JEUS Web Service
안내서

Tmax Soft

Document info

Document name: JEUS Web Service 안내서

Document date: 2005-06-06

Manual release version: 3

Software Version: JEUS 5

차 례

그림 목차

표 목차

매뉴얼에 대해서

매뉴얼의 대상

본 문서는 JEUS Web Container 와 JEUS EJB Container 에서 웹서비스 어플리케이션을 개발, 배치 그리고 유지보수 하는 개발자들은 반드시 읽어야 한다.

매뉴얼의 전제 조건

본 매뉴얼을 읽기 위해서는 아래의 세 가지 사전 지식들이 필요하다.

1. Servlet 개발과 패키징에 대한 기본적인 이해가 요구된다. 만약 Servlet 관련 지식이 부족하다면 스펙(http://java.sun.com)을 참조하거나 관련 도서를 읽기 바란다.

2. EJB 개념에 관한 기본적인 이해가 요구된다. 간단히 말하면, 본 매뉴얼에서는 J2EE 스펙에 대한 언급이 거의 없다.

3. XML, SOAP 프로토콜 그리고 WSDL 의 개념에 관한 이해가 요구된다 .

참고: 본 매뉴얼은 Servlet 또는 EJB 기술처럼 기본적인 J2EE 에 대한 자세한 설명은 생략한다. 본 매뉴얼을 읽기 전에 아래의 문서를 살펴 보기를 바란다.

- JEUS Server 안내서 (기본 개념을 위해)
- JEUS Web Container 안내서 (JEUS Servlet 매뉴얼)
- JEUS EJB 안내서 (만약 EJB 형태의 웹서비스를 구현하기 원한다면)
- 기본적인 웹서비스 관련 지식은 XML, SOAP 그리고 WSDL 이면 충분하다(아래 절의 관련문서를 참조하라).

매뉴얼의 구성

이 매뉴얼은 기본적인 웹서비스의 개념 설명에서 부터 시작하여 J2EE 플랫폼에서 프로그래밍이 가능한 개발자라면 웹 서비스를 모르는 상태에서도 웹 서비스 생성이 가능하게 하기 위한 지침서이다.

본 매뉴얼은 아래처럼 14 개의 장으로 나누어 졌다.

1. 웹 서비스의 소개: 웹서비스의 기본 개념
2. JEUS 웹 서비스의 개요: JEUS 에서 지원하는 웹 서비스의 개념과 지원 방식
3. JEUS 웹 서비스의 설계: JEUS 에서 웹 서비스를 생성하기 위한 설계 지침
4. JEUS 웹 서비스의 구현: 웹 서비스를 구현하는 여러 방식에 대한 설명
5. 자바 클래스 웹 서비스의 생성과 배치: 자바 클래스를 엔드포인트로 가지는 웹 서비스의 생성과 배치 방법에 대한 설명
6. EJB 웹 서비스의 생성과 배치: EJB 를 엔드포인트로 가지는 웹 서비스의 생성과 배치 방법에 대한 설명
7. 웹서비스의 호출: 웹서비스를 호출하기 위한 클라이언트 작성과 호출 방법에 대한 설명
8. JEUS 서버로부터의 웹 서비스 호출 (J2EE 클라이언트): 웹 서비스를 호출하기 위한 J2EE 클라이언트 작성과 호출 방법에 대한 설명
9. SOAP 메시지 핸들러의 생성: 메시지 핸들러와 SAAJ 를 이용한 SOAP 메시지 처리에 대한 설명
10. UDDI 이용: UDDI 개요와 운영 및 사용에 대한 설명
11. 웹 서비스 DD(webservices.xml)의 작성: 표준 웹 서비스 배치 서술자 작성에 대한 설명
12. JAX-RPC 매핑 파일의 작성: JAX-RPC 매핑 파일 작성에 대한 설명

13. 데이터 타입: JAX-RPC 스펙에 근거하여 데이터 타입에 관련된 향상된 웹 서비스 프로그래밍을 주제로 기술

14. 웹서비스 보안 : 이 장에서는 JEUS 에서 웹 서비스에 보안을 적용하기 위한 방법을 설명한다.

15. **결론**

부록:

A JEUS 웹 서비스 Ant Task 참조

B JEUS 웹 서비스 커맨드 라인 툴 참조

C JEUS 웹 서비스 설정 파일 작성

D JEUS 웹 서비스 배치 서술자 (jeus-webservices-dd.xml) 작성

E JEUS 웹 서비스 클라이언트 배치 서술자(jeus-webservicesclient-dd.xml) 작성

F JEUS 4 웹 서비스 Ant Task 참조

관련 매뉴얼

관련문서와 스펙들:

- SOAP 1.1 Specification (http://www.w3.org/TR/soap11)
- SOAP 1.2 Specification (http://www.w3.org/TR/soap12)
- WSDL 1.1 Specification (http://www.w3.org/TR/wsdl)
- UDDI 2.0 Specification (http://uddi.org/pubs/ProgrammersAPI_v2.htm, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2)
- UDDI 3.0 Specification (http://uddi.org/pubs/uddi_v3.htm)
- JAX-RPC 1.1 Specification (http://java.sun.com/xml/jaxrpc)

- SAAJ 1.2 Specification (http://java.sun.com/xml/saaj)
- JAXR 1.0 Specification (http://java.sun.com/xml/jaxr)
- WS-I Basic Profile 1.0 (http://www.ws-i.org/Profiles/BasicProfile-1.0.html)
- Servlet 2.4 Specification (http://java.sun.com/products/servlet)
- EJB 2.1 Specification (http://java.sun.com/products/ejb)
- JEUS Server 안내서
- JEUS Web Container 안내서
- JEUS EJB 안내서

일러두기

표기 예	내용
텍스트	본문, 12 포인트, 바탕체 Times New Roman
텍스트	본문 강조
CTRL+C	Ctrl 와 동시에 C 를 누름
`public class myClass { }`	Java 코드.
`<system-config>`	XML 문서.
참조: / 주의:	참조 사항과 주의할 사항
Configuration 메뉴를 연다	GUI 의 버튼 같은 컴포넌트
JEUS_HOME	JEUS 가 실제로 설치된 디렉토리

	예)c:\jeus
`jeusadmin nodename`	콘솔 명령어와 문법
[파라미터]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택 사항. 예) A|B: A 나 B 중 하나
…	파라미터 등이 반복되어서 나옴
?, +, *	보통 XML 문서에 각각 "없거나, 한 번", "한 번 이상", "없거나, 여러 번"을 나타낸다.
. . .	XML 이나 코드 등의 생략
<<*FileName.ext*>>	코드의 파일명
그림 1.	그림 이름이나 표 이름

OS 에 대해서

본 문서에서는 모든 예제와 환경 설정을 Microsoft Windows™의 스타일을 따랐다. 유닉스같이 다른 환경에서 작업하는 사람은 몇 가지 사항만 고려하면 별 무리 없이 사용할 수 있다. 대표적인 것이 디렉토리의 구분자인데, Windows 스타일인 "\"를 유닉스 스타일인 "/"로 바꿔서 사용하면 무리가 없다. 이외에 환경 변수도 유닉스 스타일로 변경해서 사용하면 된다.

그러나 Java 표준을 고려해서 문서를 작성했기 때문에, 대부분의 내용은 동일하게 적용된다.

용어 설명

다음에 소개되는 용어는 본 문서 전체에 걸쳐서 사용되는 용어이다. 용어가 이해하기 어렵거나 명확하지 않을 때는 아래 정의를 참조하기 바란다.

용어	정의
DII client	Dynamic Invocation Interface 클라이언트의 줄임말. DII 클라이언트는 JAX-RPC 클라이언트 API에 의해 작성된 웹서비스 클라이언트이다.
EJB	Enterprise JavaBeans 의 줄임말.
In/out parameter	In/out 파라미터는 입력과 출력에 사용되는 웹서비스 operation 의 파라미터이다.
JAX-RPC	Java API for XML-Based RPC 의 약자. JAX-RPC 는 JCP(Java Community Process)에서 정의한 Java 웹서비스를 위한 표준 API 이다 .
JAX-RPC Holder Class	JAX-RPC Holder 클래스는 입력/출력 파라미터를 지원하는 Java 클래스이다.
JAX-RPC value type	JAX-RPC 값 타입은 웹서비스의 요청과 응답 값에 의해 네트웍 상으로 전송되는 내용의 데이터 타입 이다.
JEUS	*Java Enterprise User Solution* 의 줄임말. 본 문서에서 기술된 JEUS version 5 는 J2EE1.4 를 준수한 WAS 이다.
Proxy Client	proxy 클라이언트는 wsdl2java Ant task 에 의해 WSDL 문서로부터 생성된 stub 코드를 통해서 작성된 웹서비스 클라이언트이다.

용어	정의
SAAJ	*SOAP message with Attachments API for Java 의 줄임말*. SAAJ 는 JCP(Java Community Process)에 의해 정의된 SOAP 메시지 처리를 위한 표준 Java API 이다.
SOAP	*Simple Object Access Protocol* 의 줄임말. SOAP 은 프로그래밍언어와 운영체제와 상관없이 네트웍 상에서 XML 데이터를 운반하기 위해 W3C 에 의해 정의된 표준 프로토콜이다.
WAS	Web Application Server 의 줄임말.
WSDL	*Web Service Description Language* 의 줄임말. WSDL 은 웹서비스의 인터페이스와 구현을 기술하기 위해 W3C 에 의해 정의된 표준 XML 인스턴스이다.

