronized the configuration with JEUS Master
Server.

```

1.2. 클래스 로더의 구조

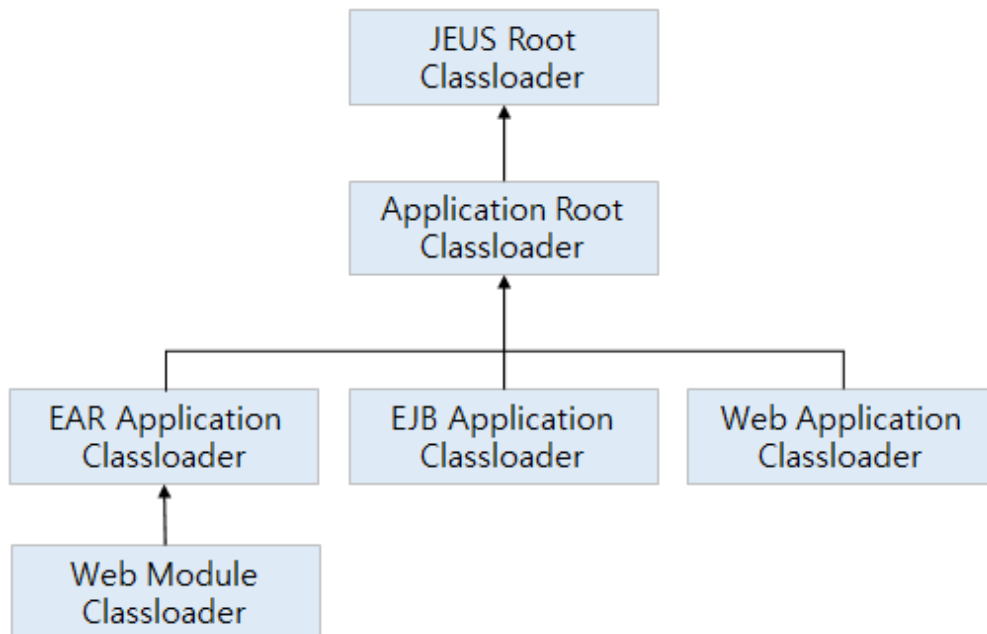
본 절에서는 JEUS에서 지원하는 Isolated Class Loader에 대해서 설명한다.



JEUS 5 이하 버전에서는 Shared Class Loader(이하 Shared) 방식을 기본적으로 사용하기 때문에 하위 호환성을 위해 그 방식을 계속 제공한다. 하지만 Shared 방식을 사용하는 것을 권장하지 않기 때문에 아무런 설정을 하지 않으면 기본적으로 Isolated 방식을 사용하게 된다. 본 안내서에는 Shared Class Loader에 대한 설명은 하지 않는다.

JEUS는 애플리케이션 간에 클래스가 중복되어 발생하는 문제를 방지하기 위해 애플리케이션마다 서로 다른 클래스 로더(Class Loader)를 사용한다. 이를 **Isolated Class Loader**(이하 Isolated)라고 하는데, Jakarta EE 표준은 이 방식을 기본으로 사용할 것을 권장하고 있으며 JEUS도 그 권고를 따르고 있다.

다음은 Isolated의 계층 구조이다.



Isolated Class Loader 계층 구조

이 구조에서는 각 애플리케이션의 웹 모듈의 클래스 로더가 해당 애플리케이션의 EJB 클래스 로더의 하위에 존재한다. 그리고 한 애플리케이션의 클래스 로더는 다른 애플리케이션의 클래스 로더에 클래스를 요청하지 않는다. Jakarta EE 표준은 애플리케이션이 다른 애플리케이션의 인터페이스를 필요로 하는 경우 그 인터페이스들을 같이 패키징하도록 규정하고 있으므로 다른 애플리케이션의 클래스 로더를 참조할 수 없다.

클래스를 공유해서 사용하지 않기 때문에 한 애플리케이션을 다시 deploy할 때 다른 애플리케이션도 같이 다시 deploy할 필요가 없어진다. 이 클래스 로더 구조를 제대로 사용하기 위해서는 결국 연관된 애플리케이션 간에 EAR로 묶어서 하나의 애플리케이션으로 만드는 작업이 필요하다.

1.3. 서버 디렉터리 구조

서버는 DOMAIN_HOME 하위에 각자 자신의 디렉터리를 갖는다. DOMAIN_HOME 하위에 servers 디렉터리에는 도메인에 속한 서버들의 디렉터리가 존재한다. 각 서버별로 DOMAIN_HOME/servers 하위에 서버 이름으로 디렉터리를 생성하고, 자신의 서버만의 공간에 운영 중에 필요한 정보나 서버 시작을 위해 필요한 정보 등을 저장한다. 이를 SERVER_HOME이라고 한다.

SERVER_HOME의 디렉터리 구조는 다음과 같다.

```

{DOMAIN_HOME}
|--servers
|   |--server1
|       |--.workspace
|       |   |--deployed
|       |   |--ejbtimerdb
|       |   |--session
  
```

```

|--tmlog
|--tmp
|--bin
|--lib
|--application
|--logs

```

.workspace

JEUS가 사용하는 서버별 공간으로 사용자가 변경해서는 안 된다.

다음은 하위 디렉터리에 대한 설명이다.

디렉터리	설명
deployed	<p>서버에 deploy된 애플리케이션의 압축을 푼 이미지 파일과 deploy할 때 생성되는 파일이 위치한다. 이 디렉터리 하위에는 애플리케이션 ID로 디렉터리가 생기고 Master로부터 받아 온 애플리케이션 파일이 저장된다. 애플리케이션이 archive된 경우 애플리케이션 ID 하위에 압축을 푼다(deploy image).</p> <p>deployed 디렉터리 하위에 '_generated_'라는 디렉터리가 생성되고 애플리케이션을 deploy할 때 생성되는 파일들이 위치한다. 디렉터리 하위에 애플리케이션 ID로 디렉터리를 생성하고 deploy 시점에 필요한 파일을 생성한다.</p>
ejbtimerdb	EJB 엔진의 타이머 서비스에서 DB 설정이 없는 경우 내부적으로 파일 기반의 DB인 JEUS에 내장된 Apache derby를 사용하기 위한 디렉터리이다. 타이머 서비스에 대한 자세한 설명은 JEUS EJB 안내서의 "EJB Timer Service"를 참고한다.
session	분산식 세션 서버에서 사용하는 파일 DB가 위치한다. 분산식 세션 서버에서는 passivation된 세션과 primary 세션 서버에서 전송한 백업 세션들을 이 파일 DB에 저장한다. 자세한 설명은 JEUS 세션 관리 안내서의 "분산 세션 서버"를 참고한다.
tmlog	<p>트랜잭션 복구를 위해 남기는 트랜잭션 로그가 남는 디렉터리이다.</p> <p>"_serverName_LOCATION_type_port_ipaddress_virtualport 이름"으로 하위 디렉터리가 생성된다. 생성되는 디렉터리 이름에 type과 virtualhost에는 다음의 값이 설정된다.</p> <ul style="list-style-type: none"> ◦ type : IPv4의 경우는 0이 설정되고, IPv6의 경우는 1이 설정된다. ◦ virtualport : 서버 이름의 hash 값이 설정된다. <p>자세한 내용은 복구 관련 로그 파일을 참고한다.</p>
tmp	서버 운영 중에 임시로 저장할 파일이 있을 때 사용하는 디렉터리로 사용자가 특별히 접근할 일은 없다.

bin

서버의 시작/종료 스크립트를 포함하고 있다. JEUS_HOME/bin의 스크립트와 동일한 기능을 수행하지만 도메인 이름과 서버 이름을 설정할 필요가 없다.

Master일 경우에 startMasterServer/stopserver, MS일 경우에 startManagedServer/stopserver가 존재한다.

lib/application

서버에 적용하고 싶은 애플리케이션 라이브러리가 위치한다.

DOMAIN_HOME/lib/application에 존재하는 도메인 범위의 라이브러리와 충돌될 경우 이 디렉터리에 존재하는 파일이 적용된다. 도메인의 애플리케이션 라이브러리와 충돌되었다는 경고 메시지를 남기고 도메인 범위의 라이브러리는 이 서버에서 무시된다. lib/application에 대한 자세한 설명은 JEUS Applications & Deployment 안내서의 "lib/application 디렉터리"를 참고한다.

logs

서버의 Launcher 로그, 서버 로그, 액세스 로그 파일이 저장된다. 각 로그 파일은 로테이션 룰에 따라 "로그 이름_날짜.log00000"으로 생성된다. 00000은 1부터 99999까지의 숫자를 다섯자리로 표현한 것으로 logging에 대한 자세한 내용은 [Logging](#)을 참고한다.

1.4. Launcher

MASTER와 MS는 모두 Launcher를 통해서 기동된다. 스크립트를 통해서 서버를 시작시키거나 MASTER를 통해서 MS를 시작시킬 때 모두 Launcher 프로세스를 실행한다. Launcher에서는 서버를 띄우기 위한 준비 작업과 실제 서버를 기동하는 작업을 수행한다.

Launcher의 역할은 크게 2가지가 있다.

- MASTER로부터 도메인 설정 파일을 받아오는 작업
- 서버를 기동시켜주는 작업

Launcher는 먼저 MASTER로부터 도메인의 설정 파일들을 받아온다. 대상이 되는 파일은 **domain.xml**과 **security 설정 파일**이다. 설정 파일을 받아오는 작업이 완료되면 Launcher에서는 받아온 설정 파일을 기반으로 서버 JVM을 기동한다. 서버를 기동할 때는 설정 파일에서 시작시킬 서버에 설정되어져 있는 JVM 옵션이나 클래스 패스를 읽어서 서버 JVM을 생성할 때 옵션으로 설정한다.

만약 Launcher 프로세스에서 설정을 받아오는 작업을 실패한 경우 머신에 캐시되어 있는 도메인 설정 파일이 존재할 경우 이 파일을 기반으로 서버 JVM을 기동시킬 것이다. Launcher에서 설정을 받아오는 작업도 실패하고, 머신에도 캐시해놓은 설정 파일이 존재하지 않는다면 서버는 부팅될 수 없다.

Launcher 프로세스는 위의 2가지 역할 외에도 서버 부팅 로그의 logging 역할을 한다.

서버가 부팅하면서 발생한 로그는 Launcher 로그에도 logging이 된다. Launcher 프로세스는 서버 JVM을 기동시키고 서버의 부팅 작업이 모두 완료될 때까지 대기하면서 서버를 부팅할 때 발생한 로그를 logging한다. 만약 서버가 로거를 초기화하기 전에 부팅에 실패했다면 Launcher 로그에서 부팅 실패 원인을 확인할 수 있다.

일반적으로 Launcher 프로세스는 서버의 JVM을 기동하고 해당 서버의 부팅이 완료됨을 확인하면 종료된다. Launcher를 통해 기동된 서버 프로세스는 백그라운드로 수행된다는 것에 유의해야 한다. 또한 서버 프로세스는 Launcher 프로세스의 자식 프로세스로 기동되기 때문에 백그라운드 프로세스로 생성되기 때문에 서버 프로세스는 자신의 콘솔 화면을 갖지 못한다.

따라서 서버의 로그를 콘솔 화면으로 출력하고 싶다면 서버를 실행할 때 -verbose 옵션을 추가해야 한다. -verbose 옵션이 존재하면 Launcher 프로세스는 서버가 종료될 때까지 종료되지 않고 서버의 로그를 화면에 출력하는 역할을 하게 된다. Launcher 로그에 대한 자세한 내용은 [Launcher 로거](#)를 참고한다.

2. JEUS 설정

본 장에서는 도메인에 서버를 추가하는 방법과 서버를 추가할 때 꼭 필요한 설정에 대해서 설명한다. 또한 서버에 설정을 변경할 때의 동작 방법에 대해서 설명한다.

2.1. 개요

JEUS에서는 콘솔 툴을 통해 서버의 설정을 변경할 수 있다. 서버의 엔진에서 동작하는 애플리케이션이 보안 인증과 권한 검사를 필요로 하지 않는다면 JEUS에서 보안 기능을 사용하지 않도록 설정한다.

위와는 별도로 하위 구성 요소들에 대한 튜닝도 확인해야 한다. 이에 대한 설명은 각 구성 요소에 대한 장을 참고한다.

2.2. 서버 추가

본 절에서는 콘솔 툴을 통해 도메인에 서버를 추가하는 방법과 서버를 추가할 때 반드시 필요로 하는 최소 설정에 대해 설명한다. 서버 설정에 대한 자세한 내용은 [서버 설정](#)을 참고한다.

2.2.1. 콘솔 툴 사용

콘솔 툴에서는 **add-server** 명령어를 통해 서버를 추가할 수 있다. 하지만 이 명령어로는 몇 가지 설정에 대해서만 추가할 수 있기 때문에 서버를 추가하고 난 뒤에 별도의 명령어들을 통해 설정을 변경해야 한다. 만약 엔진 설정을 변경하는 경우 웹 엔진, EJB 엔진, JMS 엔진 명령어를 사용한다.



웹 엔진 설정을 변경하는 명령어(JEUS Reference 안내서의 "웹 엔진 관련 명령어"), EJB 엔진 설정을 변경하는 명령어(JEUS Reference 안내서의 "EJB 엔진 관련 명령어"), JMS 엔진 설정을 변경하는 명령어(JEUS Reference 안내서의 "JMS 엔진 관련 명령어")에 대한 자세한 설명은 관련 안내서를 참고한다.

다음은 콘솔 툴에서 서버를 추가하는 예제이다.

```
[MASTER]domain1.adminServer>server-info
Information about Domain (domain1)
```

Server	Status	Node Name	PID	Clu ster	Latest Start Time / Shutdown Time	Need to Restart	Listen Ports	Running Engines
adminS erver (*)	RUNNING (00:18:2 0)	nod e1	572 02	N/A	2024-09-25 (목) 오후 01:28:24 KST	false	base-0.0. 0.0:9736 http-serv er-0.0.0.0 :8088	jms, web, ejb

Server	Status	Node	PID	Cluster	Latest Start Time / Shutdown Time	Need to Restart	Listen Ports	Running Engines
server1	RUNNING	node1	58925	N/A	2024-09-25 (목) 오후 01:46:36 KST	false	base-0.0.0.0:9836 http-server-0.0.0.0:8188	jms, web, ejb

```
[MASTER]domain1.adminServer>add-server server2 -addr 192.168.15.59 -port 9936 -node node1 -jvm "-Xmx512m -XX:MaxPermSize=128m"
```

Successfully performed the ADD operation for server (server2).

NOTICE : base-addr [192.168.15.59] base-port [9936] http-port [8088]

Check the results using "list-servers or add-server".

```
[MASTER]domain1.adminServer>modify-system-thread-pool server2 -max 200
```

Successfully performed the MODIFY operation for the system thread pool of the server (server2), but all changes were non-dynamic. They will be applied after restarting.

Check the results using "modify-system-thread-pool server2 or show-system-thread-pool server2".

```
[MASTER]domain1.adminServer>server-info
```

Information about Domain (domain1)

Server	Status	Node	PID	Cluster	Latest Start Time / Shutdown Time	Need to Restart	Listen Ports	Running Engines
adminServer (*)	RUNNING	node1	57202	N/A	2024-09-25 (목) 오후 01:28:24 KST	false	base-0.0.0.0:9736 http-server-0.0.0.0:8088	jms, web, ejb
server1	RUNNING	node1	58925	N/A	2024-09-25 (목) 오후 01:46:36 KST	false	base-0.0.0.0:9836 http-server-0.0.0.0:8188	jms, web, ejb
server2	SHUTDOWN	node1	N/A	N/A	2024-09-25 (목) 오후 02:04:50 KST	N/A	N/A	N/A

2.3. 서버 설정

JEUS에서 콘솔 툴을 사용하여 서버 설정을 변경할 수 있다.

2.3.1. 환경변수 설정

본 절에서는 서버 및 JEUS에서 제공하는 각종 콘솔 툴들이 동작하기 위해 필요한 환경변수에 대하여 설명한다. 환경변수는 콘솔 툴을 통한 설정을 지원하지 않는다.

JEUS가 동작하기 위하여 필요한 환경변수는 설치 시 UNIX 환경인 경우 JEUS_HOME/bin/jeus.properties 파일에 자동적으로 기본값이 설정된다. 해당 파일들은 운영체제 계열에서 제공하는 셸 스크립트로 작성되어 있으며, 필요한 경우 사용자가 해당 파일을 수정하여 환경변수를 추가하거나 수정할 수 있다.

JEUS에서 사용하는 주요 환경변수에 대한 목록은 다음과 같다.

환경변수	설명
JEUS_HOME	JEUS 설치 디렉터리를 설정한다. (예: /home/jeus)
JEUS_LIBPATH	JEUS 라이브러리 파일 경로를 설정한다. (예: /home/jeus/lib/system)
VM_TYPE	Java HotSpot JVM 사용 유무를 설정한다. (예: hotspot or old)
JEUS_USERNAME	Administrator 계정의 ID를 설정한다.
JEUS_PASSWORD	Administrator 패스워드를 설정한다.
JAVA_HOME	JDK 설치 디렉터리 경로를 설정한다. (예: /usr/jdk17)
JAVA_ARGS	JDK 파라미터를 설정한다.
JAVA_VENDOR	JDK 벤더를 설정한다. (예: Sun, IBM, HP)

jeus.properties에 지정한 환경변수는 해당 스크립트를 참조하여 실행하는 모든 서버 및 툴에 적용된다.

서버별로 환경변수를 다르게 지정하고자 하는 경우에는 다음과 같은 과정을 수행한다.

1. 다음의 경로에 환경변수 파일 또는 셸 스크립트 파일을 생성한다.

```
JEUS_HOME/bin/<DOMAIN_NAME>.<SERVER_NAME>.properties
```

2. 생성한 파일(과정 1)을 수정하여 환경변수를 지정한다. 파일을 작성할 때 jeus.properties 또는 jeus.properties.cmd 파일을 참고한다. 환경변수를 지정할 때에는 변수이름 앞 주석처리 부분(#)을 제거하여 사용한다.

다음은 환경변수 파일의 예이다.

```
#####
# This part is for booting JEUS automatically.                                #
# BE CAREFUL!! THIS IS ONLY FOR TEST AND DEVELOPMENT ENVIRONMENT.            #
#####

# Set up administrator name
# JEUS_USERNAME=

# Set up administrator password
# JEUS_PASSWORD=
```

다음은 서버 기동하는 예제이다.

```
JEUS_HOME/bin$startMasterServer -server adminServer
```

2.3.2. 기본 설정

본 절에서는 기본적인 설정에 대해서 설명한다. 설명하지 않은 설정들에 대해서는 관련 서비스에 대한 안내서를 참고한다.

2.3.2.1. 기본 설정 동적으로 변경

서버의 기본 설정 중 Class FTP, Use MEJB, Log Stdout To Raw Format은 동적 변경 가능하다. 동적 반영되는 설정은 콘솔 툴에서 **help** 명령어를 통해 **modify-server** 명령어를 실행하면 조회할 수 있다. 각 명령어에 대한 자세한 내용은 "JEUS Reference 안내서"를 참고한다.

콘솔 툴 사용

다음은 콘솔 명령어를 사용하여 동적 변경 가능한 기본 설정을 변경하는 예이다.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node           | node1           |
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING         |
| Stdout to Raw Format | true           |
| MEJB           | false          |
| Class FTP      | false          |
| Server Log Home Directory | none           |
+-----+-----+
=====

[MASTER]domain1.adminServer>modify-server server2 -logStdoutToRawFormat false -mejb true -classFtp
true
Successfully performed the MODIFY operation for server (server2).
Check the results using "list-servers server2 or modify-server server2"

[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node           | node1           |
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING         |
| Stdout to Raw Format | false          |
| MEJB           | true           |
| Class FTP      | true           |
| Server Log Home Directory | none           |
+-----+-----+
=====
```

```
+-----+-----+
=====
```

2.3.2.2. Action On Resource Leak 설정

Action On Resource Leak은 서버에서 사용되는 리소스가 닫혔는지 여부를 확인해서 설정한 동작을 수행하고 사용자에게 Resource Leak이 있다는 것을 알려주는 기능이다. 서버에서 이런 역할을 담당하는 것을 **Invocation Manager**라고 한다.

Invocation Manager는 서버에서 Servlet/JSP, EJB Stateless Session Bean, 그리고 MDB와 같은 Stateless 메소드를 호출하는 동안 사용하는 외부 리소스(external resource)인 JDBC Connection과 WebT Connection을 추적하여 Connection이 닫히지 않은 경우 모드에 따라 사용하는 리소스에 대해 logging을 남기거나 반환하는 작업을 한다. 이 구성요소를 위해 3가지 모드 중 하나를 선택할 수 있다.

모드	설명
NoAction	반환되지 않은 리소스가 있더라도 아무런 동작을 하지 않는다.
Warning	컴포넌트 호출 후에 반환되지 않은 리소스에 대한 로그를 남긴다 (기본값). 추가적으로 SMTP Handler 설정을 통해 이메일 안내를 받도록 할 수 있다.
AutoClose	컴포넌트 호출 후에 반환되지 않은 리소스에 대한 로그를 남기고 이를 닫아준다. 추가적으로 SMTP Handler 설정을 통해 이메일 안내를 받도록 할 수 있다.

다음은 콘솔 톨의 modify-server 명령어를 통해 Action On Resource Leak을 변경하는 예이다.

1. 서버를 부팅하기 전에 **modify-server** 명령어를 통해 현재 설정을 변경할 서버의 설정을 조회한다.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node           | node1           |
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING        |
| Stdout to Raw Format | false          |
| MEJB           | true            |
| Class FTP      | true            |
| Server Log Home Directory | none          |
+-----+-----+
=====
```

2. server2의 Action On Resource Leak 설정을 AutoClose로 설정한다.

```
[MASTER]domain1.adminServer>modify-server server2 -actionOnResourceLeak AutoClose
Successfully performed the MODIFY operation for server (server2).
Check the results using "list-servers server2 or modify-server server2".
```



동적 반영이 되지 않는 옵션들의 경우는 서버를 부팅하기 전에 변경해야 한다. 운영 중인 서버에 이런 설정을 수정하게 되면 동적으로 반영되지 않기 때문에 서버를 재기동해야 한다.

3. **modify-server** 또는 **list-servers** 명령어를 통해 변경한 설정이 잘 반영되었는지를 확인한다.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node           | node1           |
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | AUTO_CLOSE      |
| Stdout to Raw Format | false           |
| MEJB           | true            |
| Class FTP      | true            |
| Server Log Home Directory | none            |
+-----+-----+
=====
```

4. 'server2'의 Action On Resource Leak 설정을 다시 원래 값인 WARNING으로 변경한다.

이 설정은 서버가 운영 중이기 때문에 설정 변경만 되고 실제로 서버에 적용되지는 않는다. 변경한 설정이 적용되길 원한다면 서버를 재기동해야 한다.

```
[MASTER]domain1.adminServer>modify-server server2 -actionOnResourceLeak Warning
Successfully performed the MODIFY operation for server (server2), but all changes were non-
dynamic. They will be applied after restarting.
Check the results using "list-servers server2 or modify-server server2".
```

5. **modify-server** 또는 **list-servers** 명령어를 통해 변경한 설정을 확인한다.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node           | node1           |
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING         |
| Stdout to Raw Format | false           |
| MEJB           | true            |
| Class FTP      | true            |
| Server Log Home Directory | none            |
+-----+-----+
=====
```



위에서 사용한 콘솔 툴 명령어에 대한 자세한 설명은 JEUS Reference 안내서의 "Server

Management 관련 명령어"에서 해당 명령어에 대한 내용을 참고한다.

2.3.2.3. Jvm Config 설정

Jvm Config는 서버를 실행하기 위해 개별적인 JVM에 추가할 파라미터들을 선언하는 데 사용된다. 여기에 설정한 값들을 Launcher 프로세스에서 서버를 시작하기 전에 이 값을 읽어서 서버 JVM을 생성할 때 파라미터로 추가한다. 지정 가능한 JEUS 파라미터들의 목록은 JEUS Reference 안내서의 "서버 시스템 프로퍼티"를 참고한다. 표준 JVM 파라미터들도 설정 가능하다.

서버에 적용할 JVM 옵션, 시스템 프로퍼티와 더불어 JEUS에서 제공하는 시스템 프로퍼티도 여기에 설정할 수 있다. 주로 JVM 메모리나 옵션들을 설정하는데, 이는 서버의 운영 환경에 맞는 적절한 값을 설정해야 한다. JVM Config 관련 추가적인 설명은 JEUS Domain 안내서의 "서버의 JVM 설정변경"을 참고한다.



서버가 운영 중일 때는 이 값을 변경해도 반영되지 않는다. 동적 설정 반영의 대상이 아니기 때문에 운영 중인 서버에 이 값을 변경한 경우에는 서버를 재기동해야 적용된다.

콘솔 툴 사용

콘솔 툴의 **modify-server**, **add-jvm-option**, **modify-jvm-option**, **remove-jvm-option**을 이용하여 서버의 JVM 설정을 변경할 수 있다. 명령에 대한 자세한 설명은 JEUS Domain 안내서의 "서버의 JVM 설정변경", JEUS Reference 안내서의 "Server Management 관련 명령어"를 참고한다.

다음은 콘솔 툴의 **modify-server** 명령어를 통해 서버의 JVM 설정을 변경하는 예이다.

1. 서버를 부팅하기 전에 **modify-server** 명령어를 통해 현재 설정을 변경할 서버의 설정을 조회한다.

```
[MASTER]domain1.adminServer>modify-server server1
Shows the current configuration.
server (server1)
=====
+-----+-----+
| Node                | node1 |
| Action On Resource Leak | WARNING |
| Stdout to Raw Format  | true  |
| MEJB                 | false |
| Class FTP            | false |
| Server Log Home Directory | none  |
+-----+-----+
=====
```

2. server1에 JVM 옵션을 추가한다. JVM의 최대 Heap 메모리를 512MB로 설정하고, 최대 Permanent 메모리를 128MB로 설정한다.

```
[MASTER]domain1.adminServer>modify-server server1 -jvmOptions "-Xmx512m -XX:MaxPermSize=128m"
Successfully performed the MODIFY operation for server (server1).
Check the results using "list-servers server1 or modify-server server1".
```



동적 반영이 되지 않는 옵션들의 경우는 서버를 부팅하기 전에 변경해야 한다. 운영 중인 서버의 설정 정보를 변경하면 동적으로 반영되지 않기 때문에 서버를 재기동해야 한다.

3. **modify-server** 또는 **list-servers** 명령어를 통해 변경한 설정의 정상 반영을 확인한다.

```
[MASTER]domain1.adminServer>modify-server server1
Shows the current configuration.
server (server1)
=====
+-----+-----+
| Node                | node1                |
| JVM Configs         | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING              |
| Stdout to Raw Format | true                 |
| MEJB                | false                |
| Class FTP           | false                |
| Server Log Home Directory | none                 |
+-----+-----+
=====
```

4. 'server1'에 JVM 옵션을 추가한다.

서버에 OutOfMemoryError가 발생한 경우 Heap Dump 파일을 남기는 옵션을 추가하였다. 이 설정은 서버가 운영 중이기 때문에 설정 변경만 되고 실제로 서버에 반영되지는 않는다. 변경한 설정이 반영되길 원한다면 서버를 재기동해야 한다.

```
[MASTER]domain1.adminServer>modify-server server1 -jvmOptions "-XX:+HeapDumpOnOutOfMemoryError"
Successfully performed the MODIFY operation for server (server1).
Check the results using "list-servers server1 or modify-server server1".
```

5. **modify-server** 또는 **list-servers** 명령어를 통해 변경한 설정을 확인한다.

```
[MASTER]domain1.adminServer>modify-server server1
Shows the current configuration.
server (server1)
=====
+-----+-----+
| Node                | node1                |
+-----+-----+
| JVM Configs         | -Xmx512m -XX:MaxPermSize=128m, |
|                     | -XX:+HeapDumpOnOutOfMemoryError |
+-----+-----+
| Action On Resource Leak | WARNING              |
+-----+-----+
| Stdout to Raw Format | true                 |
+-----+-----+
| MEJB                | false                |
+-----+-----+
| Class FTP           | false                |
+-----+-----+
| Server Log Home Directory | none                 |
+-----+-----+
```

+-----+-----+
=====

2.3.2.4. 클래스 패스 설정

서버에 클래스 패스를 추가하기 위해서는 domain.xml 설정을 변경해야 한다.

```
<server>
  ...
  <user-interceptor>
    <jeus-classloader-append-class-path>/home/jeus/lib/mylib.jar</jeus-classloader-append-class-
path>
    <jeus-classloader-append-dirs>/home/jeus/lib/append</jeus-classloader-append-dirs>
    <boot-classloader-append-class-path>/home/jeus/boot/classes:/home/jeus/boot/a.jar</boot-
classloader-append-class-path>
  </user-interceptor>
</server>
```

각 설정 항목에 대한 설명은 다음과 같다. 클래스 패스에 추가할 항목이 여러 개인 경우 각 항목은 경로 구분자를 사용하여 구분한다.

항목	설명
jeus-classloader-append-class-path	JEUS 최상위 클래스로더(JEUS Root Classloader)의 클래스 패스에 추가할 항목들을 지정한다.
jeus-classloader-append-dirs	JEUS 최상위 클래스로더(JEUS Root Classloader)의 클래스 패스에 지정한 디렉터리에 속한 모든 클래스 패스 항목들을 추가한다.
boot-classloader-append-class-path	서버 JVM의 시스템 클래스로더 클래스 패스에 추가할 항목들을 지정한다.

2.3.3. Listener 설정

본 절에서는 서버에서 사용하는 네트워크 Listener 설정에 대해서 설명한다.

네트워크 Listener는 서버에서 수행되는 시스템 서비스들이나 각종 엔진에서 참조하여 그 설정을 사용하기 위한 것이다. 이 Listener는 포트 통합 서비스가 적용되어 있기 때문에 각자 다른 서비스나 엔진에서 하나의 Listener를 공유해서 사용하도록 설정할 수도 있다. 극단적인 예로는 기본 Listener 하나만을 열어두고 모든 서비스를 해당 Listener를 통해서 서비스하는 것도 가능하다.



서버가 운영 중일 때 이 값을 변경하려고 하면 반영되지 않는다. 동적 설정 반영의 대상이 아니기 때문에 운영 중인 서버에 이 값을 변경한 경우에는 서버를 재기동해야 적용된다.

콘솔 툴 사용

jeusadmin에서 **add-listener** 명령어를 사용하여 기본 Listener를 설정할 수 있다.

```
[MASTER]domain1.adminServer>add-listener -server server1 -name TestListener -addr 192.168.14.145
-port 8188
Executed successfully, but some configurations were not applied dynamically. It might be necessary to
restart the server.
Check the result using 'list-server-listeners -server server1 -name TestListener.'
```

2.3.4. Thread Pool 설정

본 절에서는 서버에서 사용하는 공용 Thread Pool에 대한 설정을 설명한다.

서버에서 사용되는 서비스들이 전용 Thread Pool을 별도로 설정해서 사용하지 않는다면 모두 이 공용 Thread Pool을 사용한다. 공용 Thread Pool을 사용하는 서비스는 트랜잭션 서비스와 JNDI 서비스, 스케줄러 서비스가 있다. 이 서비스들은 설정에 따라 공용 Thread Pool을 사용할 수도 있고, 전용 Thread Pool을 사용할 수도 있다. 공용 Thread Pool을 사용한다면 서비스에서 사용할 최소 Thread 개수를 미리 할당해 놓을 수 있다.



애플리케이션의 요청을 처리하는 엔진에서는 공용 Thread Pool을 사용하지 않는다. 서비스에서 전용 Thread Pool을 사용하는 방법에 대한 설명은 [JNDI Naming Server](#)와 [트랜잭션 매니저](#)와 "JEUS Scheduler 안내서"를 참고한다.

콘솔 툴 사용

본 절에서는 콘솔 툴로 공용 Thread Pool을 조회하고 변경하는 예제를 설명한다.



예제에서 사용한 명령어들에 대한 자세한 설명은 JEUS Reference 안내서의 "Thread Management 관련 명령어"를 참고한다. 서비스들에서 사용하는 Thread Pool에 대한 설정은 각 서비스의 매뉴얼을 참고한다.

다음은 Thread Pool의 max를 100에서 200으로 변경하고 keep alive time을 5분에서 10분으로 변경하는 예제이다.

```
[MASTER]domain1.adminServer>show-system-thread-pool server1
Shows the current configuration.
the system thread pool of the server (server1)
=====
+-----+-----+
| Min           | 0           |
| Max           | 100          |
| Keep-Alive Time | 300000       |
| Queue Size    | 4096         |
| Max Stuck Thread Time | 3600000     |
| Action On Stuck Thread | IGNORE_AND_REPLACE |
| Stuck Thread Check Period | 300000      |
+-----+-----+
```

Reserved Threads for the Service transaction	0
Reserved Threads for the Service namingserver	0

[MASTER]domain1.adminServer>**modify-system-thread-pool server1 -max 200 -keep 600000**
 Successfully performed the MODIFY operation for the system thread pool of the server (server1), but all changes were non-dynamic. They will be applied after restarting.
 Check the results using "modify-system-thread-pool server1 or show-system-thread-pool server1".

[MASTER]domain1.adminServer>**show-system-thread-pool server1**

Shows the current configuration.

the system thread pool of the server (server1)

Min	0
Max	200
Keep-Alive Time	600000
Queue Size	4096
Max Stuck Thread Time	3600000
Action On Stuck Thread	IGNORE_AND_REPLACE
Stuck Thread Check Period	300000
Reserved Threads for the Service transaction	0
Reserved Threads for the Service namingserver	0

다음은 JNDI 서비스에서 공용 Thread Pool에서 Thread를 미리 할당하도록 설정하는 예제이다.

[MASTER]domain1.adminServer>**show-system-thread-pool server1**

Shows the current configuration.

the system thread pool of the server (server1)

Min	0
Max	200
Keep-Alive Time	600000
Queue Size	4096
Max Stuck Thread Time	3600000
Action On Stuck Thread	IGNORE_AND_REPLACE
Stuck Thread Check Period	300000
Reserved Threads for the Service transaction	0
Reserved Threads for the Service namingserver	0

[MASTER]domain1.adminServer>**modify-system-thread-pool server1 -service namingserver -r 10**
 Successfully performed the MODIFY operation for The namingserver thread pool of the server (server1)., but all changes were non-dynamic. They will be applied after restarting.
 Check the results using "show-system-thread-pool server1 -service namingserver or modify-system-thread-pool server1 -service namingserver".

[MASTER]domain1.adminServer>**show-system-thread-pool server1 -service namingserver**

Shows the current configuration.

the system thread pool of the server (server1)

Min	0
Max	200
Keep-Alive Time	600000
Queue Size	4096
Max Stuck Thread Time	3600000
Action On Stuck Thread	IGNORE_AND_REPLACE
Stuck Thread Check Period	300000
Reserved Threads for the Service transaction	0
Reserved Threads for the Service namingserver	0

Min	0	
Max	200	
Keep-Alive Time	600000	
Queue Size	4096	
Max Stuck Thread Time	3600000	
Action On Stuck Thread	IGNORE_AND_REPLACE	
Stuck Thread Check Period	300000	
Reserved Threads for the Service transaction	0	
Reserved Threads for the Service namingserver	10	
+-----+-----+		

```
[MASTER]domain1.adminServer>modify-system-thread-pool server1 -service namingserver
Shows the current configuration.
The namingserver thread pool of the server (server1).
```

+-----+-----+		
Reserved Threads for the Service namingserver	10	
+-----+-----+		

2.3.5. Lifecycle Invocation 설정

JEUS에서는 서버의 Lifecycle에 맞게 사용자가 원하는 작업을 할 수 있도록 Lifecycle Invocation 기능을 제공하고 있다. 서버는 부팅, 다운 과정에서 각 단계에 사용자가 설정한 이벤트를 호출해준다.



서버가 운영 중일 때 class name을 변경하려고 하면 반영되지 않는다. class name 은 동적 설정 반영의 대상이 아니기 때문에 운영 중인 서버에 이 값을 변경한 경우에는 서버를 재기동해야 적용된다. 그 외의 Invocation 설정은 동적반영된다.

다음은 Lifecycle Invocation에 등록하는 클래스는 일반 Java 클래스 예제이다. 이 클래스를 jar로 패키징 하여 서버의 Lifecycle에 맞게 호출하고 싶은 대상 서버의 SERVER_HOME/lib/application에 위치시킨다.

Lifecycle Invocation에 등록하는 클래스 : <LifeCycleTester.java>

```
package lifecycle;

public class LifeCycleTester {
    public void boot() {
        System.out.println("Boot");
        // do somethig
    }

    public void beforeDeploy() {
        System.out.println("Before Deploy");
        // do somethig
    }

    public void afterDeploy() {
        System.out.println("After Deploy");
        // do somethig
    }
}
```

```

public void ready() {
    System.out.println("Ready");
    // do somethig
    try {
        System.out.println("Sleeping for 15 seconds ....");
        Thread.sleep(15000L);
    } catch (Exception e) {
        //ignored
    }
}

public void beforeUndeploy() {
    System.out.println("Before Undeploy");
    // do somethig
}

public void afterUndeploy() {
    System.out.println("After Undeploy");
    // do somethig
}
}

```

콘솔 툴 사용

콘솔 툴을 사용해서 위에서 만든 예제를 Lifecycle Invocation 클래스로 설정하는 방법에 대해 설명한다.

콘솔 툴에서 **add-lifecycle-invocation** 명령어를 통해 서버의 Lifecycle Invocation을 추가하고, **add-invocation-library** 명령어 및 **add-invocation** 명령어를 통해 Library Ref 및 Invocation을 추가할 수 있다. 이에 대한 자세한 설명은 JEUS Reference 안내서의 "add-lifecycle-invocation", "add-invocation-library", "add-invocation"을 참고한다.

```

[MASTER]domain1.adminServer>add-lifecycle-invocation lifecycle.LifecycleTester -s adminServer -m boot
-type BOOT

```

Successfully performed the ADD operation for Lifecycle Invocation Class [lifecycle.LifecycleTester] and Invocation [boot](Invocation ID = 0), but all changes were non-dynamic. They will be applied after restarting.

Check the results using "list-lifecycle-investigations".

```

[MASTER]domain1.adminServer>list-lifecycle-investigations

```

List of Lifecycle investigations

```

=====
+-----+-----+-----+-----+
| Target | Lifecycle Invocation Class | Invocation | Invocation |
|         |                             | Library Ref |             |
+-----+-----+-----+-----+
| [Server]admin | test.lifecycle.invocation.Lif | lib1 | [1]boot |
| Server        | eCycleInvocation              |      |          |
+-----+-----+-----+-----+
| [Server]admin | lifecycle.LifecycleTester |      | [0]boot |
| Server        |                             |      |          |
+-----+-----+-----+-----+
=====

```

Use the "lifecycle-invocation-info" command for more information about Lifecycle invocation.

2.3.6. Resource Reference 설정

서버의 애플리케이션에서 공통으로 사용할 리소스에 대한 매핑 정보를 설정한다.

애플리케이션에서 사용하는 리소스는 해당 애플리케이션이 서비스되는 서버의 JNDI 저장소에 이름이 등록된다. 리소스 매핑을 설정하면 등록되는 이름에 상관없이 애플리케이션에서는 항상 같은 이름의 리소스를 Lookup해서 사용할 수 있다.



서버가 클러스터에 포함되어 있는 경우에는 Resource Reference 설정은 클러스터에 설정된 값이 우선 적용된다.

Resource Reference는 다음과 같이 domain.xml을 편집하여 설정할 수 있다.

```
<server>
  <name>server1</name>
  ...
  <res-ref>
    <jndi-info>
      <ref-name>/jdbc/DB1</ref-name>
      <export-name>db1</export-name>
    </jndi-info>
  </res-ref>
</server>
```

2.4. Server Template 설정

클러스터를 생성할 때에 사용할 서버들의 공통적인 설정들을 서버 템플릿으로 저장해 둘 수 있다. 이를 이용하여 클러스터 생성 시에 공통 설정을 가지는 다수의 서버를 자동으로 생성할 수 있어서 간단하게 서버 클러스터 환경을 구성할 수 있다.

콘솔 툴 사용

콘솔 툴을 이용하여 서버 템플릿을 추가할 수 있다. **add-server-template** 명령어를 통해 서버 템플릿을 추가하고, 각종 옵션을 주어 세부 항목을 설정한다. add-server-template에 대한 자세한 설명은 JEUS Reference 안내서의 "add-server-template"을 참고한다.

```
[MASTER]domain1.adminServer>add-server-template template1 -m true
Successfully performed the ADD operation for server template (template1).
NOTICE : base-addr [0.0.0.0] base-port [9736] http-port [8088]
Check the results using "list-servers or show-server-template or add-server-template".

[MASTER]domain1.adminServer>show-server-template
Shows the current configuration.
Server template list
=====
+-----+-----+
| server templates | template1 |
+-----+-----+
```

=====

3. JEUS 서버 제어 및 모니터링

본 장에서는 JEUS 서버를 제어하고 모니터링하는 방법을 설명한다. 여기에서 설명하는 서버는 MS에 한정되며, 관리를 담당하는 MASTER가 정상 동작 중이라고 가정한다.



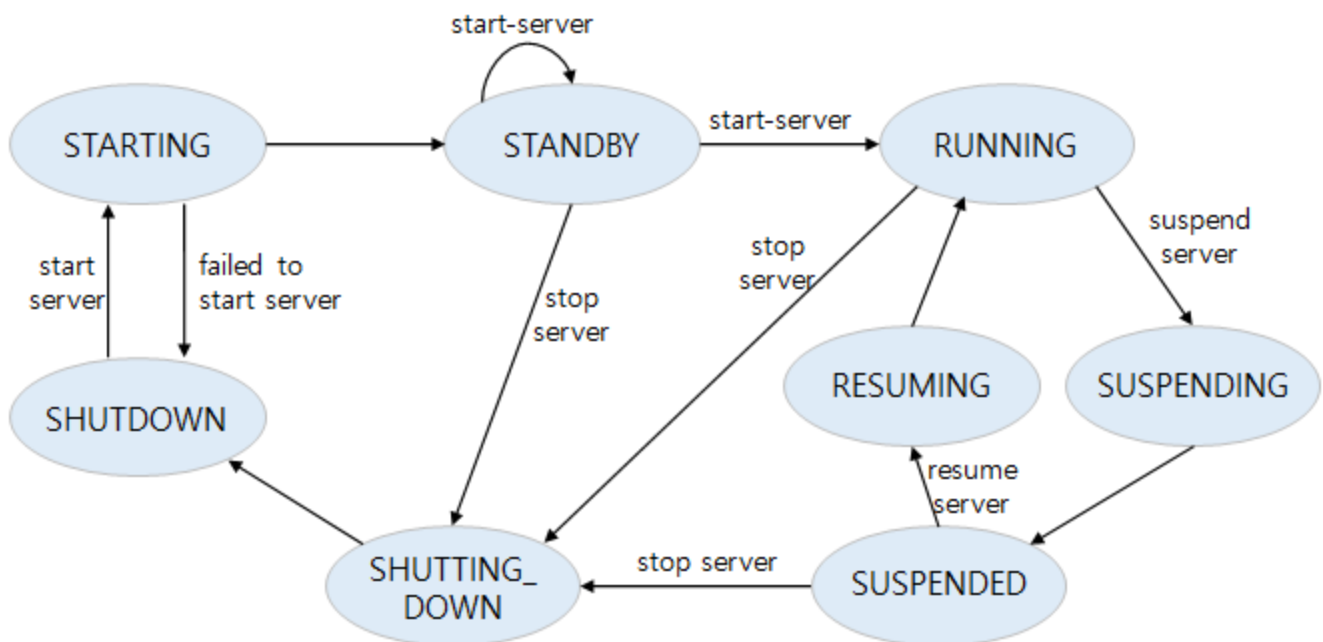
서버 실행 스크립트와 콘솔 툴에 관한 자세한 사항은 JEUS Reference 안내서의 "JEUS 서버 기동 및 종료"와 "Local 명령어"를 참고한다.

3.1. 서버 제어 및 모니터링

본 절에서는 서버 제어 및 모니터링에 대해서 설명한다.

3.1.1. Managed Server의 Lifecycle

MS는 다음과 같은 동작 상태를 갖는다. MS의 상태는 MASTER 또는 사용자에게 의해 변경될 수 있다.



MS의 상태 전이도

구분	설명
SHUTDOWN	서버가 아직 부팅되지 않았거나 정상적으로 종료된 상태이다.
STARTING	서버가 시작 명령을 받고 부팅 중인 상태이다. 이 단계에서 서버는 엔진(WEB, EJB, JMS) 생성, JEUS 서비스(보안, SCF 등) 실행, 그리고 등록된 애플리케이션 deploy 등의 작업을 수행한다.
STADNBY	서버에서 실행되어야 하는 JEUS 서비스가 시작된 상태이다. 만약 서버가 이 상태로 부팅을 완료했다면 서버에 등록된 애플리케이션이 deploy를 실패한 것이다. 서버에 등록된 애플리케이션 distribute가 실패한 경우에도 서버는 STADNBY 상태에 머무르게 된다.

구분	설명
RUNNING	서버가 부팅이 완료되고 등록된 애플리케이션들도 정상적으로 서비스되는 상태이다. 만약 STANDBY 상태의 서버를 -force 옵션을 통해 시작시킨 상황이라면 서버에 등록된 모든 애플리케이션이 서비스되지 못하는 상태일 수도 있다.
SUSPENDING	RUNNING 상태인 서버가 suspend 명령을 수행 중인 상태이다. 이 단계에서는 서버에서 서비스되고 있는 모든 애플리케이션의 서비스를 중단된다. 단, 애플리케이션 서비스를 위한 Listener는 내리지 않는다.
SUSPENDED	서버에 deploy된 애플리케이션을 모두 중지시켜 서비스가 되지 않도록 멈춘 상태이다. 단, 애플리케이션 서비스를 위한 Listener는 내리지 않고 그대로 있다.
RESUMING	SUSPENDED 상태의 서버가 resume 명령을 수행 중인 상태이다. 이 상태에서는 중지된 모든 애플리케이션을 다시 시작시켜 서비스 가능한 상태로 만든다. resume 명령이 성공적으로 수행되면 서버는 RUNNING 상태가 된다.
SHUTTING_DOWN	서버가 종료되고 있는 상태이다. deploy된 애플리케이션을 undeploy하고 부팅할 때 실행시켰던 JEUS 서비스들을 모두 종료시킨다. 서버를 종료할 때 적절한 타임아웃을 설정해서 서비스 중인 요청이 처리될 수 있도록 보장할 수 있다. 이를 Graceful Shutdown이라고 한다.



1. MASTER에서 관리하는 MS의 Lifecycle에 대한 자세한 내용은 JEUS Domain 안내서의 "서버 Life Cycle 상태 확인"을 참고한다.
2. Graceful Shutdown에 대한 자세한 내용은 [Managed Server 종료](#)를 참고한다.

3.1.2. Managed Server 시작

MASTER와 MS는 기본적으로 Launcher를 통해 기동된다. 스크립트나 MASTER를 통한 서버 실행 명령은 Launcher를 실행하게 되고, Launcher에서는 서버를 띄우기 위한 준비작업과 실제 서버를 기동하는 작업을 수행한다. Launcher에 대한 자세한 내용은 [Launcher](#)를 참고한다. 콘솔 툴을 통해 서버를 시작하는 방법은 JEUS Domain 안내서의 [Managed Server\(MS\) 시작](#)을 참고한다. 본 절에서는 스크립트로 서버를 실행하는 예제만 설명한다.

예제

Launcher에서는 실제 서버의 JVM을 기동하고 해당 서버의 부팅이 완료된 것을 확인한 뒤에 종료된다.

다음과 같이 Launcher 로그를 통해 서버가 정상적으로 기동되었는지 확인할 수 있다.

```

JEUS_HOME/bin$ ./startManagedServer -domain jeus_domain -server server1 -u jeus -p jeus -masterurl
localhost:9736
*****
- JEUS Home       : /home/jeus
- Java Vendor     : Sun
- Added Java Option :
*****

```



```
...
[2024.09.25 22:35:04][2] [launcher-1] [SERVER-0201] Successfully connected to the JEUS Master
Server(localhost:9736).
[2024.09.25 22:35:04][2] [launcher-1] [Launcher-0058] All local configurations are up-to-date.
...
[2024.09.25 22:35:07][0] [server1-1] [SERVER-0242] Successfully started the server.
[2024.09.25 22:35:07][2] [server1-1] [SERVER-0248] The JEUS server is RUNNING.
[2024.09.25 22:35:07][2] [launcher-22] [Launcher-0034] The server[server1] initialization completed
successfully[pid : 81719].
[2024.09.25 22:35:07][0] [launcher-1] [Launcher-0040] Successfully started the server[server1]. The
server state is now RUNNING.
JEUS_HOME/bin$
```

MS가 자신이 갖고 있는 애플리케이션 파일을 부팅 중 deploy할 때 문제가 발생할 경우 STANDBY 상태에 머무를 수 있다. MS가 부팅 중 deploy하는 애플리케이션은 보통 한 번 이상 정상적으로 서비스되던 것이다. 그렇기 때문에 이러한 문제가 발생했다면 이전과 비교하여 파일이 잘못 변경되었을 가능성이 크다.

특정 MS에서 JEUS 내부 디렉터리인 .workspace에 존재하는 애플리케이션의 xml 태그를 잘못 수정할 경우 아래와 같이 STANDBY 상태에 머무를 수 있다. 도메인 구조의 deploy에 대한 자세한 내용은 JEUS Applications & Deployment 안내서의 "도메인 환경에서 애플리케이션 관리"를 참고한다.



애플리케이션의 관리는 MASTER를 통해서 하는 것이 기본 정책이므로, MS의 .workspace 디렉터리에 직접 접근하여 작업하지 않도록 한다.

```
JEUS_HOME/bin$ ./startManagedServer -domain jeus_domain -server server1 -u jeus -p jeus -masterurl
localhost:9736
...
[2024.09.25 23:20:07][2] [launcher-1] [SERVER-0201] Successfully connected to the JEUS Master
Server(localhost:9736).
...
[2024.09.25 23:20:09][2] [server1-1] [Deploy-0095] Distributing the application[healthcheck].
java.lang.RuntimeException: [was class com.ctc.wstx.exc.WstxParsingException] Unexpected close tag
</urlpattern>; expected </url-pattern>.
[2024.09.25 23:20:09][0] [server1-1] [SERVER-0522] An exception occurred while processing
[SERVER_HOME/.workspace/deployed/healthcheck/1657613257716/healthcheck_war___/WEB-INF/web.xml].
<<__Exception__>>
...
<<__!Exception__>>
[2024.09.25 23:20:09][2] [server1-1] [SERVER-0248] The JEUS server is STANDBY.
[2024.09.25 23:20:09][0] [server1-1] [SERVER-0250] Starting server (server1) failed. Staying in
STANDBY.
[2024.09.25 23:20:09][2] [server1-1] [SERVER-0401] The elapsed time to start: 1775ms.
[2024.09.25 23:20:09][2] [launcher-22] [Launcher-0034] The server[server1] initialization completed
successfully[pid : 86118].
[2024.09.25 23:20:09][0] [launcher-1] [Launcher-0042] Completed starting the server but the server
state is still STANDBY.
JEUS_HOME/bin$
```

이 경우 다음과 같이 **server-info** 명령어를 통해 MS가 STANDBY 상태임을 확인할 수 있다.

```
[MASTER]jeus_domain.adminServer>server-info
```

Information about Domain (jeus_domain)

Server	Status	Node Name	PID	Cluster	Latest Start Time / Shutdown Time	Need to Restart	Listen Ports	Running Engines
adminServer (*)	RUNNING (00:52:37)	node1	81482	N/A	2024-09-25 (목) 오후 10:34:03 KST	false	base-0.0.0.0:9736 http-server-0.0.0.0:8088	jms, web, ejb
server1	STANDBY (00:06:32)	N/A	86118	N/A	2024-09-25 (목) 오후 11:20:08 KST	false	base-0.0.0.0:9836 http-server-0.0.0.0:8188	jms, web, ejb

3.1.3. Managed Server 종료

MS의 종료는 **jeusadmin** 명령어를 통해 가능하다. 특정 MS에 접속한 콘솔 툴에서 local-shutdown 명령어를 통해 접속한 MS를 종료할 수도 있으며, MASTER에 접속한 콘솔 툴을 통한 명령어로 종료할 수도 있다. MS를 종료하는 방법에 대한 자세한 내용은 JEUS Domain 안내서의 "Managed Server(MS) 종료"를 참고한다.



MS는 MASTER의 관리를 받는 것이 기본 정책이므로 MASTER를 통해 종료하는 것을 권장한다.

예제

MS를 종료할 때 현재 수행 중인 요청에 대한 처리를 보장해주는 Graceful Shutdown 기능이 존재한다. 서버 종료를 요청한 시점부터는 새로운 요청은 처리하지 않으나, 현재 수행 중인 요청에 대해서는 정해진 시간만큼 처리가 끝나길 기다려 줄 수 있다.

다음은 MS 종료 명령에 Timeout 옵션을 통해 대기시간을 설정하는 예제이다.

