

Jakarta Connectors 안내서

JEUS 9.1

TMAXSOFT

저작권 공지

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

회사 정보

(주)티맥스소프트

주소 : 경기도 성남시 분당구 정자일로 45, 티맥스소프트타워

기술 서비스 센터: 1544-8629

홈페이지: <https://www.tmaxsoft.com>

제한된 권리

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

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

상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

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

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

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

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

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

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

오픈 소스 소프트웨어 공지

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

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

`${INSTALL_PATH}/license/oss_licenses`

유지 보수

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

안내서 이력

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

목차

1. JCA 표준	1
1.1. 개요	1
1.2. 커넥터 아키텍처(Connector Architecture)	1
1.2.1. 아웃바운드(Outbound)	2
1.2.2. 인바운드(Inbound)	3
1.3. 리소스 어댑터와 JDBC 드라이버	4
1.4. CCI(Common Client Interface)	4
2. 아웃바운드 관리	6
2.1. Connection Pool과 Connection 관리	6
2.1.1. 애플리케이션의 Connection 요청 패턴	6
2.1.2. 리소스 어댑터와 Connection Pool의 관계	7
2.1.3. Connection Pool의 장점	8
2.1.4. Connection Pool 설정	8
2.1.5. Connection Pool 기능	12
2.2. 트랜잭션 관리	13
2.2.1. 로컬 트랜잭션 리소스의 글로벌 트랜잭션(XA) 참여 기능	14
2.2.2. 글로벌 트랜잭션(XA)과 Connection Sharing	14
2.3. Connection Pool 설정 예제	14
2.3.1. Connection Factory가 1개인 경우	14
2.3.2. Connection Factory가 2개 이상인 경우	16
2.4. Connection Pool 모니터링 및 제어	18
2.4.1. Connection Pool 제어	18
2.4.2. Connection Pool 모니터링	19
3. 인바운드 관리	22
3.1. Work Manager 관리	22
3.1.1. 기본 개념	22
3.1.2. Work Manager 설정	22
3.2. Message Inflow	23
4. 리소스 어댑터	26
4.1. 보안 관리	26
4.1.1. Connection 인증	26
4.2. 패키징	27
4.3. Deploy	27
4.3.1. SHARED 모드 클래스 로딩	28
4.3.2. Redeploy	28
4.4. 리소스 어댑터를 리소스로 등록	28
Appendix A: 환경설정 주의사항	29

1. JCA 표준

본 장에서는 JCA 표준 및 리소스 어댑터에 대해 설명한다.

1.1. 개요

Jakarta™ Connectors(JCA)는 Jakarta EE 플랫폼과 기업 정보 시스템(EIS, Enterprise Information System)을 연결할 수 있는 표준 아키텍처를 제공한다.

JCA는 Web Application Server(WAS)와 WAS에서 실행하는 Jakarta EE 애플리케이션을 EIS에 통합하기 위한 확장성 있고, 안전한 수단을 제공하기 위해 J2EE 1.3에서 처음 도입되었으며 이후 지속적으로 새로운 기능이 추가되고 있다.

1.2. 커넥터 아키텍처(Connector Architecture)

커넥터 아키텍처(Connector Architecture)는 Mainframe, ERP(Enterprise Resource Planning), TP Monitor, Legacy Database System 등을 포괄하는 기업 정보 시스템(EIS, Enterprise Information System)과의 연동을 위해 정의된 표준이다. 표준이 없는 경우에는 각 EIS 벤더와 Web Application Server(이하 WAS) 벤더 사이에 별도의 커스텀 드라이버를 구현해야 하므로 N by M 문제를 초래한다. 이는 곧 Jakarta EE 환경의 이식성과 확장성에 심각한 제약을 가한다.

이 문제를 해결하기 위하여 JCA 표준은 커넥터 아키텍처라는 개념을 도입하고 리소스 어댑터(Resource Adapter)라는 상호 연동에 필요한 어댑터를 정의하였다. EIS 벤더에서 리소스 어댑터를 제공하면 커넥터 아키텍처에 의하여 여러 WAS에서 코드 레벨의 수정 없이 상호 호환성을 가지고 동작할 수 있게 된다. 리소스 어댑터에 대한 자세한 내용은 [리소스 어댑터와 JDBC 드라이버](#)를 참고한다.



커넥터 아키텍처에 관한 자세한 내용은 본 안내서에서 다루지 않으므로 반드시 JCA 2.0 표준이나 그에 준하는 서적을 참고한다.

커넥터 아키텍처는 크게 아웃바운드와 인바운드로 나눌 수 있다. '아웃'과 '인'의 의미는 WAS를 기준으로 EIS와의 커뮤니케이션 방향을 나타낸 말이다.

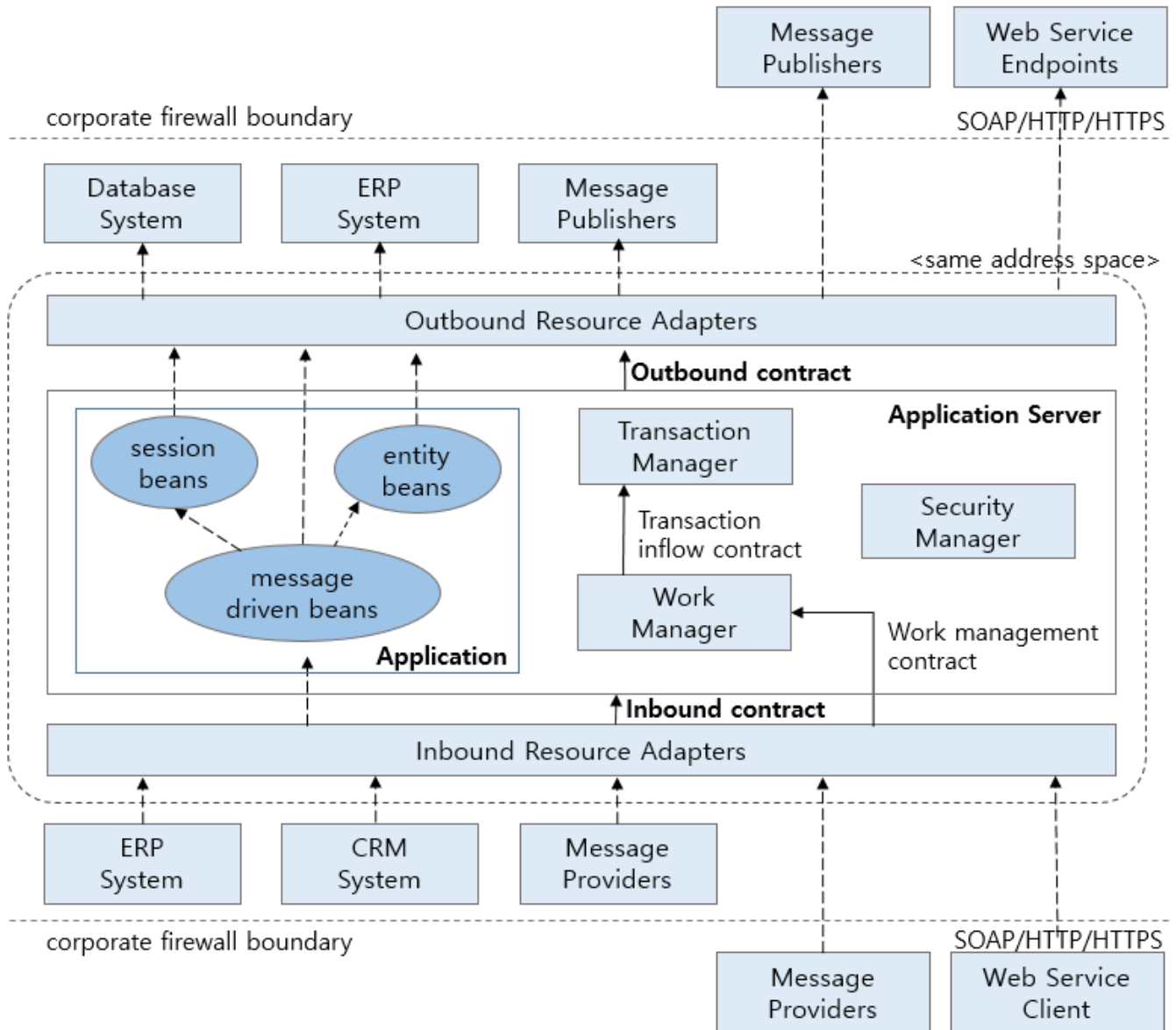
- 아웃바운드(Outbound)

WAS에 deploy된 애플리케이션에서 EIS로 가는 흐름을 의미한다.

- 인바운드(Inbound)

EIS에서 WAS의 애플리케이션으로 가는 흐름을 의미한다.

이러한 아웃바운드와 및 인바운드 커뮤니케이션을 위해서 WAS와 리소스 어댑터가 해야 하는 일을 정의한 것이 커넥터 아키텍처이다.