연락처

Korea
Tmax Soft Co., Ltd
18F Glass Tower, 946-1, Daechi-Dong, Kangnam-Gu
Seoul 135-708
South Korea
Email: info@tmax.co.kr
Web (Korean):http://www.tmax.co.kr

USA
Tmax Soft, Co.Ltd.
2550 North First Street, Suite 110
San Jose, CA 95131
USA
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com

Japan
Tmax Soft Japan Co., Ltd.
6-7 Sanbancho, Chiyoda-ku,
Tokyo 102-0075
Japan
Email: info@tmaxsoft.co.jp
Web (Japanese): http://www.tmaxsoft.co.jp

China
Beijing Silver Tower, RM 1507, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
Tel: 86-10-64106148 Fax: 86-10-64106144
E-mail : info@tmaxchina.com.cn
Web(Chinese): http://www.tmaxchina.com.cn

1 웹 서비스의 소개

웹 서비스는 애플리케이션들이 플랫폼과 프로그래밍 언어와는 독립된 방식으로 서로 통신할 수 있도록 하는 표준화된 기술이며 표준 XML 메시징을 통해 네트워크로 접근 되어 질 수 있는 오퍼레이션들을 기술하는 소프트웨어 인터페이스이다. 또 웹서비스는 인터넷에만 연결되어 있다면 서비스에 대한 권한을 가지고 있는 사용자 누구에게라도 비즈니스를 공개하고 사용되어 질 수 있도록 메시징 프로토콜, 프로그래밍 표준, 서비스 발견을 위한 편의 환경 등을 정의하고 있다.

1.1 웹 서비스의 이해

웹 서비스는 인터넷 상의 URI 로 접근 가능한 응용 프로그램이며 웹 서비스의 인터페이스와 바인딩은 XML 문서로 정의되어지고, 기술되며, 발견되어 질 수 있다. 하나의 웹 서비스는 인터넷에 공개되어진 또 다른 웹 서비스와 상호 연동이 가능할 뿐 아니라 기존의 백엔드(back-end) 응용프로그램들과도 연동이 가능하다.

지금까지의 응용 프로그램 아키텍쳐는 두 가지의 범주에 속해 있었다. 하나는 메인프레임을 기반으로 작동하는 단일화 된 시스템이며, 다른 하나는 데스크 탑에서 작동하는 클라이언트-서버(client-server) 기반의 시스템이었다. 이들 시스템들은 물론 모두 다 잘 작동하지만, 이러한 아키텍쳐 위에 동작하는 프로그램들은 그 시스템에서만 작동할 수 있게 맞추어진 프로그램들이라는 한계점을 가지고 있다. 이러한 아키텍쳐 내에서의 프로그램들은 아주 폐쇄적이어서 접근이 용이하지 않으므로 웹 상에 존재하는 무수히 많은 사용자들에게는 무용지물이다.

그래서 소프트웨어 산업은 서비스지향 아키텍쳐 (SOA:Service-Oriented Architecture) 라는 방향으로 진화하게 되었고, 이 기반위에서의 응용 프로그램들은 웹 상에서 동적으로 상호 작용 할 수 있게 되었다. 응용프로그램은 큰 규모로 이루어져 있던 소프트웨어 시스템을 더 작게 모듈화된 여러 서브 시스템으로 구성하게 만들었고, 이렇게 작게 모듈화된 서브 소프트웨어 시스템들은 각각 다양한 기술들로 구현할 수 있게 되었으며, 하나의 컴퓨터에 존

재하지 않아도 되며, 재사용 할 수 있게 되었다. 또한 XML 과 HTTP 같은 표준 웹 프로토콜을 사용하여 인터넷 상의 어떠한 사용자라도 쉽게 접근 할 수 있게 만들었다.

이러한 서비스 지향 기술은 이미 수년전부터 RMI, COM, 그리고 CORBA 와 같은 여러 다양한 형태의 기술들로 구현되어져 왔으므로 최근에야 발생한 완전한 새로운 형태의 것이라고 보기는 힘들다. 하지만 이렇게 언급한 기술들은 벤더에 종속적이거나 구현된 기술에 종속적이라는 단점이 있다. 하지만 웹 서비스는 이러한 단점들을 극복한다.

웹 서비스는 다음과 같은 특징이 있다.

- 웹 서비스는 웹을 통해 접근 할 수 있다.
- 웹 서비스는 스스로 표방하고 서술한다. – 웹 서비스는 스스로의 역할과 기능, 속성에 대해서 서술함에 따라서 웹 서비스 클라이언트가 서비스에 대한 이해를 할 수 있게한다. WSDL(Web Service Description Language)이라는 파일에 서비스를 서술하여 이를 공개함으로써 다른 응용프로그램에서 서비스를 이용할 수 있게 한다.
- 웹 서비스는 HTTP 와 같은 표준 인터넷 프로토콜에 의해 전달되는 XML 메시지를 통해 웹 서비스 클라이언트와 교신한다. – 웹 서비스 클라이언트는 응용프로그램 일 수도 있고, 다른 웹 서비스 일 수도 있다.
- 웹 서비스는 request-response 혹은 one-way 방식으로 작동하며, 동기 혹은 비동기 통신으로 호출 되어 질 수 있다. 이러한 작동방식과 통신방식에는 무관하게 웹 서비스와 웹 서비스 클라이언트간에 교환되는 근본적인 단위는 메시지이다.

1.2 웹 서비스의 이점

웹 서비스에는 다음과 같은 이점이 있다.

- 인터넷 공개 표준을 지원한다. 웹 서비스는 SOAP(Simple Object Access Protocol), WSDL, UDDI(Universal Description, Discovery, and Integration)와 같은 공개 표준을 정하여 이를 근간으로 하여 상호 작용이 이루어 지므로 웹 서비스를 지원하는 응용 프로그램들은 상호 작동에 문제가 없게된다.

- 플랫폼과 언어에 독립적이다.
- HTTP 와 같은 웹 프로토콜을 사용하므로 방화벽과 같은 장애에도 문제가 없어 응용프로그램에 대한 접근이 용이하다.

1.3 웹 서비스 표준

J2EE (Java 2 Platform, Enterprise Edition) 버전 1.4 에 이르러서는 웹 서비스의 중요성이 반영되어 마침내 웹 서비스 스펙을 포함하기에 이르렀다. 웹 서비스는 이제 J2EE 플랫폼의 많은 서비스 채널 중의 하나가 되었고, 기존의 J2EE 콤포넌트들은 웹 서비스로 공개될 수 있다. 이식성(portability), 확장성(scalability), 신뢰성(reliability), 플랫폼 독립성과 같은 J2EE 플랫폼의 많은 장점들이 이제 웹 서비스 환경에서 적용 가능하게 되어졌다. 이는 J2EE 1.3 환경에서 제공되어지던 웹 서비스와의 가장 큰 차이점이다. 예를들면 트랜잭션 지원, 데이터 베이스 연결, 생성 주기(life cycle) 관리등과 같은 여러 확장성있는 서비스들을 J2EE 컨테이너로부터 응용프로그램의 소스 코드 수정 없이 지원받을 수 있게 되었다.

현재 J2EE 웹 서비스는 다음과 같은 표준을 요구하고 있다.

- SOAP(Simple Object Access Protocol) 표준
- WSDL(Web Service Description Language) 표준
- UDDI(Universal Description, Discovery and Integration) 표준
- JAX-RPC(Java API for XML-based RPC) 표준
- SAAJ(SOAP with Attachments API for Java) 표준

1.3.1 SOAP 표준

SOAP(Simple Object Access Protocol) 은 분산 환경에서의 정보 교환을 목적으로 하는 경량의 XML 기반 프로토콜이다. SOAP 은 RPC(Remote Procedure Call) 방식과 메시지 기반(Message-oriented) 방식등 여러 방식의 정보 교환 방식을 지원한다. RPC 방식의 정보 교환은 요청-응답 프로세스를 허용하며, 서비스 엔드포인트는 프로시져 중심의 메시지를 받아서 적절한 응답 메시지를 작성하여 되돌려 준다. 메시지 기반 방식은 송신자가 서비스의 응답을 즉각적으로 요구하지 않아도 될 때에도 사용이 가능하며, 주로 비즈니스와 여

러가지 형태의 문서 타입을 교환할 필요가 있는 경우 사용된다. 메시지 기반 방식의 정보 교환은 문서(Document) 방식의 정보 교환이라고 불리기도 한다.

SOAP 프로토콜은 다음과 같은 요소들로 구성된다.

- SOAP 메시지를 서술하는 엔벨로프(Envelope) - 엔벨로프는 누가 어떻게 메시지를 처리해야 하는지에 대한 정보를 가지고 있으며, 메시지의 몸체(Body)를 감싸고 있다.
- 응용 프로그램에 사용되는 데이터 타입의 객체를 설명하는 인코딩 법칙(Encoding rule)
- 원격 프로시져 호출과 응답을 나타내는 데 필요한 준수 사항들

그리고 SOAP 은 다음과 같은 특성이 있다.

- 프로토콜에 독립적이다.
- 구현 언어에 독립적이다.
- 플랫폼과 운영시스템(Operating System)에 독립적이다.
- SOAP XML 메시지에 MIME 타입과 같은 형태로 추가적인 메시지를 붙여 보낼 수 있다.

1.3.2 WSDL 표준

WSDL(Web Service Description Language)는 웹 서비스 기술 언어라고 하며, RPC 기반과 메시지 기반의 서비스를 서술하는 XML 형태를 의미한다. WSDL 파일은 개발 툴이나 서비스 개발자 혹은 배치자-서비스 제공자-에 의해서 생성되며 인터넷 상에 공개된다면 그 때부터 서비스는 다른 클라이언트에 의해 알려지게 되어 이는 클라이언트로 하여금 서비스에 접근할 수 있는 환경을 구축할 수 있게 한다.