```
[MASTER]jeus_domain.adminServer>stop-server server1 -to 60000
Stop server message to server [server1] was successfully sent.
```

콘솔 툴을 통해 해당 MS에 직접 접속한 경우 역시 Timeout 지정이 가능하다.

```
jeus_domain.server1>local-shutdown -to 60000
Executing this command affects the service. Do you want to continue? (y/n)y
The server [server1] has been shut down successfully.
```

```
offline>
```

정상적으로 종료된 서버에서는 다음과 같은 로그를 확인할 수 있다.

```
...
[2024.09.25 23:39:28][2] [server1-25] [SERVER-0248] The JEUS server is SHUTDOWN.
[2024.09.25 23:39:28][2] [server1-24] [NET-0214] Closing http-server.
[2024.09.25 23:39:28][0] [server1-25] [SERVER-0265] The JEUS server has exited.
[2024.09.25 23:39:28][0] [server1-1] [SERVER-0099] The server[server1] has been shut down.
[2024.09.25 23:39:28][0] [server1-20] [SERVER-0565] The JVM process is shutting down.
[2024.09.25 23:39:28][0] [server1-20] [SERVER-0566] The JVM process will be terminated.
[2024.09.25 23:39:28][2] [server1-22] [NET-0214] Closing base.
```

3.1.4. Managed Server 일시 정지

Managed Server의 종료는 애플리케이션 서비스를 모두 내리고 Managed Server JVM을 종료하는 과정이다. 이와 달리 서비스하고 있던 모든 애플리케이션의 일시 정지 기능이 제공된다. 여기서도 마찬가지로 Graceful Timeout을 적용하여 현재 수행 중인 요청 처리 완료를 기다릴 수 있다. 콘솔 툴의 **suspend-server** 명령어를 통해서 동작 가능하다.

다음은 콘솔 툴을 통해 서버를 일시 정지하는 예제이다.

```
[MASTER]jeus_domain.adminServer>suspend-server -servers server1
Executing this command affects the service. Do you want to continue? (y/n)y
Successfully suspended server(s).
```

server-info 명령어로 서버가 SUSPENDED 상태가 된 것을 확인할 수 있다.

```
[MASTER]jeus_domain.adminServer>server-info
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Server | Status | Node | PID | Clu | Latest Start | Need | Listen | Runni |
|         |         | Name |     | ster | Time /       | to   | Ports  | ng    |
|         |         |      |     |      | Shutdown Time | Restart |        | Engines |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| admin | RUNNING | nod | 880 | N/A | 2024-09-25 | false | base-0.0. | jms, |
| Server | (00:22:1 | e1 | 29 |   | (목) 오후   |      | 0.0:9736 | web, |
| (*)   | 4)      |    |    |   | 11:38:36 KST |      | http-serv | ejb  |
|         |         |    |    |   |              |      | er-0.0.0.0 |      |
|         |         |    |    |   |              |      | :8088      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| serve | SUSPEND | N/A | 891 | N/A | 2024-09-25 | false | base-0.0. | jms, |
| r1    | ED(00:08 |    | 46 |   | (목) 오후   |      | 0.0:9836 | web, |
|        | :01)     |    |    |   | 11:52:49 KST |      | http-serv | ejb  |
|        |         |    |    |   |              |      | er-0.0.0.0 |      |
|        |         |    |    |   |              |      | :8188      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
=====
```

정상적으로 일시 정지된 서버에서는 다음과 같은 로그를 확인할 수 있다.

```
[2024.09.25 00:15:16][2] [server1-25] [SERVER-0248] The JEUS server is SUSPENDING.
[2024.09.25 00:15:16][2] [server1-25] [JMS-7375] The persistence store manager for the JMS server
'server1' has been shut down.
[2024.09.25 00:15:16][2] [server1-25] [SERVER-0248] The JEUS server is SUSPENDED.
```



MASTER의 제어를 받지 않고 있는 경우에도 해당 서버에 직접 콘솔 툴에 접속하여 MS를 일시 정지시킬 수 있다.

3.1.5. Managed Server 복귀

일시 정지되었던 서버의 애플리케이션들을 다시 서비스하도록 복귀시킬 수 있다.

MASTER의 제어를 받지 않고 있는 경우에도 해당 서버에 직접 콘솔 툴에 접속하여 SUSPENDED 상태인 MS를 다시 시작시킬 수 있다.

다음은 콘솔 툴을 통해 서버를 복귀시키는 예제이다.

```
[MASTER]jeus_domain.adminServer>resume-server -servers server1
Successfully resumed the servers.
```

server-info 명령어를 통해 서버가 다시 RUNNING 상태가 된 것을 확인할 수 있다.

```
[MASTER]jeus_domain.adminServer>server-info
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Server | Status | Node | PID | Clu | Latest | Need | Listen | Running |
|         |         | Name |     | ster| Start Time | to   | Ports  | Engines |
|         |         |      |     |     | / Shutdown| Restart |        |          |
|         |         |      |     |     | Time      |      |        |          |
+-----+-----+-----+-----+-----+-----+-----+-----+
| adminS | RUNNIN | nod | 121 | N/A | 2024-09-25 | false | base-0.0. | jms, |
| erver  | G(00:04 | e1 | 360 |     | (금) 오전 |      | 0.0:9736 | web, ejb|
| (*)   | :21)   |    |     |     | 11:30:11 KST |      | http-serv |      |
|         |         |    |     |     |      |      | er-0.0.0.0 |      |
|         |         |    |     |     |      |      | :8088      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
| server1 | RUNNIN | N/A | 122 | N/A | 2024-09-25 | false | base-0.0. | jms, |
|         | G(00:01 |    | 223 |     | (금) 오전 |      | 0.0:9836 | web, ejb|
|         | :07)   |    |     |     | 11:33:26 KST |      | http-serv |      |
|         |         |    |     |     |      |      | er-0.0.0.0 |      |
|         |         |    |     |     |      |      | :8188      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
=====
```

정상적으로 애플리케이션 서비스가 다시 시작된 서버에서는 다음과 같은 로그를 확인할 수 있다.

```
[2024.09.25 11:34:29][2] [server1-26] [SERVER-0248] The JEUS server is RESUMING.  
[2024.09.25 11:34:29][2] [server1-26] [SERVER-0248] The JEUS server is RUNNING.
```

3.2. 서버 엔진 설정

JEUS 서버에는 애플리케이션을 서비스하는 엔진이 내부적으로 존재한다. 웹 애플리케이션 서비스, EJB 애플리케이션 서비스, JMS 서비스를 위한 엔진이 각각 존재한다. 이들 각 엔진의 사용 여부는 초기화 시점을 설정을 통해 제어할 수 있다. 단, 현재 이들 설정은 모두 서버를 재기동해야 적용된다.

3.2.1. 엔진 사용 여부 설정

JEUS 서버의 각 엔진의 사용 여부를 설정한다. 각 엔진에 대하여 사용 설정을 할 경우 각 엔진들은 서버가 기동되거나 각 엔진에서 서비스될 첫 번째 애플리케이션이 deploy될 때 필요한 엔진이 초기화된다. 사용 설정을 하지 않은 경우 서버가 기동된 후에도 엔진은 초기화되지 않으므로 해당 엔진에서 서비스될 애플리케이션들을 deploy하면 실패하게 된다.



엔진 사용가능할 때 정상적으로 deploy된 애플리케이션을 이후 엔진 미사용을 한 후 서버를 기동할 경우 해당 애플리케이션은 deploy 실패하게된다. 그리고 서버의 상태도 서비스 가능상태(RUNNING)가 아닌 대기(STANDBY)로 전환됨으로 관리자가 이를 확인하여 엔진의 상태와 애플리케이션의 상태를 수정해주어야 한다.

콘솔 툴에서 서버 내 각 엔진 사용 여부를 설정하는 방법은 다음과 같다.

```
[MASTER]jeus_domain.adminServer>disable-engines adminServer -all  
[adminServer]  
Change Engine to Disabled: Web EJB JMS  
  
Applying configuration ...  
=====
```

Result
Successfully changed only the JEUS Domain Configuration.
Restart the server to apply the changes.

```
=====
```

...

```
[MASTER]jeus_domain.adminServer>enable-engines adminServer -all  
[adminServer]  
Change Engine to Enabled: Web EJB JMS  
  
Applying configuration ...  
=====
```

Result
Successfully changed only the JEUS Domain Configuration.
Restart the server to apply the changes.

```
=====
```

```

+-----+
=====
...
[MASTER]jeus_domain.adminServer>disable-engines adminServer -web -ejb
[adminServer]
Change Engine to Disabled: Web EJB

Applying configuration ...
=====
+-----+
|                                     Result                                     |
+-----+
| Successfully changed only the JEUS Domain Configuration.                    |
| Restart the server to apply the changes.                                    |
+-----+
=====

...
[MASTER]jeus_domain.adminServer>enable-engines adminServer -web -ejb
[adminServer]
Change Engine to Enabled: Web EJB

Applying configuration ...
=====
+-----+
|                                     Result                                     |
+-----+
| Successfully changed only the JEUS Domain Configuration.                    |
| Restart the server to apply the changes.                                    |
+-----+
=====

```



JEUS 콘솔 툴의 서버 내 엔진 설정 명령어에 대한 자세한 정보는 JEUS Reference 안내서의 "disable-engines"와 "enable-engines"를 참고한다.

3.2.2. 엔진 초기화 시점 설정

JEUS 서버의 각 엔진의 사용 여부를 설정한다. 각 엔진에 대하여 사용 설정을 할 경우 각 엔진들은 엔진 초기화와 동시에 또는 각 엔진에서 서비스될 첫 번째 애플리케이션이 deploy될 때 필요한 엔진이 초기화된다. 사용 설정을 하지 않은 경우 서버가 기동된 후에도 엔진은 초기화되지 않으므로 해당 엔진에서 서비스될 애플리케이션들을 deploy할 경우 실패하게 된다. 콘솔 툴을 통해 각 엔진별 사용 여부를 설정할 수 있다.

콘솔 툴에서 서버 내 각 엔진의 초기화 시점을 설정하는 방법은 다음과 같다.

```

[MASTER]jeus_domain.adminServer>disable-engine-init-on-boot adminServer,server1
EngineInitOnBoot was successfully disabled.
Applying configuration ...
=====
+-----+
|                                     Result                                     |
+-----+
| Successfully changed only the JEUS Domain Configuration.                    |
| Restart the server to apply the changes.                                    |
+-----+
=====

```

```

+-----+
=====

[MASTER]jeus_domain.adminServer>enable-engine-init-on-boot adminServer
EngineInitOnBoot was successfully enabled.
Applying configuration ...
=====
+-----+
|                                     |
|                               Result |
|                                     |
+-----+
| Successfully changed only the JEUS Domain Configuration. |
| Restart the server to apply the changes.                  |
|                                     |
+-----+
=====

```



JEUS 콘솔 툴의 서버 내 엔진 설정 명령어에 대한 자세한 정보는 JEUS Reference 안내서의 "disable-engine-init-on-boot"와 "enable-engine-init-on-boot"를 참고한다.

3.3. Thread 모니터링 및 제어

JEUS에서는 서블릿 Thread와 EJB 리모트 Thread, System Thread Pool에 대한 모니터링 기능과 Thread의 Stack을 확인할 수 있는 기능, 그리고 특정 Thread에 Interrupt Signal을 보낼 수 있는 기능을 제공한다.

3.3.1. Thread 모니터링

JEUS에서는 Thread를 모니터링하는 기능을 제공한다. 모니터링의 대상이 되는 Thread는 서블릿 Thread, EJB 리모트 요청 Thread 그리고 System Thread Pool이다.

서블릿 Thread나 EJB 리모트 요청 Thread의 모니터링 기능을 사용하면 Thread ID, Thread 이름, Thread 상태, 처리시간 등의 정보를 알 수 있고 Thread Pool 모니터링 기능을 사용하면 System Thread Pool의 core size, max size, keep alive time 그리고 System Thread Pool에 있는 Thread 정보를 알 수 있다.

Thread 정보 확인

콘솔 툴에서 Thread 정보를 확인하는 방법은 다음과 같다.

```

[MASTER]jeus_domain.adminServer>thread-info -server adminServer

Thread information for the server [adminServer]
There are no EJB RMI threads for the server [adminServer].
There is no batch thread pool in server [adminServer].

=====
The web engine threads for 'adminServer_web-WebContainerThreadPool'.

+-----+-----+-----+-----+-----+
| tid | name | state | elapsed | uri |
+-----+-----+-----+-----+-----+

```

50	adminServer_web-WebContainerThreadPool-1	waiting	20755	
59	adminServer_web-WebContainerThreadPool-10	waiting	20745	
51	adminServer_web-WebContainerThreadPool-2	waiting	20403	
52	adminServer_web-WebContainerThreadPool-3	waiting	20395	
53	adminServer_web-WebContainerThreadPool-4	waiting	9505	
54	adminServer_web-WebContainerThreadPool-5	waiting	15399	
55	adminServer_web-WebContainerThreadPool-6	waiting	20386	
56	adminServer_web-WebContainerThreadPool-7	waiting	4507	
57	adminServer_web-WebContainerThreadPool-8	waiting	20765	
58	adminServer_web-WebContainerThreadPool-9	waiting	23346	

elapsed: Elapsed time (ms)

Thread statistics for the 'adminServer_web-WebContainerThreadPool'.

	total	active	idle	blocked	reconn
The number of threads.	10	0	10	0	0

total = active + idle, reconn: reconnecting

The threads for the 'jeus.ejb.asyncpool' thread pool.

tid	name	thread state	active thread
(No data available)			

The statistics for the 'jeus.ejb.asyncpool' thread pool.

pool name	minimum pool size	maximum pool size	current pool size	work queue size	remaining work queue size
jeus.ejb.a syncpool	0	30	0	4096	4096

The threads for the 'EJBTimerService' thread pool.

tid	name	thread state	active thread
48	EJBTimerService-1	WAITING	false
49	EJBTimerService-2	WAITING	false

=====

The statistics for the 'EJBTimerService' thread pool.

pool name	minimum pool size	maximum pool size	current pool size	work queue size	remaining work queue size
EJBTimer Service	2	30	2	4096	4096

=====

The threads for the 'threadpool.System' thread pool.

tid	name	thread state	active thread
72	threadpool.System-1	TIMED_WAITING	false
73	threadpool.System-2	TIMED_WAITING	false
83	threadpool.System-3	TIMED_WAITING	false

=====

The statistics for the 'threadpool.System' thread pool.

pool name	minimum pool size	maximum pool size	current pool size	work queue size	remaining work queue size
threadpool.System	0	100	3	4096	4096

=====

The threads for the 'LPQ-INTERNAL' thread pool.

tid	name	thread state	active thread
(No data available)			

=====

The statistics for the 'LPQ-INTERNAL' thread pool.

pool name	minimum pool size	maximum pool size	current pool size	work queue size	remaining work queue size
LPQ-INTERNAL	0	50	0	4096	4096

=====

The threads for the 'JMS-INTERNAL' thread pool.

tid	name	thread state	active thread
38	JMS-INTERNAL-3	WAITING	false
41	JMS-INTERNAL-6	WAITING	false
42	JMS-INTERNAL-7	WAITING	false
43	JMS-INTERNAL-8	WAITING	false
45	JMS-INTERNAL-10	WAITING	false
40	JMS-INTERNAL-5	WAITING	false
44	JMS-INTERNAL-9	WAITING	false
37	JMS-INTERNAL-2	WAITING	false
39	JMS-INTERNAL-4	WAITING	false
36	JMS-INTERNAL-1	WAITING	false

The statistics for the 'JMS-INTERNAL' thread pool.

pool name	minimum pool size	maximum pool size	current pool size	work queue size	remaining work queue size
JMS-INTERNAL	10	20	10	4096	4096

특정 Thread의 Stack Trace 조회

콘솔 툴에서 특정 Thread의 Stack Trace를 조회하는 방법이다. 주로 Thread 정보를 조회하는 명령(콘솔 툴에서의 **ti** 명령어)을 통해 확인했을 때 블록된 Thread가 있을 경우 해당 Thread의 Stack을 확인하는 데 사용한다.

```
[MASTER]jeus_domain.adminServer>print-stack-trace -server server1 45
servlet thread [tid=45] Stack trace of server1_web-WebContainerThreadPool-1 tid=45
java.lang.Thread.State: WAITING
    at sun.misc.Unsafe.park(Native Method)
    at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
    at
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
    at jeus.util.pool.auto.LinkBlockingQueue.take(LinkBlockingQueue.java:450)
    at jeus.util.pool.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1418)
    at jeus.util.pool.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:1340)
    at jeus.servlet.thread.WebThreadPoolExecutor$WebRequestWorker.run(WebThreadPoolExecutor.java:273)
    at java.lang.Thread.run(Thread.java:748)
```



JEUS 콘솔 툴의 **ti** 명령어와 **strace** 명령어에 대한 자세한 정보는 JEUS Reference 안내서의 "Local 명령어"를 참고한다.

3.3.2. Thread 제어

JEUS에서는 특정 Thread에 Interrupt Signal을 보내서 수행 중인 작업을 중단할 수 있도록 Exception을 발생시키는 기능이 있다.



이 기능은 현재 실험적으로 제공되는 기능으로 Thread에 Interrupt가 걸려 있으면 수행 중인 Thread가 바로 중단되는 것이 아니라 Interrupt Status를 체크하여 Exception을 던져 더 이상 진행하지 않도록 유도하는 것이기 때문에 발생하는 Exception에 대해서는 사용자 애플리케이션에서 처리해야 한다. Exception을 제대로 처리하지 않아서 발생하는 문제에 대해서는 사용자에게 책임이 있다.

Thread Interrupt

Thread에 Interrupt를 건다는 것은 동작하고 있는 Thread에 Interrupt Signal을 보내서 현재 진행 중인 동작의 종단을 유도하기 위해 Exception을 발생시켜 이후의 작업을 더 이상 진행하지 않도록 힌트를 주는 것이다. 주로 요청이 블록되어 있거나, 예상한 시간보다 오래 지체되어 있는 경우에 사용될 수 있다.

실제로 Thread에 Interrupt가 걸린다고 해서 해당 Thread가 강제적으로 멈춘다든가, Interrupt를 발생시키는 순간에 바로 효과가 있는 것은 아니다.

JEUS에서는 서블릿 Thread, EJB 리모트 요청 Thread와 System Thread Pool의 수행 중인 Thread에 Interrupt Signal을 보낼 수 있는 기능을 지원한다. 이때 Thread가 JNDI 원격 커넥션이나 EJB operation, JDBC operation, 또는 I/O 작업을 진행 중이면 Interrupt가 걸릴 수 있다.

• 서블릿 Thread

서블릿 Thread는 다음의 경우들에 대해 Interrupt가 걸려 있으면 Exception이 발생한다. 이때 발생하는 Exception에 대한 처리는 사용자 애플리케이션에서 해야 한다.

- JNDI 원격 커넥션을 맺으려고 할 때
- JNDI 원격 operation을 수행하려고 할 때
- EJB 호출하려고 할 때
- JDBC 커넥션을 얻어올 때 또는 커넥션 객체의 메소드를 호출할 때

콘솔 툴을 통해 관리자가 직접 Interrupt를 걸 수 있으며, 서블릿 Thread의 수행시간을 체크하여 자동으로 Interrupt 되도록 하는 기능을 제공한다. 이 기능에 대한 설정 방법은 JEUS Web Engine 안내서의 "자동 Thread Pool 관리 설정(Thread 상태 통보)"를 참고한다.

• EJB 리모트 요청 Thread

EJB operation의 경우에는 다음의 시점에 Interrupt Status를 체크해서 jakarta.ejb.EJBException을 발생시킨다.

- EJB 2.x 방식으로 create와 같은 EJBHome 메소드를 호출할 때(EJB 3.0의 경우 사용자가 EJB Lookup을 하면 EJB 컨테이너 내부적으로 create 처리)
- EJB 비즈니스 메소드를 호출할 때

이때 발생하는 Exception에 대해서는 사용자 애플리케이션에서 적절하게 처리해주어야 한다. 만약 EJB 비즈니스 메소드 호출이 이미 들어간 상태에서 Interrupt Signal을 받았다면 Exception이 발생하지 않을 수 있다. 하지만 해당 메소드에서 다른 EJB를 호출한다면, JDBC operation이나 JNDI operation을 사용하는 경우라면 Interrupt가 걸릴 수도 있다.

• JDBC

JDBC operation의 경우에는 다음의 operation을 수행하려고 할 때 Interrupt Signal을 받았다면 `java.sql.SQLException`이 발생한다. 이때 발생하는 Exception에 대해서는 사용자 애플리케이션에서 적절하게 처리해주어야 한다.

- `DataSource#getConnection()`을 통해 Connection Pool에서 커넥션을 가져올 때
- Connection Pool에서 가져온 `java.sql.Connection`에 대해서 operation을 수행하려고 할 때
- JDBC Connection Pool 설정에 `use-sql-trace`나 `stmt-caching-size` 옵션을 설정하고 `executeQuery`와 같은 operation을 수행하려고 할 때(옵션에 대한 자세한 설명은 [Connection Pool 설정](#)을 참고한다.)

• JNDI

JNDI operation을 수행하는 중에도 Interrupt의 효과가 있는 경우가 있다. 다음의 JNDI operation을 수행하려고 할 때 Interrupt Signal을 받았다면 `javax.naming.NamingException`이 발생한다. 이때 발생하는 Exception에 대해서는 사용자 애플리케이션에서 적절하게 처리해주어야 한다.

- JNDI 원격 커넥션을 맺을 때(`javax.naming.InitialContext`를 생성하는 시점)
- JNDI operation을 하기 위해 원격 메시지를 보낼 때

`Lookup`, `Bind`, `Rebind`, `Rename`과 같은 JNDI operation일 경우이다. 단, 로컬(같은 JVM)에 있는 레지스트리에 접근하거나 캐시에 접근하는 경우에는 Interrupt의 효과가 없다.

• I/O 작업

I/O 작업을 할 때는 항상은 아니지만 상황에 따라 Interrupt의 효과가 있을 수 있다.

- `java.nio.channels.SocketChannel`

`java.nio.channels.SocketChannel`을 사용하는 경우에는 Interrupt에 걸리게 된다.

`java.nio.channels.SocketChannel`을 사용해서 연결할 때는 3단계로 분리된다.

첫 번째 단계인 `begin` 단계에서는 Thread에 Interrupt Status가 설정되었는지 체크하고 설정되어 있다면 채널을 종료한다. 실제 연결하는 단계에서는 Interrupt Status가 설정되어 있다면 `connect()`를 진행하지 않고, `end` 단계에서 Interrupt가 걸려 있다면 `java.nio.channels.ClosedByInterruptedException`이 발생한다(이후에 Interrupt Status는 clear된다).

- `java.nio.channels.SocketChannel#read()`/`java.nio.channels.SocketChannel#write()`

`java.nio.channels.SocketChannel#read()`나 `java.nio.channels.SocketChannel#write()`하는 경우에도 역시 3단계로 나뉘게 된다.

첫 번째 단계인 `begin` 단계에서는 Interrupt Status가 설정되었는지 체크하고 설정되어 있다면 채널을 종료한다. 그리고 실제 IO를 처리하는 단계에서는 Interrupt Status가 설정되어 있다면 실제 `Read`, `Write`를 진행하지 않고 마지막 `end` 단계에서는 `java.nio.channels.ClosedByInterruptedException`이

발생한다(이후에 Interrupt Status는 clear된다).

java.nio.channels.ClosedByInterruptedException이 발생한 후 이전에 생성했던 채널을 사용해서 Read, Write를 하려고 하면 I/O 작업이 수행되지 않고 java.nio.channels.ClosedChannelException이 발생한다.

- java.net.Socket 사용

java.net.Socket을 사용할 때는 OS, JVM에 따라서 다르다. HP-UX와 Solaris에서는 JVM의 컴파일 옵션에 따라 Socket I/O 작업에 Interrupt를 걸 수 있다.

Socket의 스트림을 통해 Read나 Write를 하려고 할 때 Interrupt Status가 체크되기 때문에 Interrupt에 걸릴 수 있고 java.net.SocketException이 발생한다(이후에 Interrupt Status는 clear된다). 하지만 이때 실제 소켓이 끊어지는 것은 아니고 Interrupt가 걸린 시점의 다음 operation만 중단된다(다음에 Read, Write를 수행하면 제대로 동작한다).

그 외 IBM AIX, Linux 등에서는 Socket I/O 작업을 하는 Thread에 Interrupt를 걸어도 효과가 없다.

- Object#wait() / Thread#sleep() / Thread#join()

Thread가 Object#wait(), Thread#sleep() 또는 Thread#join()하고 있는 상태라면 해당 Thread는 Interrupt에 걸리게 된다. 이 경우 해당 Thread는 java.lang.InterruptedException이 발생한다(이후에 Interrupt Status는 clear된다).



자세한 사항은 해당 클래스의 Javadoc API 문서를 참고한다.

Interrupt Signal 전송

JEUS에서는 콘솔 툴을 통해 Thread에 Interrupt Signal을 보낼 수 있는 명령을 제공한다.

다음은 콘솔 툴을 사용해서 Thread Interrupt Signal을 전송하는 예제이다. JEUS 콘솔 툴의 **interrupt-thread** 명령어에 대한 자세한 정보는 JEUS Reference 안내서의 "interrupt-thread"를 참고한다.

```
[MASTER]jeus_domain.adminServer>interrupt-thread -server server1 45
Sent an interrupt hint signal to the thread [tid=45] on the server server1.
```

3.4. 메모리 모니터링 및 제어

JEUS에서는 메모리를 모니터링하고 특정 사용량을 초과한 경우에 서버를 제어할 수 있는 기능을 제공한다.

3.4.1. 메모리 모니터링

콘솔 툴에서 **memory-info** 명령어를 사용하여 메모리 정보를 확인할 수 있다. memory-info 명령어에 대한 자세한 내용은 JEUS Reference 안내서의 "Server Management 관련 명령어"를 참고한다.

```
[MASTER]jeus_domain.adminServer>memory-info -servers adminServer
```

```
Domain [jeus_domain] Memory Information
```

```
Memory Information
```

```
=====
+-----+-----+-----+
|  Server  | Total Amount of Memory | The Current Amount of Memory |
+-----+-----+-----+
| adminServer | 721420288 | 315030816 |
+-----+-----+-----+
=====
```

3.4.2. 메모리 사용량에 따른 제어

JEUS에서는 자주 사용되는 기능은 아니기 때문에 현재 시스템 프로퍼티를 통해서만 메모리 관련 동작을 제어할 수 있다. 관련된 프로퍼티는 다음과 같다.

프로퍼티	설명
jeus.server.memorymonitor.enabled	MemoryMonitorService를 사용할 것인지를 설정한다. 설정이 활성화되어 있지 않다면 나머지 프로퍼티의 속성은 의미가 없다. (기본값: false)
jeus.server.memorymonitor.ratio	Memory Overflow의 기준치를 설정한다. 만약 0.8을 설정했다면 MAX 메모리를 기준으로 약 80%의 메모리를 사용할 경우 Overflow로 간주한다. (기본값: 0.8)
jeus.server.memorymonitor.interval	메모리 사용량을 측정하는 주기를 설정한다. (기본값: 2000, 단위: ms)
jeus.server.memorymonitor.duration	Memory Overflow를 판단하는 시간을 설정한다. 예를 들어 jeus.server.memorymonitor.ratio가 0.8이고 jeus.server.memorymonitor.duration이 1분이라면 MemoryMonitorService에서는 메모리가 80% 이상 1분 넘게 지속될 경우 Memory Overflow로 간주한다. (기본값: 60000, 단위: ms)
jeus.server.enable.restart.in.memory.shortage	Memory Overflow로 판정된 경우 자동으로 서버를 재기동할지 여부를 설정한다. (기본값: true)



1. Memory Overflow가 발생한 경우 오직 한 번 Heap Dump를 떨어뜨리게 되어 있다. (SERVER_HOME/logs 디렉터리에 생성)
2. IBM JDK의 경우에는 Java를 실행한 위치에 Heap Dump가 떨어지므로 경로를 다르게 설정하려면 -Xdump:heap:file 옵션을 사용해야 한다. IBM Heap Dump에 대한 더 많은 옵션은 관련 홈 페이지를 참조한다.
3. Heap Dump를 남기기 전에 Thread Dump가 로그에 남는다.

4. JNDI Naming Server

본 장은 JEUS JNDI의 기본적인 개념과 용어 그리고 환경설정 방법 및 애플리케이션을 개발 방법에 대해서 설명한다.

4.1. 개요

Java Naming and Directory Interface™ (JNDI)는 Java 애플리케이션이 네트워크에서 객체를 찾고 가져올 수 있도록 하는 표준 API이다. 애플리케이션은 객체의 논리적인 이름을 통해 해당하는 객체를 찾아서 사용할 수 있다. 사용자의 관점에서는 이전의 엔터프라이즈 환경보다 쉽게 객체를 찾아서 사용할 수 있는 환경을 제공해준다.

JEUS JNDI는 JNDI 1.2 API와 호환되며, Sun Microsystems에서 제안한 표준 JNDI API를 지원한다. 그리고 엔터프라이즈 환경에 적합하도록 JNDI Service Provider Interface(이하 SPI)도 제공한다. 즉, JNDI SPI를 구현한 제품은 JEUS JNDI 트리의 객체를 사용할 수 있다.

JEUS JNDI 서비스는 JEUS 시스템 전반적으로 사용되므로 EJB, Servlet/JSP, JMS, JDBC 등을 사용할 때마다 보게 될 것이다.

4.2. 기본 개념과 구조

JEUS JNDI는 객체를 bind하고 Lookup하는 고유의 아키텍처를 가지고 있다. 우선 JEUS JNDI의 구조와 기본 개념에 대해서 설명한다.

4.2.1. 기본 개념

MS를 시작하면 JEUS는 자동적으로 JNDI 서비스를 준비한다. JNDI 서비스는 JNDI Naming Server가 제공하는데, 이것이 실행될 때 내부적으로 JEUS Naming Service Server(이하 JNSServer)가 실행된다. 이 JNSServer와 통신하기 위한 클라이언트 역할을 하는 것이 JEUS Naming Service Client(이하 JNSClient)이다.

JNSClient는 JNSServer와 접속되어 있어서 객체의 bind와 Lookup은 JNSClient에서 먼저하고, 다음으로 JNSServer에서 진행된다. 하나의 JNSServer는 하나 이상의 JNSClient와 접속되어 있으며, 또한 JNSServer들도 다른 JNSServer와 접속하고 있어(특히 클러스터링 환경일 때) 트리와 같은 구조를 이루고 있다. 이러한 Naming Repository구조를 **JNDI 트리**라고 한다.

JNDI 트리는 객체를 bind하거나 Lookup할 때 사용되는데, 모든 객체는 서버들을 통해서 JNDI 트리로 bind되고 Lookup된다. 애플리케이션에서 JNSClient로 객체의 bind를 요청하면 JNDI 트리로 전달되어 bind되며, 이렇게 bind된 객체는 각 JNSClient를 통해 애플리케이션에서 Lookup할 수 있다.

JNDI 트리의 객체를 액세스할 때는 **InitialContext**를 통해서만 가능하다. 그러므로 애플리케이션에서 JNDI를 사용하려면 반드시 InitialContext 객체를 생성해야만 한다. 이 객체는 JNDI 트리에 접근해서 객체를 핸들링할 수 있도록 해준다. 그리고 객체를 bind하거나 Lookup할 뿐만 아니라, 객체의 목록을 가져올 수 있으며 제거할 수 있는 기능을 한다.

바인딩되어 있는 객체는 콘솔 툴을 통해 확인할 수 있다.

4.2.2. 바인딩된 객체 확인

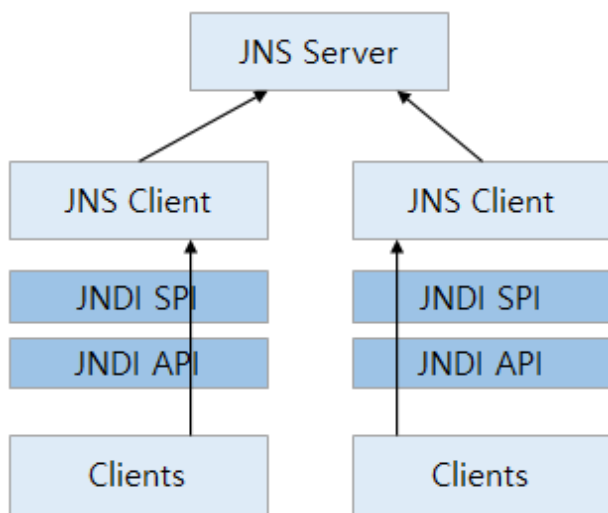
바인딩되어 있는 객체는 콘솔 툴을 통해 확인할 수 있다. 콘솔 툴을 통해 조회하는 방법은 JEUS Reference 안내서의 "jndi-info"를 참고한다.

4.2.3. JNDI Naming Server 아키텍처

본 절에서는 JNDI 트리 구조가 실제로 어떻게 작동되는지 자세히 설명한다.

JNDI 트리는 JNServer와 JNSClient로 구성되어 있다. JNServer는 MS의 JVM에 존재하며, JNSClient는 MS 또는 클라이언트의 JVM에 존재한다. JNServer는 JEUS JNDI 아키텍처의 메인으로 JNDI 트리를 생성하고 관리한다. JNServer는 그 하위에 JNSClient를 뒤서 관리한다.

다음 그림은 JNServer와 JNSClient의 관계를 나타낸다.



JNServer와 JNSClient의 관계

전체 JNDI 트리를 액세스하기 위해서는 JNSClient에서 JNSServer로 요청을 보내야 한다.

JNServer는 다른 MS의 JNServer와 연결되어서 클러스터링을 구성한다. JNSClient는 JNServer와 MS 내에서 상호 작용을 하며, 클라이언트의 액세스 요청을 처리한다. 클라이언트는 JNServer로 바로 접근하는 것이 아니라 JNSClient를 통해서 객체를 bind하고 Lookup한다.

JNServer

JNServer는 JNDI 트리를 관리하며, JNSClient가 JNDI 트리를 액세스할 수 있도록 하는 독립적인 Naming Server이다. JNDI 트리를 확장하기 위해서 여러 개의 JNServer를 연결할 수 있다. JNServer는 다른 MS의 JNServer와 직접적으로 연결할 수 있기 때문이다. JEUS에서는 MS가 시작되면 JNServer는 자동적으로 JNSClient의 접속을 기다린다.

JNSClient

JNSClient의 기본적인 기능은 JNSServer로 접속하여 애플리케이션의 요청을 전송하고 JNSServer의 결과를 다시 돌려주는 것이다. 각 JVM에서는 하나의 서버에 대해 하나의 JNSClient Singleton 인스턴스만 존재한다. 그래서 Lookup을 처리할 때 하나의 JNSClient만 사용하므로 엔터프라이즈 환경에서 EJB와 Servlet을 사용할 때 효과적이다.

다음은 JNSClient의 중요 기능이다.

- JNDI 트리 접속

JNSClient는 JNSServer에 접속해서 MS가 관리하는 JNDI 트리로 접속하는 방법을 제공한다. bind되고 Lookup되는 객체는 전체 JNDI 트리에서 공유되거나, 클라이언트의 설정에 따라서 특정 클라이언트만 액세스할 수도 있다.

- Lookup된 객체의 캐싱

JNSClient는 자주 사용되는 객체를 캐싱해서 클라이언트가 더 빠르게 사용할 수 있도록 한다. JNSClient는 JNSServer와 통신을 하면서 객체를 캐싱한다.

- JNSServer와의 연결 관리

JNSClient는 클라이언트의 요청을 받아서 JNSServer로 전달하고 그 결과를 받아서 리턴한다. 클라이언트가 있는 JVM에 JNSClient가 존재하므로 별다른 I/O 없이 효율적으로 통신할 수 있다.

JNSClient에는 JNSServer의 위치(같은 MS인지 아니면 원격지에 있는지)에 따라서 2가지 종류로 나뉜다.

JNSClient	설명
Server-side	MS의 각 엔진에 있는 EJB Bean, Servlet, JSP 등이 사용한다. Server-side JNSClient를 사용하려면 <code>java.naming.factory.initial</code> 프로퍼티 값을 <code>jeus.jndi.JNSContextFactory</code> 로 설정한다. 이 값은 기본적으로 MS가 기동될 때 설정되므로 별다르게 고려할 필요는 없다.
Client-side	MS에서 동작하지 않는 애플릿, 애플리케이션 클라이언트 등이 사용한다. Client-side JNSClient를 사용하려면 <code>java.naming.factory.initial</code> 프로퍼티값을 <code>jeus.jndi.JNSContextFactory</code> 로 설정한다. Client-side JNSClient는 일정시간 동안 통신이 없을 때 커넥션을 끊고, 다시 필요할 때 커넥션을 생성해서 리소스를 효율적으로 관리한다. 단, 클라이언트 컨테이너에서 동작하는 애플리케이션은 클라이언트 컨테이너가 실행될 때에는 이 값을 시스템 프로퍼티로 설정하기 때문에 애플리케이션에서 별도로 설정할 필요는 없다.

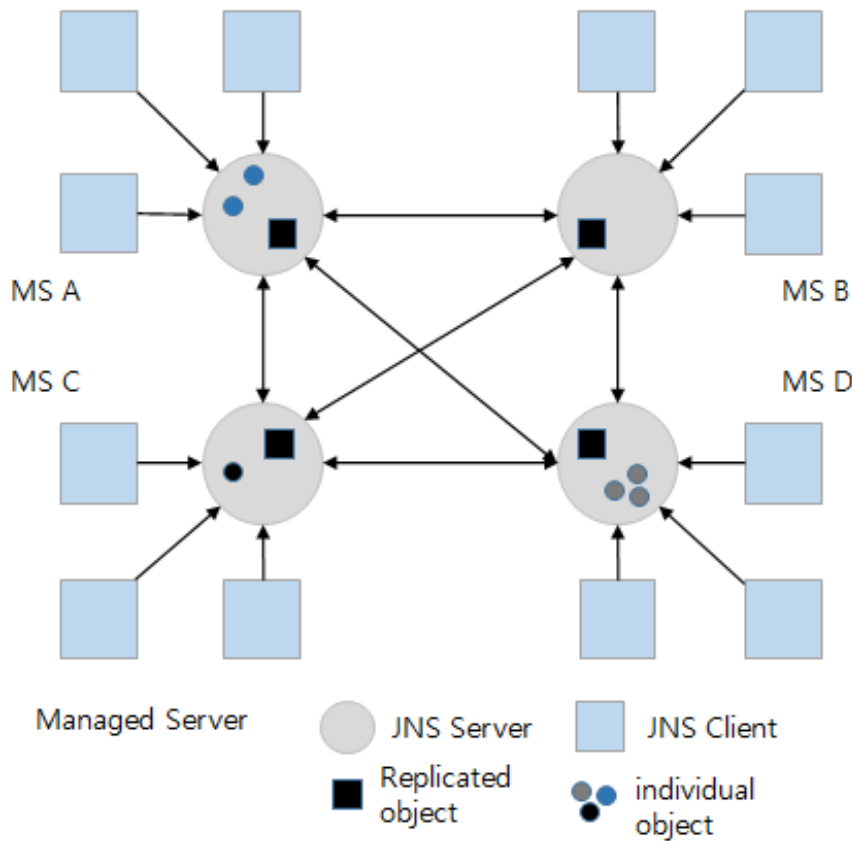
4.2.4. JNDI 클러스터링

JNDI 트리는 JNSServer와 JNSClient 간의 연결로 구성되어 있다. 이 구조는 하나의 컴포넌트(JNSClient)에서 객체에 대한 정보의 변화가 있을 때 다른 컴포넌트(JNSClient)로 전달될 수 있도록 하며, 또한 여러 개의 MS를 지원(클러스터링)할 수 있도록 한다. 즉, JNDI Naming 클러스터링은 각각의 독립적인 JNDI를 하나의 확장된 JNDI 트리로 구성하는 것이라 할 수 있으며 이때도 마찬가지로 각 JNDI 트리에서 발생하는 객체에 대한 정보의 변화가 있을 때에는 나머지 다른 JNDI로 그 내용이 전달된다. 따라서 JNDI Naming 클러스터링 환경에서는 하나의 Naming Server에서 바인딩한 객체를 다른 여러 Naming Server에서 이 객체를 Lookup할 수 있게 된다.

예를 들어 다음 그림처럼 A, B, C, D라는 4개의 MS가 클러스터링을 구성하고 있는 환경에서 MS A에 obj1이라는 객체를 'objName1'이라는 이름으로 바인딩하면, 이는 나머지 3개의 MS B, C, D에도 이 객체가 전달된다. 따라서 처음에 클라이언트가 직접 바인딩을 시도한 MS A가 아닌 다른 3개의 MS에서도 'objName1'으로 Lookup하면

obj1이라는 객체를 얻어올 수 있게 된다.

그러나 바인딩한 모든 객체가 클러스터링에 속한 MS로 복제되는 것은 아니고, 클러스터링 대상으로 바인딩한 객체만 복제된다. 예를 들어 데이터소스같이 각 MS에 특화된 객체는 각 MS의 JNS Server에만 바인딩된다.



클러스터링 환경에서 JEUS JNDI 아키텍처

각 MS는 각각의 JNSServer를 관리할 책임을 가지고 있다. 각 JNSServer는 JEUS 시스템이 기동될 때 시작되어서 다른 MS의 JNSServer와 연결된다. 각 JEUS 엔진은 InitialContext를 가져옴과 동시에 JNSClient가 JNSServer로 연결된다. 클라이언트가 JNDI 트리의 객체를 Lookup할 때 JNSClient로 요청하고 이어서 해당 MS의 JNSServer로 요청이 보내진다. 그리고 이에 대한 객체를 클라이언트가 받게 된다.

클러스터링 환경에서 원격으로 Lookup

별도의 설정을 하지 않은 경우 JNDI Lookup은 자신이 포함된 JEUS JNDI 클러스터링 영역에 대해서 수행된다. 그러나 애플리케이션이 클러스터링에 있지 않은 다른 MS의 JNDI 서버에 있는 내용을 Lookup할 때에는 PROVIDER URL(JEUS Reference 안내서의 "JNDI 시스템 프로퍼티"참고)을 지정해서 Context를 생성하거나 Lookup할 때 JEUS에서 생성한 jh(JEUS Host) 프로토콜을 사용한 이름으로 Lookup해야 한다. 이에 대한 내용은 [JNDI 프로그래밍](#)을 참고한다.

JNSServer Replication

JEUS의 각 JNSServer는 다른 서버와 연결되어서 상호 작용하기를 기다리고 있다. 기존의 클러스터로 새로운 MS가 들어오면, 새로 들어온 MS의 JNSServer는 시작할 때 이미 존재하는 다른 JNSServer로 통보를 보내게 된다. 이때 각 JNSServer는 자신의 데이터를 새로 들어온 JNSServer로 전송하게 되며, 이렇게 해서 새로 들어온 JNSServer에서도 기존에 bind되어 있는 객체를 Lookup할 수 있게 된다.

이런 확장성으로 인해 이상 작동하여 재기동된 JNSServer는 다른 JNSServer로부터 JNDI 트리 정보를 받아 정상 상태로 동작할 수 있다.

4.3. JNDI Naming Server 설정

JNDI Naming Server는 JNSServer와 JNSClient로 구성되어 있다. 이 둘은 서로 다른 설정을 가진다.

JNSServer에서는 JNSClient의 커넥션을 받아들이는 설정과 다른 JNSServer와 접속하기 위한 설정이 필요하고, JNSClient는 JNSServer와 접속하기 위한 것과 JNDI 트리의 반영을 위한 설정이 필요하다.

본 절에서는 JNSServer와 JNSClient를 설정하는 방법에 대해서 설명한다.

4.3.1. JNSServer 설정

공용 Thread Pool 설정

콘솔 툴을 사용해서 서버 전반적으로 공유할 Thread Pool을 설정할 수 있다. Thread Pool에 대한 기본적인 설명은 [Thread Pool 설정](#)을 참고한다.

- 콘솔 툴 사용

콘솔 툴을 통해서 공용 Thread Pool을 설정할 수 있다.

```
[MASTER]domain1.adminServer>modify-system-thread-pool adminServer -service namingserver -r 10
Successfully performed the MODIFY operation for The namingserver thread pool of
the server (adminServer)., but all changes were non-dynamic. They will be applie
d after restarting.
Check the results using "show-system-thread-pool adminServer -service namingserv
er or modify-system-thread-pool adminServer -service namingserver"
```

```
[MASTER]domain1.adminServer>modify-system-thread-pool adminServer -service namingserver
show the current configuration.
The namingserver thread pool of the server (adminServer)
```

```
=====
+-----+-----+
| Reserved Threads for the Service namingserver          | 10 |
+-----+-----+
=====
```

서비스 전용 Thread Pool 설정

콘솔 툴을 사용해서 서비스별 전용 Thread Pool을 설정할 수 있다.

- 콘솔 툴 사용

콘솔 툴을 통해서 서비스 전용 Thread Pool을 설정할 수 있다.

```
[MASTER]domain1.adminServer>modify-service-thread-pool server1 -service namingserver -min 0 -max
20
Successfully performed the MODIFY operation for The namingserver thread pool of
```

the server (server1)., but all changes were non-dynamic. They will be applied after restarting.

Check the results using "show-service-thread-pool server1 -service namingserver or modify-service-thread-pool server1 -service namingserver"

```
[MASTER]domain1.adminServer>modify-service-thread-pool server1 -service namingserver
```

Shows the current configuration.

The namingserver thread pool of the server (server1).

```
=====
+-----+-----+
| Min                | 0      |
| Max                | 20     |
| Keep-Alive Time    | 60000  |
| Queue Size         | 4096   |
| Max Stuck Thread Time | 3600000 |
| Action On Stuck Thread | NONE   |
| Stuck Thread Check Period | 300000 |
+-----+-----+
=====
```

다음은 **Stuck Thread Handling** 관련 설정 항목에 대한 설명이다.

항목	설명
Max Stuck Thread Time	Thread를 Stuck Thread로 판단하는 기준이 되는 값이다. 여기에 설정된 시간 이상 계속 점유된 상태이면 그 Thread를 Stuck Thread로 간주한다. (기본값: 3600000(1시간), 단위: ms)
Action On Stuck Thread	Stuck Thread로 판단된 경우 그 스레드에 대해 특정 액션을 취하기 위한 설정이다. 다음 중 하나의 값을 설정한다. <ul style="list-style-type: none">• None: 아무 액션도 취하지 않는다.• Interrupt: Interrupt 신호를 보낸다.• IgnoreAndReplace: Stuck Thread를 무시하고 새로운 스레드로 교체한다.
Stuck Thread Check Period	Stuck Thread의 상태를 체크하는 주기를 설정한다. 여기에 설정된 주기마다 Thread의 상태를 체크하여 Stuck Thread로 판단할지 여부를 결정한다. (기본값: 300000(5분), 단위: ms)



서비스의 Thread Pool 설정을 수정하는 것은 동적 반영 가능하기 때문에 서버를 재기동하지 않아도 된다. 하지만 공용 Thread Pool을 사용하다가 전용 Thread Pool을 사용하도록 변경하려면 서버를 재기동해야 한다.

4.3.2. JNSClient 설정

JNSClient는 놓이는 위치에 따라서 설정하는 방법이 달라진다.

- Server-side JNSClient 설정

Server-side JNSClient는 JNSClient가 MS와 같이 실행되는 것을 의미하며 JNSSever의 설정을 따른다.

- Client-side JNSClient 설정

Client-side JNSClient는 JVM이 서로 다른 JNSServer를 액세스한다. 그래서 JNSServer와 연결되어서 JNDI 트리의 내용을 반영하는 Thread가 존재한다. JEUS JNDI는 Thread Pool로 Thread를 관리한다. 이 Thread Pool은 JEUS 프로퍼티를 사용해서 설정하며, 기본값을 사용해도 무방하다. Client-side JNSClient의 프로퍼티를 설정하려면 JVM의 시스템 프로퍼티를 사용하거나 InitialContext의 Hash Table을 사용해야 한다.

다음은 Client-side JNSClient의 프로퍼티의 종류이다.

- JNSContext.RESOLUTION
- JNSContext.CONNECT_TIMEOUT
- JNSContext.CONNECTION_DURATION



EJB와 Servlet/JSP 같은 서버 측 객체에서 JNDI를 사용한다면 MS에 의해서 Server-side JNSClient를 사용해서 bind나 Lookup되기 때문에 이런 프로퍼티를 설정할 필요가 없다. 프로퍼티에 대한 자세한 내용은 JEUS Reference 안내서의 "JNDI 시스템 프로퍼티"를 참고한다.

4.4. 클러스터링 환경에서 JNDI

MS 간에 클러스터링 환경이 구성되어 있다면 JNDI를 클러스터링하기 위해서 별도의 작업은 필요없다. 만약 MS가 클러스터링된다면 각 JNSServer도 자동적으로 클러스터링 환경을 구성한다. MS들을 클러스터링하는 것에 대해서는 JEUS Domain 안내서의 "JEUS 클러스터링"을 참고한다.

기본적으로 JNSClient는 로컬 JNSServer로 접속한다. JNSClient는 MS의 JNSServer 주소를 Context.PROVIDER_URL로 제공받는데, 클러스터링 환경인 경우에도 클러스터링에 참여하고 있는 모든 MS의 주소를 다음과 같이 적어 주어야 한다.

JNSClient는 제공된 Context.PROVIDER_URL를 바탕으로 Load Balancing과 Failover를 수행하기 때문이다.

```
Hashtable ht = new Hashtable();  
ht.put(Context.PROVIDER_URL, "host1:9736,host2:9736"); //cluster에 host1, host2가 참여
```

만약 Context.PROVIDER_URL에서 적혔는 MS 중에 특정 MS가 FAILED 상태가 되면 JNSClient는 FAILED 상태가 된 MS로 JNSServer의 JNDI 오퍼레이션할 때 이를 감지하여 다른 JNSServer로 Failover를 한다. FAILED 상태로 판단된 MS는 JNSClient에서 주기적으로 체크하여 RUNNING 상태가 되었는지 확인한다.

-Djeus.jndi.cluster.recheckto 옵션으로 설정 가능하다. (기본값: 5, 단위: 분)

만약 JNSClient가 MASTER의 관리를 받는 MS에서 동작하고 있다면, 다음과 같이 MASTER에서 설정한 클러스터 이름을 적어서 사용할 수 있다.

```
Hashtable ht = new Hashtable();  
ht.put(Context.PROVIDER_URL, "jeus://cluster1"); //cluster1은 MASTER에 설정한 클러스터 이름
```

클러스터 이름을 적어서 사용한 경우에는 JNSClient의 "jeus.jndi.cluster.recheckto" 옵션과 상관없이 MASTER로부터 늘 MS 서버의 최신 상태 정보를 받아서 클러스터링을 할 수 있다. 즉, 클러스터에 속한 MS가 FAILED 상태가 되거나 새로운 MS가 추가된 경우에 JNSClient에서 JNDI 오퍼레이션을 수행하기 전에 최신의 서버 상태로 업데이트되어 동작된다. **따라서 JNSClient가 MASTER의 관리를 받는 MS에서 동작된다면 클러스터 이름을 사용할 것을 권장한다.**

Lookup

클러스터에 있는 클라이언트는 JNDI 트리에 bind되어 있는 어떤 객체든 Lookup할 수 있다. 만약 클라이언트가 JNSClient에 객체를 bind한다면 즉시 JNSServer를 통해서 클러스터링된 모든 MS에서 그것을 공유하게 된다. 또한 데이터 삭제나 변경이 발생하면 즉시 각 MS의 JNSClient에도 반영된다.

JEUS JNDI에서는 객체의 성격에 따라서 어떤 객체는 클러스터 전체에 공유되고 (Lookup을 위한 export name 같은 객체), 어떤 객체는 자신의 MS(데이터소스같은 MS에 의존적인 객체)에서만 보인다.

4.5. JNDI 프로그래밍

본 절에서는 JEUS JNDI를 사용하는 프로그래밍 방법에 대해서 설명한다.

Java 클라이언트는 InitialContext를 사용해서 JNDI 트리를 접근한다. InitialContext에 사용되는 프로퍼티는 JNDI 표준 프로퍼티와 JEUS용 프로퍼티가 있다. 먼저 JNDI 환경을 설정하고 그 다음으로 InitialContext를 사용해서 객체를 Lookup한 다음, 객체의 레퍼런스를 가져온다. 마지막으로 InitialContext를 사용한 다음에는 반드시 close시켜준다.

다음의 과정으로 Java 클라이언트는 JEUS JNDI 서비스를 사용한다.

1. JEUS 환경설정
2. InitialContext를 위한 프로퍼티 설정
3. Context를 사용한 Named Object의 Lookup
4. Named Object를 사용해서 객체의 레퍼런스 취득
5. Context 닫기

각 단계에 대한 자세한 설명은 해당 절을 참고한다.

4.5.1. JEUS 환경설정

JNDI 서비스에서 필요한 클래스를 사용할 수 있도록 JEUS 클라이언트 모듈의 경로(JEUS_CLIENT)를 클래스 패스에 설정한다.