커넥터 아키텍처 개요

1.2.1. 아웃바운드(Outbound)

아웃바운드를 나타내는 대표적인 용어가 Connection이다. WAS의 애플리케이션이 EIS에 요청을 하고 그 결과를 받고 싶을 때 Connection을 맺고 그 요청을 보낸다.

대표적인 예가 서블릿이나 EJB 컴포넌트에서 데이터베이스로 SQL을 보내는 경우이다. 이는 [커넥터 아키텍처 개요](#)에서 Session Beans 또는 Entity Beans에서 아웃바운드 리소스 어댑터를 통해서 EIS로 요청을 하는 흐름에 해당한다.

JEUS에서는 이러한 Connection의 효율적인 관리를 위해서 다음과 같은 기능을 제공한다.

- Connection Pool 제공
 - Connection Pool을 통해 Connection을 매번 생성하지 않고 재활용함으로써 성능의 효율성을 제고한다.
 - 부하 상황에서는 WAS에서 리소스로 보내는 요청이 무한정 증가하지 않도록 제어한다.
- 트랜잭션 관리
 - 글로벌 트랜잭션(XA)이 필요한 경우에는 자동적으로 트랜잭션 매니저와의 연동을 지원한다.

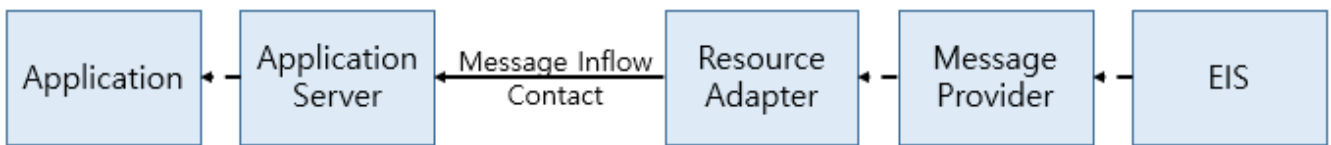
- 글로벌 트랜잭션(XA)의 경우 Connection Sharing(커넥션 공유)을 지원한다.

이외에도 JEUS에서 제공하는 여러 가지 부가 기능들이 있다. 아웃바운드에 관한 자세한 사항은 [아웃바운드 관리](#)를 참고한다.

1.2.2. 인바운드(Inbound)

JCA 표준은 JMS(Jakarta Messaging Service)를 포함하는 다양한 종류의 메시지 기반 서비스와 연동할 수 있는 매커니즘을 정의하고 있다. 이 경우 리소스 어댑터는 메시지 생산자(Message Provider)의 역할을 하고, 엔드 포인트(Endpoint) 애플리케이션은 메시지 소비자 역할을 맡게 된다. 전달되는 메시지의 타입은 WAS와 무관하며 실제적 메시지 전달이 이루어지는 메소드를 정의한 메시지 인터페이스는 리소스 어댑터가 임의로 결정할 수 있다.

다음은 인바운드의 Message Inflow의 구성을 나타낸다.



인바운드 Message Inflow

메시지 생산자는 메시지 소비자의 직접적인 요청에 의해서만 메시지를 전달하며, 두 영역 사이에는 어떠한 컨텍스트의 전달 및 공유도 이루어지지 않는다. 만약 EIS에서 발생한 트랜잭션 또는 인증 관련 정보를 WAS에 전달하기 위해서는 Work Manager를 이용하여 컨텍스트를 전달해야 한다.

JEUS에서의 인바운드 커뮤니케이션 관리에 관한 자세한 사항은 [인바운드 관리](#) 또는 [Message Inflow](#)를 참고한다.



컨텍스트 전달에 관한 자세한 내용은 JCA 표준을 참고한다.

Work Manager

아웃바운드 메시지만을 처리하는 리소스 어댑터의 경우는 일반적으로 별도의 Thread가 필요하지 않으나 인바운드 메시지를 처리하는 리소스 어댑터의 경우 Thread가 필요할 수 있다. 이때 JCA 표준은 리소스 어댑터 내에서 임의로 Java Thread를 생성하지 말고 WAS가 제공하는 Work Manager를 이용할 것을 권장하고 있다. 특히 EIS에서 시작한 트랜잭션이나 인증 정보를 연동해서 일을 실행하는 경우에는 반드시 Work Manager를 사용해야 한다.

JEUS의 Work Manager 구현체는 Thread Pool을 기반으로 한다. 리소스 어댑터가 Work Instance를 보내면 Thread Pool에서 Worker Thread를 얻어서 해당 작업을 수행한다. 만약 리소스 어댑터에서 Worker Thread로부터 작업 시작, 종료, 예외 이벤트를 받기 원하는 경우에는 이벤트 리스너를 전달할 수 있다. 그리고 작업이 수행될 때 필요한 트랜잭션, 인증 정보 등을 컨텍스트로 전달할 수 있다.

JEUS Work Manager 구현체에 관한 사항은 [Work Manager 관리](#)를 참고한다. Work Manager에 대한 기본적인 개념은 JCA 표준을 참고한다.

1.3. 리소스 어댑터와 JDBC 드라이버

JDBC가 관계형 데이터베이스(RDB)와의 연동을 표준으로 정의한 것이라면 리소스 어댑터는 RDB를 포함하여 여러 정보 시스템과의 연동 표준을 정의하고 있다. 즉, 리소스 어댑터는 WAS와 연동하기 위한 외부 리소스에 대해 JDBC 드라이버보다 좀 더 일반적이고 광범위하게 정의한 드라이버이다.

리소스 어댑터와 JDBC 드라이버의 큰 차이점이라면 리소스 어댑터는 Jakarta EE 표준에서 정의한 개념이고, JDBC 드라이버는 먼저 Java SE를 염두해서 만든 개념이라는 것이다. 이 때문에 JDBC 드라이버는 Standalone Java 애플리케이션에서 직접 사용할 수 있다.

또한 JDBC 드라이버는 JAR 파일을 시스템 클래스 패스로 잡아서 사용하며 드라이버를 업데이트하려면 현재 실행 중인 JVM을 다운시켜야 한다. 이에 반해서 리소스 어댑터는 Jakarta EE에서 정의한 애플리케이션이기 때문에 RAR 파일로 패키징을 하도록 되어 있고 WAS에 deploy할 수 있다.

그리고 리소스 어댑터의 버전을 업그레이드하려면 WAS를 중단할 필요없이 redeploy를 하면 된다. 물론 리소스 어댑터도 JDBC 드라이버처럼 JAR 파일로 변경한 후 시스템 클래스 패스에 설정하여 사용할 수 있지만 기본적으로 redeploy가 가능한 Jakarta EE 애플리케이션이다.

다음은 리소스 어댑터에 대한 이해를 돕기 위해 JDBC 드라이버와 비교하여 정리한 표이다.

	리소스 어댑터	JDBC 드라이버
개념	Jakarta EE 표준에서 정의한 개념	Java SE를 위한 개념(JDBC 드라이버는 Standalone Java 애플리케이션에서 직접 사용 가능)
연동	관계형 데이터베이스(RDB)를 포함하여 여러 정보 시스템과의 연동 표준을 정의	관계형 데이터베이스(RDB)와의 연동을 표준으로 정의
Deploy	RAR 파일로 패키징하도록 되어 있고 WAS에 Deploy 가능	JAR 파일을 시스템 클래스 패스로 설정하여 사용
업데이트	WAS를 중단할 필요없이 가능	현재 실행 중인 JVM을 다운시킨 후 업데이트

리소스 어댑터는 JDBC 드라이버와 비교했을 때 좀 더 일반적이고 광범위하게 정의한 드라이버이며, 보다 편리하게 사용이 가능하다.

1.4. CCI(Common Client Interface)

CCI는 아웃바운드 리소스 어댑터를 통일된 API로 제공하기 위한 것으로 JCA 표준은 리소스 어댑터를 개발할 때 CCI를 사용할 것을 권장하고 있다.

CCI는 서비스의 호출을 Interaction으로 개념화하여 서비스 방식에 관한 내용은 Interaction Spec이라 불리는 EIS에 종속적인 인스턴스로 추상화한다. 서비스 호출의 인자와 리턴값은 레코드 형식으로 제공된다.

Interaction Spec은 일반적으로 서비스 함수 이름(FunctionName)과 서비스 형식(InteractionVerb) 2가지를 포함할 것을 권장한다. CCI는 동기화 보내기, 동기화 받기, 동기화 보내고 받기의 3가지 서비스 형식을 지원하며, 비동기 서비스는 지원하지 않는다.



CCI는 애플리케이션과 리소스 어댑터 간의 인터페이스도 통일하자는 취지에서 나온 인터페이스이다. WAS는 기본적으로 애플리케이션과 리소스 어댑터 간의 인터페이스에 관여하지 않으며, 기본적으로는 `java.lang.Object`로 취급한다.