클라이언트 프로그래머-서비스 소비자-들은 공개된 WSDL 을 이용하여 제공되어지는 웹 서비스에 대한 정보를 얻게 되고 그 정보를 바탕으로 웹 서비스에 접근할 수 있도록 프록시(proxy)나 템플릿(template)등을 생성할 수 있다.

1.3.3 UDDI 표준

UDDI(Universal Description, Discovery and Integration)표준은 웹 서비스 관련 정보의 공개와 탐색을 위한 표준이다. 서비스 제공자는 UDDI 라는 서비스

소비자에게 이미 알려진 온라인 저장소에 그들이 제공하는 서비스들을 저장하게 되고, 서비스 소비자들은 그 저장소에 접근함으로써 원하는 서비스들의 목록을 찾을 수 있게 된다.

1.3.4 JAX-RPC 표준

JAX-RPC(Java API for XML-based RPC)표준은 Sun Microsystems 에서 제공하는 웹 서비스 API 를 정의하는 표준이다. 웹 서비스가 해결해야 하는 당면 과제 중 하나는 상호 호환성을 가지는 서비스의 개발이다. 이러한 문제를 해결하기 위해 WS-I, OASIS, W3C 그리고 SOAPBuilders 같은 다양한 표준 단체들이 수많은 노력들을 경주해 왔다.

JAX-RPC 는 이러한 상호 호환성의 문제를 해결하고 웹 서비스 구현에 대한 유연한 프로그래밍 API 를 제공하기 위한 자바 진영의 노력의 산물이다. 이러한 API 를 채택함으로 인해 JAX-RPC 는 서비스 사용자와 개발자들로 하여금 상호 호환성이라는 아주 큰 짐을 런-타임(run-time) 프레임웍으로 덜게 해 주었다.

1.3.5 SAAJ 표준

SAAJ(SOAP with Attachments API for Java) 표준은 SOAP 프로토콜으로의 직접 접근 가능한 프로그래밍 인터페이스를 제공한다. 로-레벨(low-level) 프로토콜 처리에 익숙하다면, SOAP 메시지를 큰 어려움 없이 작성할 수 있다.

1.4 SOAP 메시지 교환과 SOAP 메시지 인코딩

SOAP 은 RPC 방식과 문서 방식의 정보 교환을 지원한다. SOAP 메시지는 RPC 방식이든 문서 방식이든 SOAP 메시지의 엘리먼트에 정의되어 있는 `encodingStyle` 속성에 의해 정의되는 인코딩 방식을 쓰게 된다. 여기서는 이러한 SOAP 메시지의 특성에 대해 언급한다.

1.4.1 SOAP 메시지의 구성 요소

SOAP 메시지는 SOAP `Envelope` 를 가지며 Envelope 는 `Header` 와 `Body` 라는 두 개의 하위 요소들을 가지게 된다. Header 는 필수적으로 포함되어야 하는 것은 아니다. Header 의 하위 요소들은 Header 블록이며 각각의 Header 블록은 데이터의 논리적인 묶음으로 볼 수 있다. SOAP Body 는 SOAP 메시지의 필수 요소이며 SOAP 메시지의 궁극적으로 전달하고자 하는 내용들이 담겨 있는 곳이다.

1.4.2 SOAP 메시지의 인코딩

SOAP 은 RPC 방식과 문서 방식의 정보 교환을 지원한다.

1.4.2.1 RPC 방식

RPC 방식은 원격 프로시져 호출(RPC) 방식의 메시지 교환 방식을 가리키며, 클라이언트와 서버가 잘 정의된 프로그래밍 모델로 만들어져야 하며, 클라이언트는 인자를 가지는 메쏘드를 호출하고 서버는 응답으로 하나의 값을 반환한다. RPC 방식의 메시지 교환 방식에서는 SOAP 은 메시지 Body 의 형태를 따로 정의한다.

1.4.2.2 문서(Document) 방식

문서 방식에서는 XML 문서가 교환되고, 각각의 엘리먼트의 의미는 서버와 클라이언트가 해석의 몫으로 남겨둔다. 즉 SOAP 메시지의 Body 의 구조에 대한 제약 사항은 SOAP 에서 규정하고 있지 않으며, 응용 프로그램이나 혹은 별도로 정해진 XML 스키마에 의해 SOAP Body 속에 있는 XML 문서의 구조가 결정된다.

2 JEUS 웹 서비스의 개요

이번 장에서는 JEUS 웹서비스와 지원되는 스펙의 개념을 기술한다. 또한 JEUS 웹서비스를 위한 설정 파일들과 툴들 그리고 시스템변수에 대해서 살펴 본다.

2.1 JEUS 웹서비스

JEUS 5 에 이르러서 가장 두드러진 차이점 중에 하나가 바로 웹 서비스이다. J2EE 1.4 스펙이 J2EE 1.3 에 비해 두드러진 변화가 웹 서비스의 표준화 채택이며 J2EE 1.4 인증을 세계 최초로 받은 WAS(Web Application Server)인 JEUS 5 는 J2EE 1.4 에서 요구하는 스펙을 모두 준수한다. 따라서 JEUS 웹 서비스는 J2EE 1.4 스펙을 준수하는 벤더들과의 웹 서비스 상호 호환성이 보장된다.

아래 그림은 JEUS 웹서비스의 구조를 보여준다[그림 1].

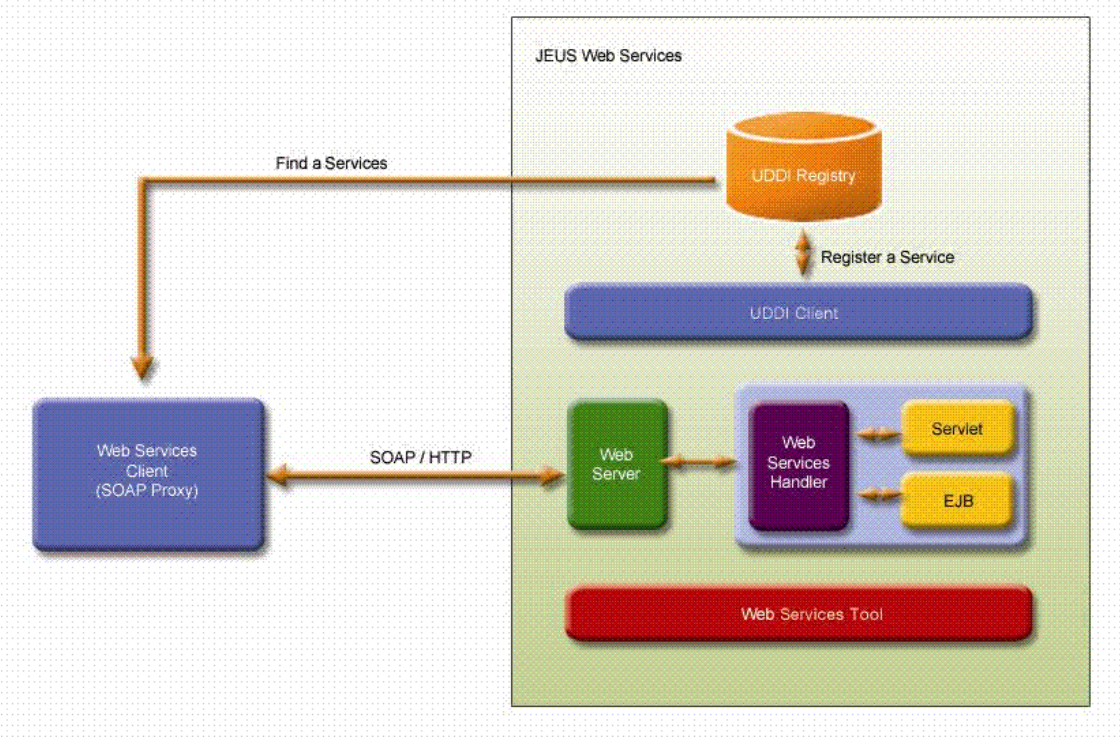


그림 1. JEUS 웹서비스의 구조.

2.2 웹서비스 스펙 지원

JEUS 웹서비스는 아래와 같은 스펙을 지원한다:

- SOAP 1.2 스펙
- WSDL 1.1 스펙
- JAX-RPC 1.1 스펙
- SAAJ 1.2 스펙
- UDDI 3.0 스펙
- JAX-R 1.0 스펙

2.3 JEUS 웹서비스와 관련된 XML 설정파일

JEUS 웹서비스는 뒷 단의 비즈니스 로직으로 EJB 와 Java 클래스를 사용한다. EJB 는 EJB Container 의 내부에서 실행되고 Java 클래스는 Web 컨테이너의 내부에서 실행된다. 그래서 'JEUS EJB 안내서' 와 'JEUS Web Container 안내서' 에서 설명한 XML 구성 파일들은 JEUS 웹서비스와 관련 있다.

JEUS 웹서비스는 webservices.xml 과 같은 웹서비스 Deployment Descriptor 를 사용한다. 파일 작성법은 본 매뉴얼에서 설명한다.

2.4 웹서비스에 관련된 툴과 유틸리티

JEUS 웹서비스와 WSDL 파일 관리를 위한 Java 클래스를 생성하기 위해서 커맨드 라인 툴과 Apache Ant 툴을 사용한다(부록 A JEUS 웹 서비스 Ant Task 참조). 커맨드 라인 툴과 Ant 툴 사용법은 다음 여러 장에 걸쳐서 설명할 것이다. Ant 툴 기능을 사용하기 위해서는 Apache Ant 1.6.1 이상을 인스톨 해야 한다. 'http://ant.apache.org'에서 Apache Ant 를 다운로드할 수 있다.