```
-classpath ${JEUS_CLIENT};
```

4.5.2. InitialContext를 위한 프로퍼티 설정

Java 클라이언트가 JEUS JNDI 서비스를 사용하기 전에 우선 InitialContext의 환경 프로퍼티를 설정한다.

설정하는 방법은 다음의 2가지가 있다.

- JVM의 -D 옵션을 사용한다.
- Hash Table을 생성하여 InitialContext의 생성자에게 넘긴다.

위의 2가지 방법 중 Hash Table을 생성하여 InitialContext의 생성자에게 넘기는 방법보다 JVM의 -D 옵션을 사용하는 방법이 우선한다.

JEUS JNDI의 InitialContext를 생성하기 위해서 설정해야 하는 프로퍼티는 다음과 같다.

- Context.INITIAL_CONTEXT_FACTORY (required)
- Context.URL_PKG_PREFIXES
- Context.PROVIDER_URL
- Context.SECURITY_PRINCIPAL
- Context.SECURITY_CREDENTIALS



프로퍼티에 대한 자세한 사항은 JEUS Reference 안내서의 "JNDI 시스템 프로퍼티"를 참고한다.

이 프로퍼티들은 Hash Table에 넣어서 InitialContext를 생성할 때 사용한다. 만약 서버 측 객체 내(EJB나 Servlet/JSP)에서만 InitialContext를 사용한다면 별도의 설정 없이 기본으로 설정된 InitialContext를 사용한다.

클라이언트 프로그램에서는 다음과 같이 설정한다.

```
Context ctx = null;
Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY, "jeus.jndi.JNSContextFactory");
ht.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
ht.put(Context.PROVIDER_URL, "<hostname>");
ht.put(Context.SECURITY_PRINCIPAL, "<username>");
ht.put(Context.SECURITY_CREDENTIALS, "<password>");

try {
    ctx = new InitialContext(ht);
    // use the context
} catch (NamingException ne) {
    // fail to get an InitialContext
} finally {
    try{
        ctx.close();
    } catch (Exception e) {
        // an error occurred
    }
}
```

클러스터링 환경에서 JNDI를 사용할 때는 JNDI 트리를 구성하고 JNSClient의 내부적인 작동을 효과적으로 관리하기 위해서 jeus.jndi.JNSContext의 추가적인 환경 프로퍼티를 사용할 수 있다.

- Hash Table을 사용해서 Context 생성하기

InitialContext의 환경 프로퍼티를 설정할 때 Hash Table(java.util.Hashtable)의 키로 프로퍼티 이름을 넣고, 값으로 프로퍼티의 데이터를 넣은 후에 InitialContext 생성자의 파라미터로 넣어준다.

- Security Context 생성하기

JEUS Security 도메인의 사용자를 실행 Thread에 적용시키기 위해서 몇 가지 보안 관련 환경 프로퍼티를 사용할 수 있다. InitialContext가 생성될 때 Security Context는 다음의 프로퍼티로 구성한다.

- 사용자명 프로퍼티 : java.naming.security.principal
- 패스워드 프로퍼티 : java.naming.security.credentials

인증이 성공하면 인증된 사용자의 정보가 실행 Thread에 설정되며, JEUS Security Manager가 관리하는 리소스를 액세스할 수 있다. 만약 인증에 실패하면 Thread는 guest 사용자로 인식된다.

Security Context가 설정되어도 InitialContext가 생성되기 전에 JEUS Security API를 사용해서 로그인한 경우 Security Context는 무시된다. 즉, 이미 로그인된 subject를 사용해서 JNDI 통신을 수행한다.



1. 사용자 정보 설정에 관한 더 자세한 내용과 JEUS Security Service에 대한 내용은 "JEUS Security 안내서"를 참고한다.
2. InitialContext 설정 관련 프로퍼티와 환경 프로퍼티에 대한 자세한 내용은 JEUS Reference 안내서의 "JNDI 시스템 프로퍼티"를 참고한다.

4.5.3. Context를 사용한 Named Object의 Lookup

JNDI 트리에 bind된 객체는 JNDI Context의 Lookup() 메소드를 사용해서 찾는다. 다음은 Lookup하려는 객체의 모듈 이름이 countermod이고, EJB 이름이 Count인 경우이다.

```
try {
    Context ctx = new InitialContext();
    Count count = (Count)ctx.lookup("java:global/countermod/Count");
    // Count count = (Count)ctx.lookup("java:global/countermod/Count!my.ejb3.tx.Count");
    // Count count = (Count)ctx.lookup("Count");
    // successfully got the object
} catch (NameNotFoundException ex) {
    // no such binding exists
} catch (NamingException e) {
    // an error occurred
}
```

비즈니스 인터페이스가 여러 개인 경우 "!인터페이스이름"을 뒤에 더 붙여서 Lookup하면 된다.

```
Count count = (Count)ctx.lookup("java:global/countermod/Count!my.ejb3.tx.Count");
```

export name을 준 경우는 export name으로 Lookup할 수 있다.


```
Count count = (Count)ctx.lookup("Count");
```

다음은 EJB 2.x 객체인 경우를 Lookup하는 예제이다.

```
try {
    Context ctx = new InitialContext();
    CountHome countHome = (CountHome)ctx.lookup("Count");
    // successfully got the object
} catch (NameNotFoundException ex) {
    // no such binding exists
} catch (NamingException e) {
    // an error occurred
}
```

4.5.4. Named Object 사용

EJB 3.x인 경우에는 Lookup한 객체를 다음과 같이 바로 사용한다.

```
count.service();
```

EJB 2.x인 경우 Lookup한 EJB 홈 객체의 create() 메소드를 사용해서 EJB 리모트 객체의 Reference를 가져온다. 다음과 같이 메소드를 실행해서 사용하려는 객체의 Home Reference를 가져오고, 그 Reference의 메소드를 실행한다.

```
Count count = countHome.create();
count.service();
```

4.5.5. Context 닫기

다음과 같이 Context를 사용한 다음에는 close() 메소드를 실행해서 Context를 닫아준다.

```
try {
    cx.close();
} catch (Exception e) {
    // an error occurred
}
```

4.5.6. 클러스터링 Context 생성

JEUS 클러스터링 환경에서 사용할 수 있는 Context를 생성하기 위해서는 다음과 같이 Context.PROVIDER_URL에 여러 개의 호스트를 설정한다.

```
Hashtable ht = new Hashtable();
ht.put(Context.PROVIDER_URL, "host1:9736,host2:9736");
```

도메인에 설정한 클러스터 이름을 알고 있다면 다음과 같이 설정하여 사용할 수도 있다.

```
Context.PROVIDER_URL에 'jeus://' + <클러스터 이름>
```

단, 클러스터에 속한 MS에서만 사용이 가능하고, 독립된 클라이언트(Standalone Client)에서는 클러스터 정보를 알 수 없어 사용할 수 없다.

```
Hashtable ht = new Hashtable();
ht.put(Context.PROVIDER_URL, "jeus://cluster1");
```

클러스터링된 Context에서 Lookup하는 경우 어떤 MS에서 객체를 가져올 것인지에 대한 정책을 정할 수 있다.

jeus.jndi.clusterlink.selection-policy 프로퍼티를 제공하며 다음과 같은 3가지 값을 설정할 수 있다. 또한 이 값들은 System property를 주어 설정할 수도 있으며, 이 경우에는 Hashtable을 통한 설정이 우선된다.

구분	설명
locallinkPreference	로컬 MS에 있는 객체를 사용한다.
roundrobin	처음 요청에서는 random하게 선택한 MS의 객체를 사용하고, 그 후 요청부터는 하나씩 증가하면서 서버를 선택한다.
random	클러스터링된 MS들 중에서 random하게 하나를 선택해서 사용한다.

다음은 설정 예제이다.

```
Hashtable ht = new Hashtable();
ht.put(Context.PROVIDER_URL, "host1:9736,host2:9736");
ht.put("jeus.jndi.clusterlink.selection-policy", "random");
```

4.5.7. 원격으로 Lookup 실행

애플리케이션이 클러스터링에 있지 않은 다른 MS의 JNDI 서버에 있는 내용을 Lookup할 때에는 PROVIDER URL을 지정해서 Context를 생성하거나 Lookup할 때 JEUS에서 생성한 jh(JEUS Host) 프로토콜을 사용한 이름으로 Lookup해야 한다.

다음은 JEUS의 클러스터링 환경에서 원격으로 Lookup을 실행할 수 있도록 하는 특별한 Lookup 구문이다. 이 구문은 자신이 속한 JEUS JNDI 클러스터링의 영역을 벗어나 원격지의 JEUS JNDI 클러스터링에 있는 객체를 Lookup할 때 사용한다.

```
jh:<remote JEUS host name>:<remote JEUS base port>/<export name>
```

```
("jh" = JEUS host)
```

다음은 Lookup 구분의 사용법으로 JNDI Context 문자열 내에 다음과 같은 구문을 입력한다.

```
try {  
    Context ctx = new InitialContext();  
    CountHome countHome = (CountHome)ctx.lookup("jh:dev:9736/Count");  
    // successfully got the object  
} catch (NameNotFoundException ex) {  
    // no such binding exists  
} catch (NamingException e) {  
    // an error occurred  
}
```

5. External Resource

본 장에서는 JEUS와 연동하여 하나의 시스템을 구축할 수 있는 다양한 외부 리소스(External Resource)에 대한 소개와 설정 방법에 대해 설명한다. 외부 리소스에 대한 설정이나 사용법에 관한 자세한 내용들은 각 외부 리소스의 안내서를 참고한다.

5.1. 리소스 종류

External Resource는 애플리케이션이 JEUS를 통해서 접근할 수 있는 JEUS 외부에 존재하는 리소스로 DB가 대표적인 예이다. 이러한 리소스들은 JEUS에 관련 설정을 추가함으로써 연결이 가능하다.



만약 External Resource에서 JCA 표준 호환의 리소스 어댑터를 제공하는 경우에는 리소스 어댑터를 deploy해서 사용하길 권장한다.

다음은 JEUS에 설정할 수 있는 리소스이다.

- **데이터소스**

데이터소스(Datasource)는 JDBC 호환 DB를 의미한다. 데이터소스는 클라이언트에서 직접적으로 접근할 수 있는데 이런 경우에는 특별히 JEUS에 설정하지 않아도 된다. 그러나 데이터소스를 설정하면 JNDI를 이용해서 JDBC Connection Pool을 사용할 수 있으므로 애플리케이션이 더욱 편리하게 DB에 접근할 수 있다. 데이터소스의 설정은 [DB Connection Pool](#)과 [JDBC](#)를 참고한다.

- **Mail Resource**

메일 리소스(Mail Resource)는 SMTP와 같은 메일 프로토콜을 이용하여 클라이언트 애플리케이션으로부터 이메일(e-mail)을 전송할 때 사용한다. JEUS에서는 JNDI Export name에 이메일 호스트의 정보를 bind하고, 클라이언트에서 간접적으로 접근하여 호스트를 사용하도록 한다. JNDI를 Lookup하면 jakarta.mail.Session 타입의 메일 소스를 가져온다.

- **URL Resource**

URL 리소스는 애플리케이션에서 외부 URL 객체를 JNDI를 통해 접근할 수 있도록 한다. URL이 변경되는 경우 해당 URL 설정을 수정함으로써 애플리케이션의 소스 수정 없이 그대로 사용할 수 있도록 한다. JNDI를 Lookup하면 java.net.URL 타입의 URL 객체를 가져온다.

- **Message Bridge**

Message Bridge는 여러 JMS 벤더의 Destination 사이에 연결을 위해 사용된다. Jakarta Messaging 2.0 스펙을 충족하는 MQ라면 무엇이든 설정 가능하며 자세한 내용은 JEUS MQ 안내서의 "JEUS MQ Message Bridge"를 참고한다.

- **Custom Resource**

Custom Resource는 Java Bean 형태의 리소스를 JNDI repository에 bind시킬 수 있다. lookup할 때 JNDI ObjectFactory를 통해 등록된 서비스를 사용할 수 있도록 하는 일반적인 리소스이다.

- **External Source - IBM MQ, Sonic MQ 등**

JEUS와 연결할 수 있는 비정규화된 소스들을 말한다. 일반적으로 JEUS에 설정할 수 있는 것으로는 IBM MQ, Sonic MQ 등의 Jakarta Messaging 제품들과 Tmax TP Monitor 등이 있다. 이 리소스들은 JEUS에 설정하지 않아도 Java API를 통해서 직접적으로 액세스할 수도 있다. 그러나 트랜잭션 매니저에서 이 소스들을 관리하려면 설정을 해야 한다([트랜잭션 매니저](#) 참조).

- **External Resource**

JEUS위에서 동작하는 리소스들을 말한다. 주로 JEUS와 연동되는 WebT나 jTmax, 또는 InifiniteCache에서 사용한다. JEUS에서 이런 External Resource를 사용하려면 jeus.external.ResourceBootstrapper를 구현한 class를 등록해야 한다.

- **Concurrency Utilities Resource**

Jakarta Concurrency와 관련한 리소스를 정의한다. 이를 통해 애플리케이션 서버 상에서 관리 가능한 작업들을 정의하고, 작업이 실행될 때 컨텍스트를 유지하며 동작할 수 있다. 자세한 설명은 "JEUS Jakarta Concurrency 안내서"를 참고한다.

5.2. 리소스 설정

본 절에서는 각각의 리소스를 설정하는 방법에 대해 설명한다.

리소스는 도메인 범위에 설정하고, 서버가 부팅할 때 이 정보를 읽어 자신의 서버에 리소스를 등록한다.

5.2.1. 데이터소스 설정

데이터소스는 DB 관련 설정을 다루게 된다. 이 내용은 [DB Connection Pool](#)과 [JDBC](#)에 기술되어 있으므로 해당 절을 참고한다.

5.2.2. 메일 소스 설정

JEUS에 클라이언트 애플리케이션에서 메일을 보낼 때 사용될 SMTP 호스트를 설정할 수 있다. 이메일 호스트에 대한 정보는 Jakarta Mail 사양을 참고한다.

domain.xml을 다음과 같이 편집하여 메일 소스를 설정할 수 있다.

```
<servers>
  <server>
    <name>server1</name>
    ...
    <resources>
      <mail-source>
        <mail-entry>
          <export-name>mailSource</export-name>
          <mail-property>
            <name>mail.smtp.auth</name>
            <value>true</value>
```

```

        </mail-property>
      </mail-entry>
    </mail-source>
  </resource>
</server>
</servers>

```



'Main Property'에 설정하는 속성명은 Jakarta Mail 사양을 따르므로 해당 사양을 참고해서 지정한다.

5.2.3. URL 소스 설정

다음은 JNDI 이름은 PRIMARY_URL과 SECONDARY_URL이고 각각에 bind되는 URL이다.

```
http://www.foo.com
```

```
http://www.bar.com
```

URL 소스 설정을 위해 domainl.xml을 다음과 같이 편집한다.

```

<resources>
  <url-source>
    <url-entry>
      <export-name>PRIMARY_URL</export-name>
      <url>http://www.foo.com</url>
    </url-entry>
    <url-entry>
      <export-name>SECONDARY_URL</export-name>
      <url>http://www.bar.com</url>
    </url-entry>
  </url-source>
</resources>

```

5.2.4. Custom Resource 설정

Custom Resource는 Java Bean 형태의 리소스를 JNDI ObjectFactory를 통해 Lookup하여 사용할 수 있도록 하는 일반적인 리소스이다. 본 절에서는 Custom Resource 구현 방법과 등록 방법에 대해서 설명한다.

다음은 Java Bean 형태의 리소스 클래스와 리소스 인스턴스를 생성할 ObjectFactory 클래스에 대한 예제이다. 이 클래스는 SERVER_HOME/lib/application 또는 DOMAIN_HOME/lib/application에 있어야 한다.

Custom Resource를 생성할 팩토리 클래스의 예제

```
package dog;
```

```

import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.Name;
import javax.naming.spi.ObjectFactory;

public class DogFactory implements ObjectFactory {
    public DogFactory() {}

    public Object getObjectInstance(Object obj, Name name, Context nameCtx, Hashtable<?, ?>
environment)
        throws Exception {
        Dog dog = Dog.getInstance();
        System.out.println("Creating a dog whose name is " + dog.getName() + ",
and age is " + dog.getAge());
        return dog;
    }
}

```

Custom Resource 예제 클래스

```

package dog;

public class Dog implements java.io.Serializable {
    public static final String DOG_NAME = "wangwang";
    public static final int DOG_AGE = 1;

    private static Dog instance = new Dog();

    private int age = DOG_AGE;
    private String name = DOG_NAME;

    public Dog() {}

    public static Dog getInstance() {
        return instance;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean equals(Object anObject) {
        if (this == anObject) {
            return true;
        }
        if (anObject instanceof Dog) {

```

```

        Dog anotherDog = (Dog) anObject;
        return (this.age == anotherDog.age && this.name.equals(anotherDog.name));
    }
    return false;
}
}

```



Custom Resource를 동적으로 추가하기 위해서는 이미 해당 클래스들이 서버의 클래스 로더에 클래스 패스로 잡혀있어야 가능하다. 만약 서버의 클래스 로더에서 이 클래스를 로딩할 수 없으면 동적 추가 명령은 pending 처리된다. 이럴 경우 Custom Resource 클래스를 SERVER_HOME/lib/application 또는 DOMAIN_HOME/lib/application에 추가하고 서버를 재기동해야 한다.

Custom Resource를 삭제하는 동작은 Graceful하게 수행되지 않는다. 즉, 진행 중인 요청이 있더라도 이를 완료하지 않기 때문에 사용자 애플리케이션에서는 에러가 발생할 수 있음을 유의한다.

콘솔 툴 사용

Custom Resource는 콘솔 툴을 통해서 서버에 등록된 리소스에 대한 조회 명령이 가능하다. 또한 새로운 Custom Resource를 동적으로 추가, 삭제하는 것이 가능하다.

```

[MASTER]domain1.adminServer>add-custom-resource custom/dog -resource dog.Dog -factory dog.DogFactory
Successfully performed the ADD operation for A custom resource.
Check the results using "list-custom-resources or add-custom-resource"

```

```

[MASTER]domain1.adminServer>list-custom-resources
List of Custom Resources

```

```

=====
+-----+-----+-----+-----+
| Export Name | Resource Class | Factory Class | Properties |
+-----+-----+-----+-----+
| custom/dog | dog.Dog        | dog.DogFactory | [test=1,   |
|            |                |                | test1=2]   |
+-----+-----+-----+-----+
=====

```

```

[MASTER]domain1.adminServer>add-custom-resource-to-servers custom/dog -servers server1
Successfully performed the ADD operation for A custom resource.
Check the results using "list-custom-resources"

```

```

[MASTER]domain1.adminServer>remove-custom-resource custom/dog
Successfully performed the REMOVE operation for A custom resource.
Check the results using "list-custom-resources or remove-custom-resource"

```

```

[MASTER]domain1.adminServer>list-custom-resources
List of Custom Resources

```

```

=====
+-----+-----+-----+-----+
| Export Name | Resource Class | Factory Class | Properties |
+-----+-----+-----+-----+

```


(No data available)

=====

5.2.5. External Source 설정

외부 소스는 크게 JMS 소스와 Connector로 나뉜다.

JMS Source

JMS 소스를 추가하기 위해서는 다음과 같이 domain.xml의 편집이 필요하다.

```
<resources>
  <external-source>
    <jms-source>
      <vendor>ibmmq</vendor>
      <factory-class-name>FirstClassName</factory-class-name>
      <resource-type>QCF</resource-type>
      <export-name>exportName</export-name>
      <queue>MQ</queue>
      <queueManager>MQManager</queueManager>
    </jms-source>
  </external-source>
</resources>
```

다음은 JMS 소스 설정 항목에 대한 설명이다.

항목	설명
Vendor	JMS 벤더를 설정한다. 다음의 값 중 하나를 설정한다. <ul style="list-style-type: none">◦ ibmmq : IBM사의 제품이다.◦ sonicmq : Sonic MQ이다.◦ others : 그 외의 제품들이다.
Factory Class Name	해당 JMS 리소스의 Factory Class의 이름을 지정한다.

항목	설명
Resource Type	<p>해당하는 JMS의 타입을 결정한다.</p> <p>다음의 8가지 값 중에 하나를 설정한다.</p> <ul style="list-style-type: none"> ◦ QCF ◦ TCF ◦ Q ◦ T ◦ XAQCF ◦ XATCF ◦ LOCALXAQCF ◦ LOCALXATCF
Export Name	JNDI에 바인딩될 이름을 지정한다. 사용자는 이 이름을 이용하여 JMS의 ConnectionFactory, Destination 등을 이용할 수 있다.
Queue	'Resource Type'이 'Q'일 때만 사용한다.
Queue Manager	'Resource Type'이 'Q'일 때만 사용한다.
Topic	'Resource Type'이 'T'일 때만 사용한다.
Property	JMS 리소스에 필요한 속성들을 기록한다. 이 설정은 name, type, value로 구성된다.



각 태그에 대한 자세한 내용은 자세한 것은 IBM MQ나 Sonic MQ의 매뉴얼을 참조한다.

Connector 추가

domain.xml을 다음과 같이 편집하여 Connector를 추가할 수 있다.

```
<resources>
  <external-source>
    <connector>
      <resource-adapter-module-id>connectorModuleId</resource-adapter-module-id>
      <worker-pool>
        <min>0</min>
        <max>5</max>
        <keep-alive-time>60000</keep-alive-time>
        <queue-size>4096</queue-size>
        <shutdown-timeout>-1</shutdown-timeout>
      </worker-pool>
    </connector>
  </external-source>
</resources>
```

Connector에 대한 자세한 설명은 "JEUS Jakarta Connectors 안내서"를 참고한다.

5.2.6. External Resource 설정

JEUS에서는 Tmax나 InfiniCache와 같이 타 제품을 JEUS에서 연동하기 위해서 External Resource를 설정할 수 있다. Tmax와의 연동을 위해서는 JEUS에서 Tmax로 연결해서 Tmax의 트랜잭션 서비스를 사용하는 아웃바운드(Outbound)인 WebT와 Tmax에서 JEUS로 온 서비스 요청을 받아주는 인바운드(Inbound)인 JTmax를 설정할 수 있다. 자세한 내용은 관련된 Tmax 매뉴얼의 "JTmax Server Guide"를 참고한다.

본 절에서는 External Resource 구현 방법과 등록 방법에 대해서 설명한다.

다음은 jeus.external.ResourceBootstrapper 인터페이스이다. 이 인터페이스를 구현한 클래스는 SERVER_HOME/lib/application 또는 DOMAIN_HOME/lib/application에 있어야 한다.

jeus.external.ResourceBootstrapper

```
package jeus.external;

import javax.naming.Context;
import java.util.Map;

/**
 * 외부 resource를 JEUS에서 사용할 수 있도록 하는 bootstrapper이다.
 */
public interface ResourceBootstrapper {
    /**
     *
     * @param propertyMap resource의 설정을 가진 Map
     */
    void setProperties(Map propertyMap) throws InvalidPropertyException;

    /**
     * resource를 bind한다.
     * @param context
     */
    void initResources(Context context);

    /**
     * 사용할 수 있는 property의 정보를 return한다.
     * @return
     */
    ResourcePropertyInfo[] getPropertyInfo();

    /**
     *
     * @param propertyMap 변경되는 property가 있는 Map
     */
    void modifyProperties(Map propertyMap) throws InvalidPropertyException;

    /**
     * resource를 다시 bind할때 호출된다.
     * @param context
     */
    void reconfigResources(Context context);
}
```

```

* resource를 제거할때 사용한다.
* @param context
*/
void destroyResources(Context context);
}

```



External Resource를 동적으로 추가하기 위해서는 이미 해당 클래스들이 서버의 클래스 로더에 클래스 패스로 잡혀있어야 가능하다. 만약 서버의 클래스 로더에서 이 클래스를 로딩할 수 없으면 동적 추가 명령은 pending 처리된다. 이럴 경우 External Resource 클래스를 SERVER_HOME/lib/application 또는 DOMAIN_HOME/lib/application에 추가하고 서버를 재기동해야 한다.

External Resource를 삭제하는 동작은 Graceful하게 수행되지 않는다. 즉, 진행 중인 요청이 있더라도 이를 완료하지 않기 때문에 사용자 애플리케이션에서는 에러가 발생할 수 있음을 유의한다.

콘솔 툴 사용

External Resource는 콘솔 툴을 통해서 서버에 등록된 리소스에 대한 조회 명령이 가능하다. 또한 새로운 External Resource를 동적으로 추가, 삭제하는 것이 가능하다.

```

[MASTER]domain1.adminServer>add-external-resource test/ext -resource
test.ext.TestResourceBootstrapper
Successfully performed the ADD operation for A external resource.
Check the results using "list-external-resources or add-external-resource"

[MASTER]domain1.adminServer>list-external-resources
List of External Resources
=====
+-----+-----+-----+-----+
| Name      | Resource Class                               | Properties |
+-----+-----+-----+-----+
| test/ext  | test.ext.TestResourceBootstrapper           | []         |
+-----+-----+-----+-----+
=====

[MASTER]domain1.adminServer>add-external-resource-to-servers test/ext -servers server1
Successfully performed the ADD operation for A external resource.
Check the results using "list-external-resources"

[MASTER]domain1.adminServer>remove-external-resource test/ext
Successfully performed the REMOVE operation for A external resource.
Check the results using "list-external-resources or remove-external-resource"

[MASTER]domain1.adminServer>list-external-resources
List of External Resources
=====
+-----+-----+-----+-----+
| Name      | Resource Class                               | Properties |
+-----+-----+-----+-----+
(No data available)

```

=====

6. DB Connection Pool과 JDBC

본 장에서는 JEUS에서 제공하는 JDBC Connection Pooling의 기본 메커니즘과 JDBC Connection Pool의 사용 방법 그리고 JEUS 도메인 구조에서의 데이터소스 관리 방법 등에 대해 설명한다.

6.1. 개요

웹 애플리케이션은 정보 저장에 필요할 때 주로 DB를 이용한다. 또한 JEUS와 같은 웹 애플리케이션 서버(web application server, 이하 WAS)는 애플리케이션에 Connection Pooling 등 DB 의존적인 서비스를 제공하기 위하여 DB와 통신할 필요가 있다. 이러한 필요를 보다 체계적이고 효율적으로 충족시키기 위해 DB 클라이언트들과 DB 간 인터페이스를 정의한 것이 바로 Java Database Connectivity(JDBC) 표준이다.

JDBC 표준은 애플리케이션의 DB Connection 사용 방법 및 SQL 작업 수행 방법에 대해 기술하고 관련 API를 제공한다. JDBC 표준에 관한 자세한 내용은 [Oracle JDBC 사이트](#) 등을 참고한다.

JEUS는 JDBC 4.0을 기반으로 애플리케이션에 Connection Pooling 및 기타 부가 서비스를 제공한다.

6.2. 데이터소스와 JDBC Connection Pooling

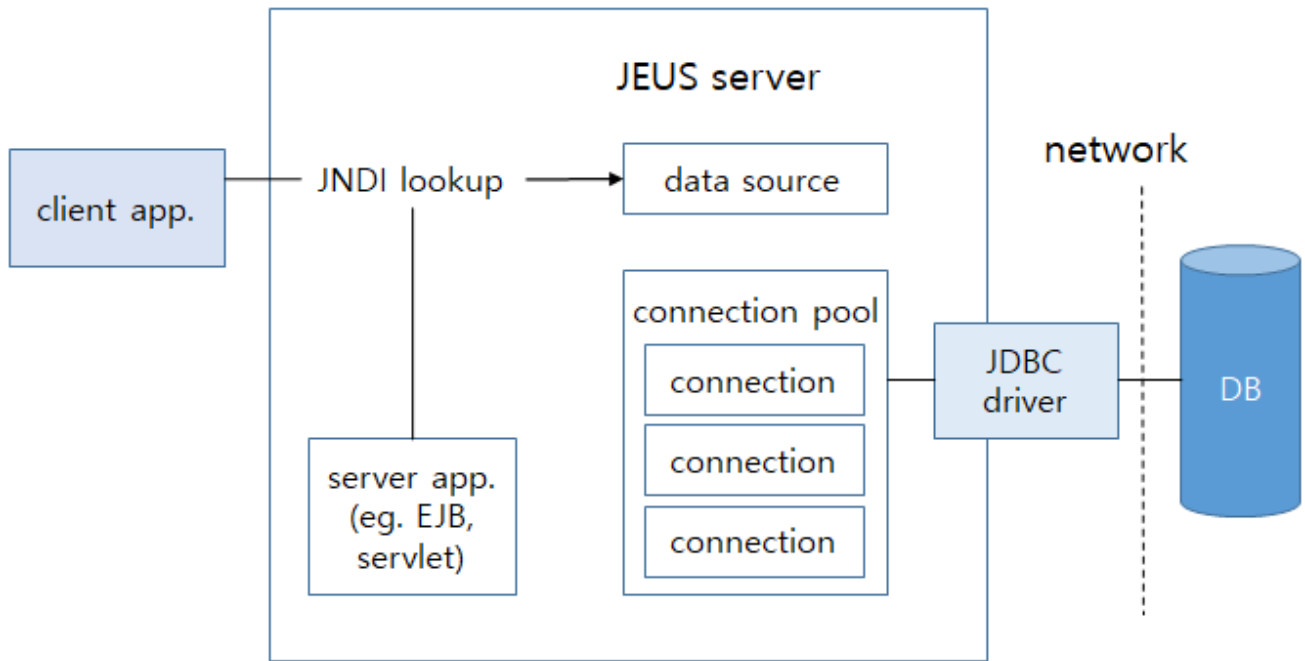
데이터소스는 애플리케이션에 JDBC Connection을 제공하는 Factory로 추상화된 객체로서 내부적으로 JDBC Connection Pool을 구성하여 Connection Pooling 서비스를 제공한다.

JDBC Connection Pooling은 DB로부터 일정 수의 Connection을 얻어 보관하면서 애플리케이션이 필요로 할 때 제공하고, 사용 이후의 Connection을 회수하여 재활용하는 JDBC Connection 관리 서비스다.

애플리케이션은 DB에 직접 Connection을 요청하는 것이 아니라 JNDI Lookup을 통해 DB와 연관된 데이터소스를 얻고 그것에 Connection을 요청한다. Connection 요청을 받은 데이터소스는 자신의 Connection Pool에서 Connection을 꺼내 애플리케이션에 전달함으로써 Connection 요청을 처리한다.

애플리케이션이 Connection 사용을 마치면 JEUS는 Connection을 수거하여 필요한 Connection 정리 작업을 수행하고 이후 재활용할 수 있도록 다시 Connection Pool에 넣어 관리한다.

다음은 JEUS 서버에서의 JDBC Connection Pooling 메커니즘을 간략히 표현한 그림이다.



JEUS의 Connection Pooling



JEUS는 boot 과정에서 Connection Pool을 생성하고 초기화하는 것이 아니라 애플리케이션으로부터 데이터소스에 대한 Connection 요청이 최초 발생할 때 Connection Pool을 생성하고 초기화한다.

6.2.1. JDBC 드라이버

JDBC 드라이버는 DB와의 통신에 필요한 API들의 구현체 집합으로 JEUS는 JDBC 인증을 받은 드라이버들과 연동하여 Connection Pooling 및 기타 서비스들을 제공할 수 있다. 인증된 드라이버는 각 DB 벤더들이 홈페이지를 통해 제공하고 있다.

JEUS의 JDBC 환경 구성은 어떤 벤더의 DB를 사용하느냐에 따라 다를 수 있다. 이는 각 드라이버마다 요구하는 속성이 다르기 때문이며 따라서 조정이 필요한 세부 속성은 각 벤더의 JDBC 드라이버 매뉴얼을 확인해야 한다.

6.2.2. JDBC Connection Pool

JDBC Connection Pool은 DB Connection을 보다 효율적으로 사용하고 관리하기 위한 하나의 프레임워크로 실제 Connection Pooling 서비스를 제공하는 주체다.

Connection Pool 사용의 이점은 다음의 2가지로 요약할 수 있다.

- **보다 높은 성능**

시스템의 성능을 향상시킬 수 있다. DB Connection 생성 및 제거는 본래 시스템에 부담을 주는 작업이므로 최초 Connection Pool에 Connection을 생성하여 채우고 이를 재활용하면 Connection Pooling 없이 Connection을 빈번히 생성하고 제거하는 경우 비롯되는 오버헤드를 크게 줄일 수 있다.

- **연결 관리**

동시 Connection들의 수를 제어할 수 있다. 동시 Connection들의 수를 설정한 값 이하로 제한하여 관리함으로써 DB에 무리한 부하를 주는 문제로부터 자유로울 수 있다.

6.2.3. 데이터소스

데이터소스는 애플리케이션과 Connection Pool 사이의 인터페이스로 애플리케이션은 데이터소스 객체를 DB Connection의 Factory로 바라보게 된다. 데이터소스를 통한 Connection 사용 방식은 `java.sql.DriverManager`를 통한 Connection 사용 방식보다 많은 장점을 제공한다. 본 절에서 설명하는 데이터소스의 특성 및 기능을 잘 고려하여 그에 맞는 적절한 데이터소스를 사용하면 서비스의 성능 향상을 꾀할 수 있다.

데이터소스 타입

데이터소스는 다음의 3가지의 타입으로 분류된다.

- 기본 데이터소스

`javax.sql.DataSource` 타입을 의미한다. 이 타입은 Connection Pooling을 위해 사용될 수는 없는 기본 데이터소스 타입이다.

- Connection Pool 데이터소스

`javax.sql.ConnectionPoolDataSource` 타입을 의미한다. JEUS는 이 타입에 대해 Connection Pool을 제공한다.

주로 다음과 같은 상황에서 사용할 수 있다.

- XA를 사용할 필요없이 간단하게 DB에 접근하기 위한 애플리케이션을 작성하는 경우
- Auto Commit을 false로 설정하고 직접 로컬 트랜잭션을 컨트롤하는 경우

```
import java.sql.Connection;

Connection conn = datasource.getConnection();
conn.setAutoCommit(false);
...DB 접근 코드... // JDBC 드라이버 내부적으로 로컬 트랜잭션 진행
conn.commit();
```

- XA 데이터소스

`javax.sql.XADataSource` 타입을 의미한다. JEUS는 이 타입에 대해 Connection Pool을 제공한다. 이 타입의 데이터소스는 Connection Pooling과 더불어 글로벌 트랜잭션(이하 XA) 연동을 지원한다.

주로 다음과 같은 상황에서 사용할 수 있다.

- Servlet/EJB와 같은 Jakarta EE 컴포넌트 로직이 2개 이상의 DB로 접근해야 하는 경우
- Jakarta EE 컴포넌트에서 하나의 DB로 접근하더라도 관련 로직들이 일련의 동작으로 묶여야 하는 경우

Connection Pool 데이터소스에 대한 XA 에뮬레이션

XA 처리 성능을 높이기 위해서 로컬 트랜잭션 최적화 기능을 사용할 수 있다. 이는 Connection Pool 데이터소스를 통해서 얻은 Connection을 XA에 참여하도록 에뮬레이션 해주는 것으로 JEUS 6에서는 로컬 트랜잭션 데이터소스라는 타입을 별도로 정의해 제공하던 기능이다. JEUS 9에서는 로컬 트랜잭션 데이터소스 타입을 별도로 정의하지 않고 Connection Pool 데이터소스에 XA 에뮬레이션 플래그 설정을 두어 Connection Pool 데이터소스의 XA 에뮬레이션 지원 여부를 결정하게 된다.

이 기능은 주로 다음과 같은 상황에서 사용할 수 있다.

- DB가 XA를 지원하지 않거나 JDBC 드라이버가 javax.sql.XADataSource 구현체를 제공하지 않는 경우
- 애플리케이션에서 XA가 필요하지만 성능 문제로 XA 데이터소스를 사용하고 싶지 않은 경우(즉, 로컬 트랜잭션 최적화가 필요한 경우)

최근에는 XA를 지원하지 않는 DB 또는 JDBC 드라이버는 거의 없다고 볼 수 있기 때문에 주로 로컬 트랜잭션 최적화를 위해서 사용하게 될 것이다. 그러나 반드시 염두에 두어야 할 제약 사항이 있는데, 하나의 XA에는 최대 하나의 로컬 XA 데이터소스만 참여할 수 있다는 것이다. 따라서 로컬 XA 데이터소스는 되도록 다음과 같은 상황에서 사용한다.

- Servlet/EJB와 같은 Jakarta EE 컴포넌트 로직이 하나의 DB만 사용하기 때문에 굳이 XA 데이터소스를 사용할 필요가 없을 때
- Jakarta EE 컴포넌트에서 여러 개의 DB를 사용하더라도 그 중 하나를 로컬 트랜잭션으로 처리해서 XA 성능을 높이고 싶을 때

여러 개의 DB를 사용하는 상황에서 주의할 점은 XA 에뮬레이션 기능을 사용하는 Connection Pool 데이터소스는 실제로 2 Phase Commit(이하 2PC)을 지원하는 것이 아니므로 트랜잭션 복구가 제대로 되지 않을 수 있다는 것이다.



Connection Pool 데이터소스의 XA 에뮬레이션은 Auto Commit을 끄고 사용하는 로컬 트랜잭션으로 다루어지며 이는 DB 관점에서 JEUS 트랜잭션 매니저가 관리하는 XA 트랜잭션과는 서로 다른 트랜잭션이다. 대신 JEUS가 그 로컬 트랜잭션이 XA 트랜잭션에 참여할 수 있도록 에뮬레이션하므로 애플리케이션 관점에서는 하나의 트랜잭션으로 보게 된다.

6.2.4. 클러스터 데이터소스

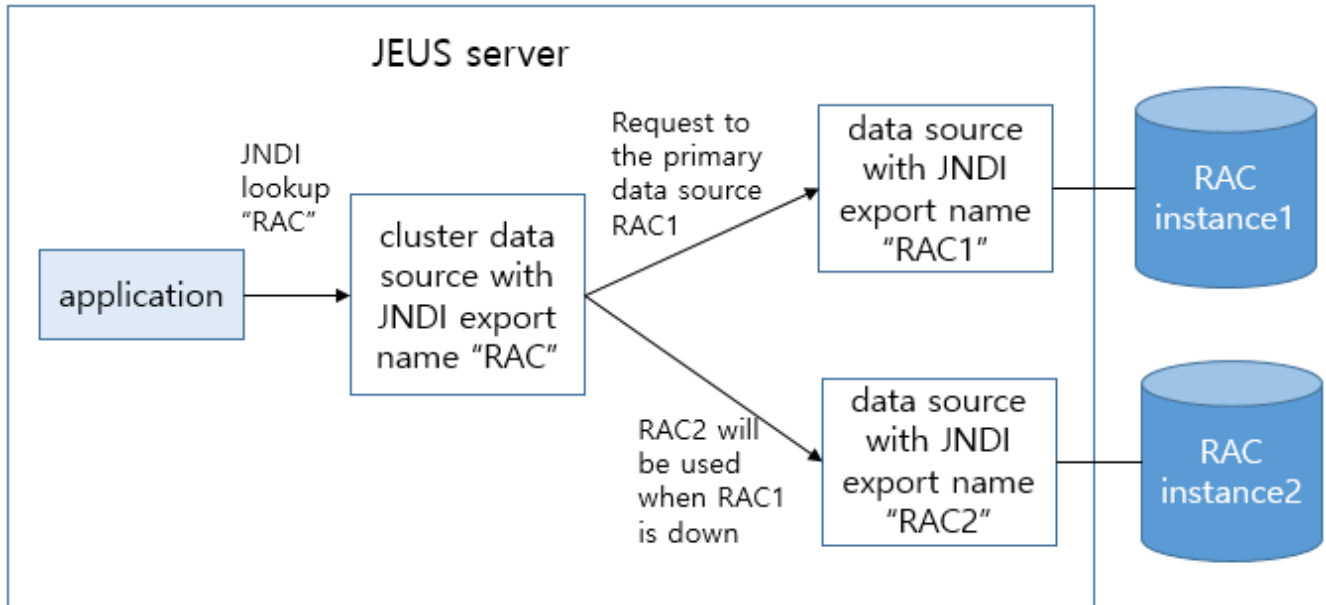
본 절에서는 클러스터 데이터소스의 장애복구 기능과 DataSource Affinity 기능, ONS와 결합기능에 대해서 설명한다.

6.2.4.1. 장애복구 기능

JEUS 레벨에서 RAC 인스턴스 간의 Failover 및 Failback을 지원하기 위해서 클러스터 데이터소스를 제공한다. RAC(Real Application Cluster)는 Oracle에서 제공하는 DB 클러스터링 기능이다. RAC에 관한 자세한 내용은 Oracle의 문서를 참고한다.

클러스터 데이터소스는 근본적으로는 하나의 JNDI 이름을 가진 데이터소스 인스턴스로서 복수의 개별

데이터소스들을 논리적으로 묶어 관리한다. 클러스터 데이터소스에 속한 개별 데이터소스(이하 컴포넌트 데이터소스)들 또한 각자의 JNDI 이름을 갖는 독립적인 데이터소스인데 이들은 반드시 RAC 인스턴스의 데이터소스로 설정되어야 한다. 클러스터 데이터소스는 애플리케이션으로부터 자신에게 들어온 Connection 요청을 이들 컴포넌트 데이터소스 중 하나에 위임하는 방식으로 동작한다. Connection 요청 위임은 Connection 요청 처리에 문제가 없음이 확인된 컴포넌트 데이터소스에 대해서 이루어지므로 클러스터 데이터소스는 애플리케이션에 투명하게 Failover를 수행할 수 있다.



RAC에 대한 클러스터 데이터소스의 Failover

Oracle JDBC 드라이버 레벨에서 제공하는 CTF(Connect Time Failover)보다는 JEUS의 클러스터 데이터소스를 사용하길 권장한다. Oracle CTF의 경우에는 Connection별로 Failover를 하기 때문에 데이터소스 전체가 문제가 생겼을 경우 Connection Pool에 있는 모든 Connection을 복구하려면 오랜 시간이 걸릴 수 있다. 그러나 JEUS 클러스터 데이터소스는 컴포넌트 데이터소스에 문제가 생긴 것을 감지하여 데이터소스 단위로 Failover를 수행하므로 보다 효율적이다. 더불어 JEUS 클러스터 데이터소스는 자동으로 Failback하는 기능도 제공한다.

JEUS 클러스터 데이터소스를 사용하는 방식과 Oracle JDBC 드라이버에 RAC 속성을 설정하는 방식은 그 동작이 서로 다르다. 전자는 Failover를 JEUS가 처리하는 것이기 때문에 각 RAC 인스턴스의 컴포넌트 데이터소스의 fail 여부를 감시하기 위해서 check-query와 check-query-period를 설정해야 한다. 그러나 후자는 드라이버가 Failover를 책임지므로 check-query가 필수적인 것은 아니다. 클러스터 데이터소스 설정에 관한 자세한 내용은 [클러스터 데이터소스 설정](#)을 참고한다.

6.2.4.2. DataSource Affinity 기능

글로벌 트랜잭션이 RAC에서 이루어지게 되는 경우에는 물리적으로 분산된 서로 다른 RAC 인스턴스들에서 트랜잭션 처리가 되는 것보다 되도록이면 하나의 RAC 인스턴스에 한정되어 트랜잭션 처리가 되는 것이 성능상 유리하다. 또한 XA Emulation을 설정한 ConnectionPoolDataSource를 멤버 데이터소스로 갖는 클러스터 데이터소스를 사용하여 강제로 로컬 트랜잭션을 RAC에서 처리하는 경우에는 반드시 하나의 RAC 인스턴스만 사용할 필요가 생긴다.

글로벌 또는 로컬 트랜잭션 처리 상황에서 클러스터 데이터소스는 최초의 커넥션 요청이 어떤 컴포넌트 데이터소스 및 연관된 RAC 인스턴스에서 처리되었는지 기억하고 이후의 커넥션 요청은 무조건 해당 컴포넌트 데이터소스로 위임함으로써 DataSource Affinity, 즉 하나의 RAC 인스턴스에 한정되어 글로벌 또는 로컬 트랜잭션이 처리됨을 보장한다.

복수의 클러스터 데이터소스가 동일한 RAC를 바라보고 있는 경우에도 각각의 클러스터 데이터소스로의 커넥션 요청은 서로 다른 클러스터 데이터소스에 대한 커넥션 요청이라고 할지라도 동일한 RAC 인스턴스에서 처리되도록 한다. 또한 만약 복수의 RAC가 연관되어 있다면 각각의 RAC에 대해서도 역시나 DataSource Affinity가 보장된다. 가령 서로 다른 클러스터 데이터소스 a와 클러스터 데이터소스 b가 각각 서로 다른 RAC A와 RAC B를 바라보고 있다면 클러스터 데이터소스 a로의 커넥션 요청은 RAC A에서 DataSource Affinity를 보장하고 클러스터 데이터소스 b로의 커넥션 요청은 RAC B에서 DataSource Affinity를 보장한다. 서로 다른 RAC인 RAC A와 RAC B 간에는 DataSource Affinity를 보장하는 것이 근본적으로 불가능함에 유의하기 바란다.

DataSource Affinity가 설정되어 있을 경우 기존의 Failover, Failback, Loadbalancing 방식은 무시되고 트랜잭션에 묶인 RAC 인스턴스와 동일한 데이터소스를 우선 선택한다. 트랜잭션 하에서 선택할 데이터소스가 존재하지 않을시 기존의 위 정책들로 데이터소스를 선택하게 된다.

6.2.4.3. ONS와 결합된 클러스터 데이터소스

ONS는 RAC node 간 서로의 상태 정보 공유를 가능하게끔 하는 Oracle의 Notification Service다. RAC에 ONS가 설정되어 있으면 기본적으로 RAC 노드들은 각자의 상태를 모든 RAC 노드들과 공유하게 된다. 한편 RAC 서버 컴포넌트가 아니더라도 ONS 클라이언트로써 ONS에 참여하면 ONS로부터 RAC 노드 상태 정보들을 일부 얻을 수 있는데 JEUS는 이를 이용하여 ONS와 결합된 보다 향상된 기능의 클러스터 데이터소스를 제공한다. ONS와 결합된 클러스터 데이터소스는 내부적으로 각 컴포넌트 데이터소스와 연관된 RAC 인스턴스를 매핑하고 있다. 이와 같은 구조는 ONS로부터 얻은 RAC 인스턴스 정보를 연관된 컴포넌트 데이터소스 관리에 이용하는 것을 가능케 한다.

JEUS가 ONS 클라이언트로써 ONS로부터 얻을 수 있는 상태 정보는 크게 두 가지다.

- RAC 인스턴스 Up & Down Notification

RAC 인스턴스가 기동했는지 혹은 종료했는지 알려준다.

- RAC 인스턴스별 가용량 Percentage

Runtime Load Balancing Advisory라고도 불리는 이 정보는 RAC 전체의 가용량을 100으로 하여 RAC 인스턴스별 가용량을 Percentage로 알려준다.

ONS와 결합된 클러스터 데이터소스 기능

다음은 ONS와 결합된 클러스터 데이터소스에서 제공하는 기능에 대한 설명이다.

- 보다 효과적인 컴포넌트 데이터소스 상태 관리

DB의 상태를 파악하기 위한 기존 방식은 Polling이었는데 이는 요청 처리 와중에 수행됨으로 인해 성능상 부담이 되거나 네트워크 단절 상황에서는 DB 상태 파악이 되지 않는 문제가 있었다. RAC 인스턴스 Up & Down Notification을 이용하게 되면서는 그러한 문제를 피하면서 컴포넌트 데이터소스의 실패 및 복구 여부를 보다 효과적으로 감지할 수 있게 되었다.

- Runtime Load Balancing Advisory를 이용한 효율적인 Load Balancing

기존에는 DB의 가용량을 판단할 수 없었으므로 단순한 Round-Robin 방식의 Load Balancing만을 제공할 수밖에 없었다. 그러나 이제는 Runtime Load Balancing Advisory를 통해 실시간으로 RAC 인스턴스 별 상대적 가용량을 알 수 있게 됨으로써 그를 바탕으로 한 효율적인 Load Balancing이 가능해졌다. 가령 컴포넌트 데이터소스 A와 컴포넌트 데이터소스 B가 각각 RAC 인스턴스 A와 RAC 인스턴스 B와 연관되어 있으며 이들의

가용량이 각각 Runtime Load Balancing Advisory를 통해 60%, 40%로 알려지면 이후의 커넥션 요청은 3:2의 비율로 컴포넌트 데이터소스 A와 컴포넌트 데이터소스 B로 위임된다.

ONS 설정

JEUS를 ONS 클라이언트로 기능하게 하기 위해서는 우선 ONS 라이브러리를 설치해야 한다.

Oracle 지원 사이트에서 ons.jar를 다운로드 받아 JEUS_HOME/lib/datasources 디렉터리에 두면 된다. 이제 클러스터 데이터소스에 ONS 관련 설정을 함으로써 ONS와 결합된 클러스터 데이터소스를 사용할 수 있다. 먼저 기본적으로 ONS상의 각 RAC 노드들이 ONS 통신에 사용하는 IP, 포트를 설정해야 한다. 여기까지 설정하면 클러스터 데이터소스는 ONS와 결합된다. 이제 Failover-Failback 방식이든 Load Balancing 방식이든 클러스터 데이터소스는 컴포넌트 데이터소스의 실패 및 복구 여부를 ONS를 통하여 보다 효과적으로 감지할 수 있다. 특히나 Load Balancing 방식일 때는 Runtime Load Balancing Advisory를 이용하여 좀 더 효율적인 Load Balancing을 제공하게 된다.

6.3. 데이터소스 및 Connection Pool 관리

JEUS가 도메인 구조로 확장됨에 따라 데이터소스 및 Connection Pool 관리 구조에도 일부 변화가 생겼다. 그러나 확장된 도메인 구조에 대한 이해가 바탕이 되면 변화된 데이터소스 및 Connection Pool 관리 구조에 대한 학습은 크게 어렵지 않은 수준에서 이루어질 수 있다.

JEUS에서 서비스를 수행하는 최소 단위는 **서버**이며 경우에 따라 복수의 서버가 클러스터로 묶여 서비스를 수행할 수도 있다. 다시 이들 서버 및 클러스터는 하나의 관리 단위로 묶여질 수 있는데 그러한 관리 단위가 바로 **도메인**이다. 즉, 도메인은 연관되어 관리될 필요가 있는 서버 및 클러스터들의 집합을 의미한다. 도메인에는 도메인에 속한 서버 및 클러스터들을 관리하고 그와 관련된 제반 서비스를 제공하는 특별한 역할의 유일한 서버가 존재하는데 이를 **MASTER**라고 한다. MASTER를 제외한 도메인에 속한 모든 서버는 **MS**라고 한다. MASTER는 MASTER로 기능하는 동시에 MS로도 기능할 수 있으나 서비스는 MS를 통해 제공하고 MASTER는 본연의 도메인 관리 역할만 수행하는 방식으로 사용할 것을 권장한다. JEUS의 도메인 구조에 대한 보다 자세한 내용은 "JEUS Domain 안내서"를 참고한다.

본 절에서는 도메인 구조에서 데이터소스 및 Connection Pool이 관리되는 방식에 대해서 설명한다.

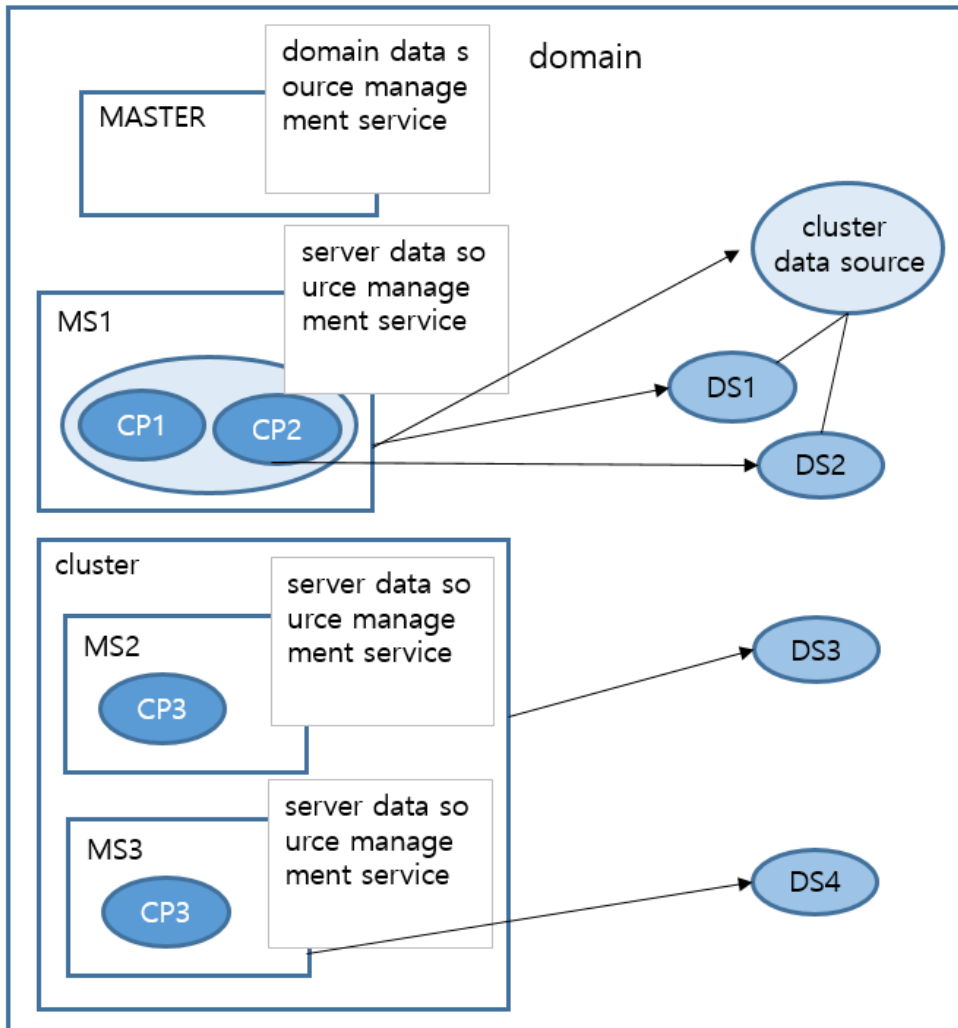
데이터소스는 기본적으로 도메인 범주에서 노출되는 자원이다. 도메인에 속한 서버 및 클러스터(보다 정확히 얘기하면 클러스터에 속한 서버)가 노출된 데이터소스 설정을 참조하여 자신들만의 Connection Pool을 생성해 Connection Pool 서비스를 제공하게 되는 것이다. 클러스터는 서버들의 묶음을 추상화한 것일 뿐 서비스를 제공하는 실질적 주체는 아니므로 클러스터에서 참조하는 데이터소스는 실제로는 그것에 속한 서버들에서 유효하다. 그렇기 때문에 클러스터에 속한 서버들은 클러스터에서 참조하는 데이터소스를 참조할 수 있다. 또한 클러스터에 속한 서버는 자체적으로 데이터소스를 참조하도록 설정될 수 있다. 이 경우 서버는 클러스터에서 참조하는 데이터소스와 자신이 참조하는 데이터소스 모두를 유효한 데이터소스로 사용할 수 있게 된다.

서버 및 클러스터의 데이터소스 참조는 서버 및 클러스터 자신이 참조할 데이터소스의 ID를 명시하여 등록함으로써 가능해진다. 즉, 서버 및 클러스터가 특정 데이터소스 ID를 명시하면 해당 서버 및 해당 클러스터에 속한 서버는 명시한 데이터소스의 설정을 읽어 Connection Pool을 생성하는 데 필요한 정보들을 구성하고 그것을 데이터소스의 JNDI 이름과 매핑하여 자신의 JNDI Repository에 bind한다. 이 과정이 이루어지고 나면 서버에 deploy된 애플리케이션은 bind한 JNDI 이름으로 데이터소스를 lookup하여 Connection Pool을 생성하고 사용할 수 있게 된다.



데이터소스의 JNDI bind는 서버 단위로 이루어지므로 서로 다른 데이터소스라할지라도 동일한 서버에 JNDI bind되지 않음을 보장할 수 있으면 동일한 JNDI 이름을 가질 수 있다. 이는 곧 동일한 JNDI 이름을 가진 서로 다른 데이터소스들을 임의의 서버에서 동시에 참조하도록 하는 설정은 허용되지 않음을 의미한다. 도메인에서 데이터소스 ID로서 데이터소스를 구별하는 이유는 이처럼 서로 다른 데이터소스가 동일한 JNDI 이름을 가질 수 있기 때문이다. 데이터소스 ID는 도메인에서 유일한 값으로 설정되어 데이터소스의 식별자로서 기능할 수 있어야 한다.

다음은 좀 더 구체적인 예로 MASTER와 3개의 MS로 이루어진 도메인 구성이다.



JETC 도메인 구조에서의 데이터소스 및 Connection Pool 관리

MS1은 클러스터 데이터소스를 등록하고 있다. 클러스터 데이터소스의 컴포넌트 데이터소스는 DS1과 DS2로서 MS1이 클러스터 데이터소스를 온전히 사용하기 위해서는 클러스터 데이터소스의 컴포넌트 데이터소스인 DS1과 DS2 역시 MS1에 등록되어야 한다. 이와 같이 설정함으로 MS1은 DS1과 DS2 Connection Pool을 각각 생성한 후 클러스터로 묶어 클러스터 Connection Pool 서비스를 제공할 수 있다.

한편 MS2와 MS3는 클러스터로 묶여있다. 클러스터는 DS3를 등록하고 있으므로 DS3는 클러스터에 속한 MS2와 MS3에서 모두 유효하다. 이는 곧 MS2와 MS3 각각이 DS3 Connection Pool을 생성해 Connection Pool 서비스를 제공할 수 있음을 의미한다.

또한 MS3는 단독으로 자신에 DS4를 등록하고 있다. 따라서 MS3는 클러스터에서 등록한 DS3와 더불어 자신이 등록한 DS4에 대해서도 Connection Pool 서비스를 제공할 수 있다.

각각의 MS는 서버 수준에서의 데이터소스 및 Connection pool 관리 서비스를, MASTER는 도메인 수준에서의 데이터소스 및 Connection Pool 관리 서비스를 수행한다. MASTER 서비스는 각각의 MS 서비스들과 연계하여 도메인에 존재하는 모든 데이터소스 및 Connection Pool을 총체적으로 관리하며 콘솔 툴부터 데이터소스 및 Connection Pool 관련 요청을 처리하게 된다.

6.4. 데이터소스 설정

데이터소스 및 JDBC Connection Pool을 실제 JEUS에서 사용하기 위해 가장 먼저 해야할 일은 JEUS_HOME/lib/datasource/ 디렉터리 내에 JDBC 드라이버 라이브러리가 존재하는지 확인하고 데이터소스 구성에 필요한 정보를 설정하는 것이다. 데이터소스 설정은 크게 JDBC 드라이버 setup 등에 필요한 기본 설정과 Connection Pool 설정으로 이루어진다.

본 절에서는 콘솔 툴을 사용한 설정 방법에 대해서 간단히 설명한다. 콘솔 툴을 사용한 자세한 설정 방법은 JEUS Reference 안내서의 "add-data-source"를 참고한다.