다음은 CCI를 이용하여 EIS의 서비스를 호출하기 위한 준비 과정이다.

```
// get Connection to EIS by lookup ConnectionFactory in JNDI
javax.naming.Context nc = new InitialContext();
jakarta.resource.cci.ConnectionFactory cf =
(ConnectionFactory)nc.lookup("java:/comp/env/eis/ConnectionFactory");
jakarta.resource.cci.Connection cx = cf.getConnection();

// create Interaction
jakarta.resource.cci.Interaction ix = cx.createInteraction();

// create Interaction Spec
com.wombat.cci.InteractionSpecImpl spec = .....
spec.setFunctionName("<EIS_FUNCTION_NAME>");
spec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE)
. . .
```

서비스에 필요한 데이터를 레코드에 저장하여 다음과 같이 서비스를 호출한다.

```
// call EIS service
boolean ret = ix.execute(spec, input, output);
```



CCI의 자세한 사항은 JCA 표준 또는 CCI 관련 서적을 참고한다.

2. 아웃바운드 관리

본 장에서는 애플리케이션에서 EIS로 향하는 아웃바운드(Outbound) 커뮤니케이션에서 JEUS의 역할과 기능들을 Connection Pool과 트랜잭션 연동을 중심으로 설명한다.

2.1. Connection Pool과 Connection 관리

아웃바운드 커뮤니케이션에서 애플리케이션이 EIS로 요청을 하고 그에 대한 결과를 받는 경우 Connection이 필요하다. JEUS에서는 애플리케이션이 Connection을 효율적으로 사용할 수 있도록 Connection Pool을 제공한다.

2.1.1. 애플리케이션의 Connection 요청 패턴

애플리케이션은 Connection Factory를 요청한 다음 해당 Factory에서 Connection을 얻을 수 있다. 이는 JDBC Connection을 얻을 때 데이터소스를 lookup하고 데이터소스로부터 Connection을 얻는 방식과 같다.

다음은 애플리케이션의 Connection 요청 패턴에 대한 예제이다.

```
// obtain the initial JNDI Naming context
Context initctx = new InitialContext();
// perform JNDI lookup to obtain the connection factory
jakarta.resource.cci.ConnectionFactory connectionFactory =
    (jakarta.resource.cci.ConnectionFactory)initctx.lookup("java:comp/env/eis/sampleEIS");
jakarta.resource.cci.Connection conn = connectionFactory.getConnection();
try {
    // do some works
} finally {
    conn.close();
}
```



예제에서는 CCI를 예로 들었지만 실제로 Connection Factory는 특별히 정의된 인터페이스가 있는 것이 아니라 리소스 어댑터에서 독자적으로 정의하고 이를 애플리케이션이 사용할 수 있다.

만약 위의 애플리케이션이 EJB라고 한다면 Annotation 또는 ejb-jar.xml의 **<resource-ref>**를 다음과 같이 설정해야 한다.

Connection 요청 : <ejb-jar.xml>

```
<ejb-jar>
. . .
<enterprise-beans>
. . .
  <session>
. . .
    <resource-ref>
      <res-ref-name>eis/sampleEIS</res-ref-name>
      <res-type>
```

```

jakarta.resource.cci.ConnectionFactory
</res-type>
<res-auth>Container</res-auth>
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
. . .

```

이때 jeus-ejb-dd.xml에는 'eis/sampleEIS'가 실제 매핑될 JNDI 이름을 설정해야 한다.

다음 예제의 'sampleConnectionPool'은 리소스 어댑터를 deploy할 때 jeus-connector-dd.xml에 기술해주는 Connection Pool의 JNDI 이름이다.

Connection 요청 : <jeus-ejb-dd.xml>

```

<jeus-ejb-dd>
. . .
<beanlist>
  <jeus-bean>
    . . .
    <res-ref>
      <jndi-info>
        <ref-name>eis/sampleEIS</ref-name>
        <export-name>sampleConnectionPool</export-name>
      </jndi-info>
    </res-ref>
  </jeus-bean>
. . .

```



jeus-connector-dd.xml에 관한 자세한 사항은 [리소스 어댑터](#)를 참고한다.

2.1.2. 리소스 어댑터와 Connection Pool의 관계

애플리케이션이 JNDI를 통해서 Connection Factory를 요청하면, JEUS는 리소스 어댑터로 Connection Factory 생성을 요청한다.

JEUS가 Connection Factory의 생성을 요청할 때는 리소스 어댑터로 Connection Manager를 전달한다. 이 Connection Manager의 구현체가 바로 **JEUS Connection Pool**이다.

리소스 어댑터가 Connection Factory를 생성해서 JEUS로 넘겨주면 이를 애플리케이션에 전달한다.

애플리케이션이 전달받은 Connection Factory에 Connection을 요청하면 리소스 어댑터는 JEUS가 넘겨준 Connection Manager에 Connection을 요청한다. 이렇게 되면 JEUS Connection Manager는 Pooling하고 있던 물리적 Connection(또는 ManagedConnection)을 하나 꺼내서 Connection Handle을 생성한 다음 이를 리소스 어댑터로 리턴한다. 그러면 리소스 어댑터가 이를 애플리케이션으로 전달한다. 물리적 Connection과 Connection Handle은 JCA 표준에 정의되어 있는 개념이다.

애플리케이션은 Connection이 더 이상 필요 없을 때는 반드시 반납해야 한다. 이를 일반적으로 "Connection을 닫는다"라고 표현한다. 만약 애플리케이션이 제대로 Connection을 닫지 않을 때는 항상 Connection Leak 현상이 발생하게 된다.

JEUS가 넘겨준 Connection Manager를 사용하는 것은 리소스 어댑터의 선택이다. 만약 리소스 어댑터가

Connection Manager를 사용하지 않고 내부적으로 Connection을 생성한다면 JEUS는 Connection 관리 및 애플리케이션과 EIS 간의 커뮤니케이션 과정에 관여하지 않는다.



1. Connection Manager에 관한 자세한 내용은 `jakarta.resource.spi.ConnectionManager` 및 `jakarta.resource.spi.ManagedConnectionFactory`의 Javadoc 및 JCA 표준을 참고한다.
2. 물리적 Connection과 Connection Handle의 자세한 내용은 JCA 표준의 "Chapter 6. Connection Management"를 참고한다.

2.1.3. Connection Pool의 장점

JCA에서 제공하는 JEUS Connection Pool의 기능은 DB Connection Pool과 거의 유사하나 다음과 같은 장점을 갖는다.

- 보다 높은 성능

데이터베이스와의 Connection뿐만 아니라 외부 리소스와의 Connection 생성은 일반적으로 처리 과정이 느리다. 그러나 Connection Pool의 물리적 Connection들은 매번 새로 생성하는 것이 아니라 미리 생성해놓거나 재활용하기 때문에 Connection 생성의 오버헤드를 감소시킨다. 또한 애플리케이션이 Connection을 더 이상 사용하지 않고 닫았을 때에는 실제로 끊어 버리는 것이 아니라 Pool에 반환시켜서 Connection 종단의 오버헤드를 감소시킬 수 있다.

- 동시 연결 관리

동시에 리소스를 요청하는 클라이언트의 수를 제어할 수 있다. 부하 상황에서 리소스로 너무 많은 처리가 몰려서 리소스가 서비스 불능이 되는 상태를 방지할 수 있다.



DB Connection Pool에 관한 자세한 사항은 JEUS Server 안내서의 "DB Connection Pool과 JDBC"를 참고한다.

2.1.4. Connection Pool 설정

JCA Connection Pool은 리소스 어댑터를 deploy할 때 `jeus-connector-dd.xml`을 기술해서 설정한다.

이때 애플리케이션에서 Connection Pool을 사용하기 위해서는 JNDI 이름이 꼭 필요하기 때문에 JEUS에서 정한 규칙에 따라 자동으로 생성한다. 그러나 사용자가 직접 JNDI 이름을 지정하는 경우에는 `jeus-connector-dd.xml`에 반드시 기술해야 한다. `jeus-connector-dd.xml`의 `<jeus-connector-dd>` 하위의 `<connection-pool>`을 사용하여 설정한다. `jeus-connector-dd.xml`를 RAR에 포함시키는 방법은 [리소스 어댑터](#)를 참고한다.

JCA 표준에 의해 하나의 리소스 어댑터에는 여러 개의 Connection Pool을 설정할 수 있다. 이러한 Connection Pool의 기준이 되는 것은 `ra.xml`의 `<connection-definition>`과 그 하위 요소인 `<connectionfactory-interface>`이다. 만약 `<connection-definition>`이 5개 설정되어 있다면 JEUS에는 최소 5개의 Connection Pool이 구성된다.



ra.xml에 정의하는<connectionfactory-interface>는 중복될 수 없다. 중복되는 값이 있을 경우에는 ra.xml의 유효성 검사가 실패하게 되어 deploy되지 않는다.

다음은 jeus-connector-dd.xml의 각 태그에 대한 설명이다. 실제 설정에 대한 예는 [Connection Pool 설정 예제](#)를 참고한다.

- **<connectionfactory-interface>**

- ra.xml에 <connection-definition>이 2개 이상인 경우에는 반드시 이 값을 설정해야 한다. 설정하지 않으면 Connection Pool 생성을 할 수 없기 때문에 deploy 에러가 발생한다.
- ra.xml의 각 <connection-definition>에 정의된 <connectionfactory-interface> 값을 그대로 입력한다. <connectionfactory-interface>의 자세한 사항은 [리소스 어댑터](#)를 참고한다.