2.5 JEUS 웹서비스와 관련된 시스템 환경 변수들

JEUS 웹서비스를 위한 특별한 시스템 환경 변수는 없다. JEUS 웹서비스의 작동을 위해서는 'JEUS Web Container 안내서' 와 'JEUS EJB 안내서'에서 기술된 시스템 환경 변수만으로 충분하다.

2.6 본 매뉴얼에서 사용된 샘플들

JEUS 를 설치한 후에 JEUS_HOME\samples\manual_examples\webservice 디렉토리에는 소스 코드와 관련된 XML 파일들이 있다.

3 JEUS 웹 서비스의 설계

3.1 웹 서비스 Back-end 의 선택

웹서비스 컴포넌트는 Web 컨테이너나 EJB 컨테이너에서 실행되는 J2EE 컴포넌트이다. 웹서비스의 Back-end 는 아래의 개체들을 공개할 수 있다.

- Java 클래스 파일
- 무상태 세션 EJB(Stateless Session EJB)

Java 클래스 파일은 Web 컨테이너에서 실행되고 무상태 세션 빈은 EJB 컨테이너에서 실행된다.

3.1.1 Java 클래스 웹 서비스 Back-end

Java 클래스는 EJB 를 생성하는 것보다 보통 작업이 더 빠르고 간단하게 끝난다. 영속성(Persistence), 보안, 트랜잭션 등과 같은 EJB 컨테이너에서 제공하는 기능들이 크게 중요하지 않을 때, Java 클래스 웹 서비스 Back-end 가 좋은 선택이다.

3.1.2 무상태 세션 빈(Stateless Session Bean) 웹 서비스 Back-end

EJB 웹 서비스를 언급하기에 앞서, EJB 는 트랜잭션이 중요한 시스템에서 선택될 수 있는 프로그래밍 모델이다. 데이터 베이스에 업데이트나 삭제 같은 작업들을 많이 하는 시스템에서는, 비즈니스 로직을 EJB 로 구현한다면 트랜잭션 관리에 효율적이므로 좋은 선택이라고 볼 수 있다.

위와 같이 트랜잭션이 중요한 경우가 아닐 때에도 EJB Back-end 는 웹 서비스 Back-end 로서 좋은 선택이 될 수 있는데 바로 CMP(Container Managed Persistence)가 그 예이다. CMP 에 대해 더 세부적인 사항을 알아보려면 Enterprise Java Beans 매뉴얼을 참조한다.

이미 언급한 대로 기존의 비즈니스 로직은 필요에 의해 EJB 로-무상태 세션 빈이든 CMP 이든- 구성되어 있을 수 있다. 이미 무상태 세션빈으로 비즈니스 로직이 구현되어 있고, 이러한 상황에서 기존의 EJB 로직을 웹 서비스로

공개하고 싶다면 무상태 세션 빈 웹 서비스 Back-end 를 선택하는 것이 좋을 것이다.

일반적으로 EJB 웹서비스는 무상태 세션 빈만을 Back-end 로 허용하므로 CMP 는 웹 서비스와 연관이 없어 보일 수도 있지만, 무상태 세션 빈이 CMP 로 구현되어 있는 비즈니스 로직으로의 인터페이스 역할을 하도록 하고, 무상태 세션 빈이 웹서비스로 공개 된다면 CMP 비즈니스 로직을 웹 서비스로 공개하게 되는 것이다. 기존의 CMP 비즈니스 로직의 장점을 그대로 이용하면서 웹서비스 기능을 추가하게 될 때, 무상태 세션 빈 웹 서비스 Back-end 는 좋은 선택이라 볼 수 있다.

3.2 RPC 방식과 Document 방식의 선택

JEUS 웹 서비스는 RPC 방식과 문서(Document) 방식, 두 가지 방식을 지원한다. WSDL 1.1 스펙에 의하면, RPC 방식은 SOAP 메시지가 웹 서비스 오퍼레이션의 인자와 리턴 값을 포함하고, 문서 방식은 XML 문서를 SOAP 메시지가 포함한다.

또, RPC 방식은 원격 프로시져 호출(RPC) 방식의 메시지 교환 방식을 가리키며, 클라이언트와 서버가 잘 정의된 프로그래밍 모델로 만들어져야 하며, 클라이언트는 인자를 가지는 메쏘드를 호출하고 서버는 응답으로 하나의 값을 반환한다. 문서 방식에서는 XML 문서가 교환되고, 각각의 엘리먼트의 의미는 서버와 클라이언트가 해석의 몫으로 남겨둔다. 즉 SOAP 메시지의 Body 의 구조에 대한 제약 사항은 SOAP 에서 규정하고 있지 않으며, 응용 프로그램이나 혹은 별도로 정해진 XML 스키마에 의해 SOAP Body 속에 있는 XML 문서의 구조가 결정된다.

JEUS 웹 서비스에서 제공하는 문서 방식에는 표준 문서 방식과 WRAPPED 방식, 두가지가 존재한다.

표준 문서 방식에서는 각각의 웹 서비스 오퍼레이션은 한 개의 인자-XML 문서-만 가질 수 있다. WRAPPED 방식에서는 여러 인자들을 하나의 복합 데이터 타입(complex data type)으로 포장하여 하나의 인자로 만들어 SOAP 메시지에 담게 되므로 웹 서비스 오퍼레이션들은 여러 개의 인자들을 가질 수 있다.

RPC 방식에서는 인자들의 개수에 제한이 없다.

3.3 웹 서비스 구현 방식 선택

웹 서비스 구현은 서비스 Endpoint 로부터 시작하거나 WSDL 로부터 시작할 수 있다. 이것은 개발자의 취향과 개발 환경에 따라서 선택되며 각각 장점을 가지고 있다.

- 서비스 Endpoint 로부터 시작 : 서비스 구현이 단순하고 쉬우며 직관적으로 구현 가능함.
- WSDL 로부터 시작 : 구현 언어에 중립적으로 상호 운영성이 가능한 서비스를 구현하기에 적합함.

3.3.1 서비스 Endpoint 로부터 시작

서비스 Endpoint 를 먼저 구현하고 이것으로부터 WSDL 을 생성하는 방법이다. 이러한 방법은 서비스 구현을 쉽고 직관적으로 진행할 수 있는 장점이 있다. 이것은 개발자가 자신에게 익숙한 개발 환경, 즉 자바 환경에서 다른 플랫폼과의 상호 운영성에 대한 걱정없이 서비스 Endpoint 와 서비스 구현체를 개발할 수 있게 한다. 그 다음에 작성된 자바 코드로 부터 플랫폼 독립적인 WSDL 을 도출하게 된다. 이렇게 생성된 WSDL 이 플랫폼에 독립적으로 정의되기 위하여 JAX-RPC 는 어떻게 서비스 Endpoint 로부터 WSDL 을 도출하여야 하는지에 대한 지침을 제공한다. 이 방식은 특히, 자바 개발자에게는 웹 서비스 개발을 시작하는 가장 쉬운 방법이다.

이 방식에서의 유의할 사항을 서비스 설정에 관한 정보를 기술(description) 해야 한다는 것이다. JAX-RPC 스펙은 자바로부터 WSDL 로의 매핑에 대한 세세한 정의를 하고 있다. 그러나, 서비스 Endpoint 인터페이스 만으로는 완전한 WSDL 을 생성할 수 없다. 서비스 Endpoint 인터페이스는 WSDL 의 serivce 와 binding 에 대한 정보를 제공하지 못한다. JEUS 웹서비스 구현에서는 별도의 서비스 설정 파일을 작성하여 이러한 정보를 Java-to-WSDL 툴킷에 제공해야 한다. 웹서비스 구현은 제 4 장에 설명하고 있다.

3.3.2 WSDL 로부터 시작

WSDL 을 먼저 작성하고 이것으로부터 웹서비스 구현을 생성하는 방법이다. 웹 서비스의 중심 목표 중 하나는 플랫폼 사이의 상호 운영성이다. 이러한 방법은 개발 언어에 중립적으로 상호 운영성 중심적인 서비스를 생성하기 좋은 방법이다. 웹 서비스 구현으로부터 생성된 WSDL 은 플랫폼 특성이 반영된다. 즉 자바 편향된다. 하지만, WSDL 로부터 시작된 웹 서비스의 기술(description)은 웹서비스 기술에 사용된 XML 타입과 용어에서 보다 중립적

이다. 즉, 자바 환경에 익숙하지 않은 개발자에게도 친근할 수 있는 일반적 명명법으로 WSDL 을 작성할 수 있게 한다. 그러나 이 방식은 개발자로 하여금 WSDL 에 대한 보다 깊은 이해를 요구한다. 웹서비스 구현은 제 4 장에 설명하고 있다. WSDL 로부터 자바 코드의 생성은 '부록 A JEUS 웹 서비스 Ant Task 참조'의 wsdl2java 와 '부록 B JEUS 웹 서비스 커맨드 라인 툴 참조'의 wsdl2java 에서 설명하고 있다.

3.4 SOAP 메시지 핸들러의 생성

SOAP 메시지의 헤더(Header)나 몸체(Body), 혹은 부착물(Attachment)을 직접 다루고자 할 때, SOAP 메시지의 JAX-RPC 표준 핸들러를 생성하고, SAAJ(SOAP with Attachments API for Java)를 이용하여 직접 메시지를 다룰 수 있다. 보다 자세한 설명은 9 장 - SOAP 메시지 핸들러의 생성 - 을 참조하라.

4 JEUS 웹 서비스의 구현

JEUS 웹 서비스 구현작업은 웹서비스를 구성하는 백앤드(back-end) 구성 요소인 자바 코드를 작성 및 컴파일하는 것을 의미한다. 기본적으로 JEUS 웹 서비스는 자바 클래스, EJB 와 같은 두 가지 타입의 웹서비스 백앤드 구성 요소를 지원한다. JEUS 웹 서비스를 구현 하는 주요 절차는 다음과 같다. 각각의 세부 설명은 계속 이어지는 매뉴얼 본문에 언급된다.