```
[MASTER]domain1.adminServer>add-data-source -id ds1 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 61.77.153.4 -pn 1521 -dn orcl -user scott
-password tiger --property driverType:java.lang.String=thin
Successfully performed the ADD operation for data source [ds1] to domain.
Check the results using "add-data-source".
[MASTER]domain1.adminServer>addds
Shows the current configuration.
Data sources in domain
=====
+-----+-----+
| ds1   | common data source          |
+-----+-----+
=====

[MASTER]domain1.adminServer>
```

DataSourceAccountProvider 인터페이스

JEUS는 기본적으로 데이터소스에 DB 접속 계정 정보를 설정할 수 있게 하여, JEUS 내 설정 저장소에 보관함으로써 사용자 애플리케이션에서 JDBC connection 요청시 매번 파라미터로 DB 접속 계정 정보를 전달하지 않아도 되게끔 하고있다. 또한 이렇게 사용자가 저장하는 DB 접속 계정 정보는 안전한 관리를 위해 JEUS Security와 연동하여 암호화 과정을 거친 후 저장소에 보관하는 것이 가능하다.

이 외에도, JEUS는 데이터소스에서 사용할 DB 접속 계정 정보를 JEUS 설정이 아닌 외부 저장소에 보관할 수도 있도록 지원하며, DB 접속 계정을 보호하기 위한 보안 모듈 또한 JEUS Security 외에도 다른 외부 보안 모듈을 사용할 수 있도록 지원한다. 이와 같은 외부 시스템과의 유연한 연동을 위해 JEUS는 jeus.jdbc.helper.DataSourceAccountProvider 인터페이스를 명시하고 있다. 사용자는 자신이 원하는 외부 시스템과 연동하도록 이 인터페이스를 적절히 구현함으로써 다양하고 자유로운 방식으로 DB 접속 계정 정보를 관리할 수 있다.

jeus.jdbc.helper.DataSourceAccountProvider 인터페이스의 명세는 다음과 같다.

jeus.jdbc.helper.DataSourceAccountProvider

```
public interface DataSourceAccountProvider {  
    String getUser(Map<String, String> dataSourceConfigurationMap)  
        throws DataSourceAccountProviderException;  
    String getPassword(Map<String, String> dataSourceConfigurationMap)  
        throws DataSourceAccountProviderException;  
}
```

메소드	설명
getUser(Map<String, String> dataSourceConfigurationMap)	JEUS로부터 파라미터로 데이터소스 설정이 담긴 Map을 넘겨받아 필요시 이를 적절히 활용하여 JEUS에게 DB 접속을 위한 평문 user 값을 리턴해준다.
getPassword(Map<String, String> dataSourceConfigurationMap)	JEUS로부터 파라미터로 데이터소스 설정이 담긴 Map을 넘겨받아 필요시 이를 적절히 활용하여 JEUS에게 DB 접속을 위한 평문 password 값을 리턴해준다.

앞서 소개한 jeus.jdbc.helper.DataSourceAccountProvider 인터페이스의 각 API에 파라미터로 전달되는 Map에는 JEUS 데이터소스 설정값 중 일부가 들어가며, 구현체에서는 다음과 같은 key 값으로 해당되는 value를 얻을 수 있다.

key	value
DATA_SOURCE_ID	JEUS 데이터소스 설정에 입력된 Data Source ID
EXPORT_NAME	JEUS 데이터소스 설정에 입력된 Export Name
VENDOR	JEUS 데이터소스 설정에 입력된 Vendor
DATA_SOURCE_CLASS_NAME	JEUS 데이터소스 설정에 입력된 Data Source Class Name
SERVER_NAME	JEUS 데이터소스 설정에 입력된 Server Name
PORT_NUMBER	JEUS 데이터소스 설정에 입력된 Port Number
DATABASE_NAME	JEUS 데이터소스 설정에 입력된 Database Name
USER	JEUS 데이터소스 설정에 입력된 User
PASSWORD	JEUS 데이터소스 설정에 입력된 Password

앞서 소개한 jeus.jdbc.helper.DataSourceAccountProvider 인터페이스의 각 API가 던지는 DataSourceAccountProviderException은 다음과 같은 상황에서 발생해야 한다.

Exception 이름	Exception이 발생하는 상황
jeus.jdbc.datasource.DataSourceAccountProviderException	DB에 접속하기 위해 필요한 평문 user 또는 password 값을 얻는데 실패하였을 때 발생

끝으로, 사용자의 의도대로 구현된 jeus.jdbc.helper.DataSourceAccountProvider 인터페이스 구현체를 사용하기 위해서는 그 클래스 이름을 다음 jvm-option으로 JEUS 서버에 지정해 주어야 한다.

jvm-option 이름	jvm-option 값
jeus.jdbc.config.data-source-account-provider-class-name	jeus.jdbc.helper.DataSourceAccountProvider 인터페이스에 대한 구현체의 전체 이름

만약에 사용자가 위의 jvm-option을 설정하지 않는다면, JEUS는 JEUS Security와 연동하는 기본 구현체를 사용한다.

위의 명세를 바탕으로 다음과 같이 Customize 된 DataSourceAccountProvider 구현이 가능하다. 아래 예시들에서는 DB 접속 계정 정보를 scott/tiger라고 가정한다.

DataSourceAccountProvider 구현 예1 - DB 접속 계정 정보를 JEUS 설정에 평문으로 저장한 경우

```
public class MyDataSourceAccountProvider implements DataSourceAccountProvider {
    @Override
    public String getUser(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // JEUS 설정에서 넘겨받은 user 값을 그대로 다시 리턴.
        return dataSourceConfigurationMap.get(USER); // scott이 얻어짐.
    }

    @Override
    public String getPassword(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // JEUS 설정에서 넘겨받은 password 값을 그대로 다시 리턴.
        return dataSourceConfigurationMap.get(PASSWORD); // tiger가 얻어짐.
    }
}
```

DataSourceAccountProvider 구현 예2 - DB 접속 계정 정보를 별도의 파일에 저장한 경우

```
public class MyDataSourceAccountProvider implements DataSourceAccountProvider {
    @Override
    public String getUser(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // JEUS 설정에서 넘겨받은 SERVER_NAME 값을 이용해서 파일에서 적절한 파싱을 통해 user를 얻어옴.
        File file = getFile(); // DB 접속 계정 정보가 적힌 파일을 얻어옴.
        // 파일을 잘 파싱해서 user 값인 scott을 얻어오도록 자체적으로 구현.
        return getUserFromFileWithSID(file, dataSourceConfigurationMap.get(SERVER_NAME));
    }

    @Override
    public String getPassword(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // JEUS 설정에서 넘겨받은 SERVER_NAME 값을 이용해서 파일에서 적절한 파싱을 통해 password를 얻어옴.
        File file = getFile(); // DB 접속 계정 정보가 적힌 파일을 얻어옴.
    }
}
```



```

    // 파일을 잘 파싱해서 password 값인 tiger를 얻어오도록 자체적으로 구현.
    return getPasswordFromFileWithSID(file, dataSourceConfigurationMap.get(SERVER_NAME));
}
}

```

DataSourceAccountProvider 구현 예3 - 외부 보안 모듈인 ExternalSecurity와 연동하여 DB 접속 password만 얻어오는 경우

```

public class MyDataSourceAccountProvider implements DataSourceAccountProvider {
    @Override
    public String getUser(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // JEUS 설정에서 넘겨받은 user 값을 그대로 다시 리턴.
        return dataSourceConfigurationMap.get(USER); // scott이 얻어짐.
    }

    @Override
    public String getPassword(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // 외부 보안 모듈인 ExternalSecurity에 파라미터로 전달할 JEUS 데이터소스 설정 값들을 얻음.
        // JEUS 데이터소스 설정상의 data source ID를 얻음.
        String dataSourceID = dataSourceConfigurationMap.get(DATA_SOURCE_ID);
        // JEUS 데이터소스 설정상의 server name을 얻음.
        String serverName = dataSourceConfigurationMap.get(SERVER_NAME);
        // JEUS 데이터소스 설정상의 user를 얻음.
        String user = dataSourceConfigurationMap.get(USER);

        // 외부 보안 모듈인 ExternalSecurity에 파라미터로 dataSourceID, serverName,
        // user를 전달하고, ExternalSecurity에서는 그에 맞는 password를 리턴해준다고 가정.
        return ExternalSecurity.getPassword(dataSourceID, serverName, user); // tiger가 얻어짐.
    }
}

```

6.4.1. Connection Pool 설정

jeusadmin에서 list-data-sources 명령어로 현재 datasource 설정을 확인할 수 있다.

```

[MASTER]domain1.adminServer>list-data-sources -id ds1
The configuration of the data source [ds1]
=====
+-----+-----+
| Configuration Name | Configuration Value |
+-----+-----+
| id                 | ds1                  | |
| export-name        | ds1                  |
| data-source-class-name | oracle.jdbc.pool.OracleConnectionPoolDataSource |
| data-source-type    | ConnectionPoolDataSource |
| server-name        | 61.77.153.4          |
| port-number        |                       | 1521 |
| database-name       | orcl                 |
| user               | scott                |
| password           | tiger                |
| login-timeout       |                       | 0    |
| auto-commit        | DRIVER              |

```

```

| stmt-query-timeout          | 0 |
| pool-destroy-timeout        | 10000 |
| property                    | [driverType;java.lang.String;thin] |
| support-xa-emulation        | false |
| min                          | 2 |
| max                          | 30 |
| step                         | 1 |
| period                       | 3600000 |
| enable-wait                  | false |
| wait-time                    | 10000 |
| max-use-count                | 0 |
| dbaTimeout                   | -1 |
| stmt-caching-size            | -1 |
| stmt-fetch-size              | -1 |
| connection-trace             | false |
| get-connection-trace         | true |
| auto-commit-trace           | false |
| use-sql-trace                | false |
| keep-connection-handle-open | false |
+-----+-----+
=====
[MASTER]domain1.adminServer>

```

다음은 각 설정 항목에 대한 설명이다.

• Pooling

JDBC Connection Pool의 사이즈 및 그의 조정에 관련된 설정들을 정의한다.

항목	설명
Min	Connection Pool에 Pooling되는 Connection의 최솟값을 지정한다.
Max	Connection Pool에 Pooling되는 Connection의 최댓값을 지정한다.
Step	Connection Pool에 Connection이 부족할 때 현재 Connection의 개수가 최댓값 이하인 경우 DB로부터 Connection을 새로 받아와 채우는데 이때 새로 받아오는 Connection의 개수를 지정한다.
Period	Connection Pool의 크기를 최솟값에 맞춰 조정하는 주기를 설정한다. Connection Pool의 크기가 최솟값을 초과하는 경우에는 사용하지 않는 Connection들을 닫아주고 Connection Pool의 크기가 최솟값에 미치지 못하는 경우에는 DB로부터 Connection을 새로 받아와 채운다. (단위: ms)

• Wait Free Connection

Connection Pool에 있는 모든 Connection들이 점유되어 있을 때 Connection 요청을 핸들링하는 메소드를 정의한다.

항목	설명
Enable Wait	<p>Connection Pool에 사용 가능한 Connection이 없고 Connection도 더 이상 늘릴 수 없을 때 Connection 요청을 처리하는 방법을 결정한다.</p> <ul style="list-style-type: none"> • true : 이용 가능한 Connection을 얻기 위해 기다린다. • false : 새로운 Connection을 만들어서 제공하지만 그 Connection은 반환되었을 때 Pooling되지 않고 버려진다. 이를 일회용(disposable) Connection이라고도 한다.
Wait Time	<p>'Enable Wait'이 true인 경우에만 유효한 설정으로 Connection을 얻기 위해 대기하는 한계 시간을 나타낸다. 만약 이 시간이 지나도 Connection을 얻지 못하면 JEUS는 타임아웃 예외를 발생시킨다. (단위: ms)</p>

• Connection Validation

애플리케이션이 Connection을 요청했을 때 Connection을 애플리케이션에 넘겨주기 전 특정 쿼리를 수행하여 Connection의 상태를 점검(validation)하는 기능이다. JDBC Connection 내부 에러로 인한 끊김, 방화벽에 의한 소켓 끊김 현상 등을 체크할 때 유용하다.

Connection의 상태에 이상이 있는 경우 Connection을 DB로부터 새로 받아 애플리케이션에 전달한다. 만약 RAC를 위한 클러스터 데이터소스에 속한 데이터소스라면 반드시 이 설정을 해야 한다.

항목	설명
Check Query	<p>Connection 상태 점검에 사용될 쿼리를 설정한다. 보통 DB와의 연결 유효성만을 확인하면 되므로 간단한 select 쿼리를 사용하는 것을 권장한다.</p> <p>한편 JEUS 7 Fix#2부터는 DB와의 연결 유효성을 확인할 때 check-query를 수행하는 대신 JDK1.6에서 java.sql.Connection에 추가된 isValid 메소드를 이용할 수 있다. 쿼리문 대신 "use isValid method"를 적으면 된다.</p> <p>그 밖의 Connection Validation 관련 설정들(Check Query Timeout, Check Query Period 등)은 isValid 메소드를 사용할 때도 동일하게 적용된다.</p>
Check Query Timeout	<p>Connection 점검을 위해 Check Query를 수행했을 때 DB가 응답이 없어 드라이버가 계속 기다리는 상황이 발생할 수 있다. 이런 경우를 피하기 위해 Check Query에 대해 쿼리 타임아웃을 적용한다. (단위: ms)</p> <p>이것은 JDBC API에서 정의한 java.sql.Statement#setQueryTimeout 메소드를 호출함으로 가능하다.</p> <p>1000ms보다 적을 경우 0으로 설정되므로 주의한다.</p>
Non Validation Interval	<p>Connection 점검이 너무 잦아서 오버헤드가 발생하는 경우 설정한다. Connection 점검을 수행하기 직전의 시각과 가장 최근의 Connection 점검 시각과의 차이가 설정한 시간 간격 이내면 Connection 점검을 생략하도록 하는 설정이다. (단위: ms)</p> <p>예를 들어 이 설정값이 5000ms인 경우 어떤 Connection의 마지막 Connection 점검 시간으로부터 아직 5초가 지나지 않았다면 그 Connection에 대한 점검이 생략된 채 애플리케이션에 전달된다.</p>

항목	설명
Check Query Period	Connection Pool의 Connection들을 설정한 주기마다 체크하여 문제가 있는 Connection을 제거한다. 클러스터 데이터소스에 속한 데이터소스는 자신의 상태 체크에 사용하므로 반드시 설정해야 한다. (단위: ms)
Check Query Class	<p>사용자나 개발자가 Connection 점검 기능을 Customize하고 싶을때 그것을 위해 구현한 클래스의 패키지 이름을 포함한 이름을 적어준다.</p> <p>이때 그 클래스는 반드시 jeus.jdbc.connectionpool.JEUSConnectionChecker 인터페이스를 구현해야 한다. 자세한 내용은 "JEUSConnectionChecker 인터페이스"를 참고한다.</p>
Check Query Retrial Count	<p>Connection 점검은 기본적으로 Destroy Policy On Check Query가 FAILED_CONNECTION_ONLY로 설정되어 있을 경우 한 번 수행된다.</p> <p>Destroy Policy On Check Query가 ALL_CONNECTIONS로 설정되어 있을 경우에는 최초의 Connection 점검에서 Connection 이상이 확인되면 또 다른 Connection에 대해서 한 번 더 Connection 점검이 이루어져 총 두 번의 Connection 점검이 수행될 수 있다. 이 설정값이 이러한 기본 Connection 점검 수행 횟수에 더해져 최종 Connection 점검 수행 횟수가 정해진다.</p>
Destroy Policy On Check Query	<p>Connection이 유효하지 않은 것으로 확인되었을 때 Connection Pool에 있는 다른 Connection들에 대한 처리 정책을 설정한다.</p> <ul style="list-style-type: none"> • FAILED_CONNECTION_ONLY : 유효하지 않은 것으로 확인된 Connection만 제거한다. • ALL_CONNTECTIONS : 유효하지 않은 것으로 확인된 Connection을 제거하고 Connection Pool에 있는 다른 Connection의 유효성을 한 번 더 확인한다. 그조차 유효하지 않은 것으로 확인되면 Connection Pool의 모든 Connection을 제거한다.

• Connection Pool

기타 Connection Pool의 부가 기능들을 설정한다.

항목	설명
Delegation Datasource	<p>트랜잭션과 연동하지 않은 상태에서는 XA 데이터소스를 통해 Connection을 얻기보다 Connection Pool 데이터소스를 통해 Connection을 얻는 것이 낫다.</p> <p>기능상 차이가 없을 뿐더러 트랜잭션 연동을 위한 기능을 포함하고 있는 XA Connection은 아무래도 시스템에 부담을 더 주기 때문이다. 이를 위해 XA 데이터소스인 경우 이 설정을 통하여 트랜잭션과 연동하지 않은 상태에서의 Connection 요청을 위임할 Connection Pool 데이터소스를 지정한다.</p> <p>한편 Oracle, DB2 등에서 XA Connection을 트랜잭션 없이 사용도 하고 트랜잭션에 연동도 하면서 사용하다 보면 XA를 시작할 수 없는 예외가 발생하기도 하는데 정확한 원인은 알 수 없기 때문에 이를 회피하기 위한 방편으로서도 이 설정을 이용한다.</p>
Max Use Count	Connection의 최대 사용 횟수이다. 이 사용 회수 이상이 되면 새로운 Connection으로 교체한다. (기본값: 0, Connection을 교체하지 않겠다는 의미)

항목	설명
Delegation Dba	<p>DB의 세션을 강제로 죽일 수 있는 권한(DBA 권한)을 가진 데이터소스(이하 DBA 위임 데이터소스)의 JNDI 이름을 설정한다. 이 설정을 한 데이터소스로부터 얻어진 Connection을 이용한 쿼리 수행이 일정 시간 이상 지체되면 JEUS는 위임 DBA 데이터소스를 통해 해당 Connection과 연관된 DB 세션을 강제로 제거하도록 하는 쿼리를 DB에 날린다.</p> <p>이후 애플리케이션이 사용 불가능해진 Connection으로 인해 발생한 예외를 처리하고 Connection을 닫아주게 되면 JEUS는 그 Connection을 제거하고 DB로부터 새로운 Connection을 얻어 Connection Pool에 넣는다. 현재 Tibero, Oracle, Sybase에 대해서 이 기능을 지원한다.</p> <p>이 기능은 JDBC 2.0 이하 JDBC 드라이버에서 쿼리 수행이 지나치게 오래 걸릴 때 그것을 중단시킬 방법으로서 고안된 것이다. 그러나 JDBC 3.0 또는 그 이상의 버전을 구현한 JDBC 드라이버는 <code>java.sql.Statement#setQueryTimeout</code>을 구현하므로 이 기능을 통해 강제로 DB 세션을 제거하기보다는 Stmt Query Timeout 설정을 이용하는 것을 권장한다.</p> <p>특히나 XA 데이터소스의 경우 XA가 정상적으로 진행하는 도중에 DB 세션이 제거되면 XA 처리에 문제가 발생할 수 있기 때문에 Stmt Query Timeout 설정과 서버의 트랜잭션 타임아웃 설정을 적절하게 사용하도록 한다.</p>
Dba Timeout	<p>위임 DAB 데이터소스는 이 설정으로 지정한 시간 동안만 Connection의 쿼리 수행을 기다려준다. 설정한 시간이 경과하면 해당 Connection과 연관된 DB 세션을 강제로 제거하도록 하는 쿼리를 DB로 날린다. (단위: ms)</p> <p>Delegation DBA가 설정된 경우에만 유효하다.</p>
Stmt Caching Size	JDBC 드라이버는 애플리케이션에서 PreparedStatement를 요청할 때마다 파라미터로 넘어온 SQL 문장을 파싱하게 된다. 이 파싱 작업이 성능에 영향을 줄 수 있기 때문에 이를 피하기 위해서 JEUS 내부적으로 PreparedStatement를 캐시하는 기능을 제공한다. 이 설정은 캐싱할 PreparedStatement의 개수를 지정한다.
Stmt Fetch Size	JDBC 드라이버 Statement의 fetch 사이즈를 설정한다.
Use Sql Trace	<p>Connection별로 사용하고 있는 SQL 쿼리를 보여주는 기능이다. <code>jeus.jdbc.sql</code> 로거의 레벨을 FINE으로 설정할 경우 서버 로그를 통해서 SQL 쿼리 히스토리를 확인할 수 있다.</p> <p>이 기능을 사용할 경우 JDBC 드라이버의 Statement 구현체를 JEUS의 Statement 구현체로 감싸게되므로 JDBC 드라이버의 Statement 객체를 캐스팅해서 사용하는 애플리케이션은 이 기능을 사용할 수 없다.</p>

항목	설명
Keep Connection Handle Open	<p>Connection을 Pooling하는 동안 Connection 핸들(또는 논리적 Connection)을 항상 열어두고 사용할 때 설정한다.</p> <p>[참고]</p> <ol style="list-style-type: none"> 1. IBM DB2에서 제공하는 Universal Driver(JCC) type 4의 XA 데이터소스를 사용할 경우 이 기능을 사용할 것을 권장한다. 여러 Thread들이 서로 다른 물리적 Connection에 대한 Connection 핸들을 open & close하면서 사용하다가 hang up되는 문제가 발생하기 때문이다. <p>이들 Thread들은 내부적으로 같은 vector(java.util.Vector)를 공유해서 사용하고 있는데 한 Thread가 vector에 대한 락을 선점한 상태에서 무한루프에 빠지고 다른 Thread들은 그 lock을 기다리게 되는 것이 표면적 원인이다. DB2 내부 로직을 알 수 없기에 보다 근본적인 원인은 알 수 없지만 테스트 결과 Connection 핸들을 항상 열어두면 문제가 발생하지 않는 것을 확인하였다. Connection 핸들을 항상 열어두면 내부 공유 vector에 접근하는 일이 Connection 핸들을 가장 처음 생성할 때와 물리적 Connection을 닫을 때 외에는 없기 때문이다.</p> <ol style="list-style-type: none"> 2. DB2 드라이버 3.53.95부터 위의 버그가 해결되었다. 이에 대한 IBM 공식 버그 리포트 넘버는 APAR IZ41181이며 DB2 9.5 Fixpak 4 문서에서 확인할 수 있다. <p>[주의]</p> <p>이 기능을 사용하면 Connection 핸들이 닫히지 않으므로 Connection을 닫을 때 드라이버가 해주는 클리어 작업이 이뤄지지 않는다. 예를 들어 Oracle JDBC 드라이버의 경우 Auto Commit을 false로 해놓고 사용하다가 commit이나 rollback을 하지 않고 Connection을 닫으면 무조건 commit을 하도록 되어 있는데 이러한 처리가 되지 않는다.</p>
Init Sql	Connection을 생성한 후 가장 처음으로 수행할 SQL 쿼리를 설정한다.

• Connection Trace

Connection 관련 부가 정보 제공 여부를 결정한다.

항목	설명
Enabled	<p>Connection 관련 부가 정보 제공 여부를 결정한다.</p> <ul style="list-style-type: none"> • false : 'Get Connection Trace'와 'Auto Commit Trace' 설정이 모두 무효하다.
Get Connection Trace	애플리케이션이 java.sql.DataSource#getConnection을 호출했을 때의 Stack Trace를 확인할 수 있도록 한다.
Auto Commit Trace	java.sql.Connection#setAutoCommit이 호출되었을 때 관련 로그와 Stack Trace를 서버 로그에 기록하도록 한다. 단, jeus.jdbc.connection-trace 로거의 로그 레벨을 FINE으로 설정해야 한다.

JEUSConnectionChecker 인터페이스

Check Query Class 설정에서 언급한 jeus.jdbc.connectionpool.JEUSConnectionChecker 인터페이스의 명세는 다음과 같다. Customize된 check-query 기능을 사용하기 원할 경우 이 인터페이스를 구현하고 해당 클래스를 Check Query Class로서 설정한다.

jeus.jdbc.connectionpool.JEUSConnectionChecker

```
public interface JEUSConnectionChecker {
    void setConnectionPoolID(String connectionPoolID);
    void setQueryString(String query);
    void setQueryTimeout(int timeout);
    void checkConnection(Connection vcon) throws SQLException;
}
```

메소드	설명
setConnectionPoolID()	데이터소스 ID 설정값이 인자로 호출된다. 정보성 자료로 활용할 수 있다.
setQueryString()	Check Query가 설정된 경우 설정값이 인자로 호출된다. 설정한 쿼리가 커넥션 점검에 사용된다.
setQueryTimeout()	Check Query Timeout이 설정된 경우 설정값이 인자로 호출된다. 설정한 쿼리 타임아웃이 커넥션 점검에 적용된다.
checkConnection()	실제 Connection 점검을 수행할 때 불려진다. 그러므로 개발자는 이 메소드에 자신이 커넥션을 점검할 때 수행할 작업들을 구현한다.

JEUS는 JEUSConnectionChecker 인터페이스에 대한 구현체로

jeus.jdbc.connectionpool.DefaultConnectionChecker와

jeus.jdbc.connectionpool.TimeLimitedConnectionChecker 두 가지를 기본적으로 제공한다.

사용자가 Check Query Class에 별도의 설정을 하지 않았을 경우 가장 기본적인 구현체인

DefaultConnectionChecker가 자동으로 설정되어 동작하고, 사용자가 TimeLimitedConnectionChecker를

설정하였다면 JDBC 드라이버 뿐만 아니라 JEUS 자체적으로도 Check Query Timeout을 처리하는 방식으로

Connection Validation이 동작하게 된다. 이렇게 TimeLimitedConnectionChecker를 사용하는 경우에는 JDBC

드라이버가 쿼리 수행 중 hang에 빠진 경우에도 JEUS가 Connection Validation을 계속 이어나갈 수 있지만, 좀비

스레드가 생성될 수 있으므로 문제 상황을 로그로 확인할 수 있도록 하기 위해 jeus.connectionpool.time-limited-connection-checker 로거의 레벨을 최소 FINE 보다 세밀하게 설정해야 한다.

6.5. 클러스터 데이터소스 설정

본 절에서는 클러스터 데이터소스의 설정에 대하여 알아본다. 콘솔 툴을 사용한 설정 방법은 JEUS Reference 안내서의 "add-cluster-data-source"를 참고한다.

6.5.1. 클러스터 데이터소스 설정

```
[MASTER]domain1.adminServer>add-cluster-data-source -id cds1 -cds ds1,ds2
Successfully performed the ADD operation for cluster data source [cds1] to domain.
Check the results using "add-cluster-data-source".
[MASTER]domain1.adminServer>addcds
Shows the current configuration.
Data sources in domain
=====
+-----+-----+-----+
| ds1 | common data source |
| ds2 | common data source |
| cds1 | cluster data source |
+-----+-----+-----+
=====
```

다음은 각 설정 항목에 대한 설명이다.

• 기본 정보

항목	설명
Data Source Id	클러스터 데이터소스의 ID이다. 하나의 도메인에서 클러스터 데이터소스 ID는 클러스터 데이터소스의 유일한 식별자로서 동작하도록 설정해야 한다.
Export Name	클러스터 데이터소스의 JNDI 이름이다. 서로 다른 두 클러스터 데이터소스가 서로 다른 서버에 JNDI bind되는 것을 보장할 수 있으면 해당 클러스터 데이터소스들은 서로 같은 JNDI 이름을 가질 수 있다. 이는 임의의 서버에서 동일한 JNDI 이름을 가지는 서로 다른 데이터소스를 허용하지 않음을 의미한다. 설정되지 않으면 클러스터 데이터소스 ID를 JNDI 이름으로 사용한다.
Data Source Selector	클러스터 데이터소스로부터 커넥션을 얻을 때 사용자나 개발자가 특정 컴포넌트 데이터소스 선택에 대한 정책을 직접 정의할 수 있다. jeus.jdbc.helper.DataSourceSelector 인터페이스를 구현하고 그 구현 클래스의 패키지 이름을 포함하는 이름을 적어준다. 자세한 내용은 " DataSourceSelector 인터페이스 "를 참고한다. 이 항목을 설정하면 Load Balance 설정은 의미가 없다. 정책을 정의할 때는 대체로 동기화를 고려해야 하며 이는 구현자의 몫이다.
Load Balance	Load Balancing 여부를 설정한다. 이 설정값이 true이면 ' Use Failback ' 설정은 적용되지 않는다.
Component Data Sources	클러스터 데이터소스에 속한 컴포넌트 데이터소스들의 데이터소스 ID를 명시한다. 명시된 순서대로 주 데이터소스의 역할을 맡게 된다.

• 고급 선택사항

항목	설명
Is Pre Conn	클러스터 데이터소스에 속한 컴포넌트 데이터소스들의 Connection Pool을 미리 생성할지 여부를 결정한다. 컴포넌트 데이터소스들의 Connection Pool을 미리 생성해놓으면 성능상으로 이점이 있으나 리소스의 절약면에서는 좋지 못하다.

항목	설명
Use Failback	<p>이전 버전의 JEUS에서는 Failover만을 지원했으므로 이에 대한 호환성을 위해 제공하는 옵션이다.</p> <p>보조 데이터소스로 Failover한 후에 주 데이터소스로 Failback할 것인지의 여부를 설정한다. 기본적으로 Failback을 시도한다. Failback을 위해서는 반드시 주 데이터소스에 대하여 'Check Query' 및 'Check Query Period' 항목을 설정해야 한다.</p>
DataSource Affinity	<p>트랜잭션 사용 중 데이터소스에 대한 Affinity 설정 여부를 정한다. 이 설정이 켜지면 트랜잭션 처리가 하나의 멤버 데이터소스 인스턴스에 한정하여 이루어져 글로벌 트랜잭션은 처리 성능 향상을 도모할 수 있고, XA Emulation을 설정한 로컬 트랜잭션도 클러스터 데이터소스 사용 간 보장 될 수 있다.</p>
Ons Support	<p>ONS와 결합된 클러스터 데이터소스를 설정한다. ONS와 결합된 클러스터 데이터소스는 컴포넌트 데이터소스의 실패 및 복구 여부를 ONS를 통하여 보다 효과적으로 감지할 수 있다. 특히나 Load Balancing 방식일 때는 Runtime Load Balancing Advisory를 이용하여 좀 더 효율적인 Load Balancing을 제공하게 된다.</p> <p>다음과 같은 하위 설정이 있다.</p> <ul style="list-style-type: none"> • Ons Config <p>ONS상의 각 RAC 노드들이 ONS 통신에 사용하는 IP, 포트를 설정한다. 클러스터 데이터소는 설정된 IP, 포트들에 소켓 연결을 맺어 ONS 클라이언트로 동작하게 된다.</p> <p>다음과 같은 형식으로 적는다.</p> <pre>nodes=host1:6200,host2:6200</pre>

DataSourceSelector 인터페이스

Data Source Selector 설정에서 언급한 jeus.jdbc.helper.DataSourceSelector 인터페이스의 명세는 다음과 같다. 컴포넌트 데이터소스 선택에 대한 정책을 직접 정의하려면 이 인터페이스를 구현하고 해당 구현 클래스를 Data Source Selector로서 설정하면 된다.

jeus.jdbc.helper.DataSourceSelector

```
public interface DataSourceSelector {
    public void setComponentDataSourceList(List<String> componentDataSourceList);
    public String selectDataSource();
}
```

메소드	설명
setComponentDataSourceList()	클러스터 데이터소스에 속한 컴포넌트 데이터소스들의 ID 리스트를 설정한다.

메소드	설명
selectDataSource() e()	클러스터 데이터소스에 속한 컴포넌트 데이터소스 선택에 대한 정책 정의를 위해 구현해야 하는 메소드이다. 리턴값은 정의된 정책을 통하여 선택된 데이터소스의 ID가 되어야 한다. 대부분의 환경에서 클러스터 데이터소스로부터 Connection을 얻을 때는 다수의 요청 Thread들이 동시에 접근할 것이므로 정책을 정의할 때는 대체로 동기화에 대한 고려가 필요하며 그것은 구현자의 몫이다.

위의 명세를 바탕으로 다음과 같은 Customize된 DataSourceSelector 구현이 가능하다. 이 DataSourceSelector 구현체는 클러스터 데이터소스에 참여하는 2개의 데이터소스들에 대하여 2대 1의 선택 비율을 정의하고 있다.

DataSourceSelector 구현 예

```
package foo.bar;
import jeus.jdbc.helper.DataSourceSelector
public class MyDataSourceSelector implements DataSourceSelector {
    List<String> componentDataSourceList = new ArrayList<String>();

    // 동기화 보장
    AtomicInteger dataSourceIndex = new AtomicInteger(0);

    public void setComponentDataSourceList(List<String> componentDataSourceList) {
        this.componentDataSourceList.addAll(componentDataSourceList);
    }

    public String selectDataSource() {
        int reminder = (dataSourceIndex.getAndIncrement() & 0x7fffffff) % 3;
        if(reminder < 2) {
            return componentDataSourceList.get(0);
        }
        else {
            return componentDataSourceList.get(1);
        }
    }
}
```

6.5.2. 클러스터 데이터소스에 속한 컴포넌트 데이터소스 설정

클러스터 데이터소스는 자신에 속한 컴포넌트 데이터소스들에 Connection 요청 처리를 위임하는 방식으로 동작한다. Load Balance 설정이 되어있지 않다면 클러스터 데이터소스는 항상 주 컴포넌트 데이터소스로 Connection 요청 처리를 위임한다. 그러나 만약 주 컴포넌트 데이터소스에 이상이 있는 것으로 판단할 경우에는 새로운 컴포넌트 데이터소스를 주 컴포넌트 데이터소스로 삼아 Connection 요청 처리에 문제가 없도록 수습하는데 이것이 바로 클러스터 데이터소스의 Failover다.

Failover 이후 Failback하기 위해서는 최초의 주 컴포넌트 데이터소스가 복구된 것을 확인해야할 필요가 있다. 이는 해당 컴포넌트 데이터소스에 주기적으로 Check Query를 보내 이루어지므로 클러스터 데이터소스에서 Failback 기능을 이용하기 위해서는 클러스터 데이터소스에 속한 모든 컴포넌트 데이터소스들에 대하여 '**Check Query**' 및 '**Check Query Period**' 항목을 설정할 필요가 있다. 만약 Failover만을 사용하기 원한다면 '**Use Failback**' 항목을 false로 설정하면 된다. 수동으로 Failback 명령을 내릴 수 있는데 이에 대해서는 JEUS Reference 안내서의 "control-cluster-data-source"를 참고한다.



컴포넌트 데이터소스의 Destroy Policy On Check Query 설정은 ALL_CONNECTIONS로 하는 것이 좋다. 그렇지 않으면 Failover 이후에도 이상이 감지된 컴포넌트 데이터소스의 Connection Pool에 사용이 불가능한 Connection들이 한동안 남아있게 되기 때문이다. 물론 이들은 주기적인 Connection 점검에 의해서 정리가 되지만 Failover가 이뤄지는 시점에 제거되는 것이 바람직하다.

6.6. 데이터소스 관련 설정 동적 변경

JEUS 7의 주된 특징 중 하나는 일부 설정에 대한 동적 변경의 지원이다. 설정 동적 변경을 지원한다는 것은 변경한 설정을 JEUS 재기동 없이 런타임에 즉시 반영함을 의미한다. 설정 동적 변경에 대한 보다 자세한 내용은 JEUS Domain 안내서의 "도메인 설정변경"을 참고한다.

본 절에서는 예시를 통해 데이터소스와 관련된 설정의 동적 변경 방법에 대해 설명한다. 보다 나은 이해를 위해서는 JEUS 도메인 구조 및 도메인에서의 데이터소스 관리 구조 등에 대한 이해와 데이터소스의 각 설정 항목의 쓰임 등에 대한 숙지가 필요하다. JEUS 도메인 구조에 대해서는 JEUS Domain 안내서의 "구성요소"를 참고한다. 도메인에서의 데이터소스 관리 구조와 데이터소스의 각 설정 항목의 쓰임에 대해서는 각각 [데이터소스 및 Connection Pool 관리](#)와 [데이터소스 설정](#)을 참고한다.

동적 변경은 콘솔 툴을 통해서 수행될 수 있으므로 본 절에서는 두 가지 방법 모두에 대해서 함께 설명한다. 콘솔 툴을 통한 방법을 설명할 때는 편의상 축약된 명령어 및 옵션의 이름을 사용한다. 축약된 명령어 및 옵션의 이름과 명령어의 세부 사용 방법은 콘솔 툴에서 명령어 이름을 인자로 help를 수행하거나 JEUS Reference 안내서의 "데이터소스 관련 명령어" 및 "Connection Pool 제어 및 모니터링 명령어"를 참고하면 확인할 수 있다.

설명을 위해 필요한 몇 가지 사전 작업을 수행한다. 우선 domain1이라는 이름의 도메인과 adminServer라는 이름의 MASTER를 생성한다. MASTER를 기동한 후 각각 server1, server2, server3라는 이름을 갖는 3개의 MS를 domain1에 추가하고 이들 또한 기동한다. 그리고 이들 중 server2와 server3은 cluster1이라는 이름의 클러스터로 묶는다. 도메인 및 클러스터, 서버 관련 설정 방법은 JEUS Domain 안내서의 "도메인 생성" 및 "JEUS 클러스터링", JEUS Server 안내서의 "JEUS 설정"을 참고한다. 이후로는 앞서 언급한 사전 작업이 모두 완료되었음을 가정한다.



각각의 예제는 서로 연관하여 시나리오를 구성해 설명하므로 본 절의 내용은 기술된 순서대로 읽어가기 바란다.

6.6.1. 데이터소스 추가

데이터소스의 동적인 추가가 가능하다. 데이터소스의 추가란 도메인에 데이터소스의 설정을 등록하여 도메인에 속한 서버 및 클러스터에 의해 참조될 수 있는 상태로 만드는 작업이다.

콘솔 툴 사용

콘솔 툴에서 **add-data-source** 명령어를 수행하면 동적으로 데이터소스를 도메인에 추가할 수 있다. add-data-source를 수행하여 데이터소스를 추가하기 위해서는 데이터소스 각 세부 설정에 해당하는 옵션값들을 설정해야 한다. add-data-source에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-data-source"를 참고한다.

다음은 add-data-source를 수행하여 필요한 몇몇 설정들을 입력하고 데이터소스 ID가 ds1인 데이터소스를 domain1에 추가하는 예이다.

```
[MASTER]domain1.adminServer>add-data-source -id ds1 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 192.168.1.165 -pn 1521 -dn ora10g
-user jeustest1 -password jeustest1 -property driverType;java.lang.String;thin
Successfully performed the ADD operation for data source [ds1] to domain.
Check the results using "add-data-source"
```

ds1이 추가된 것은 위의 결과 메시지에서 알 수 있듯이 **add-data-source**를 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>add-data-source
Shows the current configuration
Data sources in domain
=====
+-----+-----+-----+-----+
| ds1   | common data source |
+-----+-----+-----+-----+
=====
```

이후의 예시를 위해 위와 같은 방법으로 데이터소스 ID가 각각 ds2, ds3인 데이터소스 2개를 더 추가한다. 이후로는 ds2, ds3의 domain1으로의 추가가 완료되었음 가정한다.

6.6.2. 서버에 데이터소스 등록

도메인에 데이터소스가 추가되긴 했지만 서버에서 실제로 데이터소스를 참조하여 사용하기 위해서는 데이터소스를 서버에 등록해야 한다. 데이터소스가 서버에 등록되어야 데이터소스 정보가 서버의 JNDI Repository에 bind되고 이후 서버에 deploy된 애플리케이션이 이를 lookup할 수 있다. 이 같은 서버로의 데이터소스 등록 작업은 동적으로 처리 가능하다.

콘솔 툴 사용

콘솔 툴에서 **add-data-sources-to-server** 명령어를 수행하면 동적으로 데이터소스를 서버에 추가할 수 있다. add-data-sources-to-server에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-data-sources-to-server"를 참고한다.

다음은 add-data-sources-to-server를 수행하여 ds1, ds2를 server1에 등록하는 예이다.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1 -ids ds1,ds2
Successfully performed the ADD operation for data sources to the server [server1].
Check the results using "add-data-sources-to-server -server server1"
```

ds1, ds2가 server1에 등록된 것은 위의 결과 메시지를 통해서 알 수 있듯이 '**add-data-sources-to-server -server server1**'을 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1
Shows the current configuration.
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1, ds2 |
+-----+-----+
=====
```

jndilist를 수행하면 server1의 JNDI Repository에 ds1과 ds2가 bind된 것을 확인할 수 있다. ds1과 ds2의 JNDI 이름을 별도로 입력하지는 않았으므로 각각의 데이터소스 ID가 JNDI 이름으로도 사용되고 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

6.6.3. 서버로부터 데이터소스 제거

서버에 등록된 데이터소스는 동적으로 제거될 수도 있다. 데이터소스가 서버로부터 제거될 때에는 데이터소스 정보가 JNDI unbind되며 Connection Pool이 생성되어 있는 경우 Connection Pool은 destroy된다.

콘솔 툴 사용

콘솔 툴에서 **remove-data-sources-from-server**를 수행하면 동적으로 데이터소스를 서버로부터 제거할 수 있다. remove-data-sources-from-server에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-data-sources-from-server"를 참고한다.

다음은 remove-data-sources-from-server를 수행하여 ds2를 server1으로부터 제거하는 예이다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1 -ids ds2
Successfully performed the REMOVE operation for data sources from the server [server1].
Check the results using "remove-data-sources-from-server -server server1"
```

remove-data-sources-from-server -server server1을 수행하면 server1으로부터 ds2가 제거되고 ds1만 남아있는 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1
Shows the current configuration.
```

```
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1 |
+-----+-----+
=====
```

jndilist를 수행하면 server1의 JNDI Repository로부터 ds2가 제거되고 ds1만 bind되어 있는 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

6.6.4. 클러스터에 데이터소스 등록

서버에 데이터소스를 등록했던 것처럼 클러스터에도 데이터소스를 등록할 수 있다.

클러스터에 등록된 데이터소스는 클러스터에 속한 모든 서버들에서 유효하다. 즉, 클러스터에 속한 서버는 클러스터에 등록된 데이터소스를 마치 자신이 그 데이터소스를 등록한 것처럼 사용할 수 있다. 데이터소스는 서버에 동적으로 등록될 수 있었던 것처럼 클러스터에도 동적으로 등록될 수 있다.

콘솔 툴 사용

콘솔 툴에서 **add-data-sources-to-cluster** 명령어를 수행하면 동적으로 데이터소스를 클러스터에 등록할 수 있다. add-data-sources-to-cluster에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-data-sources-to-cluster"를 참고한다.

다음은 add-data-sources-to-cluster를 수행하여 ds2, ds3를 cluster1에 등록하는 예이다.

```
[MASTER]domain1.adminServer>add-data-sources-to-cluster -cluster cluster1 -ids ds2,ds3
Successfully performed the ADD operation for data sources to the cluster [cluster1].
Check the results using "add-data-sources-to-cluster -cluster cluster1"
```

ds2, ds3가 cluster1에 등록된 것은 위의 결과 메시지를 통해서 알 수 있듯이 '**add-data-sources-to-cluster -cluster cluster1**'을 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>add-datas-sources-to-cluster -cluster cluster1
Shows the current configuration.
```

```
The data sources registered in the cluster [cluster1].
```

```
=====
+-----+-----+
| data sources | ds2, ds3 |
+-----+-----+
=====
```

이제 cluster1에 데이터소스 ds2와 ds3가 등록되었으므로 cluster1에 속한 server2와 server3는 ds2와 ds3를 자신이 등록한 것처럼 사용할 수 있다. 이는 곧 server2와 server3의 각각의 JNDI Repository에 ds2, ds3가 JNDI bind되었음을 의미한다.

jndilist 명령어를 수행하여 server2의 JNDI Repository를 확인하면 ds2와 ds3가 JNDI bind되어 있음을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server2
```

```
The JNDI list on the server2
```

```
List of the context /
```

```
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

또한 jndilist를 수행하여 server3의 JNDI Repository를 확인하면 ds2와 ds3가 JNDI bind되어 있음을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server3
```

```
The JNDI list on the server3
```

```
List of the context /
```

```
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

6.6.5. 클러스터로부터 데이터소스 제거

서버로부터 데이터소스를 동적으로 제거했던 것처럼 클러스터로부터 데이터소스를 동적으로 제거할 수 있다.

클러스터에 등록된 데이터소스가 제거되면 그것에 의존하여 데이터소스를 사용하던 클러스터 내의 모든 서버들은 더 이상 그 데이터소스를 사용할 수 없게 된다. 즉, 그러한 서버들은 클러스터로부터 데이터소스가 제거될 때 데이터소스

정보를 JNDI unbind하며 Connection Pool이 생성되어있는 경우 Connection Pool을 destroy한다.

콘솔 툴 사용

콘솔 툴에서 **remove-data-sources-from-cluster** 명령어를 수행하면 동적으로 데이터소스를 클러스터로부터 제거할 수 있다. remove-data-sources-from-cluster에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-data-sources-from-cluster"를 참고한다.

다음은 remove-data-sources-from-cluster를 수행하여 ds2를 cluster1으로부터 제거하는 예이다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1 -ids ds2
Successfully performed the REMOVE operation for data sources from the cluster [cluster1].
Check the results using "remove-data-sources-from-cluster -cluster cluster1"
```

remove-data-sources-from-cluster -cluster cluster1을 수행하면 cluster1으로부터 ds2가 제거되고 ds3만 남아있는 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1
Shows the current configuration.
The data sources registered in the cluster [cluster1].
=====
+-----+-----+
| data sources | ds3 |
+-----+-----+
=====
```

이제 cluster1으로부터 데이터소스 ds2가 제거되었으므로 cluster1에 속한 server2와 server3는 더 이상 ds2를 자신이 등록한 것처럼 사용할 수 없다. 이는 곧 server2와 server3의 각각의 JNDI Repository에서 ds2가 JNDI unbind되었으며 ds2의 Connection Pool이 생성되어 있었던 경우 이것 또한 destroy되었음을 의미한다.

jndilist를 수행하여 server2의 JNDI Repository를 확인하면 server2가 더 이상 ds2를 JNDI bind하고 있지 않음을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

또한 jndilist를 수행하여 server3의 JNDI Repository를 확인하면 server3 역시 더 이상 ds2를 JNDI bind하고 있지 않음을 확인할 수 있다.