- **<export-name>**

- Connection Pool의 JNDI 이름이다.
- 서버에서 유일한 이름이어야 하며 반드시 설정해야 한다.

- **<transaction-support>**

- 해당 Connection Pool이 지원하는 트랜잭션 타입을 다음 중에 설정한다. 이 설정은 ra.xml의 설정에 우선한다.
 - NoTransaction
 - LocalTransaction
 - XATransaction

- **<user>**

- Connection을 생성하는 경우 필요한 로그인을 Connection Pool에게 맡길 경우 설정하는 사용자 ID이다.

- **<password>**

- Connection을 생성할 때 필요한 로그인을 Connection Pool에게 맡길 경우 설정하는 패스워드이다.
- 암호화해서 저장할 때는 다음과 같은 형식으로 입력한다.

```
{algorithm}ciphertext
```

예)

```
{DES}FQrLbQ/D801LDVS71L28rw==
```

- **<use-lazy-transaction-enlistment>**

- JCA 표준에서 언급하는 트랜잭션 최적화 기능 중 하나인 "Lazy Transaction Enlistment" 옵션을 사용할 것인지 결정한다. (기본값: false)

- **<pool-management>**

Connection Pool의 주요 옵션들은 <connection-pool>의 <pool-management> 하위에 존재한다.

- **<min>** : Connection 수의 초깃값이다. (기본값: 2)
- **<max>** : Pooling하는 Connection 수의 최댓값이다. (기본값: 10)
- **<step>** : Connection 개수의 증가가 필요한 상황에서 몇 개의 Connection을 증가 단위로 할 것인지를 설정한다. (기본값: 1)
- **<period>** : 설정된 period마다 Connection Pool의 사이즈를 <min> 값에 맞춰준다. 만약 Idle Connection이 없을 경우에는 <min> 값으로 줄이는 작업을 하지 않는다. 기존의 <pooled-timeout> 대신 이 설정을 사용한다. (기본값: 10분, 단위: ms)
- **<wait-connection>** : Idle Connection이 없는 상태이고 Connection Pool 사이즈도 최댓값에 도달되었을 때 Connection 요청을 처리하는 방법을 결정한다.

태그	설명
<wait-connection>	<ul style="list-style-type: none"> ◦ true : 시스템은 이용 가능한 Connection을 얻기 위해 대기한다. 만약 대기 후에도 Connection을 가져오지 못한다면 exception을 발생시킨다. ◦ false : Connection을 새로 생성해서 애플리케이션에 제공하고 애플리케이션이 이를 반납하면 Pooling하지 않고 닫아서 버린다. 이렇게 한 번 사용하고 버린다는 의미로 JEUS에서 'disposable Connection'이라고 한다. (기본값)
<wait-timeout>	<p>사용자가 Connection을 위해 대기하는 시간으로 만약 어떠한 Connection도 사용자가 이 시간 동안 대기해서 이용할 수 없을 때는 exception을 발생한다. <wait-connection>이 true일 때만 유효하다.</p> <p>(기본값: 10초, 단위: ms)</p>

- **<use-match-connection>** : Connection Match를 사용할 것인지 결정한다. (기본값: false)
- **<allow-disposable-connection-when-match-failed>** : Connection Match가 실패한 경우 일회용 Connection을 사용할 것인지를 설정한다. Connection Match를 하지 않는 경우에는 이 값은 의미가 없다. (기본값: false)
- **<connection-validation>** : Connection 유효성 검사에 관련된 설정이다.

태그	설명
<enabled>	Connection 유효성 검사 기능을 사용할 것인지 결정한다. 이 설정값을 true로 해도 기본적으로 리소스 어댑터가 jakarta.resource.spi.ValidatingManagedConnectionFactory를 구현했을 경우에만 사용이 가능하다.
<period>	Connection 유효성 검사의 주기를 설정한다. 설정된 주기에 따라 Idle 상태에 있는 Connection들의 유효성을 검사한다. (단위: ms)
<non-validation-interval>	Connection 단위로 유효성 검사를 할 때 마지막으로 Connection을 사용한 시각과 검사할 때 시각과의 차이가 설정한 시간보다 작으면 체크하지 않는다. 이 설정을 통해서 유효성 검사로 인해 발생할 수 있는 오버헤드를 줄일 수 있다. (단위: ms)

태그	설명
<validation-retrial-count>	<p>유효성 검사는 기본적으로 destroy policy가 FailedConnectionOnly일 때는 한 번 수행하고, AllConnections일 때는 하나의 Connection에 대해서 해보고 그것이 실패하면 다른 Connection을 한 번 더 해보므로 총 두 번 수행한다.</p> <p>만약 기본적인 유효성 검사 횟수가 부족하다고 판단될 경우에는 이 설정을 통해서 체크 횟수를 늘린다.</p>
<destroy-policy-on-validation>	<p>Connection 유효성 검사가 실패했을 경우 해당 Connection Pool에 있는 Connection들을 어떻게 할지 정책을 결정하는 옵션이다.</p> <ul style="list-style-type: none"> ◦ FailedConnectionOnly : 유효성 검사가 실패한 물리적 Connection만 닫는다. (기본값) ◦ AllConnections : 유효성 검사가 실패했을 경우 Pool에 있는 다른 Connection을 한 번 더 체크해 보고 그래도 실패하면 해당 Connection Pool의 모든 Connection을 닫는다. 애플리케이션이 사용하고 있던 Connection들도 모두 닫히게 된다.

- **<action-on-connection-leak>** : 컴포넌트(주로 Stateless 컴포넌트 - Servlet/JSP, Stateless Session Beans, MDB)에서 사용한 Connection에 대한 로깅이나 반환 액션을 설정한다. 설정하지 않았을 경우 기본 동작은 서버에 설정한 Action On Resource Leak을 따른다. (기본값: Warning)
- **<connection-trace>** : Connection을 모니터링하기 위한 옵션이다. 현재 기본 기능은 어떤 애플리케이션이 Connection을 사용하고 있는지 알 수 있도록 getConnection할 때의 Stack 정보를 보여준다. 서버에 설정한 Action On Resource Leak이 동작하는 경우에도 이 정보를 보여준다.

옵션	설명
enabled	connection trace 기능을 on/off하는 옵션이다.
get-connection-trace	애플리케이션이 Connection을 얻을 때(getConnection 호출)의 Stack Trace를 저장해두는 옵션이다. (기본값: true)
local-transaction-trace	<p>애플리케이션이 리소스 어댑터와 로컬 트랜잭션 작업을 할 때의 정보를 추적할 것인지 선택하는 옵션이다.</p> <p><get-connection-trace>와 함께 사용하면 로컬 트랜잭션을 제대로 commit 또는 rollback하지 않은 애플리케이션을 추적할 때 도움을 줄 수 있다.</p>

- **<max-use-count>** : 1개의 Connection에 대해서 설정된 숫자만큼 사용하면 해당 Connection을 버리고 새로운 Connection을 맺게 된다. (기본값: 0, Connection들을 교체하지 않겠다는 의미)
- **<pool-destroy-timeout>** : Connection Pool을 destroy할 때 대기하는 시간이다. 리소스 어댑터를 undeploy할 때 Pool을 destroy하는데, Connection을 닫으면서 리소스와 네트워크 통신을 할 경우 Hang이 걸릴 가능성이 존재한다. 따라서 여기에 설정된 시간만큼 기다린 뒤에도 destroy가 진행되지 않으면 이를 무시하고 계속 undeploy를 진행한다. (기본값: 10초)

• <property>

- ManagedConnectionFactory에 적용할 프로퍼티를 추가한다. ra.xml에 설정된 값을 치환하거나 추가할 때 사용한다.



Connection Match, Lazy Transaction Enlistment와 관련된 자세한 내용은 JCA 표준을 참고한다.

2.1.5. Connection Pool 기능

Connection Pool은 Connection의 유효성을 검사할 수 있고 Connection Leak에 대해 처리할 수 있는 기능을 갖는다.

Connection 유효성 검사

애플리케이션이 Connection 요청을 했을 때 리소스 어댑터로 Connection의 유효성(Connection Validation)에 대해 검사를 요청하는 기능이다. 이 기능은 Connection의 내부적인 에러로 인한 끊김, 방화벽에 의한 소켓 끊김 현상 등을 체크할 때 유용하다. 검사에 실패하면 물리적 Connection을 새로 생성하여 그에 대한 핸들을 애플리케이션으로 리턴한다.

만약 유효성 검사 작업이 너무 잦아서 오버헤드가 발생한다면 **<non-validation-interval>**에 대해 설정한다. 이는 Connection 검사를 수행할 때의 시각과 맨 마지막에 Connection을 사용한 시각과의 차이가 이미 설정한 interval 내에 있다면 무조건 유효하다고 판단하고 검사를 하지 않도록 하는 설정이다.

예를 들어 interval을 5초(5000ms)라고 설정했을 때, Connection을 마지막에 사용한 후 아직 5초가 지나지 않았다면 그 Connection이 무조건 유효하다고 가정하는 것이다.