1. 웹 서비스를 구성하는 백엔드 구성요소인 자바 코드를 작성한다.

2. SOAP 메시지의 내용을 직접 다루기를 원하거나, SOAP 메시지의 부착물(Attachment)에 직접 접근하기를 원한다면, 직접 SOAP 메시지 핸들러나 핸들러 체인(Chain)을 생성한다.

3. 자바 코드를 컴파일한다.

4.1 자바 클래스 웹 서비스의 구현

자바 클래스 웹서비스는 J2EE 에서 웹서비스를 생성하는 가장 간단한 방법이다. 먼저 간단한 예를 통해 구현하는 방법에 대해 알아보자.

4.1.1 간단한 예제

자바 클래스 웹서비스를 생성하기 위해서는 서비스 엔드포인트 인터페이스(SEI)와 구현 클래스를 정의해야 한다. 서비스 엔드포인트 인터페이스는 자바 클래스 웹서비스 엔드포인트가 자바 메쏘드의 형식으로 지원할 웹서비스 오퍼레이션들을 정의한다. 구현 클래스는 이러한 서비스 엔드포인트 인터페이스들을 구현한다.

다음의 엔드포인트 인터페이스는 echoString 과 echoString_double 이라는 웹서비스 오퍼레이션들을 정의하고 있다.

```
package jeustest.webservices.java2wsdl.doclit;

public interface Echo extends java.rmi.Remote {
```

```
      public String echoString(String arg11) throws
java.rmi.RemoteException;
      public String echoString_double(String arg1, String arg2)
throws java.rmi.RemoteException;
}
```

엔드포인트 인터페이스는 외부에 공개되어 접근 가능한 웹서비스 오퍼레이션들을 정의한다. 즉, SOAP 이라는 프로토콜을 사용하여 접근할 수 있는 오퍼레이션들을 정의한다. 엔드포인트 인터페이스는 java.rmi.Remote 인터페이스를 직접적으로나 간접적으로 확장해야하며, 정의된 메쏘드들은 java.rmi.RemoteException 타입을 익셉션으로 던져야 한다. 위와 같이 서비스 오퍼레이션들을 정의하고 나면, 이를 구현하는 로직이 필요하다. 그 역할을 하는 것이 구현 클래스이다. 다음이 그 예이다.

```
package jeustest.webservices.java2wsdl.doclit;

public class EchoImpl implements Echo {
      public String echoString(String input0) throws
java.rmi.RemoteException {
            return input0;
      }

      public String echoString_double(String input0, String
input1) throws java.rmi.RemoteException {
            return input0+input1;
      }
}
```

구현 클래스는 엔드포인트 인터페이스를 구현하고 있다. J2EE 서버내에서 인스턴스화 되어 실행되며, 런타임시에는 웹 서비스로 동작하게 될 것이다.

4.1.2 자바 클래스의 작성 원칙

Java 클래스 backend 로서 웹서비스를 구현할 때 아래의 규칙들을 준수해야 한다.

- Service Endpoint Interface(SEI) 를 정의한다.
- `public` 클래스를 정의한다.
- 파라미터 없는 디폴트 생성자를 정의한다.

- 웹서비스에서 public, non-static 으로 노출되는 Java 클래스의 메쏘드들을 정의한다.
- 쓰레드에 안전한(Thread-safety) Java 코드를 작성한다.
- 상호 운용성을 위해서 오버로딩 된 메쏘드들을 사용하지 말아야 한다

위와 같은 조건을 만족하는 경우, 다음과 같은 절차에 따라서 웹 서비스를 구현한다.

- 웹 서비스 메쏘드를 포함하고 있는 자바 클래스를 정의한다
- 서비스로 명시적으로 공개할 메쏘드를 위한 인터페이스를 정의한다.

4.2 EJB 웹서비스의 구현

EJB 는 J2EE 웹서비스를 개발하는 데 있어 이전에 언급 되었던 자바 클래스 웹서비스 프로그래밍 모델보다 좀 더 복잡하지만 더 풍부한 기능을 가질 수 있게 하는 프로그래밍 모델이다. 이러한 복잡성은 트랜잭션을 자동으로 관리하게 됨으로써 부수적으로 따라오는 것이라고 볼 수 있다.

EJB 웹서비스를 구현하기 위해서는 EJB 에 대한 이해를 어느 정도 가지고 있어야 한다. 만약 웹서비스를 구현하는 데 있어 EJB 를 굳이 사용하지 않아도 된다면, 이 장을 그냥 넘겨도 무방하다.

4.2.1 간단한 예제

무상태 세션 EJB 또한 웹서비스로 노출될 수 있다. 이 경우에도 서비스 엔드포인트 인터페이스가 필요하다.

다음의 엔드포인트 인터페이스(HelloIF.java)는 sayHello(String) 웹서비스 오퍼레이션을 정의하고 있다.

<< HelloIF.java >>

```
package helloejb;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloIF extends Remote {
```

```
      public String sayHello(String s) throws RemoteException;
}
```

엔드포인트 인터페이스는 외부에 공개되어 접근 가능한 웹서비스 오퍼레이션들을 정의한다. 즉, SOAP 이라는 프로토콜을 사용하여 접근할 수 있는 오퍼레이션들을 정의한다. 엔드포인트 인터페이스는 java.rmi.Remote 인터페이스를 직접적으로나 간접적으로 확장해야하며, 정의된 메쏘드들은 java.rmi.RemoteException 타입을 익셉션으로 던져야 한다. 위와 같이 서비스 오퍼레이션들을 정의하고 나면, 이를 구현하는 로직이 필요하다. 여기서는 EJB 가 그 역할을 담당한다.

다음은 리모트 인터페이스이다.

<< *Hello.java* >>

```
package helloejb;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Hello extends EJBObject
{
    String sayHello(String s) throws RemoteException;
}
```

다음은 홈 인터페이스이다.

<< *HelloHome.java* >>

```
package helloejb;

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface HelloHome extends EJBHome
{
    Hello create() throws RemoteException, CreateException;
}
```

다음은 세션빈을 구현한 EJB 엔드포인트 빈 클래스이다.

<< *HelloEJB* >>

```
package helloejb;

import java.rmi.RemoteException;

import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.lang.*;

public class HelloEJB implements SessionBean {

    public HelloEJB() {}
    public String sayHello(String s) throws RemoteException {
      try {
            Thread.currentThread().sleep(500);
      } catch (Exception ex) {
            throw new RemoteException("" + ex);
      }
      return "Hello World!" + s;
    }
    public void ejbCreate() {}
    public void ejbRemove() throws RemoteException {}
    public void setSessionContext(SessionContext sc) {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
}
```

4.2.2 EJB 웹서비스 작성 원칙

무상태 세션 EJB 웹서비스를 위한 코딩 작업은 다음과 같은 점을 제외하면 일반 EJB 코딩작업과 다른 점이 없다.

- 웹서비스 오퍼레이션이 one-way 로 설정되어 있으면 EJB 메쏘드의 자바 코드는 void 타입을 리턴해야 한다.

위와 같은 조건을 만족하도록 다음과 같은 절차에 따라 웹 서비스를 구현한다.

- 웹 서비스 메쏘드를 포함하고 있는 EJB 를 구현한다

- 서비스로 명시적으로 공개할 메쏘드를 위한 인터페이스를 정의한다.

4.3 WSDL 로부터 웹서비스 구현

본 웹서비스 안내서의 대부분은 웹서비스를 구성하는 백앤드(back-end) 구성 요소인 자바 코드를 작성하는 것으로부터 시작하는 것을 가정하고 있다. 그러나, 개발 환경에 따라서 WSDL 로부터 서비스 인터페이스를 생성할 수 있다. 이 경우 사용자는 wsdl2java Ant 태스크를 수행하여 서비스 인터페이스를 얻을 수 있다. Wsdl2java Ant 태스크는 사용자가 작성했거나 가지고 있는 WSDL 파일을 입력으로 한다. 다음은 Wsdl2java Ant 를 수행하기 위한 build 파일의 예이다.

<< *bukld.xml* >>

```
…
<property name="jaxrpc.wsdl"
      value="${work.home}/wsdl/DocLitEchoService.wsdl"/>
<property name="jaxrpc.mapping"
      value="${build.dir}/DocLitEchoService-mapping.xml"/>
…
<target name="wsdl2java" depends="init">
      <wsdl2java destDir="${classes.dir}"
                     verbose="true"
                     mode="import:server"
                     doCompile="true"
                     package="echo"
                     outputMapping="${jaxrpc.mapping}"
                     wsdl="${jaxrpc.wsdl}">
                     <classpath refid="build.classpath"/>
      </wsdl2java>
</target>
```

아래와 같은 명령어를 사용하여서 웹서비스 서비스 인터페이스 소스 코드를 생성할 수 있다.

```
prompt>ant wsdl2java
```

사용자는 생성된 서비스 인터페이스에 대한 서비스 구현체를 ‘4.1 자바 클래스 웹 서비스의 구현’과 ‘4.2 EJB 웹서비스의 구현’의 구현 방식에 따라서 생성한다.

wsdl2java 태스크에 대한 더 자세한 설명은 부록 A JEUS 웹 서비스 Ant Task 참조를 이용한다.

4.4 SAAJ(SOAP with Attachments API for Java)의 사용

통상적인 SOAP 메시지는 SOAP Body 안에 포함되지만, 특정한 자바 타입을 웹 서비스 오퍼레이션을 구현하는 메쏘드의 파라미터나, 리턴 타입으로 사용할 경우에는 SOAP 부착물(Attachment) 형태로 전송된다. JEUS 웹 서비스는 자바에서 MIME 타입으로의 형 전환을 다음과 같이 정의한다.