```
[MASTER]domain1.adminServer>jndilist -server server3
```

```
The JNDI list on the server3
```

```
List of the context /
```

```
=====
```

Name	Value	Local Binding
ds3	jeus.jdbc.info.JDBCBindInfo	true
JEUSMQ_DLQ	jeus.jms.common.destination.JeusQueue	false
mgmt	jeus.jndi.JNSContext	false

```
=====
```

6.6.6. 클러스터에 서버 추가

클러스터에 동적으로 서버를 추가할 수 있다. 이는 클러스터 및 서버의 설정 변경이지만 클러스터에 서버를 추가함으로 인해 데이터소스 설정도 변경될 수 있으므로 클러스터에 서버가 동적으로 추가되는 경우 데이터소스 설정이 어떻게 영향을 받을 수 있는지 있는지 설명한다.

클러스터에 동적으로 서버가 추가되면 클러스터에 등록된 데이터소스는 클러스터에 추가된 서버에서 유효해진다. 즉, 클러스터에 추가된 서버는 클러스터에 등록된 데이터소스를 마치 자신이 해당 데이터소스를 등록한 것처럼 사용할 수 있다.

반면 서버 자신이 등록한 데이터소스는 더 이상 그 서버에서 유효하지 않게 되므로 서버 자신이 등록한 데이터소스는 JNDI unbind되며 관련 Connection Pool이 생성되어 있는 경우 Connection Pool은 destroy된다.

콘솔 툴 사용

콘솔 툴에서 **add-servers-to-cluster** 명령어를 수행하면 동적으로 서버를 클러스터에 추가할 수 있다. add-servers-to-cluster에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-servers-to-cluster"를 참고한다.

다음은 add-servers-to-cluster를 수행하여 server1을 cluster1에 추가하는 예이다.

```
[MASTER]domain1.adminServer>add-servers-to-cluster cluster1 -servers server1
Successfully performed the ADD operation for The server list for cluster(cluster1)..
Check the results using "list-clusters cluster1 or add-servers-to-cluster cluster1"
```

server1이 cluster1에 추가된 것은 위의 결과 메시지를 통해서 알 수 있듯이 '**list-clusters cluster1**' 또는 '**add-servers-to-cluster cluster1**'을 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>add-servers-to-cluster cluster1
Shows the current configuration.
The server list for cluster(cluster1).
=====
```

List of Servers	server2, server3, server1
-----------------	---------------------------

```
=====
```

=====

이제 server1이 cluster1에 추가되었으므로 server1은 cluster1에 등록된 ds3를 자신이 등록한 것처럼 사용할 수 있다. 이는 곧 server3의 JNDI Repository에 ds3이 JNDI bind 되었음을 의미한다. 또한 server1 자체적으로 등록하고 있던 ds1 역시 여전히 server1에서 유효하다.

jndilist를 수행하여 server1의 JNDI Repository를 확인하면 ds1과 더불어 ds3가 JNDI bind된 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

6.6.7. 클러스터로부터 서버 삭제

클러스터로부터 동적으로 서버를 삭제할 수 있다. 클러스터에 동적으로 서버를 추가하는 것처럼 클러스터로부터 동적으로 서버를 삭제하는 것 또한 클러스터 및 서버의 설정 변경이지만 클러스터로부터 서버를 삭제함으로써 인해 데이터소스 설정도 변경될 수 있으므로 클러스터로부터 서버가 동적으로 삭제되는 경우 데이터소스 설정이 어떻게 영향을 받을 수 있는지 설명한다.

클러스터로부터 동적으로 서버가 삭제되면 클러스터에 등록된 데이터소스는 클러스터로부터 삭제된 서버에서 더 이상 유효하지 않다. 즉, 클러스터로부터 삭제된 서버는 클러스터에 등록된 모든 데이터소스 정보를 JNDI unbind하며 Connection Pool이 생성되어있는 경우 Connection Pool을 Destroy한다.

반면 서버 자신이 등록한 데이터소스는 다시 그 서버에서 유효해지므로 서버 자신이 등록한 데이터소스는 JNDI bind되며 Connection Pool을 생성하여 사용할 수 있게 된다.

콘솔 툴 사용

콘솔 툴에서 **remove-servers-from-cluster** 명령어를 수행하면 동적으로 서버를 클러스터로부터 삭제할 수 있다. remove-servers-from-cluster에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-servers-from-cluster"를 참고한다.

다음은 remove-servers-from-cluster를 수행하여 server3을 cluster1으로부터 삭제하는 예이다.

```
[MASTER]domain1.adminServer>remove-servers-from-cluster cluster1 -servers server3
Successfully performed the REMOVE operation for The server list for cluster(cluster1)..
Check the results using "list-clusters cluster1 or remove-servers-from-cluster cluster1"
```

server3이 cluster1로부터 삭제된 것은 위의 결과 메시지를 통해서 알 수 있듯이 'list-clusters cluster1' 또는 'remove-servers-from-cluster cluster1'을 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-servers-from-cluster cluster1
Shows the current configuration.
The server list for cluster(cluster1).
=====
+-----+-----+
| List of Servers | server2, server1 |
+-----+-----+
=====
```

이제 server3가 cluster1으로부터 삭제되었으므로 server3은 더 이상 cluster1에 등록된 ds3를 자신이 등록한 것처럼 사용할 수 없다. 이는 곧 server3의 JNDI Repository에서 ds3가 JNDI unbind되었으며 ds3의 Connection Pool이 생성되어 있었던 경우 이것 또한 destroy되었음을 의미한다.

jndilist를 수행하여 server3의 JNDI Repository를 확인하면 ds3가 사라진 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server3
The JNDI list on the server3
List of the context /
=====
+-----+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+-----+
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+-----+
=====
```

6.6.8. 클러스터 삭제

도메인으로부터 클러스터를 동적으로 삭제할 수 있다. 이는 도메인 및 클러스터의 설정 변경이지만 도메인으로부터 클러스터를 삭제함으로 인해 데이터소스 설정도 변경될 수 있으므로 도메인으로부터 클러스터가 동적으로 삭제되는 경우 데이터소스 설정이 어떻게 영향을 받을 수 있는지 설명한다.

클러스터가 도메인으로부터 삭제되면 클러스터에 등록되어 있던 모든 데이터소스들은 클러스터에 속해있던 서버들에서 더 이상 유효하지 않다. 즉, 삭제된 클러스터에 속했었던 서버들은 클러스터에 등록되어 있던 모든 데이터소스 정보를 JNDI unbind하며 Connection Pool이 생성되어있는 경우 Connection Pool을 destroy한다. 반면 서버 자신이 등록한 데이터소스는 다시 해당 서버에서 유효해지므로 서버 자신이 등록한 데이터소스는 JNDI bind되며 Connection Pool을 생성하여 사용할 수 있게 된다.

콘솔 툴 사용

콘솔 툴에서 **remove-cluster**를 수행하면 동적으로 클러스터를 도메인으로부터 삭제할 수 있다. remove-cluster에 대한 보다 자세한 사용법은 JEUS Reference 터를 도메인으로부터 삭제할 수 있다. remove-cluster에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-cluster"를 참고한다.

다음은 remove-cluster를 수행하여 cluster1을 domain1으로부터 삭제하는 예이다.

```
[MASTER]domain1.adminServer>remove-cluster cluster1
Successfully performed the REMOVE operation for cluster (cluster1).
Check the results using "list-clusters or remove-cluster"
```

cluster1이 domain1으로부터 삭제된 것은 위의 결과 메시지를 통해서 알 수 있듯이 **list-clusters** 또는 **remove-cluster**를 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-cluster
Shows the current configuration.
cluster list
=====
+-----+-----+
| List of Clusters | empty |
+-----+-----+
=====
```

이제 cluster1이 domain1으로부터 삭제되었으므로 cluster1에 속해있던 server1과 server2는 더 이상 cluster1에 등록되어 있던 ds3를 자신이 등록한 것처럼 사용할 수 없다. 이는 곧 server1과 server2 각각의 JNDI Repository에서 ds3가 JNDI unbind되었으며 ds3의 Connection Pool이 생성되어 있었던 경우 이것 또한 destroy되었음을 의미한다.

jndilist를 수행하여 server1의 JNDI Repository를 확인하면 ds3는 JNDI unbind되고 자신이 등록한 ds1만 JNDI bind되어 있는 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

또한 jndilist를 수행하여 server2의 JNDI Repository를 확인하면 어떤 데이터소스도 JNDI bind되어 있지 않음을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

6.6.9. 데이터소스 삭제

도메인으로부터 동적으로 데이터소스를 삭제할 수 있다. 데이터소스가 도메인으로부터 삭제되면 삭제된 데이터소스는 그것을 사용하던 서버와 클러스터에서 더 이상 유효하지 않다. 즉 삭제된 데이터소스를 사용하던 서버들은 그 데이터소스 정보를 JNDI unbind하며 Connection Pool이 생성되어있는 경우 Connection Pool을 destroy한다.

콘솔 툴 사용

콘솔 툴에서 **remove-data-source**를 수행하면 동적으로 데이터소스를 도메인으로부터 삭제할 수 있다. remove-data-source에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-data-source"를 참고한다.

다음은 remove-data-source를 수행하여 ds1을 domain1으로부터 삭제하는 예이다.

```
[MASTER]domain1.adminServer>remove-data-source -id ds1
Successfully performed the REMOVE operation for data source [ds1] from the domain.
=====
+-----+-----+-----+
| resources.dataSource.database          | MODIFY | ACTIVATED |
| servers.server.{? name == 'server1' }.dataSources | MODIFY | ACTIVATED |
+-----+-----+-----+
=====
Check the results using "remove-data-source"
```

위의 결과를 보면 remove-data-source를 수행하여 ds1을 삭제할 때 ds1을 등록하고 있던 server1에도 어떠한 변화가 생겼음을 짐작할 수 있다. 위의 결과는 ds1이 domain1에서 삭제되는 동시에 server1에 등록되었던 ds1 정보도 제거됨을 표시하고 있다. 이와 같이 동적으로 설정을 변경하는 명령어를 수행하면 그와 연관되어 발생하는 모든 설정 변경의 런타임 반영 여부를 확인할 수 있다. 동적 설정 변경 명령어의 보다 자세한 사용법은 JEUS Domain 안내서의 "도메인 설정변경"을 참고한다.

ds1이 삭제된 것은 위의 결과 메시지를 통해서 알 수 있듯이 **remove-data-source**를 아무 옵션 없이 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-data-source
Shows the current configuration.
Data sources in domain
=====
+-----+-----+-----+
| ds2 | common data source          |
| ds3 | common data source          |
+-----+-----+-----+
=====
```

이제 ds1이 domain1으로부터 삭제되었으므로 server1은 더 이상 ds1을 사용할 수 없다. 이는 곧 server1의 JNDI Repository에서 ds1이 JNDI unbind되었으며 ds1의 Connection Pool이 생성되어 있었던 경우 이것 또한

destroy되었음을 의미한다.

jndilist를 수행하여 server1의 JNDI Repository를 확인하면 ds1이 사라진 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
```

6.6.10. 데이터소스 설정 변경

지금까지는 데이터소스, 서버, 클러스터를 추가/삭제할 때 데이터소스의 관리가 어떻게 이루어지는지 살펴보았다. 본 절에서는 데이터소스 자체의 설정 변경에 대해서 설명한다.

데이터소스 설정에서 살펴봤던 것처럼 데이터소스의 설정은 크게 드라이버 Setup 등에 필요한 기본 설정과 Connection Pool 설정으로 이루어져있다. 이 중 드라이버 Setup 설정은 동적 변경이 불가능하지만 그 외의 기본 설정 몇 가지와 Connection Pool 설정의 대부분은 동적 변경이 가능하다. 동적 변경이 불가능한 설정은 변경을 시도하는 경우 변경 사항이 설정 파일에는 기록되나 런타임에 반영되지는 않는다. 이는 곧 서버를 다시 기동해야 변경 사항을 런타임에 반영할 수 있음을 의미한다. 반면 동적 변경이 가능한 설정은 변경을 시도할 때 변경 사항이 설정 파일에도 기록되며 런타임에도 즉시 반영된다.

동적 변경이 가능한 데이터소스 설정에는 여러 가지가 있는데 그 중 비교적 동적 변경 빈도가 높을 것으로 예상되는 Connection Pool의 Connection 최솟값, 최댓값 설정을 예로 설명한다. 변경 대상이 되는 데이터소스는 ds2로 하며 실제로 Connection Pool의 Connection 최솟값, 최댓값 설정 변경이 동적으로 런타임에 반영되었는지 확인하기 위해 ds2를 server1에 등록하고 Connection Pool을 생성한다. 데이터소스를 서버에 등록하는 방법은 앞서 알아보았으므로 여기에서는 실제 ds2 Connection Pool을 server1에 생성하는 방법부터 설명한다.

콘솔 툴 사용

Connection Pool은 콘솔 툴에서 **create-connection-pool**을 수행하여 생성할 수 있다. create-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "create-connection-pool"을 참고한다.

다음은 create-connection-pool을 수행하여 ds2의 Connection Pool을 생성하는 예이다. ds2의 Connection Pool이 server1에 생성되었음을 결과 메시지에서부터 알 수 있다.

```
[MASTER]domain1.adminServer>create-connection-pool -id ds2
Servers that successfully created a connection pool : server1
Servers that failed to create a connection pool : none.
```

connection-pool-info를 수행하면 Connection Pool의 런타임 Connection 최솟값, 최댓값을 확인할 수 있다. 이 명령어에 대한 보다 자세한 설명은 JEUS Reference 안내서의 "connection-pool-info"를 참고한다.

다음은 connection-pool-info를 수행하여 server1에 생성된 ds2 Connection Pool의 런타임 Connection 최솟값, 최댓값을 확인하는 예이다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1
The connection pool information on the server [server1].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID|     |     |        |     |        |      |        |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds2    | 2   | 30  | 0       | 0   | 0.0   | 2    | 0      | 2   | false| true |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
* : has not been created, total = active + idle + disposable
=====
```

ds2 Connection Pool의 런타임 Connection 최솟값이 2, 최댓값이 30으로 설정되어 있는 것을 확인할 수 있다. 이는 ds2를 처음 도메인에 추가할 때 별도로 Connection 최솟값, 최댓값을 설정하지 않아 디폴트로 적용된 값이다.

다음은 각각 2, 30으로 설정되어 있는 ds2의 Connection 최솟값과 최댓값을 변경하는 설명이다.

modify-data-source를 수행하면 동적으로 데이터소스 설정을 변경할 수 있다. help를 수행하거나 JEUS Reference 안내서의 "modify-data-source"를 참고하면 modify-data-source를 통하여 변경할 수 있는 모든 설정들을 옵션들의 이름으로 유추하여 확인할 수 있다. 동적 변경이 가능한 설정과 관련된 옵션에는 태그를 붙여 두었으므로 이를 통해 어떤 설정이 동적으로 변경 가능한지도 알 수 있다.

다음은 modify-data-source를 수행하여 ds2의 Connection 최솟값과 최댓값을 각각 10, 50으로 변경하는 예이다.

```
[MASTER]domain1.adminServer>modify-data-source -id ds2 -min 10 -max 50
Successfully performed the MODIFY operation for configuration of the data source [ds2].
Check the results using "modify-data-source -id ds2"
```

modify-data-source를 수행하여 변경된 ds2의 설정은 위의 결과 메시지를 통해서 알 수 있듯이 'modify-data-source -id ds2'를 수행하여 확인할 수 있다. min, max가 각각 10, 50으로 설정되어있는 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>modify-data-source -id ds2
Shows the current configuration.
configuration of the data source [ds2]
=====
+-----+-----+
| id          | ds2      |
| export-name | ds2      |
| data-source-class-name | oracle.jdbc.pool.OracleConnectionPoolDataSource |
| data-source-type | ConnectionPoolDataSource |
| server-name  | 192.168.1.165 |
| port-number  | 1521      |
| database-name | ora10g    |
| user        | jeustest1  |
| password     | jeustest1  |
| login-timeout | 0         |
+-----+-----+
```

auto-commit	DRIVER	
stmt-query-timeout	0	
pool-destroy-timeout	10000	
property	driverType;java.lang.String;thin	
support-xa-emulation	false	
min	10	
max	50	
step	1	
period	3600000	
enable-wait	false	
wait-time	10000	
max-use-count	0	
dbaTimeout	-1	
stmt-caching-size	-1	
stmt-fetch-size	-1	
connection-trace	false	
get-connection-trace	true	
auto-commit-trace	false	
use-sql-trace	false	
keep-connection-handle-open	false	
+-----+-----+-----+		
=====		

다음은 다시 **connection-pool-info**를 수행하여 server1에 생성된 ds2 Connection Pool의 런타임 Connection 최솟값, 최댓값을 확인하는 예이다. ds2의 런타임 Connection 최솟값, 최댓값이 각각 10, 50으로 변경된 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1
The connection pool information on the server [server1].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID |     |     |         |     |         |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds2    | 10 | 50 | 0      | 0   | 0.0   | 10  | 0      | 10 | fal | true |
|         |    |    |         |     |        |     |        |   | se  |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
* : has not been created, total = active + idle + disposable
=====
```

6.6.11. 데이터소스 설정 확인

JEUS 7부터는 설정 파일을 열어 사용자가 직접 수정한 설정에 대해서 책임지지 않는다. 설정 파일을 열어서 설정을 확인하는 것은 그러한 변경과는 관련이 없으나 되도록이면 설정 파일에 직접 접근하는 것을 권장하지 않는다. 이를 위해 데이터소스의 설정을 명령어를 통해 확인할 수 있도록 지원한다.

본 절에서는 콘솔 툴을 사용해서 데이터소스의 설정을 확인하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **list-data-sources** 명령어를 수행하면 도메인에 존재하는 모든 데이터소스의 목록을 확인할 수 있다.

다음은 list-data-sources의 수행하여 도메인에 존재하는 모든 데이터소스 목록을 확인하는 예이다.

```
[MASTER]domain1.adminServer>list-data-sources
The list of data sources
=====
+-----+-----+-----+
| Data Source ID | JNDI Export Name | Data Source Type |
+-----+-----+-----+
| ds2            | ds2              | ConnectionPoolDataSource |
| ds3            | ds3              | ConnectionPoolDataSource |
+-----+-----+-----+
=====
```

특정 데이터소스의 ID를 옵션값으로 입력하면 해당 데이터소스의 전체 설정을 상세히 조회한다.

다음은 list-data-sources를 수행하여 ds2의 상세 설정을 확인하는 예이다.

```
[MASTER]domain1.adminServer>list-data-sources -id ds2
The configuration of the data source [ds2]
=====
+-----+-----+
| Configuration name | Configuration value |
+-----+-----+
| id                 | ds2                  |
| export-name        | ds2                  |
| data-source-class-name | oracle.jdbc.pool.OracleConnectionPoolDataSource |
| data-source-type    | ConnectionPoolDataSource |
| server-name        | 192.168.1.165        |
| port-number        | 1521                  |
| database-name       | ora10g               |
| user               | jeustest1             |
| password            | jeustest1             |
| login-timeout       | 0                     |
| auto-commit         | DRIVER                |
| stmt-query-timeout  | 0                     |
| pool-destroy-timeout | 10000                 |
| property            | [driverType;java.lang.String;thin] |
| support-xa-emulation | false                 |
| min                 | 10                    |
| max                 | 50                    |
| step                | 1                     |
| period              | 3600000               |
| enable-wait         | false                 |
| wait-time           | 10000                 |
| max-use-count        | 0                     |
| dbaTimeout          | -1                    |
| stmt-caching-size   | -1                    |
| stmt-fetch-size     | -1                    |
| connection-trace    | false                 |
| get-connection-trace | true                  |
| auto-commit-trace   | false                 |
| use-sql-trace        | false                 |
| keep-connection-handle-open | false                 |
+-----+-----+
=====
```

6.7. 클러스터 데이터소스 설정 동적 변경

본 절에서는 예시를 통해 클러스터 데이터소스와 관련된 설정의 동적 변경 방법에 대해 설명한다. 보다 나은 이해를 위해서는 JEUS 도메인 구조 및 도메인에서의 데이터소스 관리 구조 등에 대한 이해와 클러스터 데이터소스의 각 설정 항목의 사용 등에 대한 숙지가 필요하다.

JEUS 도메인 구조에 대해서는 JEUS Domain 안내서의 "구성요소"를 참고한다. 도메인에서의 데이터소스 관리 구조와 클러스터 데이터소스의 각 설정 항목의 쓰임에 대해서는 각각 [데이터소스](#) 및 [Connection Pool 관리](#)와 [클러스터 데이터소스 설정](#)을 참고한다.

동적 변경은 콘솔 툴을 통해서 수행될 수 있으므로 본 절에서는 두 가지 방법을 설명한다. 콘솔 툴을 통한 방법을 설명할 때는 편의상 축약된 명령어 및 옵션의 이름을 사용한다. 축약된 명령어 및 옵션의 이름과 명령어의 세부 사용 방법은 콘솔 툴에서 명령어 이름을 인자로 help를 수행하거나 JEUS Reference 안내서의 "Connection Pool 제어 및 모니터링 명령어"를 참고하면 확인할 수 있다.

설명을 위해 필요한 몇 가지 사전 작업을 수행한다.

우선 domain1이라는 이름의 도메인과 adminServer라는 이름의 Master를 생성한다. Master를 기동한 후 각각 server1, server2, server3라는 이름을 갖는 3개의 MS를 domain1에 추가하고 기동한다. 그리고 server2와 server3는 cluster1이라는 이름의 클러스터로 묶는다. 도메인 및 클러스터, 서버 관련 설정 방법은 JEUS Domain 안내서의 "도메인 생성" 및 "JEUS 클러스터링", JEUS Server 안내서의 "JEUS 설정"을 참고한다. 이후로는 앞서 언급한 사전 작업이 모두 완료되었음을 가정한다.



각각의 예시를 서로 연관하여 시나리오를 구성해 설명하므로 본 절의 내용은 기술된 순서대로 읽어가기 바란다.

6.7.1. 클러스터 데이터소스 추가

일반 데이터소스처럼 클러스터 데이터소스도 동적인 추가가 가능하다. 클러스터 데이터소스를 추가할 때 항상 고려해야 할 것은 클러스터 데이터소스에 묶이는 컴포넌트 데이터소스들도 함께 추가해야 한다는 사실이다.

본 절에서는 데이터소스 ID가 각각 ds1, ds2인 데이터소스들을 컴포넌트 데이터소스로 갖는 클러스터 데이터소스를 domain1에 추가하는 예를 설명한다. 클러스터 데이터소스의 데이터소스 ID는 cds1이다.

콘솔 툴 사용

cds1의 컴포넌트 데이터소스로 동작할 ds1과 ds2를 domain1에 추가해야 한다.

다음은 **add-data-source**를 수행하여 ds1과 ds2를 domain1에 추가하는 예이다. add-data-source에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-data-source"를 참고한다.

```
[MASTER]domain1.adminServer>add-data-source -id ds1 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 192.168.1.165 -pn 1521 -dn ora10g -user jeustest1
-password jeustest1 -property driverType;java.lang.String;thin
Successfully performed the ADD operation for data source [ds1] to domain.
Check the results using "add-data-source"
```

```
[MASTER]domain1.adminServer>add-data-source -id ds2 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 192.168.1.165 -pn 1521 -dn ora10g -user jeustest1
-password jeustest1 -property driverType;java.lang.String;thin
Successfully performed the ADD operation for data source [ds2] to domain.
Check the results using "add-data-source"
```

ds1과 ds2를 domain1에 추가했으므로 이제 cds1을 domain1에 추가한다. **add-cluster-data-source**를 수행하면 동적으로 클러스터 데이터소스를 도메인에 추가할 수 있다. add-cluster-data-source에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-cluster-data-source"를 참고한다. add-cluster-data-source를 수행하여 데이터소스를 추가하기 위해서는 클러스터 데이터소스 각 세부 설정에 해당하는 옵션값들을 명시해야 한다.

다음은 **add-cluster-data-source**를 수행하여 필요한 몇몇 설정들을 입력하고 클러스터 데이터소스 ID가 cds1인 데이터소스를 domain1에 추가하는 예이다.

```
[MASTER]domain1.adminServer>add-cluster-data-source -id cds1 -cds ds1,ds2
Successfully performed the ADD operation for cluster data source [cds1] to domain.
Check the results using "add-cluster-data-source"
```

cds1이 추가된 것은 위의 결과 메시지를 통해서 알 수 있듯이 **add-cluster-data-source**를 아무 옵션 없이 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>add-cluster-data-source
Shows the current configuration.
Data sources in domain
=====
+-----+-----+-----+
| ds1 | common data source |
| ds2 | common data source |
| cds1 | cluster data source |
+-----+-----+-----+
=====
```

6.7.2. 서버에 클러스터 데이터소스 등록

클러스터 데이터소스도 일반 데이터소스처럼 서버에 등록되어야 사용될 수 있다. 클러스터 데이터소스가 서버에 등록되어야 비로소 클러스터 데이터소스 정보가 서버의 JNDI Repository에 bind되고 이후 서버에 deploy된 애플리케이션이 이를 lookup할 수 있다. 이때 클러스터 데이터소스에 묶인 컴포넌트 데이터소스들 역시 서버에 등록해야 한다는 것을 주의한다.

실제 Connection Pool 서비스를 제공하는 것은 각각의 컴포넌트 데이터소스이며 이들의 Connection Pool은 결국 서버에 생성되므로 클러스터 데이터소스를 등록하는 서버 그 클러스터 데이터소스에 묶인 컴포넌트 데이터소스들을 함께 등록해야 원하는대로 클러스터 데이터소스를 사용할 수 있다.

본 절에서는 cds1을 server1에 등록하는 방법에 대해 설명한다.

콘솔 툴 사용

cds1의 컴포넌트 데이터소스인 ds1과 ds2도 server1에 함께 등록해야 한다.

다음은 **add-data-sources-to-server**를 수행하여 ds1, ds2, cds1을 server1에 등록하는 예이다. add-data-sources-to-server에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-data-sources-to-server"를 참고한다.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1 -ids ds1,ds2,cds1
Successfully performed the ADD operation for data sources to the server [server1].
Check the results using "add-data-sources-to-server -server server1"
```

ds1, ds2, cds1이 server1에 등록된 것은 위의 결과 메시지를 통해서 알 수 있듯이 '**add-data-sources-to-server -server server1**'을 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1
Shows the current configuration.
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1, ds2, cds1 |
+-----+-----+
```

jndilist를 수행하면 server1의 JNDI Repository에 ds1, ds2, cds1이 bind된 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+-----+
| cds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+-----+
```

6.7.3. 서버로부터 클러스터 데이터소스 제거

서버에 등록된 클러스터 데이터소스도 일반 데이터소스처럼 서버로부터 동적으로 제거될 수 있다.

일반 데이터소스와 마찬가지로 클러스터 데이터소스가 서버로부터 제거될 때에는 클러스터 데이터소스 정보가 JNDI unbind된다. 클러스터 데이터소스는 자신의 Connection Pool을 가지지 않고 실제 Connection Pool은 클러스터 데이터소스에 속한 컴포넌트 데이터소스들로부터 생성되므로 클러스터 데이터소스가 서버로부터 제거될 때에는 일반 데이터소스들과는 다르게 Connection Pool의 destroy는 일어나지 않는다. 클러스터 데이터소스에 속했던

컴포넌트 데이터소스들의 Connection Pool이 destroy될 것으로 생각할 수 있으나 컴포넌트 데이터소스 각각은 또한 독립적인 데이터소스로도 기능할 필요가 있을 수도 있으므로 이들에 대한 JNDI unbind나 destroy는 일어나지 않는다.

콘솔 툴 사용

콘솔 툴에서 **remove-data-sources-from-server**를 수행하면 동적으로 클러스터 데이터소스를 서버로부터 제거할 수 있다. remove-data-sources-from-server에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-data-sources-from-server"를 참고한다.

다음은 remove-data-sources-from-server를 수행하여 cds1을 server1으로부터 제거하는 예이다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1 -ids cds1
Successfully performed the REMOVE operation for data sources from the server [server1].
Check the results using "remove-data-sources-from-server -server server1"
```

cds1이 server1으로부터 제거된 것은 위의 결과 메시지를 통해서 알 수 있듯이 '**remove-data-sources-from-server -server server1**'을 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1
Shows the current configuration.
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1, ds2 |
+-----+-----+
=====
```

cds1이 제거되고 ds1과 ds2는 여전히 등록되어 있는 것을 확인할 수 있다.

jndilist를 수행하면 server1의 JNDI Repository로부터 cds1이 unbind된 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

6.7.4. 클러스터에 클러스터 데이터소스 등록

클러스터 데이터소스도 일반 데이터소스들처럼 동적으로 클러스터에 등록될 수 있다. 클러스터에 등록된 클러스터 데이터소스는 클러스터에 속한 모든 서버들에서 유효하다. 즉, 클러스터에 속한 서버는 클러스터에 등록된 클러스터 데이터소스를 마치 자신이 그 클러스터 데이터소스를 등록한 것처럼 사용할 수 있다. 클러스터 데이터소스를 서버에 등록할 때처럼 클러스터 데이터소스를 클러스터에 등록할 때에도 클러스터 데이터소스에 묶인 컴포넌트 데이터소스들을 클러스터에 함께 등록해야 한다.

본 절에서는 cds1을 cluster1에 등록하는 과정을 설명한다.

콘솔 툴 사용

cds1의 컴포넌트 데이터소스인 ds1과 ds2도 cluster1에 함께 등록해야 한다.

다음은 콘솔 툴에서 **add-data-sources-to-cluster**를 수행하여 ds1, ds2, cds1을 cluster1에 등록하는 예이다. add-data-sources-to-cluster에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "add-data-sources-to-cluster"를 참고한다.

```
[MASTER]domain1.adminServer>add-data-sources-to-cluster -cluster cluster1 -ids ds1,ds2,cds1
Successfully performed the ADD operation for data sources to the cluster [cluster1].
Check the results using "add-data-sources-to-cluster -cluster cluster1"
```

ds1, ds2, cds1이 cluster1에 등록된 것은 위의 결과 메시지를 통해서 알 수 있듯이 '**add-data-sources-to-cluster -cluster cluster1**'을 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>add-data-sources-to-cluster -cluster cluster1
Shows the current configuration.
The data sources registered in the cluster [cluster1].
=====
+-----+-----+
| data sources | ds1, ds2, cds1 |
+-----+-----+
=====
```

이제 cluster1에 ds1, ds2, cds1이 등록되었으므로 cluster1에 속한 server2와 server3는 ds1, ds2, cds1을 자신이 등록한 것처럼 사용할 수 있다.

jndilist를 수행하여 server2의 JNDI Repository를 확인하면 ds1, ds2, cds1이 JNDI bind된 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| cds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
```

mgmt	jeus.jndi.JNSContext	false	
+-----+	+-----+	+-----+	+
=====			

6.7.5. 클러스터로부터 클러스터 데이터소스 제거

클러스터에 등록된 클러스터 데이터소스도 일반 데이터소스처럼 클러스터로부터 동적으로 제거될 수 있다. 클러스터에 등록된 클러스터 데이터소스가 제거되면 그것에 의존하여 클러스터 데이터소스를 사용하던 클러스터 내의 모든 서버들은 더 이상 그 클러스터 데이터소스를 사용할 수 없게 된다. 즉, 그러한 서버들은 클러스터로부터 클러스터 데이터소스가 제거될 때 클러스터 데이터소스 정보를 JNDI unbind한다. 그러나 클러스터 데이터소스를 서버로부터 제거할 때와 마찬가지로 컴포넌트 데이터소스들의 Connection Pool destroy는 일어나지 않는다.

콘솔 툴 사용

콘솔 툴에서 **remove-data-sources-from-cluster**를 수행하면 동적으로 클러스터 데이터소스를 클러스터로부터 제거할 수 있다. remove-data-sources-from-cluster의 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-data-sources-from-cluster"를 참고한다.

다음은 remove-data-sources-from-cluster를 수행하여 cds1을 cluster1으로부터 제거하는 예이다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1 -ids cds1
Successfully performed the REMOVE operation for data sources from the cluster [cluster1].
Check the results using "remove-data-sources-from-cluster -cluster cluster1"
```

remove-data-sources-from-cluster -cluster cluster1을 수행하면 cluster1으로부터 cds1가 제거되고 ds1, ds2는 남아있는 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1
Shows the current configuration.
The data sources registered in the cluster [cluster1].
=====
+-----+-----+
| data sources | ds1, ds2 |
+-----+-----+
=====
```

이제 cluster1으로부터 클러스터 데이터소스 cds1이 제거되었으므로 cluster1에 속한 server2와 server3는 더 이상 cds1을 자신이 등록한 것처럼 사용할 수 없다. 이는 곧 server2와 server3의 JNDI Repository에서 cds1이 JNDI unbind 되었음을 의미한다.

jndilist를 수행하면 server2의 JNDI Repository로부터 cds1이 JNDI unbind된 것을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+-----+
| | | | |
```

Name	Value	Local Binding
ds1	jeus.jdbc.info.JDBCBindInfo	true
ds2	jeus.jdbc.info.JDBCBindInfo	true
JEUSMQ_DLQ	jeus.jms.common.destination.JeusQueue	false
mgmt	jeus.jndi.JNSContext	false

6.7.6. 클러스터에 서버 추가

클러스터에 동적으로 서버가 추가되면 클러스터에 등록된 클러스터 데이터소스도 클러스터에 등록된 일반 데이터소스들과 마찬가지로 클러스터에 추가된 서버들에서 유효해진다. 즉, 클러스터에 추가된 서버는 클러스터에 등록된 클러스터 데이터소스를 마치 자신이 그 클러스터 데이터소스를 등록한 것처럼 사용할 수 있다.

전체적인 메커니즘은 [클러스터에 서버 추가](#)에서의 설명과 유사하므로 별도로 설명하지 않는다. 자세한 내용은 해당 절을 참고한다.

6.7.7. 클러스터로부터 서버 삭제

클러스터로부터 동적으로 서버가 삭제되면 클러스터에 등록된 클러스터 데이터소스는 클러스터에 등록된 일반 데이터소스들과 마찬가지로 클러스터로부터 삭제된 서버에서 더 이상 유효하지 않다. 즉, 클러스터로부터 삭제된 서버는 클러스터에 등록된 모든 클러스터 데이터소스 정보를 JNDI unbind한다.

전체적인 메커니즘은 [클러스터로부터 서버 삭제](#)에서의 설명과 유사하므로 별도로 설명하지 않는다. 자세한 내용은 해당 절을 참고한다.

6.7.8. 클러스터 삭제

클러스터가 동적으로 도메인에서 삭제되면 클러스터에 등록되어 있던 클러스터 데이터소스들은 클러스터에 등록되어 있던 일반 데이터소스들과 마찬가지로 클러스터에 속해있던 서버들에서 더 이상 유효하지 않다. 즉, 삭제된 클러스터에 속해있던 서버들은 클러스터에 등록되어 있던 모든 클러스터 데이터소스 정보를 JNDI unbind한다.

전체적인 메커니즘은 [클러스터 삭제](#)에서의 설명과 유사하므로 별도로 설명하지 않으므로 해당 절을 참고한다.

6.7.9. 클러스터 데이터소스 삭제

클러스터 데이터소스도 일반 데이터소스처럼 동적으로 도메인으로부터 삭제될 수 있다. 클러스터 데이터소스가 도메인으로부터 삭제되면 삭제된 클러스터 데이터소스는 그것을 사용하던 서버와 클러스터에서 더 이상 유효하지 않다. 즉, 삭제된 클러스터 데이터소스를 사용하던 서버들은 그 데이터소스 정보를 JNDI unbind한다. 클러스터 데이터소스가 삭제될 때 서버 및 클러스터에 등록된 클러스터 데이터소스가 어떻게 처리되는지 확인하기 위해 cds1을 server1과 cluster1에 등록한다. 이후로는 server1과 cluster1에 cds1이 등록된 것을 가정한다.

본 절에서는 cds1을 domain1으로부터 삭제하는 과정을 설명한다.

콘솔 툴 사용

콘솔 툴에서 **remove-cluster-data-source**를 수행하면 동적으로 클러스터 데이터소스를 도메인으로부터 삭제할 수 있다. remove-cluster-data-source의 보다 자세한 사용법은 JEUS Reference 안내서의 "remove-cluster-data-source"를 참고한다.

다음은 remove-cluster-data-source를 수행하여 cds1을 domain1으로부터 삭제하는 예이다.

```
[MASTER]domain1.adminServer>remove-cluster-data-source -id cds1
Successfully performed the REMOVE operation for cluster data source [cds1] from domain.
=====
+-----+-----+-----+
| resources.dataSource.clusterDs          | MODIFY | ACTIVATED |
| servers.server.{? name == 'server1' }.dataSources | MODIFY | ACTIVATED |
| clusters.cluster.{? name == 'cluster1' }.dataSources | MODIFY | ACTIVATED |
+-----+-----+-----+
=====
Check the results using "remove-cluster-data-source"
```

위의 결과를 보면 remove-cluster-data-source를 수행하여 cds1를 삭제할 때 cds1이 등록되어 있던 server1과 cluster1에도 어떠한 변화가 생겼음을 짐작할 수 있다. 위의 결과는 cds1이 domain1에서 삭제되는 동시에 server1과 cluster1에 등록되었던 cds1 정보가 제거됨을 표시하고 있다.

이와 같이 동적으로 설정을 변경하는 명령어를 수행하면 그와 연관되어 발생하는 모든 설정 변경의 런타임 반영 여부를 확인할 수 있다. 동적 설정 변경 명령어의 보다 자세한 사용법은 JEUS Domain 안내서의 "도메인 설정변경"을 참고한다.

cds1이 삭제된 것은 위의 결과 메시지를 통해서 알 수 있듯이 **remove-cluster-data-source**를 수행하여 확인할 수 있다.

```
[MASTER]domain1.adminServer>remove-cluster-data-source
Shows the current configuration.
Data sources in domain
=====
+-----+-----+-----+
| ds1 | common data source |
| ds2 | common data source |
+-----+-----+-----+
=====
```

이제 cds1이 domain1으로부터 삭제되었으므로 스스로 cds1을 등록하여 사용하던 server1과 cluster1에 속하여 cluster1에 등록되어 있던 cds1를 사용할 수 있었던 server2와 server3는 더 이상 cds1을 사용할 수 없다. 이는 곧 server1, server2, server3 각각의 JNDI Repository로부 cds1이 JNDI unbind되었음을 의미한다.

jndilist를 수행하여 server1의 JNDI Repository를 확인하면 cds1은 더 이상 JNDI bind되어 있지 않음을 확인할 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
```

Name	Value	Local Binding
ds1	jeus.jdbc.info.JDBCBindInfo	true
ds2	jeus.jdbc.info.JDBCBindInfo	true
JEUSMQ_DLQ	jeus.jms.common.destination.JeusQueue	false
mgmt	jeus.jndi.JNSContext	false

jndilist를 수행하여 cluster1에 속한 server2의 JNDI Repository를 확인해 보아도 cds1은 더 이상 JNDI bind 되어있지 않음을 알 수 있다.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

6.7.10. 클러스터 데이터소스 설정 변경

지금까지는 클러스터 데이터소스, 서버, 클러스터를 추가/삭제할 때 어떻게 데이터소스의 관리가 이루어지는지 살펴보았다. 이제는 클러스터 데이터소스 자체의 설정 변경에 대해서 알아보자. 클러스터 데이터소스의 설정은 모두 동적 변경이 가능한데 그 중 클러스터 데이터소스에 묶인 컴포넌트 데이터소스 변경을 예로 들도록 하겠다.

우선 현재 클러스터 데이터소스가 도메인에 없는 상태이므로 add-cluster-data-source를 수행하여 컴포넌트 데이터소스를 ds1, ds2로 하는 클러스터 데이터소스 cds1을 domain1에 추가한다. 이후로는 ds1, ds2를 컴포넌트 데이터소스로 갖는 cds1이 domain1에 추가된 것을 가정한다.

본 절에서는 cds1의 컴포넌트 데이터소스 목록에서 ds2를 삭제하는 과정을 설명한다.

콘솔 툴 사용

콘솔 툴에서 **modify-cluster-data-source**를 수행하면 동적으로 클러스터 데이터소스 설정을 변경할 수 있다. help를 수행하거나 JEUS Reference 안내서의 "modify-cluster-data-source"를 참고하면 modify-cluster-data-source를 통하여 변경할 수 있는 모든 설정들을 옵션들의 이름으로 유추하여 확인할 수 있다. 동적 변경이 가능한 설정과 관련된 옵션에는 태그를 붙여두었으므로 이를 통해 어떤 설정이 동적으로 변경 가능한지도 알 수 있다.

다음은 modify-cluster-data-source를 수행하여 ds2를 cds1의 컴포넌트 데이터소스 목록에서 삭제하는 예이다.

```
[MASTER]domain1.adminServer>modify-cluster-data-source -id cds1 -cds ds1
Successfully performed the MODIFY operation for configuration of the cluster data source [cds1].
```

```
Check the results using "modify-cluster-data-source -id cds1"
```

cds1 컴포넌트 데이터소스 목록에서 ds2가 삭제되었음은 '**modify-cluster-data-source -id cds1**'을 수행하여 기타 다른 설정값들과 함께 확인할 수 있다.

```
[MASTER]domain1.adminServer>modify-cluster-data-source -id cds1
Shows the current configuration.
configuration of the cluster data source [cds1]
=====
+-----+-----+
| id                | cds1 |
| export-name       | cds1 |
| load-balance      | false|
| is-pre-conn       | false|
| use-failback      | true |
| component-data-sources | ds1  |
+-----+-----+
=====
```

6.7.11. 클러스터 데이터소스 설정 확인

JEUS 7부터는 설정 파일을 열어 사용자가 직접 수정한 설정에 대해서 책임지지 않는다. 설정 파일을 열어서 설정을 확인하는 것은 그러한 변경과는 관련이 없으나 되도록이면 설정 파일에 직접 접근하는 것을 권장하지 않는다. 이를 위해 클러스터 데이터소스의 설정을 명령어를 통해 확인할 수 있도록 지원한다.

본 절에서는 콘솔 툴을 사용해서 클러스터 데이터소스 설정을 확인하는 방법을 설명한다.

콘솔 툴 사용

콘솔 툴에서 **list-cluster-data-sources** 명령어를 수행하면 도메인에 존재하는 모든 클러스터 데이터소스의 목록이 조회된다.

다음은 list-cluster-data-sources의 수행하여 도메인에 존재하는 모든 클러스터 데이터소스 목록을 확인하는 예이다.

```
[MASTER]domain1.adminServer>list-cluster-data-sources
The list of cluster data sources
=====
+-----+-----+-----+
| Data Source ID | JNDI Export Name | Component Data Sources |
+-----+-----+-----+
| cds1           | cds1             | [ds1]                  |
+-----+-----+-----+
=====
```

만약 특정 클러스터 데이터소스의 ID를 id 옵션값으로 입력하면 그 클러스터 데이터소스의 전체 설정을 상세히 보여준다. 다음은 **list-cluster-data-sources**를 수행하여 cds1의 상세 설정을 확인하는 예이다.

```
[MASTER]domain1.adminServer>list-cluster-data-sources -id cds1
```

```
The configuration of cluster data source [cds1]
```

Configuration Name	Configuration Value
id	cds1
export-name	cds1
load-balance	false
is-pre-conn	false
use-failback	true
component-data-sources	[ds1]

6.8. JDBC Connection Pool 모니터링

본 절에서는 서버에 존재하는 Connection Pool의 모니터링에 대해 예시를 들어 설명한다. 보다 나은 이해를 위해서는 JEUS 도메인 구조와 도메인에서의 데이터소스 및 Connection Pool 관리 구조 등에 대한 이해가 필요하다. JEUS 도메인 구조에 대해서는 JEUS Domain 안내서의 "구성요소"를 참고하고, 도메인에서의 데이터소스 및 Connection Pool 관리 구조는 [데이터소스 및 Connection Pool 관리](#)를 참고한다.

JDBC Connection Pool 모니터링은 콘솔 툴을 통해서 수행될 수 있으므로 본 절에서는 두 가지 방법 모두에 대해서 함께 설명한다. 콘솔 툴을 통한 방법을 설명할 때는 편의상 축약된 명령어 및 옵션의 이름을 사용한다. 축약된 명령어 및 옵션의 이름과 명령어의 세부 사용 방법은 콘솔 툴에서 명령어 이름을 인자로 help를 수행하거나 JEUS Reference 안내서의 "Connection Pool 제어 및 모니터링 명령어"를 참고하면 확인할 수 있다.

설명을 위해 필요한 몇 가지 사전 작업을 수행한다. 우선 domain1이라는 이름의 도메인과 adminServer라는 이름의 MASTER를 생성한다. MASTER를 기동한 후 각각 server1, server2라는 이름을 갖는 MS 두 개를 domain1에 추가하고 이들 또한 기동한다. 또한 각각 ds1, ds2라는 데이터소스 ID를 갖는 데이터소스를 domain1에 추가한 후 server1에는 ds1과 ds2 모두를 등록하고 server2에는 ds1만 등록하도록 한다. 마지막으로 server1과 server2에 ds1의 Connection Pool을 각각 생성한다.

도메인 및 서버 관련 설정 방법은 JEUS Domain 안내서의 "도메인 생성" 및 JEUS Server 안내서의 "JEUS 설정"을, 데이터소스 관련 설정 방법은 [데이터소스 관련 설정 동적 변경](#)을 참고한다. Connection Pool 생성 방법은 [JDBC Connection Pool 제어](#)를 참고한다. 이후로는 앞서 언급한 사전 작업이 모두 완료되었음을 가정한다.



데이터소스가 등록되는 대상은 궁극적으로 서버이다. 서버 자체적으로 데이터소스를 등록하는 경우는 물론이고 클러스터가 데이터소스를 등록한 경우에도 클러스터에 등록된 데이터소스는 실질적으로 클러스터에 속한 서버들에 의미가 있다. 클러스터는 서버를 묶어주는 논리적 그룹일 뿐 실제로 데이터소스를 이용해 Connection Pool을 생성하여 사용하는 주체는 서버이기 때문이다. 이와 같은 이유로 Connection Pool의 모니터링 단위는 서버다.

6.8.1. JDBC Connection Pool 목록 확인

본 절에서는 콘솔 툴을 사용해서 JDBC Connection Pool 목록을 확인하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **connection-pool-info**를 수행하면 서버에 존재하는 모든 Connection Pool의 목록을 확인할 수 있다. connection-pool-info는 JDBC Connection Pool뿐만 아니라 JCA Connection Pool에 대해서도 동일한 기능을 수행하므로 JDBC Connection Pool에 대해서 사용하려면 -jdbc 옵션을 설정한다. connection-pool-info에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "connection-pool-info"를 참고한다.

다음은 connection-pool-info를 수행하여 server1에 존재하는 모든 JDBC Connection Pool의 목록을 확인하는 예이다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -jdbc
The connection pool information on the server [server1].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID|     |     |        |     |         |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds1    | 2   | 30  | 0       | 0   | 0.0    | 2   | 0      | 2   | fal  | true |
|         |     |     |         |     |         |     |        |     | se   |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds2 *  | 2   | 30  | 0       | 0   | 0.0    | 0   | 0      | 0   | fal  | false|
|         |     |     |         |     |         |     |        |     | se   |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
=====
```

server1이 ds1과 ds2를 모두 등록하고 있으므로 ds1과 ds2의 Connection Pool이 존재하는 것을 확인할 수 있다. 특히 사전 작업의 결과대로 ds1 Connection Pool은 생성된 상태며 ds2 Connection Pool은 생성되지 않은 상태임을 확인할 수 있다.

컬럼은 각각 Connection Pool ID, Connection의 최솟값, Connection의 최댓값, Connection Pool 생성 이후 Active Connection의 최댓값, Active Connection의 개수, 최근 1시간 동안의 Active Connection 개수의 평균, Idle Connection의 개수, Disposable Connection의 개수, Connection의 총 개수, Connection이 부족할 경우의 대기 여부, Connection Pool의 활성화 여부를 표시한다.

다음과 같이 active 옵션을 사용하면 현재 생성된 Connection Pool의 리스트만 확인할 수 있다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -active -jdbc
The connection pool information on the server [server1].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID|     |     |        |     |         |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds1    | 2   | 30  | 0       | 0   | 0.0    | 2   | 0      | 2   | fal  | true |
|         |     |     |         |     |         |     |        |     | se   |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
```

또한 jndi 옵션을 사용하면 데이터소스의 JNDI 이름도 함께 표시한다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -jndi -jdbc
```

The connection pool information on the server [server1].

Connection Pool ID	JNDI Expo rt Name	Min	Max	Act ive Max	Act ive	Act ive Aver age	Idle	Dis posa ble	Tot al	Wait se	Ena bled
ds1	ds1	2	30	0	0	0.0	2	0	2	fal se	true
ds2 *	ds2	2	30	0	0	0.0	0	0	0	fal se	fal se

* : has not been created, total = active + idle + disposable

server2에 존재하는 모든 Connection Pool의 목록도 다음과 같이 확인할 수 있다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server2 -jdbc
```

The connection pool information on the server [server2].

Connec tion Pool ID	Min	Max	Active Max	Act ive	Active Average	Idle	Dispos able	Tot al	Wait se	Enab led
ds1	2	30	0	0	0.0	0	0	0	fal se	fal se

* : has not been created, total = active + idle + disposable

server2는 ds1만을 등록하고 있으므로 ds1의 Connection Pool만 존재하며 사전 작업의 결과대로 생성된 상태임을 알 수 있다.

6.8.2. 특정 JDBC Connection Pool의 상세 정보 확인

본 절에서는 콘솔 툴을 사용해서 특정 JDBC Connection Pool의 상세 정보를 확인하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **connection-pool-info**를 수행하면 서버에 존재하는 특정 Connection Pool의 상세 정보를 확인할

수 있다. connection pool-info는 JDBC Connection Pool뿐만 아니라 JCA Connection Pool에 대해서도 동일한 기능을 수행하므로 JDBC Connection Pool에 대해서 사용하기 원한다면 -jdbc 옵션을 설정한다. connection-pool-info에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "connection-pool-info"를 참고한다.

다음은 connection-pool-info를 수행하여 server1에 존재하는 ds1 Connection Pool의 상세 정보를 확인하는 예이다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1 -jdbc
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| ds1-1         | idle  | 965.837         | 0         | pooled |
| ds1-2         | idle  | 965.806         | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
```

아직 한 번도 사용되지 않은 두 개의 Connection이 965초 정도 idle 상태로 머물러 있음을 확인할 수 있다. 두 Connection은 모두 disposable Connection이 아니다.

다음과 같이 -t 옵션을 사용하면 Connection을 소유한 Thread의 이름도 함께 확인할 수 있다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1 -t -jdbc
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type | Thread name |
+-----+-----+-----+-----+-----+-----+
| ds1-1         | active| 1105.954        | 1         | pooled | http-w1     |
| ds1-2         | idle  | 1105.923        | 0         | pooled |              |
+-----+-----+-----+-----+-----+-----+
=====
```

6.9. JDBC Connection Pool 제어

본 절에서는 서버에 존재하는 Connection Pool의 제어에 대해 예를 들어 설명한다. 보다 나은 이해를 위해서는 JEUS 도메인 구조와 도메인에서의 데이터소스 및 Connection Pool 관리 구조 등에 대한 이해가 필요하다. JEUS 도메인 구조에 대해서는 JEUS Domain 안내서의 "구성요소"를 참고하도록 하고, 도메인에서의 데이터소스 및 Connection Pool 관리 구조는 [데이터소스 및 Connection Pool 관리](#)를 참고한다.