다음은 <non-validation-interval>의 설정 예이다.

```
<non-validation-interval>5000</non-validation-interval>
```



리소스 어댑터에서 반드시 `jakarta.resource.spi.ValidatingManagedConnectionFactory`를 구현해야만 Connection의 유효성을 검사할 수 있다. 리소스 어댑터로 검사를 요청했을 때 외부 리소스 측에서 응답이 오지 않아서 계속 기다리게 되는 상황이 발생할 수 있다. 현재 WAS에서는 이러한 문제를 해결할 수 있는 방법이 없으므로 리소스 어댑터 측에서 타임아웃 옵션을 제공할 필요가 있다.

사용자는 유효성 검사에 실패한 경우 해당 Connection Pool에 있는 나머지 Connection들에 대해 Destroy 정책을 다음 중에 하나를 설정할 수 있다.

정책	설명
FailedConnectionOnly	유효성 검사에 실패한 Connection만 버린다. (기본값)
AllConnections	유효성 검사에 실패한 후 바로 Connection들을 버리는 것이 아니라, 한 번 더 Connection을 Pool에서 가져와서 검사를 요청한다. 만약 검사에 실패하면 그때 Pool에 있는 모든 Connection들을 버린다.

다음은 유효성 검사에 실패한 경우 Pool에 있는 Connection에 대한 Destroy 정책을 설정하는 예이다.

```
<destroy-policy-on-validation>AllConnections</destroy-policy-on-validation>
```



유효성 검사 횟수를 추가적으로 더 늘리는 경우에는 <validation-retrial-count>를 이용한다.

Connection Leak 처리

JEUS는 서버 또는 각 Connection Pool별로 Connection Leak에 대한 액션을 설정할 수 있다.

Servlet/JSP, Stateless Session Beans, Message Driven Bean과 같이 시작과 끝이 분명한 컴포넌트의 경우 사용한 Connection을 제대로 반환했는지 확인할 수 있다. 만약 Connection을 반환하지 않았다면 Invocation Manager를 통해서 로깅을 남기거나 강제로 닫아주는 액션을 설정할 수 있다. 단, JCA Connection의 경우 Invocation Manager의 자동 반환(AutoClose) 액션에 대해 예외가 있다.

JCA Connection Pool에서는 JDBC의 java.sql.Connection처럼 어떤 메소드가 close 역할을 하는지 알 수 없기 때문에 일반적으로는 직접 Connection을 닫아주는 것이 불가능하다. 대신 리소스 어댑터의 ra.xml에 설정된 <connection-definition>의 <connection-interface>가 다음과 같은 경우에는 close 메소드의 모양을 이미 알고 있고 메소드에 아무런 파라미터가 없기 때문에 직접 Connection을 닫아준다.

다음은 JCA Connection의 Invocation Manager의 자동 반환 액션의 예외 메소드이다.

- java.sql.Connection
- jakarta.resource.cci.Connection

위의 <connection-interface>가 아닌 경우에는 직접 Connection을 닫아주지 못하지만 jakarta.resource.spi.ManagedConnection에 정의된 cleanup 메소드를 호출한 뒤 강제로 해당 Connection을 Pool로 반환한다.

cleanup 메소드에는 리소스 어댑터가 다음과 같은 동작을 하도록 정의되어 있다.

```
The cleanup should invalidate all connection handles  
that had been created using this ManagedConnection instance.
```



Invocation Manager의 자세한 사항은 JEUS Server 안내서의 [Action On Resource Leak 설정](#)을 참고한다. JEUS Server 안내서의 "Action On Resource Leak 설정"을 참고한다.

2.2. 트랜잭션 관리

본 절에서는 JEUS에서 트랜잭션 타입이 로컬 트랜잭션, 글로벌 트랜잭션인 Connction Pool을 관리하는 방법에 대해서 설명한다.

2.2.1. 로컬 트랜잭션 리소스의 글로벌 트랜잭션(XA) 참여 기능

원칙적으로 로컬 트랜잭션은 애플리케이션과 리소스 어댑터 간의 트랜잭션이기 때문에 글로벌 트랜잭션이 없는 상황에서 리소스 어댑터로 Connection 요청을 할 경우에는 JEUS가 관여할 수 없다.

그러나 글로벌 트랜잭션 내에서 로컬 트랜잭션 타입의 Connection Pool을 사용하는 경우 리소스 어댑터의 ManagedConnection으로부터 얻을 수 있는 jakarta.resource.spi.LocalTransaction이 XAResource인 것처럼 에뮬레이션해줄 수 있다.

글로벌 트랜잭션을 지원하지 않는 리소스라도 리소스 어댑터가 로컬 트랜잭션 타입을 지원한다면 최대 하나의 리소스가 글로벌 트랜잭션에 참여가 가능하다. 이는 JDBC에서 XA 에뮬레이션 기능을 사용하는 Connection Pool 데이터소스와 거의 동일하다.



데이터소스의 자세한 사항은 JEUS Server 안내서의 "데이터소스 설정"을 참고한다.

2.2.2. 글로벌 트랜잭션(XA)과 Connection Sharing

같은 글로벌 트랜잭션 내에서는 하나의 리소스에 대해 항상 하나의 Connection만 사용하는 것을 Connection Sharing(커넥션 공유)이라고 한다. JEUS는 항상 Connection Sharing을 지원하며, 아무런 설정이 없는 경우에도 기본적으로 공유하도록 되어 있다.

만약 Connection Sharing 기능을 사용하고 싶지 않은 경우에는 각 애플리케이션별로 ejb-jar.xml, web.xml의 **<resource-ref>**에 Unshareable로 설정한다.

```
<resource-ref>
  <res-ref-name>jca/pool</res-ref-name>
  <res-type>jakarta.resource.cci.ConnectionFactory</res-type>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```



1. Connection Sharing의 자세한 사항은 JCA 표준 2.0의 "8.9 Connection Sharing"을 참고한다.
2. DB Connection Pool에서도 Connection Sharing을 제공하고 있으므로 JEUS Server 안내서의 "글로벌 트랜잭션(XA)과 Connection Sharing"을 참고한다.

2.3. Connection Pool 설정 예제

아웃바운드 리소스 어댑터를 사용해서 하나 또는 그 이상의 아웃바운드 Connection을 설정할 수 있다.

2.3.1. Connection Factory가 1개인 경우

다음은 jakarta.resource.cci.ConnectionFactory를 Connection Factory를 1개 갖는 ra.xml의 예제이다.

대부분의 경우 Connection Factory는 1개이다.

Connection Factory가 1개인 경우 Connection Pool 설정 : <ra.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="https://jakarta.ee/xml/ns/jakartaee" version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
        https://jakarta.ee/xml/ns/jakartaee/connector_2_0.xsd">
    <display-name>ConnectionManagementSample1</display-name>
    <vendor-name>TmaxSoft</vendor-name>
    <eis-type>TestResource</eis-type>
    <resourceadapter-version>2.0</resourceadapter-version>
    <license>
        <license-required>false</license-required>
    </license>
    <resourceadapter>
        <outbound-resourceadapter>
            <connection-definition>
                <managedconnectionfactory-class>
                    com.tmax.SampleManagedConnectionFactory
                </managedconnectionfactory-class>
                <connectionfactory-interface>
                    jakarta.resource.cci.ConnectionFactory
                </connectionfactory-interface>
                <connectionfactory-impl-class>
                    com.tmax.SampleConnectionFactory
                </connectionfactory-impl-class>
                <connection-interface>
                    jakarta.resource.cci.Connection
                </connection-interface>
                <connection-impl-class>
                    com.tmax.SampleConnection
                </connection-impl-class>
            </connection-definition>
            <transaction-support>
                LocalTransaction
            </transaction-support>
            <authentication-mechanism>
                <authentication-mechanism-type>
                    BasicPassword
                </authentication-mechanism-type>
                <credential-interface>
                    jakarta.resource.spi.security.PasswordCredential
                </credential-interface>
            </authentication-mechanism>
            <reauthentication-support>
                false
            </reauthentication-support>
        </outbound-resourceadapter>
    </resourceadapter>
</connector>
```

jeus-connector-dd.xml을 설정하지 않은 경우에는 ra.xml의 <connection-definition>에 해당하는 Connection Pool을 생성할 수 없다. 그러므로 jeus-connector-dd.xml의 <export-name>을 반드시 설정해야 한다.

다음은 jeus-connector-dd.xml의 설정 예이다. 각 태그에 대해서는 [Connection Pool 설정](#)을 참고한다.