표 1 Required Mappings : Java to MIME

Java Type	MIME Type
`java.awt.Image`	image/gif or image/jpeg
`java.lang.String`	text/plain
`javax.mail.internet.MimeMultipart`	multipart/*
`javax.xml.transform.Source`	text/xml or application/xml

위에 열거된 각각의 MIME 타입에 대해 JAX-RPC 엔드포인트 스텁이 특정한 자바 타입을 적절히 인코딩 된 데이터의 스트림으로 변환, 역변환 작업을 한다. 보다 자세한 설명은 9 장 - SOAP 메시지 핸들러의 생성 - 을 참조하라.

5 자바 클래스 웹 서비스의 생성과 배치

JEUS 웹 서비스 생성은 개발 편의를 제공하기 위해 커맨드 라인 툴과 Apache Ant 툴을 사용한다(부록 A JEUS 웹 서비스 Ant Task 참조, B JEUS 웹 서비스 커맨드 라인 툴 참조).

웹 서비스 생성과 배치 작업은 다음의 작업들이 순차적으로 진행된다.

- 서비스 설정 파일의 작성(service-config.xml)
- J2EE 웹서비스를 위한 WSDL 과 JAX-RPC 매핑 파일의 생성
- 웹서비스 배치 서술자 작성(webservices.xml, jeus-webservices-dd.xml)
- 생성된 웹서비스 모듈의 패키징과 배치

5.1 서비스 설정 파일의 작성

JEUS 웹 서비스는 service-config.xml 과 같이 웹 서비스 생성을 위한 설정을 xml 파일에 저장한다.

다음은 설정 파일의 한 예이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
   <service>
      <service-name>DocLitEchoService</service-name>
      <target-namespace>urn:DocLitService</target-namespace>
      <output-wsdl-file>DocLitEchoService.wsdl</output-wsdl-file>
      <output-jaxrpc-mapping-file>DocLitEchoService-mapping.xml
      </output-jaxrpc-mapping-file>
      <style>wrapped</style>
      <interface>
         <endpoint-interface-class>
            jeustest.webservices.java2wsdl.doclit.Echo
```

```
        </endpoint-interface-class>
      </interface>
</web-services-config>
```

위 예에서는 <service-name>에 설정된 DocLitEchoService 라는 이름의 웹 서비스를 생성하게 되며, 생성되는 WSDL 과 매핑 파일은 각각 DocLitEchoService.wsdl 과 DocLitEchoService-mapping.xml 이된다. 서비스 스타일은 문서 방식 중, 랩트(WRAPPED)방식이며, 웹서비스 엔드 포인트로는 jeustest.webservices.java2wsdl.doclit.Echo 라는 자바 클래스 파일로 설정 되었다.

보다 자세한 설명은 부록 C 를 참조한다.

5.2 WSDL 파일과 JAX-RPC 매핑 파일의 생성

JEUS 웹 서비스는 웹 서비스의 생성을 위해 커맨드 라인 툴 방식과 ANT Task 방식을 제공한다. 웹 서비스 생성을 위해 두 가지 방법 중 어느 것을 택해도 무방하며, 사용자 편의에 따라 결정한다.

5.2.1 커맨드 라인 툴의 이용

JEUS 웹 서비스는 웹 서비스의 생성을 위한 커맨드 라인 툴을 제공한다. 사용법은 다음과 같다.

```
Usage: java2wsdl [options] configuration_file
where [options] include:
-classpath <path>           specify where to find input class
files
-cp <path>                  same as -classpath <path>
-d <directory>              specify where to place generated
output files
-verbose                    [optional] turn verbose mode on
```

5.1 에서 생성한 service-config.xml 파일을 가지고 웹 서비스를 생성하려면 커맨드 라인에서 다음과 같이 입력하여 명령어를 수행한다. 다음 예는 컴파일된 자바 클래스 백-엔드 파일들이 `D:\sample\classes` 에 존재하는 경우이다.

```
D:\sample\>java2wsdl -cp .\classes service-config.xml
```

위와 같이 명령을 수행하면, JAX-RPC 매핑 파일인 DocLitEchoService-mapping.xml 과 WSDL 파일인 DocLitEchoService.wsdl 이 생성 된다.

5.2.2 Ant 툴의 이용

JEUS 웹 서비스는 웹 서비스의 생성을 위한 ant task 인 'java2wsdl'를 제공한다. 'java2wsdl' ant task 는 입력으로 서비스 설정 파일의 위치를 받아서, WSDL 파일과 JAX-RPC 매핑 파일을 생성한다. 'java2wsdl' task 에 관한 더 많은 정보는 부록 A JEUS 웹 서비스 Ant Task 참조를 참조한다.

먼저 자바 클래스 파일들을 다음과 같이 컴파일 한다.

```
ant compile
```

WSDL 파일과 JAX-RPC 매핑 파일은 아래의 명령어를 수행하면 `./build` 디렉토리에 자동 생성된다.

```
ant java2wsdl
```

`java2wsdl` 앤트 태스크(ant task)는 다음과 같이 구성되어 있다.

```
<target name="java2wsdl" depends="compile">
      <java2wsdl destDir="${build.dir}"
                    verbose="true"
                    configfilepath="./service-config.xml">
                    <classpath refid="build.classpath"/>
    </java2wsdl>
  </target>
```

서비스 설정 파일 경로를 <java2wsdl> 엘리먼트의 configfilepath 속성에 넣어주고, 컴파일 된 자바 클래스 파일들의 경로를 <classpath> 엘리먼트의 refid 속성에 넣어준다.

다음은 java2wsdl 을 사용하는 build.xml 의 예제이다.

```
<?xml version="1.0"?>
<project name="Echo" default="core" basedir=".">
   <property name="jeus.home" value="c:/jeus"/>
   <path id="jeus.classpath">
      <pathelement location="${jeus.home}/classes"/>
      <pathelement location="${jeus.home}/lib/system/jeus.jar"/>
```

```
      <pathelement location=
           "${jeus.home}/lib/system/jxml-impl.jar"/>
      <pathelement location=
           "${jeus.home}/lib/system/jaxb-api.jar"/>
      <pathelement location=
          "${jeus.home}/lib/system/jaxb-impl.jar"/>
      <pathelement location=
          "${jeus.home}/lib/system/jaxb-libs.jar"/>
      <pathelement location=
          "${jeus.home}/lib/system/relaxngDatatype.jar"/>
      <pathelement location=
          "${jeus.home}/lib/system/xsdlib.jar"/>
      <pathelement location="${jeus.home}/lib/system/mail.jar"/>
      <pathelement location=
          "${jeus.home}/lib/system/activation.jar"/>
      <pathelement location="${java.home}/lib/tools.jar"/>
   </path>

   <taskdef name="java2wsdl"
      classname="jeus.util.ant.webservices.Java2WsdlTask">
   </taskdef>

<!-- path to root -->
<property name="root.dir" value="../../.."/>
<property name="work.home" value="."/>

<!-- The destination directory for the build -->
<property name="build.dir" value="${work.home}/build"/>
<property name="classes.dir" value="${build.dir}/classes"/>

<property name="src" value="${work.home}/src"/>
<property name="build.wardir" value="${build.dir}/war"/>
<property name="build.jardir" value="${build.dir}/jar"/>
<property name="build.eardir" value="${build.dir}/ear"/>
<property name="warfile"
        value="${build.dir}/DocLitEchoService.war"/>
<property name="earfile" value="${build.dir}/EchoDocLit.ear"/>

<property name="jaxrpc.wsdl"
```

```
     value="${build.dir}/DocLitEchoService.wsdl"/>
<property name="jaxrpc.mapping"
   value="${build.dir}/DocLitEchoService-mapping.xml"/>

<property name="j2ee-webservice-dd"
   value="./conf/webservices.xml"/>
<property name="jeus-webservice-dd"
   value="./conf/jeus-webservices-dd.xml"/>
<property name="web-xml" value="./conf/web.xml"/>
<property name="application-xml"
     value="./conf/application.xml"/>

<path id="build.classpath">
  <path refid="jeus.classpath"/>
  <pathelement location="${classes.dir}"/>
</path>

<target name="compile">
  <mkdir dir="${classes.dir}"/>
  <javac srcdir="${src}"
         debug="${javac.debug}"
         destdir="${classes.dir}"
       includes="**">
    <classpath refid="build.classpath"/>
  </javac>
</target>

<target name="java2wsdl" depends="compile">
    <java2wsdl destDir="${build.dir}"
              verbose="true"
              configfilepath="./service-config.xml">
        <classpath refid="build.classpath"/>
  </java2wsdl>
</target>

<target name="wswar" depends="java2wsdl">
  <delete dir="${build.wardir}" quiet="true"/>
```

```
    <mkdir dir="${build.wardir}"/>
    <mkdir dir="${build.wardir}/WEB-INF"/>
    <mkdir dir="${build.wardir}/WEB-INF/wsdl"/>
    <copy todir="${build.wardir}/WEB-INF/classes">
        <fileset dir="${classes.dir}">
           <exclude name="**/*.wsdl"/>
        </fileset>
    </copy>
    <copy file="${jaxrpc.wsdl}"
       todir="${build.wardir}/WEB-INF/wsdl" />
    <copy file="${jaxrpc.mapping}"
       todir="${build.wardir}/WEB-INF" />
    <copy file="${j2ee-webservice-dd}"
       todir="${build.wardir}/WEB-INF" />
    <copy file="${jeus-webservice-dd}"
       todir="${build.wardir}/WEB-INF" />
    <copy file="${web-xml}"  todir="${build.wardir}/WEB-INF" />
    <jar jarfile="${warfile}" basedir="${build.wardir}"/>
 </target>