JDBC Connection Pool 제어는 콘솔 툴을 통해서 수행될 수 있으므로 본 절에서는 두 가지 방법 모두에 대해서 함께 설명한다.

설명을 위해 필요한 몇 가지 사전 작업을 수행한다. 우선 domain1이라는 이름의 도메인과 adminServer라는 이름의 MASTER를 생성한다. MASTER를 기동한 후 각각 server1, server2라는 이름을 갖는 MS 두 개를 domain1에 추가하고 이들 또한 기동한다. 또한 각각 ds1, ds2라는 데이터소스 ID를 갖는 데이터소스를 domain1에 추가한 후 server1, server2 각각에 ds1과 ds2 모두를 등록하도록 한다. 도메인 및 서버 관련 설정 방법은 JEUS Domain 안내서의 "도메인 생성" 및 JEUS Server 안내서의 "JEUS 설정"을 참고하고, 데이터소스 관련

설정 방법은 [데이터소스 관련 설정 동적 변경](#)을 참고한다.



Connection Pool 제어 후의 결과는 별도로 예를 들어 설명하지 않으므로 [JDBC Connection Pool 모니터링](#)을 참고하여 결과를 확인한다.

6.9.1. Connection Pool 생성

본 절에서는 콘솔 툴을 사용해서 Connection Pool을 생성하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **create-connection-pool**을 수행하면 서버에 데이터소스의 Connection Pool을 생성할 수 있다. create-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "create-connection-pool"을 참고한다.

다음은 create-connection-pool을 수행하여 server1에 ds1 Connection Pool을 생성하는 예이다.

```
[MASTER]domain1.adminServer>create-connection-pool -server server1 -id ds1
Servers that successfully created a connection pool : server1
Servers that failed to create a connection pool : none.
```

server 옵션값을 명시하지 않으면 데이터소스가 등록된 모든 서버를 대상으로 Connection Pool이 생성된다.

다음은 create-connection-pool을 수행하여 ds2가 등록된 server1, server2 각각에 ds2 Connection Pool을 생성하는 예이다.

```
[MASTER]domain1.adminServer>create-connection-pool -id ds2
Servers that successfully created a connection pool : server1, server2
Servers that failed to create a connection pool : none.
```

control-connection-pool을 수행해도 JDBC Connection Pool을 생성할 수 있다. control-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "control-connection-pool"을 참고한다.

다음은 control-connection-pool을 수행하여 server1에 ds1 Connection Pool을 생성하는 예이다.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -create
Servers that successfully created a connection pool : server1
Servers that failed to create a connection pool : none.
```

6.9.2. Connection Pool 비활성화

본 절에서는 콘솔 툴을 사용해서 Connection Pool 비활성화하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **disable-connection-pool**을 수행하면 서버에 존재하는 Connection Pool을 비활성화할 수 있다. disable-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "disable-connection-pool"을 참고한다.

다음은 disable-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool을 비활성화하는 예이다.

```
[MASTER]domain1.adminServer>disable-connection-pool -server server1 -id ds1
Servers that successfully disabled a connection pool : server1
Servers that failed to disable a connection pool : none.
```

server 옵션값을 명시하지 않으면 데이터소스가 등록된 모든 서버를 대상으로 Connection Pool 비활성화가 시도된다.

다음은 disable-connection-pool을 수행하여 ds2가 등록된 server1, server2 각각에 대해 ds2 Connection Pool 비활성화를 수행하는 예이다.

```
[MASTER]domain1.adminServer>disable-connection-pool -id ds2
Servers that successfully disabled a connection pool : server1, server2
Servers that failed to disable a connection pool : none.
```

control-connection-pool을 수행해도 JDBC Connection Pool을 비활성화할 수 있다. control-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "control-connection-pool"을 참고한다.

다음은 control-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool을 비활성화하는 예이다.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -disable
Servers that successfully disabled a connection pool : server1
Servers that failed to disable a connection pool : none.
```

6.9.3. Connection Pool 활성화

본 절에서는 콘솔 툴을 사용해서 Connection Pool 활성화하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **enable-connection-pool** 명령어를 수행하면 서버에 존재하는 Connection Pool을 활성화할 수 있다. enable-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "enable-connection-pool"을 참고한다.

다음은 enable-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool을 활성화하는 예이다.

```
[MASTER]domain1.adminServer>enable-connection-pool -server server1 -id ds1
Servers that successfully enabled a connection pool : server1
Servers that failed to enable a connection pool : none.
```

server 옵션값을 명시하지 않으면 데이터소스가 등록된 모든 서버를 대상으로 Connection Pool 활성화가 시도된다.

다음은 enable-connection-pool을 수행하여 ds2가 등록된 server1, server2 각각에 대해 ds2 Connection Pool 활성화를 수행하는 예이다.

```
[MASTER]domain1.adminServer>enable-connection-pool -id ds2
Servers that successfully enabled a connection pool : server1, server2
Servers that failed to enable a connection pool : none.
```

control-connection-pool을 수행해도 JDBC Connection Pool을 활성화할 수 있다. control-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "control-connection-pool"을 참고한다.

다음은 control-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool을 활성화하는 예이다.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -enable
Servers that successfully enabled a connection pool : server1
Servers that failed to enable a connection pool : none.
```

6.9.4. Connection Pool의 Connection 교체

본 절에서는 콘솔 툴을 사용해서 Connection Pool의 Connection을 교체하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **refresh-connection-pool** 명령어를 수행하면 서버에 존재하는 Connection Pool의 Connection들을 새로운 Connection으로 교체할 수 있다. refresh-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "refresh-connection-pool"을 참고한다.

다음은 refresh-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool의 Connection들을 새로운 Connection으로 교체하는 예이다.

```
[MASTER]domain1.adminServer>refresh-connection-pool -server server1 -id ds1
Servers that successfully refreshed a connection pool : server1
Servers that failed to refresh a connection pool : none.
```

server 옵션값을 명시하지 않으면 데이터소스가 등록된 모든 서버를 대상으로 Connection Pool의 Connection 교체가 시도된다.

다음은 refresh-connection-pool을 수행하여 ds2가 등록된 server1, server2 각각에 대해 ds2 Connection Pool의 Connection들을 새로운 Connection으로 교체하는 예이다.

```
[MASTER]domain1.adminServer>refresh-connection-pool -id ds2
Servers that successfully refreshed a connection pool : server1, server2
Servers that failed to refresh a connection pool : none.
```

control-connection-pool을 수행해도 JDBC Connection Pool의 Connection들을 새로운 Connection으로 교체할 수 있다. control-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "control-connection-pool"을 참고한다.

다음은 control-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool의 Connection들을 새로운 Connection으로 교체하는 예이다.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -refresh
Servers that successfully refreshed a connection pool : server1
Servers that failed to refresh a connection pool : none.
```

6.9.5. Connection Pool의 Connection 개수 최소화

본 절에서는 콘솔 툴을 사용해서 Connection Pool의 Connection 개수를 최소화하는 방법에 대해서 설명한다.

콘솔 툴 사용

콘솔 툴에서 **shrink-connection-pool** 명령어를 수행하면 서버에 존재하는 Connection Pool의 Connection 개수를 설정된 Connection 최솟값으로 조정할 수 있다. shrink-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "shrink-connection-pool"을 참고한다.

다음은 shrink-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool의 Connection 개수를 설정된 Connection 최솟값으로 조정하는 예이다.

```
[MASTER]domain1.adminServer>shrink-connection-pool -server server1 -id ds1
Servers that successfully shrank a connection pool : server1
Servers that failed to shrink a connection pool : none.
```

server 옵션값을 명시하지 않으면 데이터소스가 등록된 모든 서버를 대상으로 Connection Pool의 Connection 개수 조정이 시도된다.

다음은 shrink-connection-pool을 수행하여 ds2가 등록된 server1, server2 각각에 대해 ds2 Connection Pool의 Connection 개수를 설정된 Connection 최솟값으로 조정하는 예이다.

```
[MASTER]domain1.adminServer>shrink-connection-pool -id ds2
Servers that successfully shrank a connection pool : server1, server2
Servers that failed to shrink a connection pool : none.
```

control-connection-pool을 수행해도 JDBC Connection Pool의 Connection 개수를 설정된 Connection 최솟값으로 조정할 수 있다. control-connection-pool에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "control-connection-pool"을 참고한다.

다음은 control-connection-pool을 수행하여 server1에 생성된 ds1 Connection Pool의 Connection 개수를 설정된 Connection 최솟값으로 조정하는 예이다.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -shrink
Servers that successfully shrank a connection pool : server1
Servers that failed to shrink a connection pool : none.
```

6.9.6. Connection Pool의 Connection 연결 반환(Return Connection)

본 절에서는 콘솔 툴을 사용해서 Connection Pool의 Connection 연결 반환(Return Connection)을 하는 방법에 대해서 설명한다.

애플리케이션에서 데이터소스에 getConnection 요청을 한 뒤 마지막에 Connection을 close해주지 않으면 해당 Connection은 Data Source가 AutoClose 설정이 되어있지 않는 한 계속 Active 상태로 남아있게 된다. 이러한 Connection Leak이 발생하였을 때 연결 반환(Return Connection) 기능을 이용하여 Leak이 발생한 Connection을 Connection Pool에 반환할 수 있다.

콘솔 툴 사용

콘솔 툴에서 **return-connection** 명령어를 수행하면 서버에 존재하는 Connection Pool의 Active 상태인 Connection을 Connection Pool로 연결 반환(Return Connection)할 수 있다. return-connection에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "return-connection"을 참고한다.

다음은 return-connection을 수행하여 server1에 생성된 ds1 Connection Pool의 ds1-1 Connection을 Active 상태에서 Idle 상태로 Connection Pool에 반환하는 예이다. **connection-pool-info** 명령어를 사용하여 ds1-1 Connection이 현재 Leak이 발생하여 Active 상태로 머물고 있는지 확인 할 수 있고, 해당 Connection에 대해 **return-connection** 명령어를 사용한다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| ds1-2         | idle  | 22.311          | 0         | pooled |
| ds1-1         | active | 22.289          | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
[MASTER]domain1.adminServer>return-connection -server server1 -cid ds1-1
Successfully returned the connections to the connection pool.
```

같은 Connection Pool일 경우 cid 옵션에 여러 개의 Connection을 쉼표로 구분하여 한번에 연결 반환 (Return Connection)할 수도 있고, cpid 옵션을 이용하여 하나의 Connection Pool에 존재하는 모든 Connection들을 반환 시도할 수 있다.

다음은 return-connection을 수행하여 Connection Pool ds1에 존재하는 모든 Active Connection들을 Connection Pool에 반환하는 예이다.

```
[MASTER]domain1.adminServer>return-connection -server server1 -cpid ds1
Successfully returned the connections to the connection pool.
```

6.9.7. Connection Pool의 Connection 강제 끊기(Destroy Connection)

본 절에서는 콘솔 툴을 사용해서 Connection Pool의 Connection을 강제로 끊는 방법에 대해서 설명한다.

애플리케이션에서 데이터소스에 getConnection 요청을 하여서 Active 상태로 있는 Connection이 문제가 있다고 판단되면 강제 끊기(Destroy Connection) 기능을 이용하여 문제가 있다고 판단되는 Connection을 강제로 끊을 수 있다. 단, 이 기능은 실제로 물리적 연결을 강제로 끊는 것이므로 사용에 주의를 요한다.

콘솔 툴 사용

콘솔 툴에서 **destroy-connection** 명령어를 수행하면 서버에 존재하는 Connection Pool의 Active 상태인 Connection의 연결을 강제로 끊을 수 있다. destroy-connection에 대한 보다 자세한 사용법은 JEUS Reference 안내서의 "destroy-connection"을 참고한다.

다음은 destroy-connection을 수행하여 server1에 생성된 ds1 Connection Pool의 ds1-1 Connection을 Active 상태에서 강제로 연결 끊는 예이다. **connection-pool-info** 명령어를 사용하여 ds1-1 Connection이 현재 Active 상태로 머물고 있는지 확인 할 수 있고, 해당 Connection에 대해 **destroy-connection** 명령어를 사용한다.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| ds1-2         | idle  | 22.311          | 0         | pooled |
| ds1-1         | active| 22.289          | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
[MASTER]domain1.adminServer>destroy-connection -server server1 -cid ds1-1
Successfully destroyed the connections from the connection pool.
```

같은 Connection Pool일 경우 cid 옵션에 여러 개의 Connection을 쉼표로 구분하여 한번에 연결을 강제로 끊을 수도 있고, cpid 옵션을 이용하여 하나의 Connection Pool에 존재하는 모든 Connection들의 연결을 강제로 끊도록 시도할 수 있다.

다음은 destroy-connection을 수행하여 Connection Pool ds1에 존재하는 모든 Active Connection들을 강제로 끊는 예이다.

```
[MASTER]domain1.adminServer>destroy-connection -server server1 -cpid ds1
Successfully destroyed the connections from the connection pool.
```

6.10. JEUS JDBC 프로그래밍

본 절에서는 JDBC 애플리케이션 프로그래밍 규칙에 대해 간략히 설명한다.

6.10.1. 데이터소스로부터 Connection 얻기

다음의 코드는 데이터소스를 사용하여 Connection을 얻는 방법에 대한 예이다.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("ds1");
Connection con = ds.getConnection();
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from test");
...
con.close();
```



Connection을 사용한 후에는 반드시 닫아야 한다.

6.10.2. 트랜잭션 프로그래밍 규칙

JEUS는 트랜잭션 프로그래밍을 위한 표준 단계를 정의한다. UserTransaction를 사용하는 모든 애플리케이션은 표준 단계들을 따라야 한다.

1. 트랜잭션을 시작한다.
2. DB Connection을 얻어 온다.
3. 트랜잭션의 나머지를 코딩한다.



만약 다른 트랜잭션 처리를 원한다면 반드시 새로운 Connection을 다시 얻어야 한다.

6.10.3. JDBC 드라이버의 Connection 구현체 인스턴스 얻기

JEUS에서는 애플리케이션에 Connection을 넘겨줄 때 Connection의 상태 관리를 위하여 래퍼를 사용하게 된다. 그러나 Oracle의 CLOB, XMLType 등은 JDBC 드라이버의 Connection 구현체 인스턴스를 직접적으로 필요로 하기 때문에 드라이버 내부적으로 클래스 캐스팅을 하다가 ClassCastException이 발생한다. 이는 JEUS뿐만 아니라 대부분의 WAS 제품에서 발생할 수 있는 문제로 애플리케이션 작성자는 WAS에서 넘어오는 Connection이 드라이버의 Connection 구현체 인스턴스일 것이라고 가정하여 애플리케이션 코드를 작성하면 안 된다. 하지만 이를 우회적으로 피할 수 있는 방법이 있다.

애플리케이션에서는 다음과 같은 방법으로 드라이버의 Connection 구현체 인스턴스를 얻을 수 있다.

```
import java.sql.Connection;
import java.sql.DatabaseMetaData;
...
Connection conn = datasource.getConnection();
DatabaseMetaData metaData = conn.getMetaData();
OracleConnection oraConn = (OracleConnection)metaData.getConnection();
```

6.10.4. Standalone 클라이언트에서의 Connection Pool

Standalone 클라이언트에서 JEUS에 등록된 데이터소스를 Lookup하면 해당 데이터소스의 정보를 JEUS로부터 얻어 자신의 JVM에 Connection Pool을 구성할 수 있다. 이때 Standalone 클라이언트는 clientcontainer.jar 또는 jclient.jar를 클래스 패스에 추가해야 한다.

이러한 방식은 Standalone 클라이언트에서 JNDI Lookup 방식으로 편리하게 DB에 접근할 수 있다는 장점이 있으나 DB 입장에서는 WAS 외에 JDBC 드라이버로 접근하는 클라이언트가 증가하는 것이고 클라이언트가 특별히 DB Connection을 효율적으로 관리할 필요가 있는 것이 아니라면 클라이언트에 Connection Pool을 구성하는 것은 불필요한 자원 낭비가 될 수 있다. 따라서 실제 서비스를 구현할 때는 이 방식의 사용을 권장하지 않는다.



JEUS 서버에 구성된 Connection Pool에서 Connection을 가져와서 사용하는 것이 아니다. 만약 그렇게 하고 싶다면 Connection을 사용하는 로직을 EJB로 구현해서 JEUS에 deploy하고 EJB를 찾아서(Lookup) 사용해야 한다.

6.11. 글로벌 트랜잭션(XA)과 Connection Sharing

Connection Sharing(커넥션 공유)은 같은 트랜잭션 내에서는 하나의 리소스에 대해 항상 하나의 Connection만 사용하는 것을 보장한다는 개념으로 JCA 표준에 언급되어 있다. JDBC 관점에서의 리소스는 대부분 DB(또는 데이터소스)이다. JEUS는 특별한 설정 없이도 기본적으로 XA 데이터소스 및 XA 에뮬레이션 기능을 사용하는 Connection Pool 데이터소스에 대해 Connection Sharing을 제공한다.

ProFrame과 같은 Application Framework에서는 같은 트랜잭션 내에서 하나의 데이터소스에 대해 여러 번 Connection 요청을 하는 경우가 많으므로 Connection Sharing을 사용하는 것이 바람직하다. 그렇지 않으면 하나의 데이터소스에 대해 여러 개의 물리적 Connection이 트랜잭션에 참여하게 된다. 이는 DB에 불필요하게 많은 트랜잭션 Lock을 관리해야 하는 부담을 주게 되어 트랜잭션 성능이 전체적으로 떨어질 수 있다.

만약 Connection Sharing을 사용하고 싶지 않을 때는 XA 데이터소스를 사용하는 Jakarta EE 컴포넌트 설정 파일(web.xml, ejb-jar.xml 등)을 다음과 같이 작성한다. <res-ref-name>는 데이터소스의 JNDI 이름을 입력한다.

```
<resource-ref>
  <res-ref-name>jdbc/xads</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```

아무런 설정을 하지 않으면 기본적으로 <res-sharing-scope>가 Shareable이므로 Servlet이나 EJB에서 항상 Connection Sharing을 사용한다. 단, XA 에뮬레이션 기능을 사용하는 Connection Pool 데이터소스는 항상 Connection을 공유해야 하기 때문에 Unshareable로 설정하면 에러(java.sql.SQLException)가 발생한다.

6.12. 여러 사용자에게 Connection Pooling 서비스 지원

JEUS는 데이터소스 설정상 명시된 기본 사용자에게 대해서만 Connection Pooling 서비스를 지원해왔다. 만약 기본 사용자가 아닌 사용자 정보로 Connection을 요청할 경우에는 해당 Connection을 무조건 일회용(disposable)

Connection으로 다룬다. 일회용 Connection은 이름 그대로 Connection 요청이 있을 때마다 생성됐다가 사용 후 바로 Connection Pool로부터 제거되므로 그에 대한 요청이 많을 경우 Connection Pooling 서비스에는 상당한 부담이 될 수 있다.

이를 위해 여러 사용자에게 대하여 Connection Pooling 서비스를 지원한다. 이는 하나의 Connection Pool이 외부에는 투명하게 사용자별로 다시 나뉘어 관리되는 방식이므로 클라이언트는 기존의 방식과 비교하여 아무런 차이 없이 Connection Pooling 서비스를 이용할 수 있다. 즉, Connection Pool에 존재하는 Connection들은 서로 사용자 정보가 달라도 Connection Pool의 전체적인 설정과 운영 정책은 공통적으로 따르게 되는 것이다. 그러므로 다수의 사용자별 Connection을 빈번하게 사용하게 되는 경우에도 일회용 Connection을 생성해야 하는 부담 없이 보다 효율적으로 Connection Pooling 서비스를 이용할 수 있다.

7. 트랜잭션 매니저

본 장에서는 JEUS에서의 트랜잭션 매니저와 그 주변 요소들에 대해 설명한다.

7.1. 개요

JEUS 트랜잭션 매니저는 Jakarta EE 환경에서 애플리케이션에게 다양한 형태의 트랜잭션 서비스를 제공할 수 있다. JEUS 트랜잭션 매니저는 애플리케이션의 동작에 따라 필요한 서비스를 제공하며, 다양한 리소스 매니저와의 연결을 제공한다. 또한 애플리케이션은 필요에 따라 트랜잭션의 제어 권한을 매니저에 일임할 수도 있고, 자신이 직접 갖을 수도 있다. 트랜잭션 매니저는 리소스 매니저, 애플리케이션 등과 함께 트랜잭션 처리를 위한 작업들을 하며, 그 작업에서 중점적인 역할을 담당한다.

엔터프라이즈 시스템에서 트랜잭션 서비스는 리소스에 대한 대용량의 클라이언트 요청을 안전하고 효율적으로 처리하는 데 기본적이고 중요한 서비스이다. 최근에는 리소스의 종류가 다양해지고, 리소스에 대한 요청 규모가 커짐에 따라, 트랜잭션 매니저는 통일된 인터페이스를 제공하고 더욱 안정적으로 동작할 필요성이 생겼다.

JEUS의 트랜잭션 매니저는 Java Transaction Service(JTS)와의 호환성을 제공하며, Jakarta Transaction(JTA)를 지원한다. 이는 클라이언트가 트랜잭션을 이용할 때 통일된 인터페이스와 기능을 제공하기 위함이다. 자세한 내용은 관련 API 문서나, 스펙 문서를 참고한다. 또한 다양한 형태의 애플리케이션을 지원하기 위하여, 애플리케이션이 동작할 수 있는 여러 가지 상황(클라이언트, 서버 애플리케이션)에서의 트랜잭션 서비스를 제공한다.

트랜잭션 매니저는 안정적인 서비스 제공을 위하여, 문제 상황이 발생하더라도 트랜잭션의 무결성을 보장해야 한다. 이를 위해 JEUS 트랜잭션 매니저는 트랜잭션 복구(Transaction Recovery) 기능을 제공하여, 다양한 문제 상황 이후에 트랜잭션을 복구할 수 있도록 한다.

JEUS에서의 트랜잭션 작업에 참여하는 요소는 다음과 같이 크게 3가지로 나뉘어진다. 각 구성 요소는 트랜잭션 처리를 위해 표준 API인 JTA를 이용하여 서로 통신한다.

- 서비스를 받는 입장에서의 애플리케이션

애플리케이션은 웹 컨테이너나 EJB 컨테이너, 클라이언트 컨테이너를 통해서 다양한 리소스 매니저로 접속할 수 있다. 애플리케이션은 스스로 트랜잭션의 작업과 끝을 지정하여 트랜잭션을 조정할 수도 있고, 컨테이너의 트랜잭션 매니저가 그러한 작업들을 대신할 수도 있다.

- 해당 애플리케이션에 트랜잭션 관리 기능과 그에 따르는 인터페이스를 제공하는 트랜잭션 매니저

트랜잭션 매니저는 애플리케이션의 동작 위치에 따라 서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저로 나뉘고, 실제 리소스 매니저에 직접적으로 트랜잭션 작업을 준비, 반영 요청을 전달한다.

- 실제 트랜잭션들이 영향을 주는 리소스 매니저(예: DB)

리소스 매니저는 요청들을 받아들여 실제 작업을 수행한다.

본 절에서는 각 요소에 대해 설명하며, 각각에 대해 동작 방식을 알아보도록 한다. 트랜잭션의 기본적인 내용은 관련 서적이나 자료 등을 참고한다.

7.1.1. 애플리케이션

애플리케이션은 JEUS 트랜잭션 매니저를 이용하여 리소스 매니저에 트랜잭션 관련 작업을 수행한다.

- **Local Transaction**

하나의 리소스 매니저를 사용하는 트랜잭션 작업들은 하나의 로컬 트랜잭션으로 묶일 수 있다. 애플리케이션은 리소스 매니저의 드라이버를 사용해서 로컬 트랜잭션(Local Transaction)을 시작하거나 종료한다.

기본적으로 JEUS 트랜잭션 매니저는 로컬 트랜잭션 처리에는 관여하지 않는다. 애플리케이션이 작업을 글로벌 트랜잭션으로 처리할 수 있지만, 하나의 리소스 매니저를 이용하는 단순한 트랜잭션의 작업을 할 경우라면 로컬 트랜잭션이 훨씬 효율적이므로 로컬 트랜잭션을 사용하길 권장한다.

- **Global Transaction**

애플리케이션이 2개 이상의 리소스 매니저를 이용하여 일련의 트랜잭션 작업을 시도할 경우 트랜잭션 매니저는 이 트랜잭션들을 하나의 글로벌 트랜잭션(Global Transaction)으로 묶어서 처리할 수 있다. 이 글로벌 트랜잭션은 흔히 2 Phase Commit 프로토콜을 사용하여 사용 중인 리소스들에 대해 일괄적인 트랜잭션 작업을 가능하게 한다.

트랜잭션 매니저는 트랜잭션을 commit할 때 애플리케이션의 실행 흐름을 추적해서 1 Phase Commit을 할지 2 Phase Commit을 할지를 결정한다. 이 결정은 트랜잭션 내에 사용된 리소스 매니저의 개수와 타입에 따라서 결정된다. JEUS 트랜잭션은 중복된 트랜잭션을 지원하지 않으며 2개의 트랜잭션이 섞여서 진행되는 것은 허락하지 않는다.

- **User Managed Transaction**

애플리케이션은 `jakarta.transaction.UserTransaction` 객체를 사용해서 트랜잭션의 시작과 종료 시기를 지정할 수 있다. 트랜잭션을 commit 또는 rollback할 것인지는 전적으로 애플리케이션에 의해 결정된다. 그러나 commit을 했음에도 불구하고 리소스 매니저가 트랜잭션을 처리할 수 없을 때 JEUS 트랜잭션 매니저가 rollback시킬 수도 있다.

EJB에서는 이 User Managed 트랜잭션을 Bean Managed Transaction(BMT)이라고 한다.

- **Container Managed Transaction**

Container Managed 트랜잭션은 컨테이너가 트랜잭션의 시작과 종료시기를 결정하는 방식이다.

트랜잭션의 commit, rollback 등의 결정은 전적으로 컨테이너에 의해 결정된다. EJB에서만 사용할 수 있으며 Bean의 각 메소드별로 트랜잭션 속성(NotSupported, Required, Supports, RequiresNew, Mandatory, Never)을 지정해서 사용한다.

애플리케이션은 몇 개의 리소스 매니저를 이용하느냐에 따라 **로컬 트랜잭션**과 **글로벌 트랜잭션** 2가지를 사용할 수 있다. 또한 트랜잭션의 결정 권한을 갖는 주체에 따라서 설정 및 프로그래밍 방식에 따라 나뉠 수 있다.

7.1.2. JEUS 트랜잭션 매니저

JEUS는 서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저의 두 가지 트랜잭션 매니저를 제공한다.

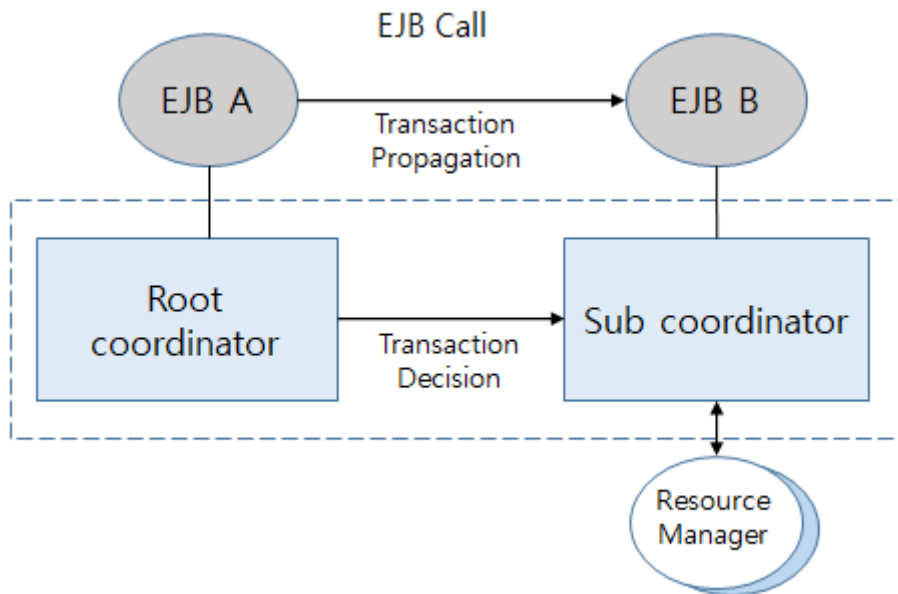
서버 트랜잭션 매니저는 글로벌 트랜잭션을 관리하기 위한 모든 기능을 제공한다. 반면에 클라이언트 트랜잭션

매니저는 서버 트랜잭션의 대행(proxy) 역할만을 수행한다.

서버 트랜잭션 매니저(Server Transaction Manager)

서버 트랜잭션 매니저는 Jakarta Transaction를 모두 구현했으며, 서버 애플리케이션이 트랜잭션을 사용할 경우 동작한다.

글로벌 트랜잭션이 시작되면 서버 트랜잭션 매니저는 글로벌 트랜잭션과 관련된 모든 정보를 수집해서, 트랜잭션의 Coordinator로서 작업을 진행한다. 상황에 따라서 **Root Coordinator**와 **Sub Coordinator**로 역할이 나뉜다.



Root Coordinator와 Sub Coordinator의 관계

- Root Coordinator

트랜잭션이 시작되는 트랜잭션 매니저가 담당하며, 여기서 시작된 트랜잭션이 전파되는 매니저는 Sub Coordinator가 된다.

- Sub Coordinator

Root Coordinator로부터 결정을 받아 해당하는 작업을 수행한다. 즉, 능동적으로 트랜잭션을 관리하는 역할이 Root Coordinator, 이 내용을 따라하는 수동적 역할이 Sub Coordinator가 할 일이다.

클라이언트 트랜잭션 매니저(Client Transaction Manager)

클라이언트 트랜잭션 매니저는 서버 트랜잭션 매니저의 대행자(proxy)로 작동한다.

글로벌 트랜잭션의 신호를 받은 서버 트랜잭션 매니저가 Root Coordinator의 역할을 하며, 해당 트랜잭션에 관한 정보를 모두 모아서 처리한다. commit 시점에 클라이언트 트랜잭션 매니저는 commit 신호를 Root Coordinator에 전송하고 commit의 결과를 받는다.

7.1.3. 리소스 매니저

애플리케이션은 다양한 리소스 매니저와 작업을 할 수 있다. 커넥션 매니저는 리소스 매니저와의 연결을 관리한다.

JEUS에서는 JDBC Connection Manager, JMS Connection Manager, WebT Connection Manager,

Connector Manager의 4가지 타입의 커넥션 매니저를 제공한다. 글로벌 트랜잭션을 위해서 커넥션 매니저는 트랜잭션과 관련되는 정보를 트랜잭션 매니저에 보고한다.

- JDBC 리소스 매니저

JDBC(Java Database Connectivity)는 Java 언어로 개발할 때 사용하는 DB와 애플리케이션 간의 연결을 정의한 것이다. 애플리케이션은 JDBC 리소스 매니저를 Naming Server로부터 Lookup해서 사용한다. 자세한 내용은 [JNDI Naming Server](#)와 [DB Connection Pool](#)과 [JDBC](#)를 참고한다.

- JMS 리소스 매니저

JMS(Jakarta Messaging)는 Queue와 Topic 그리고 거기에 저장된 메시지에 액세스하기 위한 API를 정의한 스펙이다. 자세한 내용은 "JEUS MQ 안내서"나 JMS 스펙을 참고한다.

- Tmax(WebT) 리소스 매니저

Tmax는 TmaxSoft에서 개발된 TP-monitor이다. JEUS에서는 Tmax의 트랜잭션 매니저와 투명한 양방향 서비스를 위해서 WebT라는 Bridge를 제공한다. WebT는 양방향 메소드 호출을 지원한다. 그래서 Tmax에서 일반적인 EJB 클라이언트가 작동될 수 있으며, JEUS에 있는 EJB Bean을 동일한 방법으로 호출할 수 있다.

- JCA 리소스 매니저

JCA(Jakarta Connector)는 Jakarta EE 스펙 중의 하나로 Jakarta EE 호환 플랫폼과 다양한 EIS(Enterprise Information System)와의 통합 메커니즘을 제공한다. XA 트랜잭션을 지원하는 Connector를 사용해서 애플리케이션이 다양한 트랜잭션 작업을 하나의 글로벌 트랜잭션으로 다룰 수 있다. 자세한 내용은 "JEUS Jakarta Connectors 안내서"를 참고한다.

7.2. 서버 트랜잭션 매니저 설정

본 절에서는 JEUS 트랜잭션 매니저의 설정 방법에 대해 설명한다.

JEUS 사용자는 설정을 통해 다양한 선택을 할 수 있다. 이러한 선택 사항들은 JEUS 트랜잭션 매니저의 성능 및 동작에 큰 영향을 미치게 된다. 그러므로 각각의 내용을 숙지하고 알맞은 설정을 해야 한다.

트랜잭션 매니저의 설정은 콘솔 툴을 이용할 수 있고, 나머지는 시스템 프로퍼티로 설정이 가능하다.

7.2.1. Worker Thread Pool 설정

JEUS 트랜잭션 매니저에서는 다른 트랜잭션 매니저 간의 통신을 지원하기 위해서 여러 개의 Worker Thread를 사용한다. 트랜잭션 매니저는 기본적으로 서버의 System Thread Pool을 사용한다. 트랜잭션이 중요하고 빠른 처리를 요구하는 서버에서는 Thread Pool을 설정할 수 있다.

- 공용 Thread Pool

트랜잭션이 많지는 않지만 일정한 수준으로 트랜잭션이 발생하고 다른 서비스보다 빠른 처리를 원한다면 서버 전체 서비스가 공용으로 사용하는 System Thread Pool을 설정한다.

공용 Thread Pool은 트랜잭션 매니저만 사용할 수 있는 Thread 개수를 '**Reserved Thread Num**' 항목을

이용하여 별도로 할당할 수 있다. 이때에는 다른 서비스들도 처리에 문제가 없도록 전체 Thread Pool의 크기를 고려해서 트랜잭션 매니저만의 Thread를 할당한다.



다른 서비스에서도 각각 해당 서비스만을 위한 Thread를 할당할 수 있는데 그 총합이 전체 Thread Pool 크기 이상으로 설정하면 안 된다. '**Reserved Thread Num**'을 동적으로 변경할 경우($n \rightarrow m$) n 또는 m 이 0이면 동적 반영되지 않고 Pending으로 처리되므로 재부팅해야 한다.

• 전용 Thread Pool

트랜잭션이 굉장히 많고 부하의 기복이 심하다면 System Thread를 사용하지 않고, 트랜잭션 매니저만 사용하는 전용 Thread를 생성해서 사용한다. Thread Pool을 구성하는 설정들은 System Thread Pool의 설정들과 같으므로 [Thread Pool 설정](#)을 참고한다.

위와 같이 System Thread Pool을 사용할 수도 있고, 전용 Thread Pool을 사용할 수도 있는데 서버 운영 중에는 Pool의 종류를 변경할 수는 없다. 즉, 변경 후 서버를 재시작해야 Pool의 종류를 반영할 수 있다. 그러나 Pool의 설정은 전용 Thread Pool의 큐 크기만 제외하고는 모두 서버 재시작 없이 동적 변경이 가능하다.

콘솔 툴 사용

다음은 콘솔 툴을 사용해서 Thread Pool 변경하는 설명이다.

• 공용 Thread Pool 설정

System Thread Pool에서 몇 개의 Thread를 트랜잭션 매니저를 위해 별도로 할당하려면 **modify-system-thread-pool** 명령어를 사용한다.

```
[MASTER]domain1.adminServer>modify-system-thread-pool server1 -service transaction -reservednum 10
Successfully performed the MODIFY operation for the transaction thread pool of the server (server1).
Check the results using "show-system-thread-pool server1 -service transaction or modify-system-thread-pool server1 -service transaction"
```

• 전용 Thread Pool 설정

트랜잭션 매니저 전용 Thread를 사용하려면 **modify-service-thread-pool** 명령어를 사용한다.

System Thread Pool을 사용하다가 전용 Thread Pool을 설정하면 서버를 재시작해야 적용되고, 전용 Thread Pool을 사용하다가 속성만 바꾸면 재시작 없이 반영된다.

```
[MASTER]domain1.adminServer>modify-service-thread-pool server1 -service transaction -min 10 -max 20
Successfully performed the MODIFY operation for The transaction thread pool of the server (server1), but all changes were non-dynamic. They will be applied after restarting.
Check the results using "show-service-thread-pool server1 -service transaction or modify-service-thread-pool server1 -service transaction"
```



modify-system-thread-pool과 modify-service-thread-pool 명령어에 대한 자세한 설명은 JEUS Reference 안내서의 "modify-service-thread-pool"을 참고한다.

7.2.2. 타임아웃 설정

JEUS 트랜잭션 매니저는 예외 상황 처리를 위해서 다양한 타임아웃 메커니즘을 사용한다. 타임아웃 메커니즘의 값을 조정해서 애플리케이션 시스템에 가장 적합하도록 트랜잭션 매니저를 튜닝할 수 있다. 타임아웃 설정들은 모두 서버 재시작으로 반영되는 설정들이다. 서버 시작 전에 고려하여 설정하거나 서비스 중에 변경했다면 서버를 재시작해야 한다.

다음은 설정 항목에 대한 설명이다.

• 기본 정보

항목	설명
Active Timeout	글로벌 트랜잭션이 시작되면 이 시간 안에 commit이 실행되어야 한다. 그렇지 않으면 트랜잭션 매니저가 rollback시킨다. (기본값: 600000(10분), 단위: ms)
Automatic Recovery	클러스터링 환경에서 현재의 트랜잭션 매니저가 Failure될 경우 다른 곳에서 자동으로 Indoubt 트랜잭션을 복구해주는 기능을 사용할지의 여부를 설정한다. 이 기능이 올바르게 동작하기 위해서는 클러스터링 설정이 되어 있어야 하며, Failed TM의 로그 디렉토리를 다른 곳에서 접근 가능하도록 설정되어 있어야 한다.

• 고급 선택사항

항목	설명
Prepare Timeout	Root Coordinator는 이 시간 내에 Sub Coordinator와 리소스 매니저로부터 'prepare' 신호를 받아야 한다. 만약 이 시간 내에 받지 못하면 Root Coordinator는 글로벌 트랜잭션을 rollback시킨다. (기본값: 120000(2분), 단위: ms)
Prepared Timeout	Sub Coordinator는 자신의 Root Coordinator로부터 이 시간 안에 commit을 해야 할지 rollback을 해야 할지를 나타내는 Global Decision을 받아야 한다. 만약 이 시간 내에 받지 못하면 Root Coordinator로 다시 'prepare'에 대한 응답 메시지를 보낸다. 그래도 여전히 시간 내에 Global Decision이 오지 않으면 글로벌 트랜잭션을 rollback시킨다. (기본값: 60000(1분), 단위: ms)
Commit Timeout	Root Coordinator는 이 시간 내에 Sub Coordinator와 리소스 매니저로부터 commit이나 rollback에 대한 결과를 받아야 한다. 만약 결과가 오지 않으면, Root Coordinator는 글로벌 트랜잭션을 'Incomplete List'에 기록해서 트랜잭션이 완전히 끝나지 않았음을 남겨둔다. (기본값: 240000(4분), 단위: ms)

항목	설명
Recovery Timeout	트랜잭션 복구하는 경우에 사용되는 값이다. 트랜잭션 매니저는 트랜잭션 복구를 위해서 복구될 트랜잭션 정보를 가져오려고 한다. 만약 다른 트랜잭션 매니저에서 이 시간 내에 복구 정보를 알려주지 않으면 해당 트랜잭션을 rollback시킨다. (기본값: 120000(2분), 단위: ms)
Incomplete Timeout	트랜잭션 매니저는 전체 트랜잭션 처리를 완료하기 위해 실패한 글로벌 트랜잭션의 목록을 보관한다. 완료되지 못한 글로벌 트랜잭션의 정보는 복구를 처리할 때 사용되므로 이 타임아웃 시간까지 보관된다. 그러므로 이 시간이 너무 짧으면 복구 정보가 빨리 지워지고, 트랜잭션 매니저가 해당 글로벌 트랜잭션의 무결성을 복구할 수 없게 된다. 그 결과 글로벌 트랜잭션 복구를 위해서 시스템 관리자가 많은 작업을 직접 처리해야 한다. (기본값: 86400000(1일), 단위: ms)

콘솔 툴 사용

modify-transaction-manager 명령어를 사용하여 트랜잭션 매니저의 타임아웃을 변경할 수 있다. 해당 명령어에 대한 자세한 설명은 JEUS Reference 안내서의 "modify-transaction-manager"를 참고한다.

다음은 콘솔 툴을 사용해서 타임아웃 변경하는 방법에 대한 설명이다.

```
[MASTER]domain1.adminServer>modify-transaction-manager server1 -activeTimeout 20000
Successfully performed the MODIFY operation for transaction of server (server1), but all changes were
non-dynamic. They will be applied after restarting.
Check the results using "show-transaction-manager server1 or modify-transaction-manager server1"
```

7.2.3. Root Coordinator와 Sub Coordinator 관련 설정

트랜잭션 매니저는 역할에 따라 Root Coordinator와 Sub Coordinator로 나뉘어진다. Sub Coordinator는 Root Coordinator로부터 트랜잭션 관련 사항을 전파받아야 하므로 Root Coordinator에 자신을 등록시켜 Root Coordinator가 Sub Coordinator의 존재를 알게 한다.

다음의 두 가지 시스템 프로퍼티를 사용해서 설정할 수 있다. 사용자는 퍼포먼스와 안정성을 고려해서 옵션을 적절히 사용하도록 한다.

- `-Djeus.tm.forcedReg=<true or false>`

이 설정은 Sub Coordinator가 Root Coordinator에게 언제 자신을 등록시킬 것인지에 대한 설정이다.

설정값	설명
true	트랜잭션 전파 상황에서 Sub Coordinator는 즉시 자신을 등록한다. (기본값)

설정값	설명
false	실제로 Sub Coordinator에 연결된 리소스 매니저를 사용할 때 Root Coordinator에 등록한다. 이렇게 할 경우 실제 Sub Coordinator에 등록된 리소스 매니저를 사용하지 않는 EJB Call이 많은 경우에 트랜잭션 매니저 간의 통신 횟수를 줄일 수 있다.

• `-Djeus.tm.checkReg=<true or false>`

설정값	설명
true	Sub Coordinator가 자신을 등록할 때 Root Coordinator로부터 ACK를 기다린다. 만약 일정 시간 내에 ACK가 도착하지 않으면 등록 실패로 간주하고 대기 중인 트랜잭션들을 rollback을 시킨다. (기본값)
false	Sub Coordinator가 자신을 등록할 때 Root Coordinator로부터 ACK를 기다리지 않으므로 네트워크 등의 문제로 등록되지 않은 경우에도 미리 알지 못하므로 실제 commit할 때에 문제가 된다.

7.2.4. 트랜잭션 Join 설정

JEUS 트랜잭션 매니저는 같은 리소스 매니저의 트랜잭션 리소스들은 서로 Join을 시킨다. 이렇게 할 경우 추가적인 트랜잭션 브랜치를 생성하지 않고도 작업을 하게 되어 오버헤드를 줄일 수 있다. 하지만 상황에 따라서는 이러한 방식이 문제가 될 수도 있다(예를 들면 Oracle OCI를 RAC처럼 사용할 때).

이 경우에는 다음의 옵션을 true로 지정하여 문제 발생을 대비한다.

`-Djeus.tm.disableJoin=true`

7.3. 클라이언트 트랜잭션 매니저 설정

클라이언트 트랜잭션 매니저는 클라이언트 애플리케이션이 글로벌 트랜잭션을 시작하고 종료하기 위해서 만들어졌다. 본 절에서는 클라이언트에 특별한 의미를 갖는 설정들에 대해 설명한다.

클라이언트 애플리케이션은 domain.xml의 설정 등을 사용하지 않기 때문에 애플리케이션 실행 스크립트에 시스템 프로퍼티 형태로 설정을 지정한다.

7.3.1. 트랜잭션 매니저 사용 유무 설정

클라이언트 트랜잭션 매니저는 클라이언트 애플리케이션이 JNDI Lookup을 사용할 때 초기화된다. 그러나 클라이언트 애플리케이션에 따라 트랜잭션 매니저를 사용하지 않는 경우가 있다. 이때 발생하는 추가적인 오버헤드를 없애기 위해 다음의 내용을 실행 스크립트에 저장한다.

`-Djeus.tm.not_use=true`

7.3.2. 트랜잭션 매니저 타입 설정

클라이언트 트랜잭션 매니저는 클라이언트 컨테이너에서만 사용된다. 기본적으로 클라이언트에서는 클라이언트 트랜잭션 매니저를 사용하고 특별히 서버 트랜잭션 매니저를 클라이언트에서 사용하려면 다음을 클라이언트 애플리케이션 스크립트에 추가한다. 서버 트랜잭션 매니저와 클라이언트 트랜잭션 매니저의 차이는 [JEUS 트랜잭션 매니저](#)를 참고한다.

```
-Djeus.tm.version=server
```

7.3.3. 트랜잭션 매니저의 TCP/IP Port 설정

클라이언트 트랜잭션 매니저는 다른 트랜잭션 매니저와 통신하기 위해서 TCP/IP Port를 사용한다.

기본적으로 클라이언트 트랜잭션 매니저는 적절한 포트를 스스로 찾아서 사용하나, 직접 설정하려면 다음을 클라이언트 애플리케이션 스크립트에 추가한다.

```
-Djeus.tm.port=<port number>
```

7.3.4. Worker Thread Pool 설정

Worker Thread Pool의 정보를 설정한다.

- **Min**

- Pool에 생성되는 Worker Thread의 수로 클라이언트 트랜잭션 매니저에서 이 값을 설정하려면 다음의 내용을 클라이언트 실행 스크립트에 넣어준다.

```
-Djeus.tm.tmMin=<number of threads>
```

- **Max**

- Pool에서 생성되는 Thread의 최대 개수로 클라이언트 트랜잭션 매니저에서 이 값을 설정하려면 다음의 내용을 클라이언트 실행 스크립트에 넣어준다.

```
-Djeus.tm.tmMax=<number of threads>
```

7.3.5. 타임아웃 설정

다음은 클라이언트에서 트랜잭션 매니저를 설정하는 방법에 대한 설명이다.

- **Active Timeout**

- 글로벌 트랜잭션이 시작되면 이 시간 안에 commit이 실행되어야 한다. 그렇지 않으면 트랜잭션 매니저가

rollback시킨다. (기본값: 600000(10분), 단위: ms)

- 클라이언트 컨테이너에서 이 값을 설정하려면 클라이언트 애플리케이션의 스크립트에 다음을 추가한다.

```
-Djeus.tm.activeto=<time in milliseconds>
```

• Prepare Timeout

- Root Coordinator는 이 시간 내에 Sub Coordinator와 리소스 매니저로부터 'prepare' 신호를 받아야 한다. (기본값: 120000(2분), 단위: ms)
- 만약 받지를 못하면 Root Coordinator는 글로벌 트랜잭션을 rollback시킨다.
- 클라이언트 컨테이너에서 이 값을 설정하려면 클라이언트 애플리케이션의 스크립트에 다음을 추가한다.

```
-Djeus.tm.prepareto=<time in milliseconds>
```

• Prepared Timeout

- Sub Coordinator는 자신의 Root Coordinator로부터 이 시간 안에 commit할지 rollback할지를 나타내는 Global Decision을 받아야 한다. (기본값: 60000(1분), 단위: ms)
- 만약 이 시간 내에 받지 못하면 Root Coordinator 로 다시 'prepare'에 대한 응답 메시지를 보낸다. 그래도 여전히 시간 내에 Global Decision이 오지 않으면 글로벌 트랜잭션을 rollback시킨다.
- 클라이언트 컨테이너에서 이 값을 설정하려면 클라이언트 애플리케이션의 스크립트에 다음을 추가한다.

```
-Djeus.tm.preparedto=<time in milliseconds>
```

• Commit Timeout

- Root Coordinator는 이 시간 내에 Sub Coordinator와 리소스 매니저로부터 commit이나 rollback에 대한 결과를 받아야 한다. (기본값: 240000(4분), 단위: ms)
- 만약 결과가 오지 않으면 Root Coordinator는 글로벌 트랜잭션을 'Incomplete List'에 기록해서 트랜잭션이 완전히 끝나지 않았음을 남겨둔다.
- 클라이언트 컨테이너에서 이 값을 설정하려면 클라이언트 애플리케이션의 스크립트에 다음을 추가한다.

```
-Djeus.tm.committo=<time in milliseconds>
```

• Recovery Timeout

- 트랜잭션을 복구하는 경우에 사용된다. 트랜잭션 매니저는 트랜잭션 복구를 위해서 복구될 트랜잭션 정보를 가져오려고 한다. (기본값: 120000(2분), 단위: ms)
- 만약 다른 트랜잭션 매니저에서 이 시간 내에 복구 정보를 알려주지 않으면 해당 트랜잭션을 rollback시킨다.
- 클라이언트 컨테이너에서 이 값을 설정하려면 클라이언트 애플리케이션의 스크립트에 다음을 추가한다.

```
-Djeus.tm.recoveryto=<time in milliseconds>
```

• Incomplete Timeout

- 트랜잭션 매니저는 전체 트랜잭션 처리를 완료하기 위해 실패한 글로벌 트랜잭션의 목록을 보관한다. 완료되지 못한 글로벌 트랜잭션의 정보는 복구를 처리할 때 사용되므로 이 타임아웃 시간까지 보관된다. 그러므로 이 시간이 너무 짧으면 복구 정보가 빨리 지워지고 트랜잭션 매니저가 해당 글로벌 트랜잭션의 무결성을 복구할 수 없게 된다. 그 결과 글로벌 트랜잭션 복구를 위해서 시스템 관리자가 많은 작업을 직접 처리해야만 한다. (기본값: 86400000(1일), 단위: ms)
- 클라이언트 컨테이너에서 이 값을 설정하려면 클라이언트 애플리케이션의 스크립트에 다음을 추가한다.