Connection Factory가 1개인 경우 Connection Pool 설정 : <jeus-connector-dd.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <connection-pool>
    <export-name>jcapool</export-name>
    <pool-management>
      <min>10</min>
      <max>50</max>
      <period>600000</period> <!-- 10 minutes -->
      <wait-connection>
        <wait-connection>true</wait-connection>
        <wait-timeout>30000</wait-timeout>
      </wait-connection>
      <action-on-connection-leak>Warning</action-on-connection-leak>
    </pool-management>
  </connection-pool>
</jeus-connector-dd>
```

2.3.2. Connection Factory가 2개 이상인 경우

다음은 javax.sql.DataSource과 jakarta.resource.cci.ConnectionFactory를 Connection Factory로 설정한 ra.xml 예이다.

Connection Factory가 2개인 경우 Connection Pool 설정 : <ra.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="https://jakarta.ee/xml/ns/jakartaee" version="2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
    https://jakarta.ee/xml/ns/jakartaee/connector_2_0.xsd">
  <display-name>ConnectionManagementSample1</display-name>
  <vendor-name>TmaxSoft</vendor-name>
  <eis-type>TestResource</eis-type>
  <resourceadapter-version>2.0</resourceadapter-version>
  <license>
    <license-required>>false</license-required>
  </license>
  <resourceadapter>
    <outbound-resourceadapter>
      <connection-definition>
        <managedconnectionfactory-class>
          com.tmax.DataSourceManagedConnectionFactory
        </managedconnectionfactory-class>
        <connectionfactory-interface>
          javax.sql.DataSource
        </connectionfactory-interface>
        <connectionfactory-impl-class>
          com.tmax.JeusDataSource
        </connectionfactory-impl-class>
        <connection-interface>
          java.sql.Connection
        </connection-interface>
        <connection-impl-class>
          com.tmax.JeusConnection
        </connection-impl-class>
      </connection-definition>
    </outbound-resourceadapter>
  </resourceadapter>
</connector>
```

```

</connection-definition>
<connection-definition>
  <managedconnectionfactory-class>
    com.tmax.SampleManagedConnectionFactory
  </managedconnectionfactory-class>
  <connectionfactory-interface>
    jakarta.resource.cci.ConnectionFactory
  </connectionfactory-interface>
  <connectionfactory-impl-class>
    com.tmax.SampleConnectionFactory
  </connectionfactory-impl-class>
  <connection-interface>
    jakarta.resource.cci.Connection
  </connection-interface>
  <connection-impl-class>
    com.tmax.SampleConnection
  </connection-impl-class>
</connection-definition>
<transaction-support>
  NoTransaction
</transaction-support>
<authentication-mechanism>
  <authentication-mechanism-type>
    BasicPassword
  </authentication-mechanism-type>
  <credential-interface>
    jakarta.resource.spi.security.PasswordCredential
  </credential-interface>
</authentication-mechanism>
<reauthentication-support>
  false
</reauthentication-support>
</outbound-resourceadapter>
</resourceadapter>
</connector>

```

다음과 같이 jeus-connector-dd.xml에서 ra.xml의 <connection-definition>에 해당하는 Connection Pool을 생성하려면 반드시 **<connectionfactory-interface>**를 설정해야 한다. 그렇지 않을 경우 deploy할 때 Connection Pool을 생성할 수 없다는 에러가 발생한다.

Connection Factory가 2개인 경우 Connection Pool 설정 : <jeus-connector-dd.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <connection-pool>
    <export-name>jdbcpool</export-name>
    <connectionfactory-interface>
      javax.sql.DataSource
    </connectionfactory-interface>
    <pool-management>
      <min>5</min>
      <max>20</max>
    </pool-management>
  </connection-pool>
  <connection-pool>
    <export-name>ccipool</export-name>
    <connectionfactory-interface>

```

```
jakarta.resource.cci.ConnectionFactory
</connectionfactory-interface>
<pool-management>
  <min>1</min>
  <max>10</max>
</pool-management>
</connection-pool>
</jeus-connector-dd>
```

2.4. Connection Pool 모니터링 및 제어

JCA Connection Pool의 모니터링 및 제어는 콘솔 툴(jeusadmin)을 사용한다. JCA Connection Pool의 모니터링 및 제어 방법을 알아보려면 먼저 리소스 어댑터 모듈을 deploy해야 한다. 리소스 어댑터 모듈의 deploy 방법에 대해서는 "JEUS Applications & Deployment 안내서"를 참고한다.

본 절에서 설명하는 예제에는 JCA Connection Pool의 JNDI export name이 'my_rar_cp'인 리소스 어댑터 모듈의 deploy가 완료되었음을 가정한다. JCA Connection Pool의 JNDI export name 설정에 대해서는 [Connection Pool 설정](#)을 참고한다.



콘솔 툴을 통한 모니터링 및 제어 방법은 JEUS Reference 안내서의 "Connection Pool 제어 및 모니터링 명령어"를 참고한다.

2.4.1. Connection Pool 제어

다음은 jeusadmin을 사용해서 Connection Pool을 제어하는 방법에 대한 설명이다.

```
[MASTER]domain1.adminServer>cpinfo
The connection pool information on the server [adminServer].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min   | Max   | Active | Act   | Active | Idle | Dispos | Tot   | Wait | Enab |
| tion   |       |       | Max    | ive   | Average|      | able   | al    |      | led  |
| Pool ID|       |       |        |       |         |      |        |       |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| mysql *| 20    | 20    | 0       | 0     | 0.0    | 0    | 0       | 0     | 0     | false|
|        |       |       |         |       |         |      |         |       |       | se    |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| oracle | 1      | 30    | 0       | 0     | 0.0    | 0    | 0       | 0     | 0     | true  |
| *      |       |       |         |       |         |      |         |       |       | false|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
=====
[MASTER]domain1.adminServer>create-connection-pool -id mysql
Servers that successfully created a connection pool : adminServer
Servers that failed to create a connection pool : none.
[MASTER]domain1.adminServer>cpinfo
The connection pool information on the server [adminServer].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Connec tion Pool ID	Min	Max	Active Max	Act ive	Active Average	Idle	Dispos able	Tot al	Wait	Enab led
mysql	20	20	1	0	0.0	20	0	20	fal se	true
oracle *	1	30	0	0	0.0	0	0	0	true	false

* : has not been created, total = active + idle + disposable

```
[MASTER]domain1.adminServer>control-connection-pool -id mysql -enable
Servers that successfully enabled a connection pool : adminServer
Servers that failed to enable a connection pool : none.
[MASTER]domain1.adminServer>
```

먼저 Connection Pool이 생성이 되지 않은 상태이므로 create-connection-pool 명령어로 Connection Pool을 생성한다. 이후에는 control-connection-pool 명령어로 Connection Pool을 제어할 수 있다.

다음은 control-connection-pool 관련 옵션에 대한 설명이다.

버튼	설명
[Enable]	Connection Pool을 활성화한다.
[Shrink]	Connection Pool의 커넥션 개수를 최솟값으로 조정한다.
[Disable]	Connection Pool을 비활성화한다.
[Refresh]	Connection Pool의 커넥션들을 새로운 커넥션으로 교체한다.

2.4.2. Connection Pool 모니터링

다음은 jeusadmin을 사용해서 Connection Pool을 모니터링하는 방법에 대한 설명이다.

```
[MASTER]domain1.adminServer>cpinfo
The connection pool information on the server [adminServer].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able    | al  |      | led  |
| Pool ID |     |     |        |     |        |      |         |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| mysql  | 20  | 20  | 1      | 0   | 0.0   | 20   | 0       | 20  | fal  | true |
|        |     |     |        |     |        |      |         |    | se   |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| oracle | 1   | 30  | 0      | 0   | 0.0   | 0    | 0       | 0   | true | false|
| *      |     |     |        |     |        |      |         |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
=====
```

다음은 Connection Pool 목록의 항목에 대한 설명이다.

항목	설명
Connection Pool ID	Connection Pool의 ID이다.
Min	Connection Pool의 Connection 최솟값이다.
Max	Connection Pool의 Connection 최댓값이다.
Active Max	Connection Pool 생성 이후 Active Connection의 최댓값이다.
Active	사용 중인 Connection 개수이다.
Active Average	최근 1시간 동안의 Active Connection 개수의 평균값이다.
Idle	사용 가능한 유휴 Connection 개수이다.
Disposable	Connection Pool에 유휴 Connection이 없는 경우 생성하는 일회용 Connection 개수이다. 일회용 Connection은 Wait 컬럼이 false인 경우에 한해서 생성된다.
Total	Active, Idle, Disposable의 총합이다.
Wait	Connection Pool에 사용 가능한 유휴 Connection이 없는 경우 일회용 Connection을 생성하여 사용할지의 여부이다.
Enabled	Connection Pool 활성화 여부이다.