 <target name="wsear" depends="wswar">
    <mkdir dir="${build.eardir}"/>
    <mkdir dir="${build.eardir}/META-INF"/>

    <copy file="${application-xml}"
        todir="${build.eardir}/META-INF" />
    <copy file="${warfile}"  todir="${build.eardir}/" />

    <jar jarfile="${earfile}" basedir="${build.eardir}" />
 </target>
 <target name="core" depends="compile" />
 <target name="all" depends="core" />
</project>
```

5.3 웹서비스 배치 서술자의 작성

웹서비스 DD(배치 서술자 : Deployment Descriptor)파일은 웹서비스의 배치에 관해 기술한 XML 파일이다. 웹서비스 DD 파일은 웹서비스의 배치에 관

한 정보와 어떻게 웹서비스 백-엔드(back-end)를 발견할 것인가에 대한 정보를 웹 서비스 엔진에게 제공한다.

웹 서비스 DD 파일은 J2EE 웹 서비스 스펙에 규정된 DD 파일인 `webservices.xml` 과 JEUS 웹 서비스를 위한 DD 파일인 `jeus-webservices-dd.xml` 이 있다.

5.3.1 J2EE 웹 서비스 DD(webservices.xml)의 작성

J2EE 웹 서비스 DD 파일명은 반드시 `webservices.xml` 로 해야 한다. 다음은 한 예이다.

<< *webservices.xml* >>

```
<?xml version="1.0"?>
<webservices version="1.1"
xmlns="http://java.sun.com/xml/ns/j2ee">
   <webservice-description>
     <webservice-description-name>DocLitEchoService
     </webservice-description-name>
     <wsdl-file>WEB-INF/wsdl/DocLitEchoService.wsdl</wsdl-file>
     <jaxrpc-mapping-file>WEB-INF/DocLitEchoService-mapping.xml
     </jaxrpc-mapping-file>
     <port-component>
        <port-component-name>EchoPort</port-component-name>
        <wsdl-port xmlns:ns2="urn:DocLitService">
        ns2:EchoPort
        </wsdl-port>
        <service-endpoint-interface>
        jeustest.webservices.java2wsdl.doclit.Echo
        </service-endpoint-interface>
        <service-impl-bean>
           <servlet-link>EchoServlet</servlet-link>
        </service-impl-bean>
       </port-component>
    </webservice-description>
</webservices>
```

DocLitEchoService 라는 서비스 이름을 가지고, WSDL 파일의 위치는 WAR 파일 내에서 WEB-INF/wsdl 디렉토리 내에 있으며, 그 이름은

DocLitEchoService.wsdl 이다. JAX-RPC 매핑 파일의 위치는 WEB-INF 디렉토리이며, DocLitEchoService-mapping.xml 이라는 이름을 가지고 있다. 이 서비스에 접근 하려면 WSDL 에 표기된 포트 중 EchoPort 라는 이름을 가지고 있는 포트를 사용하고, 이 포트에 대한 SEI(Service Endpoint Interface)와 서블릿이 정의된다.

보다 자세한 설명은 11 장 “웹 서비스 DD(webservices.xml)의 작성”을 참조한다.

5.3.2 JEUS 웹 서비스 DD(jeus-webservices-dd.xml)의 작성

JEUS 웹 서비스 DD 파일명은 jeus-webservices-dd.xml 로 해야 한다. 다음은 한 예이다.

<< *jeus-webservices-dd.xml* >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <service>
        <webservice-description-name>DocLitEchoService
        </webservice-description-name>
        <port>
            <port-component-name>EchoPort</port-component-name>
        </port>
    </service>
</jeus-webservices-dd>
```

보다 자세한 설명은 부록을 참조한다.

5.4 웹서비스의 배치

자바 클래스 웹서비스는 자바 클래스와 웹서비스 DD 파일을 웹 모듈처럼 패키징한다. JEUS 웹 컨테이너의 컨텍스트와 웹 어플리케이션에 대한 보다 자세한 설명은 JEUS Web Container 안내서의 'Context(웹 어플리케이션)'장을 참조한다.

웹 서비스는 웹 어플리케이션 WAR 파일과 EJB JAR 파일을 포함하는, 표준 엔터프라이즈 어플리케이션(Enterprise Application) EAR 파일로 묶이게 된다.

5.4.1 서블릿 DD 및 JEUS 웹 모듈 DD 의 작성

5.4.1.1 서블릿 DD(web.xml)의 작성

서블릿 DD(web.xml)의 통상적인 역할은 서블릿과 JSP 콤포넌트의 런타임 속성을 기술하는 것이다. 자바 클래스 웹 서비스는 런 타임에 서블릿에 임베드된 형태로 연동 되므로 web.xml 파일은 자바 클래스 웹서비스를 배치하는데 필요하다. 다음은 web.xml 파일의 한 예이다.

<< *web.xml* >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>EchoServlet</servlet-name>
    <servlet-class>jeustest.webservices.java2wsdl.doclit.EchoImpl
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
     <servlet-name>EchoServlet</servlet-name>
     <url-pattern>/DocLitEchoService</url-pattern>
  </servlet-mapping>
</web-app>
```

<servlet-class> 엘리먼트는 실제 서비스를 구현한 로직이 들어있는 자바 클래스의 풀 패키지 이름을 포함한 클래스 이름을 텍스트 노드의 값으로 가진다. <servlet-mapping> 엘리먼트의 하위 엘리먼트인 <url-pattern> 엘리먼트는 실제로 서비스에 접근하기 위한 URL 을 값으로 필요로 한다.

5.4.1.2 JEUS 웹 모듈 DD 의 작성

JEUS 웹 모듈 DD 는 배치하려는 웹 모듈의 컨텍스트를 정의한다. 파일 이름은 `jeus-web-dd.xml` 과 같다. 다음은 DocLitEchoService 를 컨텍스트로 가지는 서비스의 JEUS 웹 모듈 DD 이다.

<< *jeus-web-dd.xml* >>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
        <context-path>/DocLitEchoService</context-path>
        <docbase>DocLitEchoService</docbase>
```

```
        <enable-jsp>true</enable-jsp>
        <auto-reload>
            <enable-reload>false</enable-reload>
            <check-on-demand>false</check-on-demand>
        </auto-reload>
        <max-instance-pool-size>-1</max-instance-pool-size>
        <url-rewriting>false</url-rewriting>
        <enable-default-login>false</enable-default-login>
</jeus-web-dd>
```

5.4.2 WAR 패키징

WAR 파일은 확장명이 .war 로 끝나는 일종의 JAR 압축 파일이며, 압축과 압축 해제 알고리즘인 zlib 알고리즘 표준을 따라 압축되는 파일이다. WAR 파일은 서블릿, JSP 와 같은 웹 컴포넌트들을 묶는 용도로만 쓰이며, 특정한 디렉토리 구조를 가지고 있어야 한다(그림 2 WAR Packaging Structure).

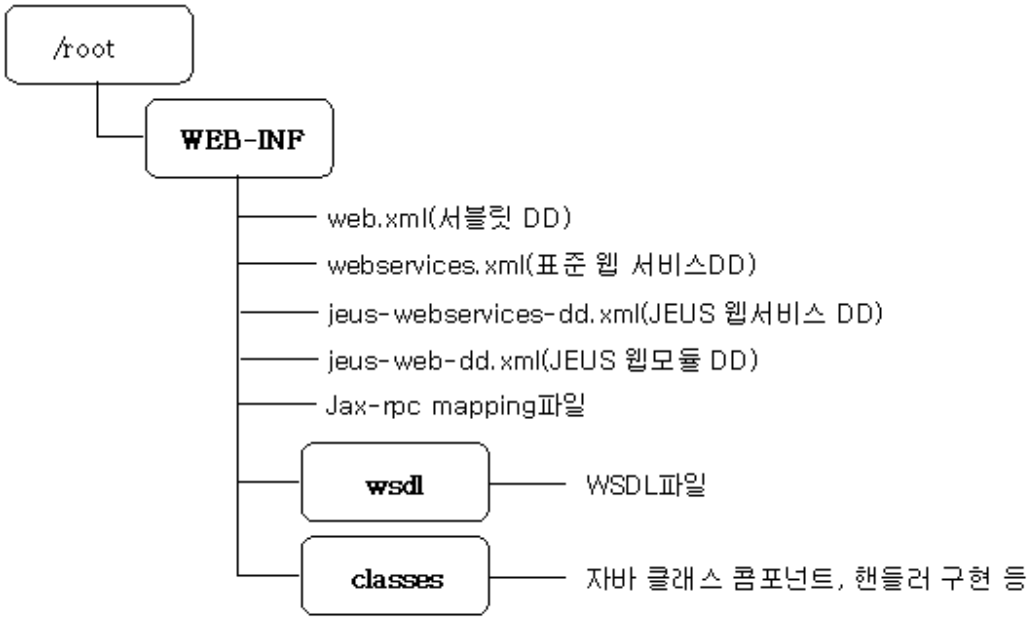


그림 2 WAR Packaging Structure

앞에서 생성한 WSDL 파일과, JAX-RPC 매핑 파일, 표준 웹 서비스 DD(webserivices.xml), JEUS 웹 서비스 DD(jeus-webservices-dd.xml), 서블릿 DD(web.xml), JEUS 웹 모듈 DD 을 WAR 파일 형태로 묶는다.

JAX-RPC 매핑파일과 WSDL webservices.xml 파일에 <wsdl-file>과 <jax-rpc-mappng-file>로 기술한 위치에 존재한다면 반드시 위의 구조를 따르지 않아도 무방하다.