```
-Djeus.tm.incompleteto=<time in milliseconds>
```

7.4. 트랜잭션 애플리케이션 작성

본 절에서는 JEUS 트랜잭션 프로그래밍 예제를 설명한다.

트랜잭션 프로그래밍 패턴에는 4가지가 있다.

- 로컬 트랜잭션(Local Transaction)
- 클라이언트 관리 트랜잭션(Client Managed Transaction)
- Bean 관리 트랜잭션(Bean Managed Transaction)
- 컨테이너 관리 트랜잭션(Container Managed Transaction)

사용자 및 애플리케이션 프로그래머는 현재 자신의 애플리케이션이 어떤 양식으로 운영되어야 하는가를 확실히 지정한 후 프로그래밍을 해야 한다. 그래야만 예측된 결과를 얻을 수 있으며 문제가 발생했을 때 쉽게 해결할 수 있다.

애플리케이션은 트랜잭션 작업을 하나의 트랜잭션으로 관리할 수 있다. 작업이 하나의 리소스 매니저를 통해서 처리된다면 로컬 트랜잭션을 형성한다. 그렇지 않다면 트랜잭션을 관리하기 위해서 글로벌 트랜잭션을 사용할 수밖에 없다.

7.4.1. 로컬 트랜잭션(Local Transaction)

로컬 트랜잭션은 하나의 리소스 매니저에 대한 트랜잭션 작업을 관리하는 데 효과적인 방법이다. 가볍고 빠르기 때문에 가능하면 로컬 트랜잭션을 사용한다. 로컬 트랜잭션은 JEUS 트랜잭션 매니저와는 아무런 관계가 없으며 모든 타입의 컨테이너에서 사용할 수 있다.

다음은 로컬 트랜잭션을 사용하는 간단한 예제이다. 비록 Java 애플리케이션이지만 코드 일부는 Servlet이나 EJB 등과 같은 다른 Jakarta EE 프로그래밍에서도 사용할 수 있다.

로컬 트랜잭션을 사용 예 : <Client.java>

```
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
```

```

import java.util.*;
import jakarta.transaction.UserTransaction;
import java.sql.*;
import javax.sql.*;

public class Client {
    private static Connection con;

    private static Connection getConnection() throws
        NamingException, SQLException {
        // get a JDBC connection
    }

    private static String insertCustomer(String id, String name,
        String phone) throws NamingException, SQLException {
        // insert a product entity given by the arguments to DB
    }

    private static void deleteCustomer(String id) throws
        NamingException, SQLException {
        // delete a product entity given by the arguments from DB
    }

    public static void main(String[] args) {
        try {
            // get a JDBC connection
            con = getConnection();

            // set the autocommit attribute as false
            con.setAutoCommit(false);

            // insert customers
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting customer id="+i+"c... from Client");
                customers[i] = insertCustomer(i+"c", "Hong Kil Dong "+i, "000-123-1234-"+i);
            }
            System.out.println("completed inserting customers!!");
            con.commit();

            // delete customers
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("deleting customerid="+customers[i]+" ... from Client");
                deleteCustomer(customers[i]);
            }
            System.out.println("completed deleting customers!!");
            con.commit();

        } catch (NamingException ne) {
            System.out.println("Naming Exception caught : " + ne.getMessage());
        } catch (SQLException sqle) {
            System.out.println("SQL Exception caught : " + sqle.getMessage());
        } catch (Exception e) {
            try {
                con.rollback();
            } catch (Exception ee) {
                System.out.println("Transaction Rollback error : " + ee.getMessage());
            }
            System.out.println("Error caught : " + e.getMessage());
        }
    }
}

```

```

        e.printStackTrace();
    } finally {
        try {
            if (con!=null) con.close();
        } catch (SQLException se) {}
    }
}
}
}

```

7.4.2. 클라이언트 관리 트랜잭션(Client Managed Transaction)

클라이언트 컨테이너의 애플리케이션에서 글로벌 트랜잭션을 사용할 수 있다. 클라이언트는 UserTransaction을 이용하여 직접 트랜잭션을 관리하고 실제 작업을 수행하는 EJB를 호출한다.

다음은 간단한 예제이다.

클라이언트

글로벌 트랜잭션을 시작하기 전에 'java:comp/UserTransaction'을 Lookup해서 jakarta.transaction.UserTransaction의 인스턴스를 가져온다. 이 인스턴스의 begin()을 호출해서 글로벌 트랜잭션을 시작한 다음, EJB Bean을 여러 번 호출한다. EJB Bean의 트랜잭션 작업을 commit하려고 UserTransaction 인스턴스의 commit() 메소드를 실행해서 트랜잭션이 완료되었음을 알린다.

정상적으로 commit되면 메소드는 끝나게 된다. 장애가 발생해서 글로벌 트랜잭션이 commit되지 못한다면 메소드에서 Exception을 던지게 된다. jakarta.transaction.RollbackException은 JEUS 트랜잭션 매니저가 글로벌 트랜잭션이 rollback되었다는 것을 보장하는 Exception이다. 이외에 다른 Exception은 JEUS 트랜잭션 매니저가 예측할 수 없는 Exception이 발생해서 글로벌 트랜잭션을 rollback하려고 한다는 것을 나타낸다. 그러나 글로벌 트랜잭션에 포함된 모든 트랜잭션 작업들이 완전히 rollback된다는 것은 아니다. 이럴 때는 catch 문 안에서 다시 한 번 직접 UserTransaction의 rollback() 메소드를 호출해 글로벌 트랜잭션을 rollback시킨다.



Standalone 클라이언트는 컴포넌트의 개념이 없기 때문에 java:comp/UserTransaction을 Standalone 클라이언트에서 다른 이름으로 Lookup할 수 있도록 java:/UserTransaction을 지원한다. 실질적으로 클라이언트에서는 java:comp/UserTransaction, java:/UserTransaction 모두 트랜잭션을 Lookup하기 위해 사용 가능하고 어느 것을 사용해도 상관은 없다.

클라이언트 관리 트랜잭션 사용 예 : <Client.java>

```

package umt;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import java.util.*;
import jakarta.transaction.UserTransaction;

public class Client {
    public static void main(String[] args) {
        UserTransaction tx = null;
    }
}

```

```

try {
    InitialContext initial = new InitialContext();
    ProductManager productManager =
        (ProductManager)initial.lookup("productmanager");
    System.out.println("");
    System.out.println("< Testing ProductManager EJBBean " + "Using User Managed Transaction >");
    System.out.println("");
    int number = 10;
    String products[] = new String[number];
    tx = (UserTransaction)initial.lookup("java:comp/UserTransaction");
    tx.begin();
    // insert products
    for (int i=0 ; i<number/2 ; i++) {
        System.out.println("inserting product id="+i+"b...");
        // bean call
        products[i] = productManager.insertProduct(i+"b","ball pen", i*10);
    }
    for (int i=number/2 ; i<number ; i++) {
        System.out.println("inserting product id="+i+"f...");
        products[i] = productManager.insertProduct(i+"f", "fountain pen", (i-number/3)*50);
    }
    System.out.println("completed inserting products!!");
    // delete products
    for (int i=0 ; i<number ; i++) {
        System.out.println("deleting productid="+products[i]+" ...");
        productManager.deleteProduct(products[i]); // bean call
    }
    System.out.println("completed deleting products!!");
    tx.commit();
} catch (jakarta.transaction.SystemException se) {
    System.out.println("Transaction System Error caught : " + se.getMessage());
} catch (jakarta.transaction.RollbackException re) {
    System.out.println("Transaction Rollback Errorcaught : " + re.getMessage());
} catch (Exception e) {
    try {
        tx.rollback();
    } catch (Exception ee) {
        System.out.println("Transaction Rollback error : " + ee.getMessage());
    }

    System.out.println("Error caught : " + e.getMessage());
    e.printStackTrace();
}
}
}
}

```

EJB

다음의 EJB Bean은 트랜잭션 작업을 하는 메소드를 가지고 있다. 클라이언트에서 글로벌 트랜잭션을 관리하기 위해서는 EJB의 트랜잭션 타입이 CMT(container-managed)여야 한다. 기본값은 CMT이므로 아무것도 지정하지 않아도 되지만 명시적으로 지정하고 싶다면 다음과 같이 TransactionManagerType 값을 지정한다.

클라이언트 관리 트랜잭션 사용 예 : <ProductManagerEJB.java>

```
package umt;
```

```
import jakarta.ejb.*;
import jakarta.annotation.*;

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class ProductManagerEJB implements ProductManager {
    ....

    public String insertProduct(String id, String name, double price) {
        // insert a product entity given by the arguments to DB
    }

    public void deleteProduct(String id) {
        // delete a product entity indicated by the argument from
        // DB
    }
    .....
}
```

7.4.3. Bean 관리 트랜잭션(Bean Managed Transaction)

Bean 관리 트랜잭션은 웹 컨테이너와 EJB 컨테이너에서 애플리케이션이 JTA를 사용해서 글로벌 트랜잭션 경계를 정할 수 있다. 애플리케이션이 글로벌 트랜잭션을 세밀하게 조절할 필요가 있을 때 유용한다. 실행 순서에 따라 클라이언트의 요청을 처리할 수 있기 때문에 애플리케이션은 자신의 판단에 따라서 commit과 rollback을 할 수 있다.

다음은 EJB 컨테이너에서 애플리케이션이 글로벌 트랜잭션을 실행하는 예제이다.

클라이언트

Bean 관리 트랜잭션(BMT)에서는 EJB Bean이 글로벌 트랜잭션 영역을 처리한다. 그러므로 모든 클라이언트는 단지 트랜잭션 작업을 하는 메소드를 호출하기만 하면 된다.

Bean 관리 트랜잭션 사용 예 : <Client.java>

```
package bmt;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class Client {
    public static void main(String[] args) {
        try {
            ProductManager productManager;
            .....
            // Getting a reference to an instance of
            // ProductManager EJB bean.
            .....
            productManager.transactionTest();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
}
```

EJB

EJB Bean은 글로벌 트랜잭션을 시작하기 위해서 `jakarta.transaction.UserTransaction`를 사용한다.

`jakarta.ejb.EJBContext`(본 예제에서는 EJB가 Session Bean이므로 `jakarta.ejb.SessionContext`)를 사용해서 `UserTransaction`의 인스턴스를 얻어온다. 글로벌 트랜잭션은 메소드의 실행으로 begin되고 commit된다.

EJB가 Bean 관리 트랜잭션으로 작동하려면 Annotation으로 `TransactionManagement` 값을 `TransactionManagementType.BEAN`으로 지정한다.

Bean 관리 트랜잭션 사용 예 : <ProductManagerEJB.java>

```
package bmt;

import jakarta.ejb.*;
import javax.naming.*;
import java.sql.*;
import java.util.*;
import jakarta.annotation.*;
import jakarta.transaction.UserTransaction;

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class ProductManagerEJB implements ProductManager {
    @Resource SessionContext ejbContext;

    public void transactionTest() {
        UserTransaction tx = null;
        try {
            int number = 20;
            String products[] = new String[number];
            tx = ejbContext.getUserTransaction();
            tx.begin();
            // insert products
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting product id="+i+"b ...");
                products[i] = insertProduct(i+"b", "ball pen", i*10);
            }
            for (int i=number/2 ; i<number ; i++) {
                System.out.println("inserting product id="+i+"f...");
                products[i] = insertProduct(i+"f", "fountain pen",
                    (i-number/3)*50);
            }
            System.out.println("completed inserting products!!");
            // delete products
            for (int i=0 ; i<number ; i++) {
                System.out.println("deleting product id="+products[i]+" ...");
                deleteProduct(products[i]);
            }
            System.out.println("completed deleting products!!");
            tx.commit();
        } catch (jakarta.transaction.SystemException se) {
            throw new EJBException(
                "Transaction System Error caught : " + se.getMessage());
        }
    }
}
```



```

    } catch (jakarta.transaction.RollbackException re) {
        throw new EJBException(
            "Transaction Rollback Error caught : " + re.getMessage());
    } catch (Exception e) {
        try {
            tx.rollback();
        } catch (Exception ee) {
            throw new EJBException("Transaction Rollback error : " + ee.getMessage());
        }
        throw new EJBException("Error caught : " + e.getMessage());
    }
}

private String insertProduct(String id, String name, double price)
    throws NamingException, SQLException {
    // insert a product entity given by the arguments to DB
}

private void deleteProduct(String id) throws NamingException, SQLException {
    // delete a product entity indicated by the argument from
    // DB
}

// some EJB callback methods
}

```

7.4.4. 컨테이너 관리 트랜잭션(Container Managed Transaction)

Jakarta EE 스펙에 의하면 애플리케이션은 글로벌 트랜잭션 경계(demarcation) 지정을 컨테이너에 위임할 수 있다. 웹 컨테이너나 EJB 컨테이너는 메소드와 관련된 글로벌 트랜잭션을 관리한다. 애플리케이션은 설정 파일에 메소드별로 트랜잭션 속성을 정할 수 있다. 이 방법은 글로벌 트랜잭션을 처리하는 가장 쉬운 방법이다.

다음은 EJB 컨테이너가 관리하는 글로벌 트랜잭션의 예이다.

클라이언트

다음은 컨테이너 관리 트랜잭션에서 서버 측 EJB 컨테이너가 EJB의 글로벌 트랜잭션 작업을 관리하는 예제이다.

컨테이너 관리 트랜잭션 : <Client.java>

```

package bmt;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class Client {
    public static void main(String[] args) {
        try {
            ProductManager productManager;
            // getting a reference to an instance of
            // ProductManager EJB bean.
            productManager.transactionTest();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
}  
}
```

EJB

컨테이너 관리 트랜잭션의 장점은 EJB Bean 개발자가 비즈니스 로직 코드에서 더 이상 트랜잭션 관련 코드를 작성하지 않도록 한다는 것이다.

EJB Bean의 메소드에 적절한 트랜잭션 속성만 지정하면 글로벌 트랜잭션 관련 작업은 모두 EJB 컨테이너에 위임된다. 다음의 예제에서 transactionTest()는 트랜잭션 관련 코드가 아무 것도 없다. EJB Bean이 컨테이너 관리 트랜잭션으로 작동하도록 EJB 컨테이너에 알려주려면 TransactionManagement Annotation을 생략하거나 TransactionManagerType.CONTAINER 값으로 지정해준다.

컨테이너 관리 트랜잭션 : <ProductManagerEJB.java>

```
package cmt;  
  
import jakarta.ejb.*;  
import javax.naming.*;  
import java.sql.*;  
import java.util.*;  
import jakarta.annotation.*;  
  
@Stateless  
@TransactionManagement(TransactionManagementType.CONTAINER)**  
public class ProductManagerEJB implements ProductManager {  
    public void transactionTest() {  
        try {  
            int number = 20;  
            String products[] = new String[number];  
            // insert products  
            for (int i=0 ; i<number/2 ; i++) {  
                System.out.println("inserting product id="+i+"b...");  
                products[i] = insertProduct(i+"b", "ball pen", i*10);  
            }  
            for (int i=number/2 ; i<number ; i++) {  
                System.out.println("inserting product id="+i+"f...");  
                products[i] = insertProduct(i+"f", "fountain pen", (i-number/3)*50);  
            }  
            System.out.println("completed inserting products!!");  
            // delete products  
            for (int i=0 ; i<number ; i++) {  
                System.out.println("deleting product id="+products[i]+" ...");  
                deleteProduct(products[i]);  
            }  
            System.out.println("completed deleting products!!");  
        } catch (Exception e) {  
            throw new EJBException("Error caught : " + e.getMessage());  
        }  
    }  
  
    private String insertProduct(String id, String name, double price)  
        throws NamingException, SQLException {  
        // insert a product entity given by the arguments to DB  
    }  
}
```

```
private void deleteProduct(String id) throws NamingException, SQLException {
    // delete a product entity indicated by the argument from
    // DB
}

// some EJB callback methods
}
```

7.4.5. 트랜잭션 매니저 사용

트랜잭션 매니저를 직접 사용하는 경우 JNDI Context에서 `java:appserver/TransactionManager`의 이름으로 Lookup하면 트랜잭션 매니저를 얻을 수 있다. 트랜잭션 매니저는 트랜잭션을 시작/종료할 수 있고 트랜잭션 객체를 얻거나 XAResource 객체를 등록하는 기능을 제공한다. 이에 대한 자세한 설명은 "Jakarta Transaction(JTA) specification"을 참고한다.



JEUS에서 트랜잭션 매니저를 얻거나 UserTransaction을 얻는 방법은 Glassfish 또는 SunOne Application Server와 호환된다. 따라서 Hibernate와 같은 3rd-party library를 사용할 때 호환되는 서버의 설정을 그대로 사용할 수 있다.

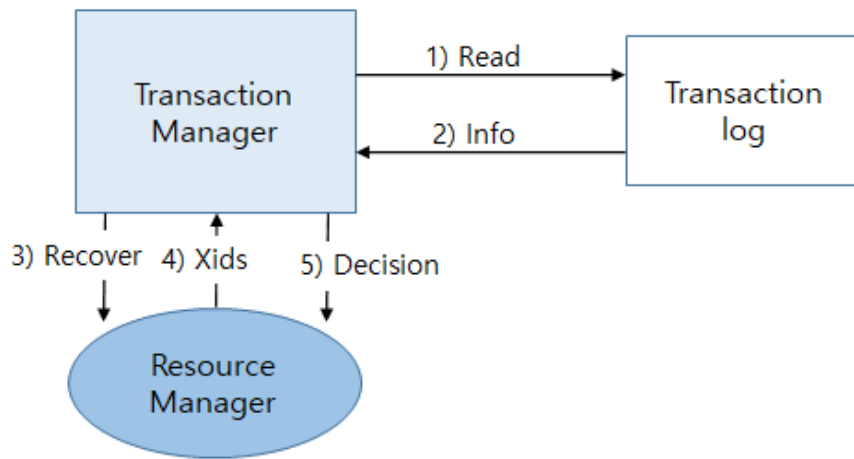
7.5. 트랜잭션 복구

본 절에서는 트랜잭션의 복구 작업에 대해 설명한다. 트랜잭션의 복구는 예상치 못한 문제 상황에 있어 트랜잭션 무결성을 지원하기 위한 중요한 기능이다. JEUS 사용자는 앞서 설명한 관련 설정 및 트랜잭션 복구 과정에 대해 숙지한다.

7.5.1. 트랜잭션 복구 과정

트랜잭션 매니저는 트랜잭션을 사용할 때 발생할 수 있는 장애에 대한 대책을 가지고 있어야 한다. 장애 대책은 주로 진행 중이던 트랜잭션의 무결성을 보장하는 데 중점을 둔다. 트랜잭션 매니저는 트랜잭션 매니저의 역할에 따라서 복구 방식에 차이를 보인다. 즉, 매니저가 Root Coordinator 또는 Sub Coordinator인지, JEUS가 아닌 다른 외부 트랜잭션 매니저와 작업 중인지에 따라서 복구 방식이 달라진다.

다음의 그림은 트랜잭션 매니저의 복구 과정을 간단히 보여준다.



단순화한 트랜잭션 복구 과정

- 1, 2) 트랜잭션 매니저가 장애 상황에서 복구될 경우 매니저는 트랜잭션 로그로부터 복구할 트랜잭션 정보를 얻어온다.
- 3) 그 이후 로그의 내용에 따라 해당 리소스 매니저로 Recover 명령을 보낸다.
- 4) 리소스 매니저는 이에 대한 응답으로 현재 자신이 처리하지 못한 트랜잭션들의 XID를 매니저로 넘겨준다.
- 5) 트랜잭션 매니저는 이 XID들과 로그로부터 얻어진 정보를 가지고 commit 또는 rollback 판정을 리소스 매니저로 전달한다.

트랜잭션 매니저의 역할에 따라 조금씩 복구 방식의 차이는 있지만 큰 흐름은 앞서 설명한 내용을 벗어나지 않는다. 이후부터는 복구에 참여하는 각각의 요소에 대한 간략한 설명과 관련 설정들에 대해 설명한다.

트랜잭션 매니저별 복구 과정

트랜잭션 복구에서 트랜잭션 매니저는 가장 중심적인 역할을 한다. 다음은 트랜잭션 매니저 역할에 따른 복구 방식의 차이에 대한 설명이다.

• Root Coordinator

트랜잭션 매니저가 Root Coordinator일 경우에는 [단순화한 트랜잭션 복구 과정](#)과 같이 동작한다.

• Sub Coordinator

Sub Coordinator는 Root Coordinator에게 하나의 리소스 매니저처럼 등록된다.

그리고 모든 Decision을 Root Coordinator로부터 받아오므로 Sub Coordinator는 자신에게 등록된 리소스 매니저와 로그 파일에 대해 XID를 얻어오는 작업까지만 수행한다. 그 이후 XID를 Root Coordinator로 보내고 판정을 기다린다. Root Coordinator로부터 판정이 넘어오면 그 내용을 리소스 매니저에게 전달해 복구 작업을 마무리한다.

• 외부 트랜잭션 매니저와의 작업

JEUS 트랜잭션 매니저는 Root Coordinator로서 동작을 하고 외부 트랜잭션 매니저는 JEUS를 하나의 리소스 매니저로 인식한다. 그러므로 판정을 내리는 작업을 제외한 모든 작업들을 수행한다. 그 이후 외부 트랜잭션 매니저가 JEUS에게 Recover 명령 및 판정을 내리면 그에 따라 복구 작업을 완료한다.

자동 트랜잭션 복구

자동 트랜잭션 복구 기능은 클러스터 환경에서 현재의 트랜잭션 매니저가 비정상 종료되면 다른 서버의 트랜잭션 매니저가 자동으로 indoubt 트랜잭션을 복구해주는 기능이다. 이 기능이 올바르게 동작하기 위해서는 서버 간에 클러스터 설정이 되어 있어야 하며 복구해주는 서버가 비정상 종료된 서버의 트랜잭션 로그에 접근 가능해야 한다.

다음은 콘솔 툴을 사용해서 자동 트랜잭션 복구를 설정하는 과정에 대한 설명이다. 이 설정은 서버 재시작 없이 변경 적용이 가능하다.

• 콘솔 툴 사용

콘솔 툴에서 **modify-transaction-manager** 명령어를 사용해서 자동 트랜잭션 복구 기능을 활성화할 수 있다. 해당 명령어에 대한 자세한 설명은 JEUS Reference 안내서의 "modify-transaction-manager"를 참고한다.

```
[MASTER]domain1.adminServer>modify-transaction-manager server1 -automaticRecovery true
Successfully performed the MODIFY operation for transaction of server (server1), but all changes
were non-dynamic.
They will be applied after restarting.
Check the results using "show-transaction-manager server1 or modify-transaction-manager server1"
```

7.5.2. 복구 관련 로그 파일

트랜잭션 복구에 사용되는 로그 파일은 다음의 2가지가 있다.

- 트랜잭션의 진행 상황을 기록한 로그
- 사용했던 XA 리소스 관련 로그

로그 파일은 기본적으로 SERVER_HOME/.workspace/tmlog의 하위 디렉터리에 생성된다. SERVER_HOME의 위치는 [서버 디렉터리 구조](#)를 참고한다. 로그 파일의 위치는 다음과 같이 변경할 수 있다.

예를 들어 위에서 설명한 자동 트랜잭션 복구를 위해서 여러 서버가 접근 가능한 위치에 로그를 남기고 싶은 경우 공용 디렉터리를 지정한다. 만약 복구가 필요없는 트랜잭션 때문에 기동 중 복구 관련 문제가 발생한다면 이 로그를 백업한 후 JEUS를 다시 실행한다.

다음은 콘솔 툴을 사용해서 복구 관련 로그 파일을 설정하는 과정에 대한 설명이다.

• 콘솔 툴 사용

콘솔 툴에서 **modify-transaction-manager** 명령어를 사용하여 트랜잭션 로그 디렉터리를 변경할 수 있다. 해당 명령어에 대한 자세한 설명은 JEUS Reference 안내서의 "modify-transaction-manager"를 참고한다.

```
[MASTER]domain1.adminServer>modify-transaction-manager server1 -txLogDir
/Users/user1/jeus/domain1/tmlogs
Successfully performed the MODIFY operation for transaction of server (server1) ,but ALL changes
were non-dynamic.
They will be applied after restarting.
Check the results using "show-transaction-manager server1 or modify-transaction-manager server1"
```

7.5.3. 복구 관련 설정

다음은 트랜잭션 복구에 관련된 설정이다. 시스템 프로퍼티에 다음의 설정 내용을 서버의 JVM 설정에 추가해야 한다. 추가하는 방법은 JEUS Domain 안내서의 "서버의 JVM 설정변경"을 참고한다.

- 시스템 관리자가 직접 복구할 경우나 성능 향상이 필요한 경우에 복구를 위한 logging을 막을 수도 있다.

```
-Djeus.tm.noLogging=true
```

- 만약 복구 작업 중에 어떤 문제에 의해 복구가 실패할 경우 트랜잭션 매니저는 그 리소스에 대해 추가적인 복구를 시도한다. 이는 2분에 한 번씩 시행되며 사용자는 시도 횟수를 지정할 수 있다.

시도 횟수 지정을 위해 다음 프로퍼티를 스크립트에 추가한다. (기본값: 30)

```
-Djeus.tm.recoveryTrial=<number of trial>
```

- 서버를 시작할 때 복구할 트랜잭션이 있다면 STANDBY가 되기 전에 복구를 시작한다.

복구 중 실패한 리소스가 있다면 그에 대해 백그라운드로 복구를 재시도한다. 이때 재시도하는 간격을 지정할 수 있다. (기본값: 120000(2분), 단위: ms)

```
-Djeus.tm.recoveryInterval=<time in milliseconds>
```

- TM 로그 파일이 지워져서 복구가 불가능한 상황일 때 JEUS의 boot는 실패하게 된다.

만약 복구가 필요하지 않는 상황이라면 JEUS가 파일이 깨진 것을 감지한 후 TM 로그 파일들을 전부 지우도록 초기화시킨 후 boot가 성공되게 한다. (기본값: false)

```
-Djeus.tm.ignore.broken.log.file=<true or false>
```

7.5.4. 리소스 매니저 장애

리소스 매니저에서 장애가 발생했다면 시스템 매니저는 콘솔 툴을 사용해서 수동으로 복구한다. 그 이유는 JEUS 트랜잭션 매니저는 리소스 매니저가 다시 트랜잭션 처리가 가능한 상태인지를 알아낼 수 없기 때문이다.

리소스 매니저의 문제를 해결한 후에 콘솔 툴의 **recover-transactions** 명령어를 수행해서 복구를 진행한다. 콘솔 툴의 자세한 사용법은 JEUS Reference 안내서의 "recover-transactions"를 참고한다.

7.6. 트랜잭션 프로파일 기능

JEUS 트랜잭션 매니저가 각 트랜잭션 과정 중 중요한 시점에 사용자의 callback을 실행시킬 수 있는 기능을 제공한다. 클라이언트에서는 사용할 수 없고 서버에서만 사용할 수 있다.

다음에 정의된 인터페이스를 상속한 사용자만의 프로파일 리스너를 구현한다.

프로파일 리스너 : <CoordinatorProfileListener.java>

```
package jeus.transaction.profile;

import jeus.transaction.info.TransactionInfo;

public interface CoordinatorProfileListener extends ProfileListener {

    public void beforePrepare( TransactionInfo info );

    public void afterPrepare( jeus.transaction.info.TransactionInfo info );

    public void beforeSetDecision( TransactionInfo info );

    public void afterSetDecision( TransactionInfo info );

    public void beforeCommit( TransactionInfo info );

    public void afterCommit( TransactionInfo info );

    public void beforeOnePhaseCommit( TransactionInfo info );

    public void afterOnePhaseCommit( TransactionInfo info );

    public void beforeRollback( TransactionInfo info );

    public void afterRollback( jeus.transaction.info.TransactionInfo info );

    public void beforeDestroy( TransactionInfo info );

    public void afterDestroy( TransactionInfo info );

    public void beforeActiveTimeout( TransactionInfo info );

    public void afterActiveTimeout( TransactionInfo info );

}
```

프로파일 리스너 : <TransactionInfo.java>

```
package jeus.transaction.info;

import jakarta.transaction.xa.Xid;

public interface TransactionInfo {

    public static final int JEUS_SPECIFIC_CURRENT_FORMAT_XID = 303077;
    public static final int UNSPECIFIED_TIME = -1;

    /**
     * 이 transaction의 xid를 반환한다. 반환된 XID는 JEUS 내부구현체의 XID이며
     * serializable하다.
     * 차후 이 XID를 string으로 표현하고자 할 경우 XidToString.getXidHexString()을
     * 이용하도록 한다.
     *
     * @return 해당 tx의 XID구현체
     */
}
```

```

public Xid getXid();

/**
 * transaction manager의 information을 string형태로 반환한다. 이 정보를 통해 해당 TX의
 * TM server 정보(ip, port등)를 알수 있다.
 *
 * @return 해당 tx의 transaction manager의 information
 */
public String getCoordinator();

/**
 * 외부에서 시작된 TX의 경우 외부의 xid를 반환한다. 반환된 XID는 JEUS 내부구현체의 XID로
 * 교체되며 serializable하다.
 *
 * @return 해당 tx의 XID구현체
 */
public Xid getExternalXid();

/**
 * active timeout설정을 반환한다. 단위는 ms이다.
 *
 * @return 해당 tx의 active timeout
 */
public long getTimeout();

/**
 * TX가 시작되고 얼마나 경과했는지에 대한 시간을 반환한다. 단위는 ms이다.
 *
 * @return 해당 tx의 tx의 경과 시간
 */
public long getElapseSinceBegin();

/**
 * TX의 상태를 string형태로 반환한다.
 *
 * @return 해당 tx의 string status값
 */
public String getStatus();

/**
 * TX에 enlist된 XAResource들을 String형태로 반환한다.
 *
 * @return 해당 tx에 enlist된 XAResource의 정보들
 */
public String[] getXaResources();

/**
 * TX을 시작한 thread를 return한다.
 * TX을 전파받은 곳에서는 이 정보를 얻을 수 없다.
 *
 * @return 해당 tx을 시작한 thread
 */
public Thread getBeginThread();
}

```

구현한 리스너를 시스템 프로퍼티로 지정한다. 여러 개의 리스너 클래스들을 공백, 세미콜론(;) 또는 콤마(,)을 separator로 하여 지정할 수 있다.


```
-Djeus.tm.profile.classes="test.profile.MyCoordinatorListener1,test.profile.MyCoordinatorListener2"
```

7.7. IP 대역이 다른 서버 간에 트랜잭션 통신 문제

JEUS 트랜잭션 매니저 간에 통신을 할 경우 IP 주소를 이용하여 접속한다. 하지만 이 경우 NAT 환경 안에 있는 JEUS 트랜잭션 매니저와 외부 환경에 있는 JEUS 트랜잭션 매니저가 통신할 때 서로 간의 IP 대역이 맞지 않아 통신이 불가능할 경우가 생긴다.

- Real IP를 Virtual IP로 매핑하는 프로퍼티를 지정해서 해결할 수 있다.

JEUS_HOME/templates/nat.properties.template을 참고하여 IP 매핑 프로퍼티 파일을 생성해서 다음과 같이 시스템 프로퍼티로 지정한다.

```
-Djeus.tm.net.address-mapping-properties=<full path of properties file>
```

8. Logging

본 장에서는 JEUS 로깅(Logging) 시스템에 대해 JEUS의 로거 구조 및 각 로거와 핸들러를 설정하는 방법, 로그 메시지의 내용에 대해서 설명한다.

8.1. 개요

JEUS 로깅(Logging)은 JEUS 실행 중에 시스템에서 수행되었던 일련의 작업들에 대한 내용을 순서대로 보관, 기록하는 작업이다. 이를 통해서 시스템 관리자는 JEUS에서 일어난 여러 작업들에 대한 내용을 이해하고, JEUS 실행 도중에 발생한 여러 에러 상황을 파악하여 그에 따른 대처를 할 수 있다.

JEUS는 시스템 및 애플리케이션에서 발생하는 여러 가지 상황들을 로그(log)를 통해 알려준다. JEUS는 Java SE에서 기본으로 제공되는 표준 Logging API(java.util.logging)를 사용한다. 따라서 로깅 시스템의 구조나 설정 방식도 로깅 API를 따르며 로거(logger), 핸들러(handler), 포맷터(formatter) 구조를 그대로 반영하고 있다. 또한, 개발자가 로깅 API를 이용하여 JEUS의 로거를 사용할 수 있다.

JEUS 로거 시스템은 Java SE Logging API를 기반으로 구현되었기 때문에 개발자는 이 API를 통해 JEUS 로깅 시스템을 Customize할 수 있다.

JEUS 로거의 종류는 JEUS Launcher와 JEUS 서버, 그리고 WEB 엔진에서 사용되는 액세스 로그가 있다. 각 로거는 "jeus 로거"(여기서 "jeus"는 로거의 이름, 이하 jeus 로거)를 기준으로 생성되어 있다. JEUS 서버는 모두 "jeus 로거"를 기본적으로 생성한다. 하위에 생기는 여러 가지 모듈들은 jeus.ejb 등과 같이 jeus 하위의 로거를 생성해서 사용한다.



"jeus 로거"는 JEUS에서 사용되는 최상위 로거의 이름이다. 사용자가 설정하지 않아도 이 로거는 기본적으로 존재한다.

이런 여러 가지 로거 중에서 JEUS 설정 파일에서 설정 가능한 로거는 다음과 같다.

- jeus 로거
- user 로거
- 웹 엔진에서 사용하는 액세스 로거
- jeus 로거의 하위 로거
- Java 로거

핸들러는 실제 로거에서 출력하려는 로그 메시지를 어떤 대상을 통해 기록하는 역할을 한다. 따라서 로거는 항상 핸들러를 동반한다. JEUS에서는 로그 메시지의 출력 대상에 따라 다음의 4가지 종류의 로거에 핸들러(handler)를 설정할 수 있고, 설정된 핸들러를 통해 로그 메시지가 기록된다. 이 중 가장 보편적으로 사용하는 것이 파일 핸들러(File Handler)이다.

구분	설명
파일 핸들러	로거를 통해 로깅되는 로그 메시지를 파일에 기록한다.
SMTP 핸들러	로그 메시지를 메일을 통해 기록한다.

구분	설명
소켓 핸들러	로그 메시지를 지정한 IP를 통해 기록한다.
user 핸들러	사용자가 작성한 핸들러를 통해 로그 메시지를 기록한다.

이 외의 핸들러에 대한 설명과 각 로거에 대한 핸들러를 JEUS 설정 파일에 설정하는 방법에 대해서는 [로거 정보 확인](#)에서 설명한다. 로깅 시스템에 대한 기본적인 이해는 Java SE Logging API를 참고한다.

버전별 주요 변경사항

다음은 각 버전별 주요 변경사항에 대한 설명이다.

1. JEUS 6까지는 콘솔 핸들러도 설정이 가능했지만 JEUS 7부터는 설정할 수 없다.
2. JEUS 7부터는 서버 프로세스의 콘솔은 존재하지 않는다. Launcher를 통해 서버를 부팅하기 때문에 Launcher 프로세스의 콘솔이지 서버 프로세스의 콘솔은 아니다.

서버에서 발생하는 로그 메시지들을 콘솔로 출력하기 위해서는 Launcher 프로세스가 서버가 shutdown될 때까지 다운되지 않아야 한다. 서버를 띄울 때 `-verbose` 옵션을 주고 띄우면 Launcher 프로세스는 서버가 다운될 때까지 다운되지 않고 서버에서 발생하는 로그 메시지들을 파이프를 통해 읽어 콘솔로 출력한다.

3. JEUS 7 Fix#4 버전부터 Asynchronous logging를 지원한다. 로깅을 log를 호출한 worker thread에서 처리하지 않고 전담 logger thread에서 처리함으로써 JEUS 전체적인 성능의 향상을 위함이다. 이 방식에서는 로깅이 지연될 수 있는 파일 핸들러 외의 나머지 핸들러는 지원하지 않는다. 만약 나머지 핸들러들을 쓰고 싶다면 `jeus.logging.useAsync` 옵션을 false로 설정한다. 또한 Asynchronous logging 방식에서는 `jeus.access.logging.skip.when.busy` 옵션을 true로 함으로써 성능에 민감한 web worker가 현재 로깅이 밀려있어 기다리게 되는 현상을 자동으로 detect하여 access logger만큼은 기다리지 않고 넘어갈 수 있도록 지원한다.
4. JEUS 8.5 버전부터 Asynchronous logger의 예외상황이 발생하는 경우에 이를 기록할 수 있는 Status logger를 도입한다.

Asynchronous logger로 로깅 작업 시에 문제가 발생했을 때에 원인을 찾을 수 있도록 하기 위함이다. 사용자가 별도로 설정할 수 없으며, 기본값으로 파일 핸들러(FileHandler)를 통해 문제 발생 시에 `JeusLoggerStatus.log` 파일에 문제 상황에 대해 로그를 기록한다. Asynchronous logger에 문제 상황이 발생하지 않는다면 `JeusLoggerStatus.log` 파일도 생성되지 않는다.

5. JEUS 8.5 버전부터 Pattern formatting을 지원해서 로깅 시에 출력할 로그의 포맷을 사용자가 지정할 수 있도록 한다.

JEUS의 서버 로그에 적용되며 총 8가지의 형식 문자열(%d, %l, %j, %T, %c, %M, %N, %m)을 지원한다. 사용자는 형식 문자열과 일반 문자를 조합하여 원하는 로깅 패턴을 설정할 수 있다(단, 형식문자열의 식별을 위하여 '%' 문자는 로깅 패턴에 포함시킬 수 없다).

JEUS 이전 버전에서 지원했던 SimpleFormatter와 SimpleMillisFormatter 또한 형식문자열과 문자열의 조합인 로깅 패턴으로 동일하게 출력할 수 있다.

- SimpleFormatter

```
[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
```

◦ SimpleMillisFormatter

```
[%d{yyyy.MM.dd HH:mm:ss:SSS}][%l] [%J-%T] [%M-%N] %m
```

로깅 패턴의 기본값은 기존의 SimpleFormatter와 동일한 패턴의 로그를 출력하는 로깅 패턴으로 되어 있다. 형식 문자열 등 자세한 설정은 [로거 정보 확인](#)에서 설명한다.

6. JEUS 21 버전부터 Asynchronous logger의 이벤트를 저장하기 위한 버퍼의 사이즈와 버퍼에 로그 이벤트가 없을 때 컨슈머가 사용할 대기 전략을 시스템 프로퍼티로 지정할 수 있다.

로그 이벤트의 양이 많다면 jeus.logging.async.bufferSize를 늘려서 이벤트 버퍼가 더 많은 이벤트를 수용할 수 있도록 할 수 있다. 기본값은 2048이며, 최대 262144까지 가능하다.

로그 이벤트 컨슈머의 대기 전략은 jeus.logging.async.waitStrategy를 통해 설정 가능하고, 기본값은 block이고 별도로 sleep, yield, busyspin, timeout을 지원한다.

7. JEUS 9 버전부터 각 서버의 log-home, File Handler의 rotation-dir 경로 설정 시 경로에 환경 변수 및 시스템 프로퍼티를 포함시킬 수 있다. 단, 변수 이름은 \${VAR}과 같은 형태로 명시되어 있어야 한다.

8.2. JEUS 로거 기본 구조

본 절에서는 로거를 사용하기 위한 기본 개념을 설명하고 실제 설정 방법에 대해서 설명한다.

8.2.1. 개요

서버에 로거를 설정하지 않아도 최상위 로거인 jeus 로거는 기본적으로 존재한다. 파일 핸들러(File Handler)를 사용하고 매일 로그 파일이 로테이션될 수 있도록 한다. 로그 로테이션을 비롯한 파일 핸들러에 대한 자세한 내용은 [로거 정보 확인](#)에서 설명한다.



jeus 로거는 로거 삭제 명령으로 삭제할 수 없다. jeus 로거를 삭제하면 다른 로거를 설정하지 않은 경우 서버에 아무런 로그 메시지도 로깅되지 않기 때문에 운영상 문제가 될 수 있다. 만약 서버에서 로거를 남기고 싶지 않다면 jeus 로거의 레벨을 OFF로 설정해야 한다.

도메인에 등록된 서버의 삭제 역시 해당 서버의 logs 디렉터리 삭제를 의미하지는 않는다. 로거는 관리자가 언제든지 열람할 수 있는 중요한 정보이기 때문에 필요없다고 판단하여 지우는 역할은 관리자의 몫이다.

로거 디렉터리 구조

로거에서 파일 이름을 별도로 지정하지 않았다면 JEUS에서는 정해진 위치에 로그 파일들을 생성한다. 로그 디렉터리의 구조는 다음과 같다.

```
SERVER_HOME
|--logs
```

```
|--servlet
|      |--access.log
|--JeusLauncher.log
|--JeusServer.log
|--JeusLoggerStatus.log
|--jvm.log
```

다음은 로그 디렉터리에 기본적으로 생성되는 로그 파일에 대한 설명이다.

servlet/access.log

웹 애플리케이션 요청에 대한 액세스 로그 파일이다. 기본적으로 서버의 모든 웹 애플리케이션으로 요청한 내용이 기록된다. 웹 엔진에 가상 호스트가 설정되어 있다면 servlet 디렉터리 하위에 가상 호스트 이름으로 디렉터리가 생성되고 그 하위에 access.log 파일을 생성하여 가상 호스트로 요청된 정보들이 로깅한다.

JeusLauncher.log

Launcher에서 서버 기동을 위해 남기는 정보와 서버를 부팅할 때 발생하는 로그 메시지를 로깅한다.

JeusServer.log

서버에서 로깅하는 기본 로그 파일이다. 서버에 로거관련 설정을 하지 않았다면 jeus 로거를 포함한 하위 모든 로거에 대한 기본 로그 파일이 된다.

JeusLoggerStatus.log

JEUS 7 Fix4부터 기본으로 설정되는 Asynchronous logger(서버 로거)의 로깅 중 예외를 기록하는 로그 파일이다. Asynchronous logger가 정상적으로 동작한다면 생기지 않지만 예외 상황이 발생하면 해당 예외를 로깅하며 생성된다.

jvm.log

서버 JVM에서 발생하는 gc 로그나 Thread Dump 등이 기록된다. Launcher에서 서버를 시작할 때 특정 JVM 옵션을 넣기 때문에 생성되는 파일이다.

Launcher에서 서버를 시작할 때 JVM 옵션에 다음을 추가한다.

```
-XX:+UnlockDiagnosticVMOptions -XX:+LogVMOutput -XX:LogFile=SERVER_HOME/logs/jvm.log
```

JVM 로그를 별도의 파일에 기록하도록 설정하는 이유는 서버가 백그라운드 프로세스로 동작하면서 서버의 Thread Dump가 더 이상 콘솔에 남지 않기 때문이다. 이 로그의 위치를 변경하려면 -XX:LogFile 옵션을 서버의 JVM 옵션에 추가해주면 기본 설정이 무시되고 사용자의 설정을 따른다.

사용자 정의 gc log file

Oracle의 경우 "-Xloggc:", IBM의 경우 "-Xverbosegclog:" 옵션을 사용하여 gc log file을 지정할 수 있다. 이때 JEUS에서는 지정된 파일 뒤에 자동으로 time stamp를 붙여서 JEUS를 띄울 때마다 생기는 gc log file을 구별할 수 있는 옵션을 제공한다.(기본값) 만약 time stamp를 붙이고 싶지 않다면 기동 스크립트에 -Djeus.logging.gclog.timestamp.on=false 라는 JVM 옵션을 추가한다.

특정 로그 디렉터리 설정

서버에서 생성하는 모든 로그(Rotation Backup 로그 포함)들을 특정 디렉터리에 저장하고 싶다면 domain.xml을

편집해야 하며 <server> 태그 하위에 <log-home> 태그를 통해서 경로를 설정할 수 있다.

```
<servers>
  <server>
    <name>server1</name>
    <node-name>node1</node-name>
    <log-home>${LOG_HOME}/logs</log-home>
    ...
```

log-home 경로에 시스템 환경 변수나 시스템 프로퍼티의 값을 넣을 수 있다. 이때 변수명은 \${ENV_NAME}와 같은 패턴이어야 하며, 등록되지 않은 환경 변수를 사용할 경우 디폴트 경로에 로그가 생성된다.

8.2.2. Launcher 로거

Launcher는 서버를 기동하기 위해 사용되는 프로세스이다. 실행 스크립트나 명령어를 통해 서버를 기동시키면 Launcher 프로세스가 실행되고, Launcher에서는 설정 파일을 읽고 실제 서버를 시작시킨다. Launcher에서는 서버가 부팅되면서 발생하는 로그 메시지를 JeusLauncher.log 파일과 콘솔에 남긴다. Launcher 프로세스는 일반적으로 서버가 부팅을 완료하면 종료하여 서버의 부팅 로그만 Launcher 로거를 통해 남긴다. 하지만 서버를 실행할 때 -verbose 옵션을 주었다면 Launcher 프로세스는 서버의 부팅이 완료된 후에도 종료되지 않고 서버가 다운될 때까지 남겨지는 모든 로그 메시지를 Launcher의 콘솔과 파일을 통해 남긴다.

서버에서 서버 로거를 통해 로깅하고 있는 로그 메시지들을 Launcher 프로세스에서 별도로 로깅하는 이유는 서버 부팅 중에 서버에서 로깅을 할 수 없는 상황에서 발생할 수 있는 오류 때문이다. 예를 들어 설정 파일 오류 등의 이유로 서버를 기동시키지 못했거나, 서버가 기동은 되었지만 부팅 실패로 인해 서버의 로거가 초기화되기 전에 종료되었다면 부팅 실패의 원인을 알 수 있는 로그 메시지들이 서버 로거를 통해 남을 수가 없다. 따라서 Launcher에서는 서버의 부팅 로그를 Launcher 자신의 로거를 통해 남겨서 사용자가 Launcher 로그를 확인하여 서버가 부팅할 때 발생한 오류를 해결할 수 있도록 한다. Launcher에 대한 자세한 내용은 JEUS Server 안내서의 "Launcher"를 참고한다.

다음은 설정 파일 오류로 인한 부팅에 실패할 때 Launcher에서 남긴 로그 메시지의 예이다.

```
[2016.08.24 12:38:52][0] [launcher-1] [XmlValidationEventHandler] [FATAL_ERROR]
Invalid content was found starting with element <system-clustering-framework>
(of parent element <domain>). One of '{password-validator, admin-server-name}'
is expected.

[2016.08.24 12:38:52][0] [launcher-1] [SERVER-0522] An exception occurred while processing
[domain.xml].
<<__Exception__>>
jeus.xml.binding.JeusJAXBException: Unmarshalling the XML descriptor failed:
domain.xml
class org.xml.sax.SAXParseException :
cvc-complex-type.2.4.a: Invalid content was found starting with element
'system-clustering-framework'. One of '{"http://www.tmaxsoft.com/xml/ns/jeus":password-validator,
"http://www.tmaxsoft.com/xml/ns/jeus":admin-server-name}' is expected..
  at jeus.xml.binding.BindingHelper.getDescriptor(BindingHelper.java:96)
  at jeus.service.descriptor.DescriptorFile.getDeploymentDescriptor(DescriptorFile.java:210)
  at
jeus.service.descriptor.JEUSDomainDescriptorFile.getConfigDescriptor(JEUSDomainDescriptorFile.java:54
)
  at jeus.launcher.Launcher.initDomainType(Launcher.java:222)
```

```
at jeus.launcher.Launcher.readDescriptor(Launcher.java:210)
at jeus.launcher.Launcher.start(Launcher.java:105)
at jeus.launcher.Launcher.main(Launcher.java:58)
```

8.2.3. 서버 로거

서버가 운영되는 동안 발생하는 로그 메시지를 로깅한다. 각 로그 메시지들은 서버가 운영 중에 하는 작업들에 대한 정보를 함축적으로 나타낸다.

JEUS의 기본 로거는 최상위 로거인 jeus 로거이고, jeus 로거의 하위 로거들은 jeus 로거의 핸들러를 통해 로깅된다. 서버에서 로깅되는 로거는 동적으로 추가, 삭제, 변경이 가능하다. 핸들러 역시 동적 추가, 삭제, 변경이 가능하다.

로그 메시지의 포맷은 formatter pattern을 통해 변경 가능하다. 아무런 설정이 없다면 다음과 같은 형태로 출력되는 JEUS에서 제공하는 기본 포맷 패턴인 '%d{yyyy.MM.dd HH:mm:ss}}[%l] [%J-%T] [%M-%N] %m'가 사용된다.

이전 버전에서 지원했던 포맷터인 SimpleFormatter와 SimpleMillisFormatter의 로깅 패턴은 다음과 같다.

- SimpleFormatter

```
[%d{yyyy.MM.dd HH:mm:ss}}[%l] [%J-%T] [%M-%N] %m
```

- SimpleMillisFormatter

```
[%d{yyyy.MM.dd HH:mm:ss:SSS}}[%l] [%J-%T] [%M-%N] %m
```

다음은 서버 로거의 형식에 대한 설명이다.

- 로그 형식

```
[%d{yyyy.MM.dd HH:mm:ss}} [%l] [%J-%T] [%M-%N] %m
```

항목	설명
%d{yyyy.MM.dd HH:mm:ss}	로그 이벤트의 날짜를 출력한다. 뒤에 '{}' 에 추가하는 DATE_FORMAT에는 출력하고자하는 날짜의 형식을 입력한다. DATE_FORMAT의 Formatting 방식은 JDK의 SimpleDateFormat 방식을 따른다.

항목	설명
%l	로그 이벤트의 로그 레벨이 그에 매핑되는 숫자로 출력된다. <ul style="list-style-type: none"> • 0 : SEVERE • 1 : WARNING • 2 : INFO • 3 : CONFIG • 4 : FINE • 5 : FINER • 6 : FINEST • 7 : ALL
%J	로그 메시지를 로깅하는 프로세스가 출력된다. Thread 정보를 나타낸다.
%T	로그 메시지를 로깅하는 Thread 번호가 출력된다. Thread 정보가 같은 로그 메시지는 같은 Thread에서 로깅한 것이다.
%M	로그를 출력하는 모듈의 이름이 출력된다. 각 모듈에 해당하는 이름은 로그 메시지 모듈 이름 을 참고한다.
%N	출력되는 로그 이벤트의 메시지 번호를 출력한다.
%m	운영 중에 발생한 일에 대한 의미를 함축하고 있는 로그 메시지이다.
%c	로그 메시지를 로깅하는 로거의 이름을 출력한다.



Launcher 프로세스에서 로깅되는 로그 메시지는 별도로 설정할 수 없으며 기본 formatter pattern을 따른다.

다음은 실제 JEUS 서버에 출력되는 로그 메시지의 예이다.

```
[2024.09.25 16:01:33][2] [ms1-1] [SERVER-0248] The JEUS server is STARTING.
[2024.09.25 16:01:34][0] [ms1-1] [SERVER-0000] Version information - JEUS 9 (9.0.0.0-b15).
[2024.09.25 16:01:34][0] [ms1-1] [SERVER-0001] java.specification.version=[17],
java.runtime.version=[17.0.2+8-86], vendor=[Oracle Corporation]
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0003] os.name=[linux], os.arch=[amd64], os.version=[4.18.0-408.el8.x86_64]
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0002] Domain=[domain1], Server=[ms1], baseport=[19736], pid=[17004]
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0004] The current system time zone : 한국 표준시
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0571] All JEUS system properties have been confirmed.
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0568] Service address='0.0.0.0:19736', hostname='JEUS-PC', representation ip='192.168.1.105'
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0561] The default RMI export port = 19743 and bind address = JEUS-PC/192.168.1.105.
[2024.09.25 16:01:34][2] [ms1-1] [NET-0002] Beginning to listen to NonBlockingChannelAcceptor: 0.0.0.0:19736.
```

우선, 첫 번째 로그 메시지는 2024년 9월 25일 오후 4시 1분 33초에 출력된 레벨 2(INFO 레벨)의 메시지라는 것을

알려준다. 이 로그 메시지는 ms1라는 서버에서 1번 Thread에 의해 로깅되었고, SERVER 모듈의 0248번 메시지가 출력되었음을 알 수 있다. 실제 이 로그 메시지는 JEUS 서버가 부팅을 시작했음을 알리는 서버의 최초 메시지이다.

다른 로그 메시지들 역시 첫 번째 메시지와 같은 프로세스에서 로깅되고, SERVER 모듈의 메시지가 출력된 것을 알 수 있다. 위 로그 메시지들은 서버가 부팅할 때 발생하는 것으로 JEUS 버전, Java 버전, 프로세스 ID, TimeZone, 네트워크 정보와 같은 환경 정보를 출력하고 있다.

JEUS에서는 서버 로거의 formatter pattern으로 기본 포맷 패턴으로 [%d{yyyy.MM.dd HH:mm:ss}] [%i] [%j-%T] [%M-%N] %m 이 설정되어있다. 이전 버전에서 SimpleFormatter가 출력하는 포맷 패턴이며 이전 버전에서 SimpleFormatter와 함께 제공했던 SimpleMillisFormatter의 포맷 패턴은 [%d{yyyy.MM.dd HH:mm:ss:SSS}] [%i] [%j-%T] [%M-%N] %m 이다.

다음은 SimpleMillisFormatter와 동일한 로그 형식을 출력하는 포맷 패턴을 설정했을 때 발생하는 로그 메시지의 예이다. 포맷터 설정에 대한 자세한 내용은 [로깅 설정](#)을 참고한다.

```
[2024.09.25 16:01:33][2] [ms1-1] [SERVER-0248] The JEUS server is STARTING.
[2024.09.25 16:01:34:385][0] [ms1-1] [SERVER-0000] Version information - JEUS 9 (9.0.0.0-b15).
[2024.09.25 16:01:34:385][0] [ms1-1] [SERVER-0001] java.specification.version=[17],
java.runtime.version=[17.0.2+8-86], vendor=[Oracle Corporation]
[2024.09.25 16:01:34:386][2] [ms1-1] [SERVER-0003] os.name=[linux], os.arch=[amd64],
os.version=[4.18.0-408.el8.x86_64]
[2024.09.25 16:01:34:387][2] [ms1-1] [SERVER-0002] Domain=[domain1], Server=[ms1], baseport=[19736],
pid=[17004]
[2024.09.25 16:01:34:387][2] [ms1-1] [SERVER-0004] The current system time zone : 한국 표준시
[2024.09.25 16:01:34:560][2] [ms1-1] [SERVER-0571] All JEUS system properties have been confirmed.
[2024.09.25 16:01:34:568][2] [ms1-1] [SERVER-0568] Service address='0.0.0.0:19736', hostname='JEUS-PC',
representation ip='192.168.1.105'
[2024.09.25 16:01:34:580][2] [ms1-1] [SERVER-0561] The default RMI export port = 19743 and bind
address = JEUS-PC/192.168.1.105.
[2024.09.25 16:01:35:030][2] [ms1-1] [NET-0002] Beginning to listen to NonBlockingChannelAcceptor:
0.0.0.0:19736.
```

8.2.4. 액세스 로거

액세스 로거는 사용자의 애플리케이션 요청이 어떻게 처리되는지를 남기기 위한 용도로 사용된다. 웹 애플리케이션으로의 요청에 대한 정보가 기록된다. 본 절에서는 웹 엔진에서의 액세스 로거에 대해 설명한다.

웹 엔진에서의 액세스 로거

웹 엔진 액세스 로거는 웹 엔진이 처리한 모든 요청을 기록한다. 액세스 로거로 기록되는 로그는 웹 애플리케이션에 접근한 정보와 기록할 내용을 지정하여 관리자가 필요한 정보를 얻을 수 있다. 웹 엔진으로 요청이 들어와 해당 요청을 모두 처리하고 그 응답을 보낸 후 액세스 로거는 설정된 정보를 기록한다.

웹 엔진 액세스 로거로 기록되는 로그들은 기본적으로 공통 로그 형식(Common Log Format)을 지원한다. 이 형식은 Apache와 같은 많은 곳에서 공통으로 사용하고 있으므로 로그를 액세스 분석하는 데 많은 정보를 얻을 수 있다. 특히 로그 분석 도구의 경우 공통 로그 형식을 분석하는 도구들이 많이 있으므로 이 도구들을 이용하면 웹 엔진의 액세스 로그를 분석하는 데 큰 도움이 된다.



JEUS 6까지는 공통 로그 형식을 지원하지 않았다.

그리고 웹 엔진의 경우 가상 호스트를 설정할 수 있는데 가상 호스트를 설정한 경우 가상 호스트별로 액세스 로거를 설정해서 별도의 액세스 로그를 수집할 수 있다. 이때 가상 호스트의 액세스 로그는 다음의 파일명으로 남게 된다.