특정 Connection Pool의 ID를 옵션으로 주면 해당 Connection Pool에 존재하는 Connection들에 대한 상세 정보를 확인할 수 있다.

```
[MASTER]domain1.adminServer>cpinfo -id mysql
Information about connections in the server [adminServer]'s connection pool [mysql].
=====
+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| mysql-14      | idle  | 7.607           | 0         | pooled |
| mysql-4       | idle  | 7.609           | 0         | pooled |
| mysql-20      | idle  | 7.605           | 0         | pooled |
| mysql-2       | idle  | 7.61            | 0         | pooled |
| mysql-6       | idle  | 7.609           | 0         | pooled |
| mysql-8       | idle  | 7.608           | 0         | pooled |
| mysql-7       | idle  | 7.609           | 0         | pooled |
| mysql-17      | idle  | 7.606           | 0         | pooled |
| mysql-12      | idle  | 7.607           | 0         | pooled |
| mysql-1       | idle  | 7.61            | 0         | pooled |
| mysql-5       | idle  | 7.609           | 0         | pooled |
| mysql-15      | idle  | 7.606           | 0         | pooled |
| mysql-16      | idle  | 7.606           | 0         | pooled |
| mysql-11      | idle  | 7.607           | 0         | pooled |
| mysql-13      | idle  | 7.607           | 0         | pooled |
| mysql-10      | idle  | 7.608           | 0         | pooled |
| mysql-3       | idle  | 7.61            | 0         | pooled |
| mysql-9       | idle  | 7.608           | 0         | pooled |
| mysql-18      | idle  | 7.606           | 0         | pooled |
| mysql-19      | idle  | 7.605           | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
```


다음은 Connection Pool 상세 정보 항목에 대한 설명이다.

항목	설명
Connection ID	Connection 식별자이다.
State	Connection의 상태이다. <ul style="list-style-type: none">◦ idle : 사용 가능한 상태이다.◦ active : 사용 중인 상태이다.
State Time	State 지속 시간이다.
Use Count	Connection 사용 횟수이다.
Type	일회용 Connection인지의 여부를 나타낸다. <ul style="list-style-type: none">◦ disposable : 일회용 Connection인 경우를 의미한다.◦ pooled : 일회용 Connection이 아닌 경우를 의미한다.

3. 인바운드 관리

본 장에서는 EIS에서 애플리케이션으로 향하는 인바운드 커뮤니케이션에서 JEUS의 역할과 기능에 대해 설명한다. 주로 Work Manager, Message Driven Beans(MDB)와 리소스 어댑터 간의 연동에 관련하여 설명한다.

3.1. Work Manager 관리

본 절에서는 Work Manager의 기본 개념과 설정 방법에 대해서 설명한다.

3.1.1. 기본 개념

WAS에서 백그라운드로 어떤 일을 실행시키거나 다른 애플리케이션으로 정보를 넘기고 싶을 때 리소스 어댑터에서 Java Thread를 생성한다. 그러나 리소스 어댑터 내에서 직접 Java Thread를 생성하는 것은 WAS 입장에서 반길만한 상황이 아니다. 그렇기 때문에 리소스 어댑터에서 어떤 일을 실행시키고 싶으면 그것을 WAS로 넘겨주고, 직접적인 Thread 관리는 WAS가 대신해 주겠다는 것이 Work Manager의 기본 의도이다. 이때 일은 `jakarta.resource.spi.work.Work` 인터페이스를 구현한 인스턴스로 나타낸다.

JEUS에서는 Thread Pool을 기반으로 Work Manager를 제공한다. Thread Pool은 리소스 어댑터가 Work Manager를 실제로 사용할 때 생성해 주기 때문에 일단 `jakarta.resource.spi.BootstrapContext`를 통해서 항상 유효한 Work Manager 인스턴스를 넘겨준다.

JEUS의 Work Manager 설정은 Thread Pool 스타일의 설정으로 되어 있다. 만약 아무런 설정을 하지 않을 경우에는 `jeus-connector-dd.xsd` 스키마에 정의된 기본값으로 생성된다.



Work Manager 및 Work에 대한 자세한 사항은 JCA 표준 2.0을 기준으로 "11. Work Management"를 참고한다.

JDK Thread Pool은 최솟값(JDK에서는 core size라고 함)만큼 Thread를 늘린 이후에는 Thread를 무조건 큐에 쌓는다. 큐가 가득 차면 Thread를 최댓값까지 늘리는 방식이다.

JEUS의 Thread Pool은 JDK에서 제공하는 Thread Pool(`java.util.concurrent.ThreadPoolExecutor`)의 동작과 거의 유사하다. 그러나 JEUS의 Thread Pool은 Thread를 늘리는 조건을 JDK 방식보다 좀더 완화해서 일의 양에 따라 Thread를 적절히 늘리는 방식을 취하고 있다. 따라서 리소스 어댑터가 Work Manager를 빈번하게 사용하는 경우에는 최솟값, 최댓값 외에도 `<keep-alive-time>`과 `<queue-size>`를 적절하게 조절해야 한다.

3.1.2. Work Manager 설정

Work Manager는 내부적으로 Thread Pool을 사용하기 때문에 결국 Thread Pool 설정과 유사하다. Work Manager 설정은 리소스 어댑터에 포함되는 `jeus-connector-dd.xml`의 `<worker-pool>`에 한다.

다음은 Work Manager의 설정 예제이다.

Work Manager 설정 : <jeus-connector-dd.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <worker-pool>
    <min>0</min>
    <max>5</max>
    <keep-alive-time>60000</keep-alive-time>
    <shutdown-timeout>-1</shutdown-timeout>
  </worker-pool>
</jeus-connector-dd>
```

다음은 <worker-pool>의 태그에 대한 설명이다.

태그	설명
<min>	Work Manager가 관리하는 Thread 수의 최솟값이다. (기본값: 0)
<max>	Work Manager가 관리하는 Thread 수의 최댓값이다. (기본값: 5)
<keep-alive-time>	최솟값 이외의 Thread의 경우 설정된 시간 동안 사용되지 않는다면 자동적으로 Thread Pool에서 제거된다. (기본값: 1분) <pooled-timeout>을 대체하는 설정이다.
<queue-size>	Thread Pool에서 필요로 하는 큐의 크기를 지정한다. (기본값: 4096)
<pre-allocation>	Work Manager가 초기화될 때 <min> 값에 설정된 수의 Thread를 미리 만들어 놓는다. (기본값: true)
<shutdown-timeout>	리소스 어댑터를 undeploy하면서 Work Manager가 종료될 때 여기에 설정된 시간 동안 기다린 후 종료된다. 기다리는 동안 새로운 요청은 받지 않는다. 즉, Graceful Shutdown을 지원한다. (기본값 : -1, 기다리지 않고 종료한다)



jeus-connector-dd.xml를 RAR에 포함시키는 방법은 [리소스 어댑터](#)를 참고한다.

3.2. Message Inflow

JCA 표준에 따라 리소스 어댑터에서 JEUS에 deploy된 애플리케이션으로 인바운드 커뮤니케이션을 하기 위해서는 MDB를 구현해야 한다. 그리고 MDB를 통해서 다른 EJB 컴포넌트를 호출하도록 권장하고 있다. [인바운드 Message Inflow](#)에 따르면 WAS에서 애플리케이션으로 가는 흐름에서 MDB가 1차적인 관문 역할을 한다.

본 절에서는 JEUS에서 MDB와 리소스 어댑터를 연동하는 방법에 대해서 설명한다.



Message Inflow의 자세한 사항은 JCA 표준 2.0을 기준으로 "14. Message Inflow" 또는 이에 관련된 서적을 참고한다.

다음은 MDB 예제이다.

```

@MessageDriven(
    activationConfig =
    {
        @ActivationConfigProperty(propertyName = "destinationType",
            propertyValue = "jakarta.jms.Queue"),
        , @ActivationConfigProperty(propertyName = "DestinationProperties",
            propertyValue = "imqDestinationName=Queue")
        , @ActivationConfigProperty(propertyName = "ProviderIntegrationMode",
            propertyValue = "jndi")
        , @ActivationConfigProperty(propertyName = "ConnectionFactoryJndiName",
            propertyValue = "XAConnectionFactory")
        , @ActivationConfigProperty(propertyName = "DestinationJndiName",
            propertyValue = "jms/QUEUE1")
    }
)

public class TestMsgBean implements jakarta.jms.MessageListener {
    ...

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void onMessage(Message message) {
        ...
    }
}

```

EJB 3.0부터는 ejb-jar.xml 대신 Annotation으로 설정이 가능하다. MDB에는 **@ActivationConfigProperty**를 이용해서 리소스 어댑터가 요구하는 프로퍼티들을 반드시 설정해야 하며, 이에 관해서는 리소스 어댑터가 제공하는 매뉴얼 등을 참조한다.

MDB와 연동할 리소스 어댑터를 jeus-ejb-dd.xml의 **<mdb-resource-adapter-name>**을 설정해야 한다.

MDB 연동 리소스 설정 : <jeus-ejb-dd.xml>

```

<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>TestMsgBean</ejb-name>
            <connection-factory-name>QueueConnectionFactory</connection-factory-name>
            <destination>jms/QUEUE1</destination>
            <mdb-resource-adapter-name>app#ra</mdb-resource-adapter-name>
            ...
        </jeus-bean>
    . . .

```

태그	설명
<mdb-resource-adapter-name>	<p>연동할 리소스 어댑터를 지정한다. 이때 리소스 어댑터가 Standalone 모듈인 경우와 EAR에 포함되어 있는 모듈인 경우에 따라 리소스 어댑터 이름이 달라진다.</p> <ul style="list-style-type: none"> ◦ Standalone 모듈인 경우: 해당 모듈의 이름 ◦ EAR에 속한 모듈인 경우: EAR 이름 + '#' + 해당 모듈의 이름 <p>위의 예제는 리소스 어댑터가 EAR에 속한 모듈인 경우이고, EAR의 이름이 'app', 리소스 어댑터 모듈의 이름이 'ra'이다.</p>



MDB를 deploy할 때 리소스 어댑터가 먼저 deploy되어 있어야 한다는 점에 주의한다.

4. 리소스 어댑터

본 장에서는 JEUS에서 제공하는 보안 관리 기능과 리소스 어댑터에 jeus-connector-dd.xml를 추가하는 방법 및 deploy할 때 유의할 사항 등에 대해 설명한다.



실제 리소스 어댑터 자체에 대한 패키징은 JCA 표준 2.0를 기준으로 "21. Packaging Requirements"를 참고한다.

4.1. 보안 관리

본 절에서는 JEUS가 리소스 어댑터의 인증 및 권한 체크와 관련하여 어떤 기능을 제공하는지 설명한다.

4.1.1. Connection 인증

JCA 표준에 명시된 바와 같이 ejb-jar.xml, web.xml 등에 기술된 내용을 바탕으로 Connection 인증을 누가할 것인지 판별하게 된다.

```
<resource-ref>
  <res-ref-name>jca/pool</res-ref-name>
  <res-type>jakarta.resource.cci.ConnectionFactory</res-type>
  <res-sharing-scope>Unshareable</res-sharing-scope>
  <res-auth>Container</res-auth>
</resource-ref>
```

- <res-auth>

각 애플리케이션 컴포넌트별로 인증 역할을 컨테이너가 담당할지 애플리케이션이 담당할지 설정한다. (기본값: Container)

설정값	설명
Container	<p>컨테이너에 의한 Connection 인증으로 <res-auth>를 'Container'로 설정한 경우에는 jeus-connector-dd.xml에 사용자명과 패스워드를 설정한다. 이때 패스워드에는 암호화된 값을 사용할 수 있다.</p> <p>다음은 패스워드 설정에 대한 예이다.</p> <pre>{DES}FQrLbQ/D8011DVS71L28rw==</pre> <p>이렇게 설정한 사용자명과 패스워드 정보는 Connection을 새로 생성할 때 리소스 어댑터로 넘겨주는 인증 정보로 사용하게 된다. 만약 사용자가 jeus-connector-dd.xml에 아무런 인증 정보를 설정하지 않았을 경우에는 빈 껍데기의 javax.security.auth.Subject 객체를 리소스 어댑터로 넘겨준다.</p> <p>패스워드 암호화에 대한 자세한 사항은 JEUS Domain 안내서의 "Security 관리"를 참고한다.</p>
Application	<p><res-auth>를 'Application'로 설정한 경우에는 애플리케이션이 Connection을 요청할 때 JEUS는 Connection 인증에 관여하지 않는다. 대신 애플리케이션과 리소스 어댑터 간에 인증 정보를 주고받게 된다. 그 정보는 보통 jakarta.resource.spi.ConnectionRequestInfo를 구현한 리소스 어댑터의 클래스를 이용하게 된다.</p>

4.2. 패키징

리소스 어댑터를 JEUS에 deploy하기 위해서는 ra.xml 이외에 JEUS에서 필요로 하는 별도의 DD(Deployment Descriptor)로 jeus-connector-dd.xml 파일을 생성해야 한다.

이 파일에는 다음과 같은 내용이 설정된다.

- Work Manager 설정: [Work Manager 설정](#)
- 아웃바운드 Connection Pool 설정: [Connection Pool 설정](#)

설정한 후 RAR 파일의 META-INF 디렉터리에 jeus-connector-dd.xml을 위치시킨다.

```
xxx.rar/META-INF
```

4.3. Deploy

리소스 어댑터는 다음과 같이 2가지 형태로 deploy가 가능하다.

- 독립적인(Standalone) 모듈 : JEUS의 모든 애플리케이션에서 사용할 수 있다.
- Jakarta EE 애플리케이션(EAR)에 속한 모듈 : EAR 내에서만 사용할 수 있다.



JEUS에서 애플리케이션을 deploy하는 방법에 관한 자세한 사항은 "JEUS Applications & Deployment 안내서"를 참고한다.

4.3.1. SHARED 모드 클래스 로딩

JCA 표준상 독립적인(Standalone) 모듈로 deploy하는 리소스 어댑터는 모든 애플리케이션이 사용 가능해야 하기 때문에 JEUS에서는 SHARED 모드의 클래스 로딩 방식을 지원한다. 이를 위해서 리소스 어댑터는 사용자의 설정에 관계없이 항상 SHARED 모드로 deploy한다. 이때 리소스 어댑터를 사용할 애플리케이션도 반드시 SHARED 모드로 deploy해야 한다.



1. SHARED 모드의 클래스 로딩 방식의 자세한 사항은 JEUS Server 안내서의 "클래스 로더의 구조"를 참고한다.
2. deploy하는 방법의 자세한 사항은 "JEUS Applications & Deployment 안내서"를 참고한다.

4.3.2. Redeploy

리소스 어댑터는 일종의 JDBC 드라이버처럼 JEUS에 등록된다. JDBC 드라이버의 경우에는 JEUS_HOME/lib/datasource 아래에 JAR 파일을 두면 서버의 클래스 패스로 등록되는 방식으로 중간에 JAR 파일을 교체해도 제대로 반영이 되지 않기 때문에 JEUS를 Shutdown해야 한다.

그러나 리소스 어댑터는 JEUS에서 관리하는 애플리케이션이기 때문에 JEUS를 Shutdown하지 않고 리소스 어댑터의 버전 업그레이드를 할 수 있고 redeploy가 가능하다.

다음은 redeploy할 때 제약 사항이다.

- 리소스 어댑터 모듈을 redeploy할 때는 그것을 사용하는 애플리케이션들을 모두 redeploy해야 한다.
기존의 리소스 어댑터를 사용하고 있던 애플리케이션들은 이미 클래스를 Cache하고 있기 때문에 redeploy한 리소스 어댑터의 클래스들을 찾지 않는다.
- 현재 JEUS에서는 SHARED 모드로 deploy한 EJB 모듈을 redeploy할 경우 그것을 사용하던 웹 모듈도 모두 자동으로 redeploy하고 있다. 그러나 리소스 어댑터 모듈에 대해서는 아직 자동 redeploy를 지원하지 않고 있으므로 수동으로 해야 한다.

4.4. 리소스 어댑터를 리소스로 등록

리소스 어댑터 모듈은 하나의 독립된 애플리케이션이라기보다는 모든 애플리케이션이 공유해서 사용하는 드라이버와 같은 개념으로 이해될 수 있다. 그와 같은 관점에서 JEUS는 리소스 어댑터 모듈을 도메인에 커넥터 리소스로서 등록하여 사용할 수 있는 기능을 제공한다.

도메인에 등록되는 리소스 어댑터의 설정 정보는 리소스 어댑터 모듈의 jeus-connector-dd.xml의 설정 정보와 동일하다. 이때 리소스 어댑터는 본래의 모듈로서의 역할만을 수행한다.

Appendix A: 환경설정 주의사항

jeus-connector-dd.xml을 작성할 때 반드시 고려해야 할 사항은 다음과 같다.

- **ra.xml에 <outbound-resourceadapter><connection-definition> 존재 유무**

리소스 어댑터는 인바운드 목적만으로 사용할 수 있기 때문에 아웃바운드 설정이 없을 수도 있다. 이런 경우에는 jeus-connector-dd.xml을 작성하지 않아도 된다. 그러나 <connection-definition>이 있을 경우에는 그에 맞춰서 Connection Pool을 생성해야 하므로 반드시 jeus-connector-dd.xml나 domain.xml의 external-source에 connector 설정이 있어야 한다.

- **ra.xml에 <outbound-resourceadapter><connection-definition>이 2개 이상 존재하는 경우**

<connection-definition>이 2개 이상 설정된 경우에는 그에 맞춰서 최소 하나 이상의 Connection Pool을 생성해야 한다. 이때 주의할 점은 각각의 <connection-definition> 설정에 맞춰 <connectionfactory-interface> 값을 jeus-connector-dd.xml나 domain.xml의 external-source에 connector 설정이 있어야 한다. [Connection Pool 설정 예제](#)를 참고한다.

만약 Connection Pool을 생성하지 않으려면 ra.xml을 조정한다.

- **Work Manager 설정을 조정하고 싶은 경우**

jeus-connector-dd.xml을 설정하지 않아도 기본적으로 리소스 어댑터에게 Work Manager를 제공한다. Work Manager는 리소스 어댑터가 필요할 때 가져가서 사용하는 것이기 때문에 JEUS는 처음부터 이를 초기화하지 않는다.

대신 실제 요청을 할 때 Work Manager를 생성한다. Work Manager를 Thread Pool 설정이라고 할 수 있는데 Thread 수를 조정하고 싶은 경우에는 jeus-connector-dd.xml을 작성하여 Work Manager를 설정한다. Work Manager 설정 방법에 대한 자세한 내용은 [Work Manager 설정](#)을 참고한다.