```
SERVER_HOME/logs/servlet/<가상 호스트명>/access.log
```

웹 엔진 액세스 로그의 기본 형식으로 액세스 로그를 남겼다면 다음과 같은 로그들이 access.log에 남는다.

```
192.168.15.57 [29/Aug/2016:17:37:02 +0900] "GET /example/test1.jsp HTTP/1.1" 200 5 38
```

위 로그는 192.168.15.57 IP로부터 /example/test1.jsp 요청의 결과 정상적으로 38Byte의 응답을 29/Aug/2016:17:37:02 +0900 시간에 보냈다는 것을 의미한다. 보다 자세한 로그 형식은 공통 로그 형식에 대한 문서를 참고한다.

기록되는 로그의 형식을 바꾸거나 그 외의 웹 엔진 액세스 로거의 설정에 대한 자세한 내용은 JEUS Web Engine 안내서의 "액세스 로그 기본 설정"을 참고한다.

8.2.5. 사용자 로거

JEUS의 각 서버마다 제공되는 사용자 로거(user logger)는 개발자가 별도의 로거를 사용할 필요없이 JEUS에서 제공하는 로거를 사용할 수 있도록 한다. Java SE Logging API의 java.util.logging.logger API를 사용해서 사용자 로거를 사용할 수 있다.

8.2.6. 로거 리스트

다음은 로거 목록에 대한 설명이다.

- EJB 관련

구분	설명
jeus.ejb.bean	EJB Home/Object Stub 관련 로거
jeus.ejb.cluster	EJB 클러스터링 관련 로거
jeus.ejb.compiler	EJB Stub Compiler 관련 로거
jeus.ejb.connector	MDB와 리소스 어댑터 관련 로거
jeus.ejb.container	EJB 컨테이너 관련 로거
jeus.ejb.ejbserver	EJB 엔진 관련 로거
jeus.ejb.interop	EJB CORBA 연동 관련 로거
jeus.ejb.persistence	CMP 관련 로거

구분	설명
jeus.ejb.schema	EJB QL 관련 로거
jeus.ejb.timer	EJB Timer 관련 로거
jeus.ejb.transaction	EJB Transaction & Synchronization 관련 로거
jeus.ejb.webserver	EJB Class FTP 관련 로거
jeus.ejb.util	기타 로거

- JPA 관련

구분	설명
jeus.persistence	JPA 모듈의 최상위 로거
jeus.persistence.provider	TopLink Essentials(default provider)의 최상위 로거
jeus.persistence.container	JPA에 대한 컨테이너 로거

- Servlet 관련

구분	설명
jeus.servlet.common	Servlet 공통 모듈 관련 로거
jeus.servlet.connector	Connector 관련 로거
jeus.servlet.connector	NIO Connector 관련 로거
jeus.servlet.deployment	Deploy 관련 로거
jeus.servlet.engine	Servlet 주요 처리 과정 관련 로거
jeus.servlet.filter	필터 관련 로거
jeus.servlet.jsp	JSP 관련 로거
jeus.servlet.listener	Servlet Listener 관련 로거
jeus.servlet.loader	클래스 로더 관련 로거
jeus.servlet.property	프로퍼티 관련 로거
jeus.servlet.servlets	JEUS에서 제공하는 Servlet 관련 로거
jeus.servlet.util	유틸리티에서 사용하는 로거
jeus.websocket	websocket 서버에 대한 모든 로거
jeus.webserver	Class FTP 서비스에서 사용하는 로거

- Session Manager 관련

구분	설명
jeus.session	Session Manager의 최상위 로거, 공통적으로 사용되는 로거
jeus.session.distributed	분산식 Session Manager의 로거

- 웹 서비스 관련

- JAX-RPC/SAAJ 로거

구분	설명
jeus.webservices.client	jeus.webservices.client 패키지(client invocation framework) 관련 로거
jeus.webservices.encoding	SOAP serialize/deserialize 관련 로거
jeus.webservices.message	SAAJ 및 SOAP 메시지 관련 로거
jeus.webservices.wsdl	WSDL 처리 관련 로거
jeus.webservices	그 외 웹 서비스 관련 로거

- UDDI 로거

구분	설명
jeus.uddi.datastore	DB 프로세싱 관련 로거
jeus.uddi.function	UDDI API 메시지 프로세싱 관련 로거
jeus.uddi.judy	Registry Server Engine 관련 로거
jeus.uddi.registry	Transport layer, Request/Response 프로세싱, Registry 엔진 관련 로거

- WS-* 로거 (JAX-RPC 기반)

구분	설명
jeus.webservices.wss	ws-security 관련 로거

- 기타

구분	설명
jeus.xml.binding. webservicesHelp er	EWS(JSR109)에 사용되는 DD 바인딩 관련 로거

- 트랜잭션 관련

구분	설명
jeus.transaction	트랜잭션 매니저 전반적으로 사용하는 로거
jeus.transaction.log ging	Recovery에 사용되는 Resource Factory 관련 로거
jeus.transaction.ots	OTS 관련 로거
jeus.transaction.rec overy	트랜잭션 Recovery 작업 내용을 기록하는 로거

- Security 관련

구분	설명
jeus.security	JEUS Security 관련 로거
jeus.security.impl.lo gin	JEUS Security 로그인 서비스 관련 로거
jeus.security.util	JEUS Security 유틸리티 관련 로거

- 기타 주요 로거

구분	설명
jeus.classloader	JEUS 클래스 로딩 관련 로거
jeus.clustering	JEUS 서버 클러스터링 관련 로거
jeus.config	JEUS 동적 설정 변경 관련 로거
jeus.connector	Jakarta EE Connector 관련 로거
jeus.converter	XML 컨버터 관련 로거
jeus.ddinit	DD Initializer 관련 로거
jeus.deploy	애플리케이션 Deploy 관련 로거
jeus.domain	도메인 관련 로거
jeus.filetransfer	설정 파일, 애플리케이션 파일 전송 관련 로거
jeus.io	JEUS Network I/O Library 관련 로거
jeus.jdbc	JDBC Connection Pool 관련 로거
jeus.jmx	JMX 관련 로거
jeus.jndi	JNDI 관련 로거

구분	설명
jeus.jnlp	JNLP 관련 로거
jeus.jtmax	JTmax 관련 로거
jeus.management	JMX MBean Framework 관련 로거
jeus.net	JEUS Network API 관련 로거
jeus.rmi	JEUS RMI 관련 로거
jeus.scheduler	JEUS 스케줄러 관련 로거
jeus.service	JEUS 서비스 MBean 관련 로거
jeus.weld	JEUS CDI 관련 로거
jeus.logger.status	JEUS Asynchronous logger(서버 로거)의 예외를 기록하는 로거

8.2.7. 로그 메시지 모듈 이름

JEUS에서 제공되는 로그 메시지는 여러 가지 정보를 제공하고 있다. 그 중 로그 메시지의 형식에 따라 모듈의 정보를 출력하는 로그 메시지 정보에서는 각 모듈의 이름을 확인할 수 있다. 본 절에서는 로그에 출력되는 각 모듈 이름과 이에 해당하는 실제 모듈들에 대해 설명한다.

모듈 이름	모듈 정보
Connector	Connector
Console	콘솔 커맨드
Config	설정 파일 동기화와 관련 모듈
CORBA	CORBA
CPOOL	Connection
D_Session	분산식 세션 서버
Deploy	애플리케이션 Deploy
Domain	도메인
EJB	EJB 엔진
JDBC	JDBC
JMS	JMS 엔진
JMSC	JMS Clustering
JMSF	JMS Failover
JMX	JMX
JMXR	JMX Remote
JNDI.Common	JNDI 공통
JNDI.Context	JNDI Context
JNDI.Local	JNDI Local 클라이언트

모듈 이름	모듈 정보
JNDI.Remote	JNDI Remote 클라이언트
JNSS	JNDI 서버
JPA	JPA
JTMAX	JTmax
Launcher	JEUS Launcher (Launcher 프로세스)
Network	JEUS 네트워크
OTS	OTS
SCF	SCF(JEUS System Clustering Framwork)
Scheduler	스케줄러
Secutiry	JEUS Security
SERVER	서버 부트, 다운, 모니터링 관련 모듈
Session	세션 서버
TM	트랜잭션 매니저
TMRecovery	트랜잭션 매니저 Recovery
UDDI	UDDI
WEB	서블릿 엔진
WebT	WebT
WebtobLight	WebtoB
WSS	웹 서비스 Security
WSVC	웹 서비스

8.3. 로깅 설정

본 절에서는 JEUS에서 로깅에 대한 설정 방법과 Customization 방법 등을 설명한다.

8.3.1. 로거 정보 확인

jeus 로거는 설정이 없더라도 기본적으로 존재한다. 그러나 jeus 로거를 제외한 다른 로거들은 설정하지 않으면 상위 로거(jeus 로거)의 핸들러(handler)를 사용해서 로그 메시지를 출력한다. jeus 로거는 설정되지 않으면 파일 핸들러(File Handler)를 사용한다. 따라서 서버에 아무런 로거 설정을 하지 않았다면 서버가 운영되면서 발생하는 로그 메시지는 디폴트 로그 파일에 남게 된다. 이때 로그 레벨(level)은 INFO이다. jeus 로거는 상위 로거의 핸들러를 사용하지 않는다.



서버의 로그 메시지를 콘솔에서 확인하고 싶다면 서버를 부팅할 때 `-verbose` 옵션을 설정한다. 옵션을 설정하면 Launcher 프로세스가 서버를 부팅시킨 후 종료되지 않고 서버가 shutdown될 때까지 살아있으면서 서버의 로그 메시지를 파이프를 통해 읽어서 콘솔에

출력하는 기능을 한다.

본 절에서는 콘솔 툴을 사용해서 서버에 기본적으로 존재하는 jeus 로거에 대한 정보를 확인하는 예를 설명한다.

콘솔 툴 사용

콘솔 툴에서 다음과 같이 **log-level**, **list-log-handlers** 명령어를 수행하면 jeus 로거와 그에 등록된 핸들러에 대한 정보를 확인할 수 있다. 각 명령어의 사용 방법에 대한 자세한 내용은 JEUS Reference 안내서의 "Server Management 관련 명령어"를 참고한다.

```
[MASTER]domain1.adminServer>log-level -server server1 jeus
The logger[jeus] information for the server [server1]
Information about the logger[jeus].
=====
Logger Name : jeus
Level : INFO
Use Parent Handlers : false

+-----+-----+-----+
| Handler Name | Handler Type | Handler Level |
+-----+-----+-----+
| jeus.util.logging.ConsoleHandler@1698156408 | ConsoleHandler | ALL |
| FileHandler | FileHandler | FINEST |
+-----+-----+-----+
=====

[MASTER]domain1.adminServer>modify-logger -server server1 jeus
Show the current configuration.
The logger[jeus] information for the server [server1]
=====
+-----+-----+
| Name | Value |
+-----+-----+
| Level | INFO |
| Use Parent Handlers | false |
| Formatter | [%d{yyyy.MM.dd HH:mm:ss}] [%l] [%J-%T] [%M-%N] %m |
+-----+-----+
=====

[MASTER]domain1.adminServer>list-log-handlers -server server1 jeus
List of Loggers
=====
+-----+-----+
| Handler Name | FileHandler |
| Handler Type | FileHandlerType |
| Handler Level | FINEST |
| Filename | JeusServer.log |
| Enable Rotation | true |
| Rotation Directory | ${SERVER_HOME}/logs |
| Valid Day | 1 |
| Buffer Size | 1024 |
| Append Logs | true |
+-----+-----+
=====
```


콘솔 툴을 통해 jeus 로거 정보를 확인했을 때 콘솔 핸들러가 보여지는 경우는 2가지이다.

- 서버를 부팅할 때 실행 스크립트에 `-verbose` 옵션을 설정한 경우
- 서버가 부팅 중인 경우

서버를 부팅할 때 `-verbose` 옵션을 설정한 경우는 Launcher 프로세스로 콘솔 로그가 출력된다. 또한 서버가 부팅 중인 경우에도 Launcher 프로세스를 통해서 로그 메시지가 콘솔 화면에 출력될 수 있다.

8.3.2. 동적으로 로거 설정

콘솔 툴을 통해서 런타임(runtime)에 동적으로 로거나 핸들러를 추가, 삭제, 변경할 수 있다.

콘솔 툴 사용

콘솔 툴에서 **add-logger**, **modify-logger**, **remove-logger** 명령어를 사용해서 로깅 설정을 동적으로 변경할 수 있다. 각 명령어에 대한 자세한 내용은 JEUS Reference 안내서의 "Server Management 관련 명령어"를 참고한다.

list-loggers 명령어를 사용하여 현재 설정되어 있는 로거 정보를 확인할 수 있다.

```
[MASTER]domain1.adminServer>list-loggers server1
```

```
List of Loggers
```

```
=====
+-----+-----+-----+-----+-----+
| Logger Name | Level | Use Parent | Filter | Formatter |
|             |       | Handlers   |        |            |
+-----+-----+-----+-----+-----+
| jeus        | INFO  | false      |        | [%d{yyyy.MM.dd HH:mm:ss}][%l] |
|             |       |            |        | [%J-%T] [%M-%N] %m            |
+-----+-----+-----+-----+-----+
=====
```

```
[MASTER]domain1.adminServer>add-logger -server server1 jeus.ejb -level FINEST
```

```
Successfully performed the ADD operation for The logger for the server(server1)..
Check the results using "list-loggers or add-logger".
```

```
[MASTER]domain1.adminServer>add-logger -server server1 jeus.ejb.clustering -level FINEST
```

```
Successfully performed the ADD operation for The logger for the server(server1)..
Check the results using "list-loggers or add-logger".
```

```
[MASTER]domain1.adminServer>list-loggers server1
```

```
List of Loggers
```

```
=====
+-----+-----+-----+-----+-----+
| Logger Name | Level | Use Parent | Filter | Formatter |
|             |       | Handlers   |        |            |
+-----+-----+-----+-----+-----+
| jeus        | INFO  | false      |        | [%d{yyyy.MM.dd |
|             |       |            |        | HH:mm:ss}][%l] [%J-%T] |
|             |       |            |        | [%M-%N] %m            |
+-----+-----+-----+-----+-----+
| jeus.ejb    | FINEST | true       |        | [%d{yyyy.MM.dd |
|             |       |            |        | HH:mm:ss}][%l] [%J-%T] |
+-----+-----+-----+-----+-----+
```

				[%M-%N] %m	
jeus.ejb.clustering	FINEST	true		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T]	
				[%M-%N] %m	

```
[MASTER]domain1.adminServer>modify-logger -server server1 jeus.ejb.clustering -level FINE
Successfully performed the MODIFY operation for The logger[jeus.ejb.clustering] information for the
server [server1].
Check the results using "modify-logger".
```

```
[MASTER]domain1.adminServer>list-loggers server1
List of Loggers
```

Logger Name	Level	Use Parent Handlers	Filter	Formatter
jeus	INFO	false		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
jeus.ejb	FINEST	true		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
jeus.ejb.clustering	FINE	true		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m

```
[MASTER]domain1.adminServer>remove-logger -server server1 jeus.ejb
Successfully performed the REMOVE operation for The logger for the server(server1)..
Check the results using "list-loggers or remove-logger".
```

```
[MASTER]domain1.adminServer>list-loggers server1
List of Loggers
```

Logger Name	Level	Use Parent Handlers	Filter	Formatter
jeus	INFO	false		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
jeus.ejb.clustering	FINE	true		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m

8.3.3. 표준 출력과 표준 에러를 로그 형식으로 출력 설정

JEUS에서는 표준 출력과 표준 에러를 JEUS 로그 형식으로 출력하는 기능을 제공한다. JEUS에서 기본으로 사용하는

포맷을 이용하여 jeus 로거와 비슷한 형식으로 표준 출력과 표준 에러를 출력할 수 있다.

본 절에서는 콘솔 툴을 통해 표준 출력과 표준 에러를 JEUS 로그 형식으로 출력하도록 설정하는 방법에 대해 설명한다.

콘솔 툴 사용

콘솔 툴에서 표준 출력과 표준 에러를 JEUS 로그 형식으로 출력하는 설정에 대한 예제이다.

```
[MASTER]domain1.adminServer>modify-server server1
Shows the current configuration.
server (server1)
=====
+-----+-----+
| JVM Configs          | -Xmx1024m -XX:MaxMetaspaceSize=512m |
| Action On Resource Leak | WARNING                               |
| Stdout to Raw Format   | true                                 |
| MEJB                  | false                               |
| Class FTP             | false                               |
| Server Log Home Directory | none                               |
+-----+-----+
=====

[MASTER]domain1.adminServer>modify-server server1 -logStdoutToRawFormat false
Successfully performed the MODIFY operation for server (server1).
Check the results using "list-servers server1 or modify-server server1".

[MASTER]domain1.adminServer>list-servers server1
List of Editable Servers
=====
+---+---+---+---+---+---+---+---+---+---+---+
| Ser | Base | Node | JVM | Action | Stdout | MEJB | Cla | Server | Type |
| ver | Listen |   | Configs | On | to Raw |   | ss | Log Home |   |
|   | Address |   |   | Resource | Format |   | FTP | Directo |   |
|   | /Port  |   |   | Leak   |   |   |   | ry     |   |
+---+---+---+---+---+---+---+---+---+---+---+
| ser | 0.0.0.0 |   | -Xmx102 | Warning | false | fal | fal | none | ser |
| ver1 | / 9836 |   | 4m      |   |   | se | se |   | ver |
|   |   |   | -XX:MaxM |   |   |   |   |   |   |
|   |   |   | etaspace |   |   |   |   |   |   |
|   |   |   | Size=512m |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+
=====
```

이 기능을 사용하면 표준 출력은 다음과 같은 포맷으로 출력된다.

```
[%d{yyyy.MM.dd HH:mm:ss}] [%l] [%J-%T] [%M-%N] [STDOUT/STDERR] %m
```

항목	설명
%d{yyyy.MM.dd HH:mm:ss}	로그 이벤트의 날짜를 출력한다. 뒤에 '{}'에 추가하는 DATE_FORMAT에는 출력하고자하는 날짜의 형식을 입력한다. DATE_FORMAT의 Formatting 방식은 JDK의 SimpleDateFormat 방식을 따른다.
%l	로그 이벤트의 로그 레벨이 그에 매핑되는 숫자로 출력된다. <ul style="list-style-type: none"> ◦ 0 : SEVERE ◦ 1 : WARNING ◦ 2 : INFO ◦ 3 : CONFIG ◦ 4 : FINE ◦ 5 : FINER ◦ 6 : FINEST ◦ 7 : ALL
%J	로그 메시지를 로깅하는 프로세스가 출력된다. Thread 정보를 나타낸다.
%T	로그 메시지를 로깅하는 Thread번호가 출력된다. Thread 정보가 같은 로그 메시지는 같은 Thread에서 로깅한 것이다.
%M	로그를 출력하는 모듈의 이름이 출력된다. 각 모듈에 해당하는 이름은 로그 메시지 모듈 이름 을 참고한다.
%N	출력되는 로그 이벤트의 메시지 번호를 출력한다.
%m	System.out이나 System.err를 통해 출력할 메시지이다. 만약 표준 출력이나 표준 에러에 대한 로그이면 메시지 앞에 [STDOUT] 혹은 [STDERR]와 같은 메시지가 추가되어 출력된다. <ul style="list-style-type: none"> ◦ STDOUT : 로깅하려는 메시지가 표준 출력인 경우이다. ◦ STDERR : Exception과 같은 표준 에러인 경우이다. ◦ UNKNOWN : 알 수 없는 경우이다.
%c	로그 메시지를 로깅하는 로거의 이름을 출력한다.

8.3.4. 로거 설정

콘솔 툴을 사용하여 로거나 핸들러를 추가, 삭제, 수정할 수 있고 이는 모두 동적 반영 가능하다. 즉, 운영 중인 서버를 재기동하지 않아도 변경한 설정 내용이 적용될 수 있다.



로거에서 동적 반영 가능한 설정은 레벨(level)과 상위 핸들러를 사용할지 여부(use-parent-handlers)이다. 핸들러에서는 레벨(level)만 동적반영 가능하다. 또한 콘솔 툴 명령어로는 파일 핸들러만 추가, 변경이 가능하다.

Logger 추가

다음과 같이 콘솔 툴을 사용하여 로거를 추가하거나 사용 중인 로거를 조회할 수 있다.

```
[MASTER]domain1.adminServer>add-logger logger2 -level SEVERE -useParentHandlers false -server server1
Successfully performed the ADD operation for The logger for the server(server1)., but all changes
were non-dynamic. They will be applied after restarting.
Check the results using "list-loggers or add-logger".
[MASTER]domain1.adminServer>list-loggers server1
List of Loggers
=====
+-----+-----+-----+-----+-----+
| Logger Name | Level | Use Parent | Filter | Formatter |
|             |       | Handlers   |        |            |
+-----+-----+-----+-----+-----+
| jeus        | INFO  | false      |        | [%d{yyyy.MM.dd |
|             |       |            |        | HH:mm:ss}][%l] [%J-%T] |
|             |       |            |        | [%M-%N] %m      |
+-----+-----+-----+-----+-----+
| logger2     | SEVERE | false      |        | [%d{yyyy.MM.dd |
|             |       |            |        | HH:mm:ss}][%l] [%J-%T] |
|             |       |            |        | [%M-%N] %m      |
+-----+-----+-----+-----+-----+
=====
```

다음은 설정 항목에 대한 설명이다.

항목	설명
server-name	로거를 추가할 서버의 이름이다.
logger-name	로거의 이름을 설정한다.
log-level	로거의 레벨(level)을 설정한다. 이 레벨 이하의 메시지만 로거를 통해 출력된다. Logging API의 레벨인 다음의 값이 설정된다. <ul style="list-style-type: none">◦ SEVERE◦ WARNING◦ INFO (기본값)◦ CONFIG◦ FINE◦ FINER◦ FINEST◦ ALL
use-parent-handlers	로거가 자신의 핸들러뿐만 아니라 상위 로거의 핸들러를 사용할 것인지의 여부를 설정한다. 기본값은 true이지만 jeus 로거의 경우는 JEUS의 최상위 로거이기 때문에 false로 설정한다. 또한 자신의 핸들러를 설정하지 않고 이 값을 false로 설정할 경우 상위 핸들러를 사용할 수 있도록 이 값을 true로 변경한다.

항목	설명
filter-class	로거가 로그 메시지를 핸들러에게 보내기 전에 수행하는 필터링(filtering)에 사용할 클래스를 지정한다. 여기에 지정된 클래스는 lib/application 디렉터리의 JAR 파일 내에 포함되어 있어야 한다.
formatter-pattern	로거가 로그 메시지를 핸들러에 보내기 전에 로그 메시지를 포맷팅하는데 이때 사용하는 포맷 패턴을 지정한다. (기본값: [%d{yyyy.MM.dd HH:mm:ss}] [%l] [%J-%T] [%M-%N] %m)



JEUS 4.x에서 사용하던 로그 레벨인 FATAL, NOTICE, INFORMATION, DEBUG 레벨은 JEUS 7부터는 사용할 수 없다.

Handler 추가

Handler에는 파일 핸들러, SMTP 핸들러, 소켓 핸들러, 사용자 핸들러가 있으며 각 핸들러에 대해 다음과 같은 설정들이 있다.

• 파일 핸들러

파일 핸들러는 파일로 로그 메시지를 출력하는 핸들러이다. 다음과 같이 jeusadmin 명령어를 통해 추가할 수 있다.

```
[MASTER]domain1.adminServer>add-handler handler2 -server server1 -logger logger2 -level FINEST
-filename logFile2.log -enable false -day 1
Successfully performed the ADD operation for The handler for the logger(logger2) on the
server(server1)., but all changes were non-dynamic. They will be applied after restarting.
Check the results using "list-log-handlers or add-log-handler".
```

항목	설명
Name	핸들러를 나타내는 이름을 지정한다. 이름은 하나의 로거 내에서 유일해야 한다. 만약 지정하지 않으면 클래스 이름과 Hash Code를 조합한 이름이 사용된다.
File Name	핸들러가 로그 메시지를 출력할 파일의 이름을 지정한다. 절대 경로로 되어 있다면 그 경로로 파일이 생성되고 상대 경로라면 각 로거의 기본 경로를 기준으로 한 상대 경로로 인식한다. 설정하지 않으면 각 로거별로 지정된 경로로 파일을 생성해서 로그 메시지를 출력한다.
Level	핸들러가 출력할 메시지의 레벨을 지정한다. 로그 메시지를 출력할 때 로거를 통과한 로그 메시지가 해당 로거가 사용하는 각각의 핸들러에게 전달되는데, 이 핸들러의 레벨에 부합하는 로그 메시지만 해당 핸들러에 의해 출력된다. 기본값으로 설정하면 로거를 통과하는 모든 로그 메시지가 핸들러에 의해 출력된다. (기본값: FINEST)
Log chown	Log 파일의 소유자, 소유그룹 설정을 한다. ,(comma)로 구분하는 소유자와 소유그룹을 순서대로 입력한다.

항목	설명
Log Permission	Log 파일의 접근 권한 설정을 한다. 9자리의 r,w,x,-로 이루어진 log permission을 주도록 한다. Unix 계열 OS를 사용하는 경우에만 유효하다.
Enable Rotation	핸들러가 로그 파일 로테이션 기능을 사용할 것인지 설정한다. 기본값으로 설정하면 로테이션 기능을 사용한다. (기본값: true)
Rotation Count	로테이션의 횟수를 설정한다. 핸들러가 로그 파일 로테이션 기능을 사용할 때만 의미가 있다. (기본값: 10)
Valid Day	핸들러가 출력할 파일을 날짜별로 생성할 경우에 사용한다. 파일 이름의 형식은 파일 끝에 "_YYYYMMDD"가 붙는다.
Valid Hour	핸들러가 출력할 파일을 시간별로 생성할 경우에 사용한다. 24의 약수(예: 3, 6)이거나 24로 나눈 나머지가 약수(예: 27, 30)인 값을 지정한다. 파일 이름의 형식은 끝에 "_YYYYMMDD_HH"가 붙는다.
Valid Size	핸들러가 출력할 파일을 크기별로 생성할 경우에 설정하는 항목으로 핸들러가 로그 파일 로테이션 기능을 사용할 때만 의미가 있다.
Encoding	핸들러가 출력하는 문자열의 인코딩을 지정한다. 기본은 시스템 인코딩으로 설정되어 있다.
Filter Class	핸들러가 로그 메시지를 필터링해서 출력하고자 할 때 이용되는 클래스이다. 로거의 Filter Class와 마찬가지로 lib/application에 해당 클래스를 포함한 JAR 파일이 존재해야 한다.
Append	서버를 기동한 뒤 로그를 파일로 출력하려고 할 때 이미 같은 이름의 파일이 존재하면 덮어쓰지, 파일 끝에 추가할지를 결정한다. <ul style="list-style-type: none"> ◦ true : 파일 끝에 계속 추가한다. (기본값) ◦ false : 로테이션을 사용 여부에 따라서 의미가 달라진다. 로테이션을 사용하고 있다면 서버를 기동할 때마다 이전 로그 파일은 백업되고 새로운 로그 파일이 생성된다. 그러나 로테이션을 사용하고 있지 않다면 서버를 기동할 때마다 이전 로그 파일이 삭제되므로 주의해서 사용해야 한다.
Rotation Dir	핸들러가 로그 파일 로테이션 기능을 사용할 때만 의미가 있다. 시스템 환경 변수나 시스템 프로퍼티의 값을 사용해 경로를 지정할 수 있다. 예를 들어 <rotation-dir>\${JEUS_HOME}/rotatedLog/\${jeus.server.name}</rotation-dir>와 같이 경로를 설정할 수 있다.
Buffer Size	파일로 출력할 때 사용할 버퍼(buffer)의 크기를 지정한다. 버퍼가 클수록 로깅의 성능은 좋아지지만 예상치 못한 상황으로 인해 JEUS가 비정상 종료될 때에는 그 버퍼 크기만큼 로그가 손실될 수도 있다. (기본값: 1024, 단위: KB) 버퍼에 로그 메시지를 축적했다가 버퍼의 크기가 설정한 크기보다 커지면 로그 메시지를 파일에 출력한다.

• SMTP 핸들러

SMTP 핸들러는 로그 메시지를 이메일로 전송하는 핸들러이다. 하나의 로그 메시지가 하나의 이메일로 전송된다.

파일 핸들러 이외의 핸들러를 추가하기 위해서는 **jeus.logging.useAsync** 옵션을 **false**로 설정해야 하며, 다음과 같이 domain.xml을 편집하여 추가할 수 있다.

```
<smtp-handler>
  <name>smtphandler</name>
  <level>WARNING</level>
  <smtp-host-address>gw.tmaxsoft.com</smtp-host-address>
  <sender-id>sender@tmaxsoft.com</sender-id>
  <sender-password>1234567</sender-password>
  <from-address>sender@tmaxsoft.com</from-address>
  <to-address>receiver@tmaxsoft.com</to-address>
  <property>
    <key>mail.smtp.socketFactory.port</key>
    <value>465</value>
  </property>
  <property>
    <key>mail.smtp.socketFactory.class</key>
    <value>javax.net.ssl.SSLSocketFactory</value>
  </property>
  <property>
    <key>mail.smtp.auth</key>
    <value>true</value>
  </property>
  <send-for-all-messages>true</send-for-all-messages>
</smtp-handler>
```

항목	설명
Name	핸들러가 툴에서 보여질 때 사용할 이름을 지정한다. 이름은 하나의 로거 내에서 유일해야 한다. 만약 지정하지 않으면 클래스 이름과 Hash Code 이름이 대체된다.
Level	핸들러가 출력할 메시지의 레벨을 지정한다. 즉, 로거를 통과한 로그 메시지가 해당 로거가 사용하는 각각의 핸들러에게 전달되는데, 이 핸들러의 레벨에 부합하는 로그 메시지만 해당 핸들러에 의해 출력된다. 기본값으로 설정하면 로거를 통과하는 모든 로그 메시지가 핸들러에 의해 출력된다. (기본값: FINEST)
Smtp Host Address	이메일을 보낼 호스트(host)의 주소를 지정한다.
From Address	이메일을 보내는 사람의 주소를 지정한다.
Sender ID	이메일을 보내는 사람의 ID를 명시한다.
Sender Password	이메일을 보내는 사람의 패스워드를 지정한다. 암호화해서 저장할 때에는 "{암호화할 알고리즘}암호화된 패스워드"와 같은 형식으로 입력한다.
To Address	이메일을 받는 사람의 주소를 지정한다.
Property	사용하는 메일 서버마다 요구하는 특정한 SMTP 프로퍼티가 있을 수 있다.
Send For All Messages	모든 메시지를 SMTP 핸들러로 보낼지를 결정한다. false이면 JEUS 시스템에서 이메일로 전송하기로 결정되어 있는 메시지만 이 핸들러를 사용해서 보내진다. 현재 이 설정은 사용자 로거에만 유효하다. (기본값: false)
Encoding	핸들러가 출력하는 문자열의 인코딩을 지정한다. 기본은 시스템 인코딩으로 설정되어 있다.

항목	설명
Filter Class	핸들러가 로그 메시지를 출력하기 전에 수행할 필터링에 이용되는 클래스이다. 로거의 <filter-class>와 마찬가지로 lib/application에 이 클래스를 포함한 JAR 파일이 존재해야 한다.
Cc Address	이메일을 참조로 받는 사람의 주소를 지정한다.
Bcc Address	이메일을 숨은 참조로 받는 사람의 주소를 지정한다.

• 소켓 핸들러

소켓 핸들러는 로그 메시지를 소켓으로 전송하는 핸들러이다.

항목	설명
Name	핸들러가 톨에서 보여질 때 사용할 이름을 지정한다. 이름은 하나의 로거 내에서 유일해야 한다. 지정하지 않으면 클래스 이름과 Hash Code 이름이 대체된다.
Level	핸들러가 출력할 메시지의 레벨을 지정한다. 즉, 로거를 통과한 로그 메시지가 해당 로거가 사용하는 각각의 핸들러에게 전달되는데, 이 핸들러의 레벨에 부합하는 로그 메시지만 해당 핸들러에 의해 출력된다. 기본값으로 설정하면 로거를 통과하는 모든 로그 메시지가 핸들러에 의해 출력된다. (기본값: FINEST)
Address	핸들러가 접속할 머신(machine)의 IP 주소를 지정한다.
Port	핸들러 접속할 머신의 포트 번호를 지정한다.
Encoding	핸들러가 출력하는 문자열의 인코딩을 지정한다. 기본은 시스템 인코딩으로 설정되어 있다.
Filter Class	핸들러가 로그 메시지를 출력하기 전에 수행할 필터링에 이용되는 클래스이다. 로거의 <filter-class>와 마찬가지로 lib/application에 이 클래스를 포함한 JAR 파일이 존재해야 한다.

• 사용자 핸들러

사용자 핸들러는 사용자가 생성한 핸들러 클래스를 지정하는 항목이다.

항목	설명
Name	핸들러가 톨에서 보여질 때 사용할 이름을 지정한다. 이름은 하나의 로거 내에서 유일해야 한다. 지정하지 않으면 클래스 이름과 Hash Code 이름이 대체된다.
Level	핸들러가 출력할 메시지의 레벨을 지정한다. 즉, 로거를 통과한 로그 메시지가 해당 로거가 사용하는 각각의 핸들러에게 전달되는데, 이 핸들러의 레벨에 부합하는 로그 메시지만 해당 핸들러에 의해 출력된다. 기본값으로 설정하면 로거를 통과하는 모든 로그 메시지가 핸들러에 의해 출력되도록 되어 있다. (기본값: FINEST)
Handler Class	사용자가 생성한 핸들러의 클래스를 지정한다. 해당 클래스는 lib/application 디렉터리의 JAR 파일에 포함되어 있어야 한다. 또한 해당 클래스는 Logging API의 java.util.logging.Handler를 상속받고 jeus.util.logging.JeusHandler를 구현해야 한다.
Handler Property	jeus.util.logging.JeusHandler의 setProperty()에 사용되는 Map 객체에 들어갈 프로퍼티를 지정한다.

항목	설명
Formatter Pattern	출력되는 로그의 형식을 지정한다. (기본값: [%d{yyyy.MM.dd HH:mm:ss}] [%l] [%j]-%T [%M-%N] %m)
Formatter Property	jeus.util.logging.JeusFormatter의 setProperty()에 사용되는 Map 객체에 들어갈 프로퍼티를 지정한다.
Encoding	핸들러가 출력하는 문자열의 인코딩을 지정한다. 기본은 시스템 인코딩으로 설정되어 있다.
Filter Class	핸들러가 로그 메시지를 출력하기 전에 수행할 필터링에 이용되는 클래스이다. 로거의 <filter-class>와 마찬가지로 lib/application에 이 클래스를 포함한 JAR 파일이 존재해야 한다.

8.3.5. 로그 파일 로테이션 설정

JEUS가 시작될 때 또는 JEUS 운영 중에 설정한 시간이 되거나 파일 사이즈를 넘으면 자동으로 이전에 로깅하던 파일의 이름은 변경하고 새로 출력할 로그들은 원래의 로그 파일에 계속 로깅할 수 있도록 **로그 파일 로테이션(Log File Rotation)** 기능을 제공한다.

콘솔 툴 사용

콘솔 툴에서 File Handler의 로그 파일 로테이션 설정은 **modify-log-handler** 명령어를 통해 설정할 수 있다. 콘솔 툴에서 로테이션 설정을 위해 사용할 수 있는 옵션으로는 **-hour**, **-size**, **-day**의 3개 항목이 있으며, 기본적으로 day 1로 설정되어 있다.

로테이션의 조건이 되는 3개의 항목 중 아무것도 설정되어 있지 않으면 매일 00시 00분이 되면 로깅하던 파일의 이름을 변경해서 백업하고 이후에 로깅되는 파일은 이전에 로깅하던 파일에 새로 로깅한다.

다음은 jeusadmin을 사용하여 로그 로테이션을 변경하는 예시이다.

```
[MASTER]domain1.adminServer>modify-log-handler fileHandler -server adminServer -logger jeus -day 2
Successfully performed the MODIFY operation for The handler(fileHandler) for the logger(jeus) in
server (adminServer), but all changes were non-dynamic. They will be applied after restarting.
Check the results using "modify-log-handler".
```

8.3.6. 프로퍼티 설정

필요한 프로퍼티 설정은 다음과 같다.

- 시스템 프로퍼티 설정

Standalone 클라이언트에서 로거 설정을 할 경우 로그 레벨을 시스템 프로퍼티로 설정할 수 있다.

- 로깅 프로퍼티 파일

Java 로깅 프로퍼티(logging.properties) 파일에 로깅 설정을 할 수 있다.

(기본 경로: JEUS_HOME/bin/logging.properties)

-Djava.util.logging.config.file = <프로퍼티 파일 경로>



JEUS 6까지 사용 가능했던 jeuslogging.properties 파일은 JEUS 7부터는 사용할 수 없다.

레벨 설정 우선순위

JEUS 툴을 이용한 동적 설정을 제외한 나머지 설정들의 레벨 설정 우선순위는 다음과 같다.

1. 시스템 프로퍼티
2. Java 로깅 프로퍼티(logging.properties) 파일
3. domani.xml의 로깅 설정



핸들러에 설정한 로그 레벨은 위에서 언급한 설정 우선순위를 치환(Override)하는 것은 아니다. 하지만 최종적으로 핸들러를 통해서 로그가 출력되기 때문에 가장 높은 순위에 있다고 할 수 있다. 참고로 핸들러의 기본 로그 레벨은 FINEST이다.

Appendix A: JEUS에서 사용하는 Port

본 부록에서는 JEUS에서 사용하는 Port를 정리한다. 방화벽을 설정하는 경우 참고한다.

A.1. 서버 Port

- BASEPORT

설명	JEUS 서버가 JNDI 서비스 및 운영을 위해 필요한 기본 서비스 Port. base listener 설정으로 설정 가능
Base Port	9736

- COS Naming Server Port

설명	COS Naming 서비스를 위해 사용하는 Port
Base Port	BASEPORT + 4 (예: 9740)

- ORB Port

설명	IIOP Port
Base Port	BASEPORT + 1

- ORB SSL Port

설명	IIOP SSL Port
Base Port	BASEPORT + 2

- ORB SSL Mutual Authorization Port

설명	IIOP 상호인증 Port
Base Port	BASEPORT + 3

- RMI Port

설명	EJB나 스케줄러 등 RMI를 사용하는 기능들이 사용하는 RMI Port
Base Port	BASEPORT + 7

Appendix B: JDBC 데이터소스 구성 예제

본 부록에서는 주요 DB 벤더들에 대한 데이터소스 설정 예제를 제공한다.

B.1. 개요

본 부록에서 설정 예제를 제공하는 JDBC 데이터소스는 다음과 같다.

- Oracle Thin
- Oracle OCI
- DB2 Type4(JCC)
- DB2 Type2(JCC)
- Sybase jConnect 5.x, 6.x
- Microsoft SQL Server 2005 Type4
- Informix Type4
- Tibero Type4
- MySQL 5.x Type4

본 부록을 참고하여 데이터소스 설정을 할 수 있다. 데이터소스를 설정하는 방법에 대한 자세한 내용은 [데이터소스 설정](#)을 참고한다.

B.2. Oracle Thin(Type4) 구성 예제

B.2.1. Oracle Thin Connection Pool 데이터소스

다음은 Oracle Thin Connection Pool 데이터소스를 구성한 예이다.

Oracle Thin Connection Pool 데이터소스 : <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>ora_thin_cpds</data-source-id>
        <export-name>ora_thin_cpds</export-name>
        <data-source-class-name>
          oracle.jdbc.pool.OracleConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>oracle</vendor>
        <server-name>192.168.1.2</server-name>
        <port-number>1521</port-number>
      </database>
    </data-source>
  </resources>
</domain>
```

```

        <database-name>orcl</database-name>
        <user>scott</user>
        <password>tiger</password>
        <property>
            <name>driverType</name>
            <type>java.lang.String</type>
            <value>thin</value>
        </property>
        <connection-pool>
            . . .
        </connection-pool>
    </database>
    . . .
</data-source>
. . .
</resources>
</domain>

```

B.2.2. Oracle Thin XA 데이터소스

다음은 Oracle Thin XA 데이터소스를 구성한 예이다.

Oracle Thin XA 데이터소스 : <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>ora_thin_xads</data-source-id>
                <export-name>ora_thin_xads</export-name>
                <data-source-class-name>
                    oracle.jdbc.xa.client.OracleXADataSource
                </data-source-class-name>
                <data-source-type>
                    XADataSource
                </data-source-type>
                <vendor>oracle</vendor>
                <server-name>192.168.1.2</server-name>
                <port-number>1521</port-number>
                <database-name>orcl</database-name>
                <user>scott</user>
                <password>tiger</password>
                <property>
                    <name>driverType</name>
                    <type>java.lang.String</type>
                    <value>thin</value>
                </property>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
            . . .
        </data-source>
        . . .
    </resources>

```

```
</domain>
```

B.2.3. java.util.Properties 설정 예제 with Oracle ASO

Oracle ASO 설정 예제를 이용해서 java.util.Properties 설정 방법에 대해 설명한다.

<property>를 사용하고, 기본 형식은 <type>은 java.util.Properties, <value>는 다음의 형식으로 입력한다.

```
[프로퍼티 이름 1]=[값 1], [프로퍼티 이름 2]=[값 2]
```

java.util.Properties 설정 예제 with Oracle ASO : <domain.xml>

```
<domain>
. . .
<resources>
  <data-source>
    <database>
      <data-source-id>oracle_CPDS1</data-source-id>
      <export-name>oracle_CPDS1</export-name>
      <data-source-class-name>
        oracle.jdbc.pool.OracleConnectionPoolDataSource
      </data-source-class-name>
      <data-source-type>
        ConnectionPoolDataSource
      </data-source-type>
      <vendor>oracle</vendor>
      <server-name>192.168.1.2</server-name>
      <port-number>1521</port-number>
      <database-name>ora9</database-name>
      <user>scott</user>
      <password>tiger</password>
      <property>
        <name>driverType</name>
        <type>java.lang.String</type>
        <value>thin</value>
      </property>
      <property>
        <name>ConnectionProperties</name>
        <type>java.util.Properties</type>
        <value>
          oracle.net.encryption_client=xxxx,
          oracle.net.encryption_types_client=(3DES168,3DES112),
          oracle.net.crypto_checksum_client=xxxx,
          oracle.net.crypto_checksum_types_client=(MD5)
        </value>
      </property>
      <connection-pool>
        . . .
      </connection-pool>
    </database>
    . . .
  </data-source>
  . . .
</resources>
```

B.3. Oracle OCI (Type2) 구성 예제

B.3.1. Oracle OCI Connection Pool 데이터소스

Oracle OCI 드라이버를 사용하기 위해서는 선행 작업이 필요한데, JEUS를 실행할 때 -Djava.library.path로 Oracle OCI 드라이버의 네이티브 라이브러리 경로를 설정해야 한다.

Oracle OCI Connection Pool 데이터소스 : <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>ora_oci_cpds</data-source-id>
        <export-name>ora_oci_cpds</export-name>
        <data-source-class-name>
          oracle.jdbc.pool.OracleConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>oracle</vendor>
        <server-name>192.168.1.2</server-name>
        <port-number>1521</port-number>
        <database-name>ora9</database-name>
        <user>scott</user>
        <password>tiger</password>
        <property>
          <name>driverType</name>
          <type>java.lang.String</type>
          <value>oci</value>
        </property>
        <property>
          <name>TNSEntryName</name>
          <type>java.lang.String</type>
          <value>ORCL</value>
        </property>
        <connection-pool>
          . . .
        </connection-pool>
      </database>
    </data-source>
  </resources>
</domain>
```


B.4. DB2 구성 예제

B.4.1. DB2 Type4(JCC) Connection Pool 데이터소스

다음은 DB2 Type4 Connection Pool 데이터소스를 구성한 예이다.

DB2 Type4(JCC) Connection Pool 데이터소스 : <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>db2_type4_cpsd</data-source-id>
        <export-name>db2_type4_cpds</export-name>
        <data-source-class-name>
          com.ibm.db2.jcc.DB2ConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>db2</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>50000</port-number>
        <database-name>TEST1</database-name>
        <user>db2inst1</user>
        <password>password</password>
        <property>
          <name>driverType</name>
          <type>java.lang.Integer</type>
          <value>4</value>
        </property>
        <connection-pool>
          . . .
        </connection-pool>
      </database>
    </data-source>
  </resources>
</domain>
```

B.4.2. DB2 Type4(JCC) XA 데이터소스

다음은 DB2 Type4 XA 데이터소스를 구성한 예이다.

DB2 Type4(JCC) XA 데이터소스 : <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>db2_type2_cpsd</data-source-id>
```

```

        <export-name>db2_type2_cpds</export-name>
        <data-source-class-name>
            com.ibm.db2.jdbc.DB2XADataSource
        </data-source-class-name>
        <data-source-type>
            XADataSource
        </data-source-type>
        <vendor>db2</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>50000</port-number>
        <database-name>TEST1</database-name>
        <user>db2inst1</user>
        <password>password</password>
        <connection-pool>
            . . .
        </connection-pool>
    </database>
    . . .
</data-source>
. . .
</resources>
</domain>

```

B.4.3. DB2 Type2(JCC) XA 데이터소스

DB2 Type 2 드라이버를 사용하기 위해서는 DB2 클라이언트를 설치하여 DB Alias를 설정한 뒤 JEUS를 실행할 때 -Djava.library.path 또는 시스템의 라이브러리 경로에 DB2 클라이언트의 네이티브 라이브러리 경로를 설정해야 한다. 다음의 예제에서 Database Name이 DB Alias가 되고 DB2 서버 주소와 포트 번호는 입력할 필요가 없다.

B2 Type2(JCC) XA 데이터소스 : <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>db2_type2_xads</data-source-id>
                <export-name>db2_type2_xads</export-name>
                <data-source-class-name>
                    com.ibm.db2.jdbc.DB2XADataSource
                </data-source-class-name>
                <data-source-type>
                    XADataSource
                </data-source-type>
                <vendor>db2</vendor>
                <server-name>192.168.1.1</server-name>
                <port-number>50000</port-number>
                <database-name>TEST1</database-name>
                <user>db2inst1</user>
                <password>password</password>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
        </data-source>
    </resources>
</domain>

```

```
. . .  
</resources>  
</domain>
```

B.5. Sybase 구성 예제

B.5.1. Sybase jConnect 5.x Connection Pool 데이터소스

다음은 Sybase jConnect 5.x Connection Pool 데이터소스를 구성한 예이다.

Sybase jConnect 5.x Connection Pool 데이터소스 : <domain.xml>

```
<domain>  
  . . .  
  <resources>  
    <data-source>  
      <database>  
        <data-source-id>syb_jconn5_cpds</data-source-id>  
        <export-name>syb_jconn5_cpds</export-name>  
        <data-source-class-name>  
          com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource  
        </data-source-class-name>  
        <data-source-type>  
          ConnectionPoolDataSource  
        </data-source-type>  
        <vendor>sybase</vendor>  
        <server-name>192.168.1.1</server-name>  
        <port-number>5000</port-number>  
        <database-name>testdb</database-name>  
        <user>sa</user>  
        <password>password</password>  
        <property>  
          <name>networkProtocol</name>  
          <type>java.lang.String</type>  
          <value>Tds</value>  
        </property>  
        <connection-pool>  
          . . .  
        </connection-pool>  
      </database>  
    . . .  
  </data-source>  
  . . .  
</resources>  
</domain>
```

B.5.2. Sybase jConnect 6.x XA 데이터소스

다음은 Sybase jConnect 6.x XA 데이터소스를 구성한 예이다.

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>syb_jconn6_xads</data-source-id>
        <export-name>syb_jconn6_xads</export-name>
        <data-source-class-name>
          com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>sybase</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>5000</port-number>
        <database-name>testdb</database-name>
        <user>sa</user>
        <password>password</password>
        <property>
          <name>networkProtocol</name>
          <type>java.lang.String</type>
          <value>Tds</value>
        </property>
        <connection-pool>
          . . .
        </connection-pool>
      </database>
    </data-source>
  </resources>
</domain>
```

B.6. Microsoft SQL Server 구성 예제

B.6.1. Microsoft SQL Server 2005 Connection Pool 데이터소스

다음은 Microsoft SQL Server 2005 Connection Pool을 구성한 예이다.

Microsoft SQL Server 2005 Connection Pool 데이터소스 : <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>mssql_2005_cpds</data-source-id>
        <export-name>mssql_2005_cpds</export-name>
        <data-source-class-name>
          com.microsoft.sqlserver.jdbc.SQLServerConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
```

```

        ConnectionPoolDataSource
    </data-source-type>
    <vendor>mssql</vendor>
    <server-name>192.168.1.1</server-name>
    <port-number>1411</port-number>
    <database-name>testdb</database-name>
    <user>jeusdb1</user>
    <password>password</password>
    <connection-pool>
        . . .
    </connection-pool>
</database>
. . .
</data-source>
. . .
</resources>
</domain>

```



ODBC 설정은 JDBC보다 먼저 설정되어야 한다.

B.7. Informix 구성 예제

B.7.1. Informix Connection Pool 데이터소스

다음은 Informix Connection Pool 데이터소스를 구성한 예이다.

Informix Connection Pool 데이터소스 : <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>infomix_cpds</data-source-id>
                <export-name>infomix_cpds</export-name>
                <data-source-class-name>
                    com.informix.jdbcx.IfxCConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <vendor>informix</vendor>
                <server-name>192.168.1.1</server-name>
                <port-number>2002</port-number>
                <database-name>skynps</database-name>
                <user>informix</user>
                <password>password</password>
                <property>
                    <name>IfxIFXHOST</name>
                    <type>java.lang.String</type>
                    <value>192.168.1.43</value>
                </property>
            </database>
        </data-source>
    </resources>
</domain>

```

```

        <connection-pool>
            . . .
        </connection-pool>
    </database>
    . . .
</data-source>
. . .
</resources>
</domain>

```

B.8. Tiberio 구성 예제

B.8.1. Tiberio Connection Pool 데이터소스

다음은 Tiberio Connection Pool 데이터소스를 구성한 예이다.

Tiberio Connection Pool 데이터소스 : <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>tiberio_cpds</data-source-id>
                <export-name>tiberio_cpds</export-name>
                <data-source-class-name>
                    com.tmax.tiberio.jdbc.ext.TbConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <vendor>tiberio</vendor>
                <server-name>192.168.1.1</server-name>
                <port-number>8629</port-number>
                <database-name>testdb</database-name>
                <user>jeusdb1</user>
                <password>password</password>
                <property>
                    <name>driverType</name>
                    <type>java.lang.String</type>
                    <value>thin</value>
                </property>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
            . . .
        </data-source>
        . . .
    </resources>
</domain>

```

B.9. MySQL 5.x 구성 예제

B.9.1. MySQL Connector/J Connection Pool 데이터소스

다음은 MySQL Connector/J Connection Pool 데이터소스를 구성한 예이다.

MySQL Connector/J Connection Pool 데이터소스 : <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>mysql_cpds</data-source-id>
        <export-name>mysql_cpds</export-name>
        <data-source-class-name>
          com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>mysql</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>3306</port-number>
        <database-name>testdb</database-name>
        <user>tester</user>
        <password>password</password>
        <connection-pool>
          . . .
        </connection-pool>
      </database>
    </data-source>
  </resources>
</domain>
```