5.4.3 EAR 패키징과 배치

5.4.3.1 EAR 패키징

J2EE 응용 프로그램은 웹 컴포넌트나 EJB, J2EE 커넥터를 사용할 수 있는 하나의 독립적인 비즈니스 솔루션이며, EAR(Enterprise ARchive)파일로 패키징 될 수 있다. EAR 파일은 응용 프로그램의 XML DD 을 가지고 있으며, J2EE 컴포넌트와 커넥터등을 EJB JAR 나 WAR, 혹은 RAR 파일 형태로 패키징하여 포함하고 있다.

자바 클래스 웹서비스는 서블릿 프로그래밍 모델 위에서 생성 되었으므로 WAR 파일 형태로 패키징 되고, EJB 엔드포인트는 SOAP 메시지를 다루는 곳이 EJB 이므로 EJB JAR 파일 형태로 패키징 된다.

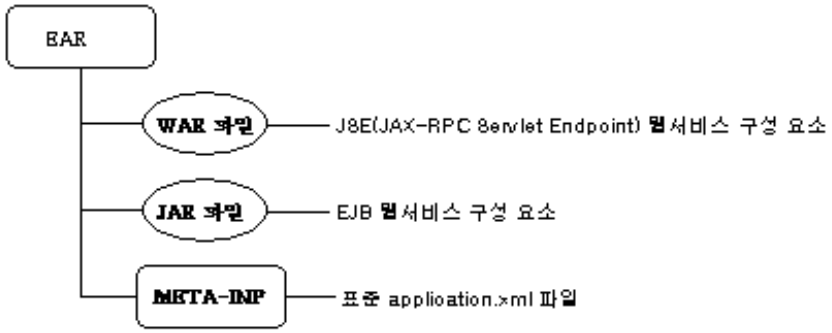


그림 3 EAR Packaging Structure

META-INF 디렉토리 안의 표준 application.xml 파일에서는 EAR 패키징한 J2EE 컴포넌트들을 기술한다. 다음은 웹 모듈을 J2EE 구성요소로 가지고 있는 EAR 응용프로그램의 application.xml 파일의 한 예이다.

<< *application.xml* >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns1:application version="1.4"
xmlns:ns1="http://java.sun.com/xml/ns/j2ee">
    <ns1:module>
```

```
        <ns1:web>
            <ns1:web-uri>DocLitEchoService.war</ns1:web-uri>
            <ns1:context-root>DocLitEchoService</ns1:context-
root>
        </ns1:web>
    </ns1:module>
</ns1:application>
```

<web> 엘리먼트에서는 웹 모듈 패키징의 이름이 무엇인지, 모듈의 컨텍스트의 정보가 무엇인지에 대한 정보를 가지고 있다.

5.4.3.2 웹 서비스의 배치(EAR 응용프로그램의 배치)

웹 서비스의 배치 작업은 일반적인 J2EE 응용 프로그램의 배치 작업과 동일하다. EAR 배치작업을 수행함으로써 웹 서비스는 인터넷에서 접근 가능한 서비스로 공개 된다. 배치된 서비스의 실제 접근 가능한 URL 주소는

`http://host:port/DocLitEchoService/DocLitEchoService`

이다.

6 EJB 웹 서비스의 생성과 배치

EJB 를 이용한 웹 서비스의 구현 작업을 마쳤다면, 실제로 EJB 의 비즈니스 로직을 웹서비스로 전환하고 배치하는 작업이 필요하다. EJB 웹 서비스 생성과 배치 작업의 전체 적인 흐름은 자바 클래스 웹서비스 생성, 배치 작업과 같다. 한편, JEUS 웹 서비스 생성은 개발 편의를 제공하기 위해 커맨드 라인 툴과 Apache Ant 툴을 사용한다(부록 A JEUS 웹 서비스 Ant Task 참조, B JEUS 웹 서비스 커맨드 라인 툴 참조).

전체 적인 작업 흐름은 다음과 같다.

- 서비스 설정 파일의 작성(service-config.xml)
- J2EE 웹서비스를 위한 WSDL 과 JAX-RPC 매핑 파일의 생성
- 웹서비스 배치 서술자 작성(webservices.xml, jeus-webservices-dd.xml, ejb-jar.xml)
- 생성된 웹서비스 모듈의 패키징과 배치

6.1 서비스 설정 파일의 작성

다음은 설정 파일의 한 예이며, 파일 이름은 service-config.xml 이다.

<< *ejb-service-config.xml* >>

```
<?xml version="1.0"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
      <service>
            <service-name>AddressBookService</service-name>
            <target-namespace>urn:AddressBookService</target-
namespace>
            <style>wrapped</style>
            <use>literal</use>
            <interface>
```

```
                <endpoint-interface-
class>address.AddressBookIF</endpoint-interface-class>

            </interface>
      </service>
</web-services-config>
```

위 예에서는 <service-name>에 설정된 AddressBookService 라는 이름의 웹 서비스를 생성하게 되며, 생성되는 WSDL 과 매핑 파일은 각각 AddressBookService.wsdl 과 AddressBookService-mapping.xml 이 된다. 서비스 스타일은 문서 방식 중, 랩트(WRAPPED) 방식이며, 웹서비스 엔드 포인트로는 address.AddressBookIF 라는 자바 클래스 파일로 설정 되었다.

보다 자세한 작성법은 부록 C 를 참조한다.

6.2 WSDL 파일과 JAX-RPC 매핑 파일의 생성

JEUS 웹 서비스는 웹 서비스의 생성을 위해 커맨드 라인 툴 방식과 ANT Task 방식을 제공한다. 웹 서비스 생성을 위해 두 가지 방법 중 어느 것을 택해도 무방하며, 사용자 편의에 따라 결정한다.

6.2.1 커맨드 라인 툴의 이용

JEUS 웹 서비스는 웹 서비스의 생성을 위한 커맨드 라인 툴을 제공한다. 사용법은 다음과 같다.

```
Usage: java2wsdl [options] configuration_file

where [options] include:
-classpath <path>   specify where to find input class files
-cp <path>          same as -classpath <path>
-d <directory>      specify where to place generated output files
-level <log-level>  specify a log level
-verbose            [optional] turn verbose mode on
```

이전 장에서 생성한 service-config.xml 파일을 가지고 웹 서비스를 생성하려면 커맨드 라인에서 다음과 같이 입력하여 명령어를 수행한다. 다음 예는 컴파일 된 자바 클래스 백-엔드 파일들은 `D:\sample\classes` 에 존재하는 경우이다.

```
D:\sample\>java2wsdl -cp .\classes ejb-service-config.xml
```

위와 같이 명령을 수행하면, JAX-RPC 매핑 파일인 AddressBookService-mapping.xml 과 WSDL 파일인 AddressBookService.wsdl 이 생성 된다.

6.2.2 Ant 툴의 이용

JEUS 웹 서비스는 웹 서비스의 생성을 위한 ant task 인 'java2wsdl'를 제공한다. 'java2wsdl' ant task 는 입력으로 서비스 설정 파일의 위치를 받아서, WSDL 파일과 JAX-RPC 매핑 파일을 생성한다. 'java2wsdl' task 에 관한 더 많은 정보는 부록 A JEUS 웹 서비스 Ant Task 참조를 참조한다.

먼저 자바 클래스 파일들을 다음과 같이 컴파일 한다.

```
ant compile
```

WSDL 파일과 JAX-RPC 매핑 파일은 아래의 명령어를 수행하면 `./build` 디렉토리에 자동 생성된다.

```
ant java2wsdl
```

`java2wsdl` 앤트 태스크(ant task)는 다음과 같이 구성되어 있다.

```
<target name="java2wsdl" depends="compile">
      <java2wsdl destDir="${build.dir}"
                     verbose="true"
                     configfilepath="./ service-config.xml">
                     <classpath refid="build.classpath"/>
      </java2wsdl>
</target>
```

서비스 설정 파일 경로를 <java2wsdl> 엘리먼트의 configfilepath 속성에 넣어주고, 컴파일 된 자바 클래스 파일들의 경로를 <classpath> 엘리먼트의 refid 속성에 넣어준다.

위와 같은 명령을 수행하고 나면 WSDL 파일과 JAX-RPC 매핑 파일이 생성되게 된다.

6.3 웹서비스 배치 서술자의 작성

웹서비스 DD(배치 서술자; Deployment Descriptor)파일은 웹서비스의 배치에 관해 기술한 XML 파일이다. 웹서비스 DD 파일은 웹서비스의 배치에 관한 정보와 어떻게 웹서비스 백-엔드(back-end)를 발견할 것인가에 대한 정보를 웹 서비스 엔진에게 제공한다.

웹 서비스 DD 파일은 J2EE 웹 서비스 스펙에 규정된 DD 파일인 `webservices.xml` 과 JEUS 웹 서비스를 위한 DD 파일인 `jeus-webservices-dd.xml` 이 있다.

6.3.1 J2EE 웹 서비스 DD(webservices.xml)의 작성

J2EE 웹 서비스 DD 파일명은 반드시 `webservices.xml` 로 해야 한다. 다음은 한 예이다.

<< *webservices.xml* >>

```
<?xml version="1.0"?>
<webservices version="1.1"
xmlns="http://java.sun.com/xml/ns/j2ee">
    <webservice-description>
        <webservice-description-name>
            AddressBookService
        </webservice-description-name>
        <wsdl-file>
            META-INF/wsdl/AddressBookService.wsdl
        </wsdl-file>
        <jaxrpc-mapping-file>
            META-INF/AddressBookService-mapping.xml
        </jaxrpc-mapping-file>
        <port-component>
            <port-component-name>
               AddressBookIFPort
            </port-component-name>
            <wsdl-port xmlns:ns2="urn:AddressBookService">
               ns2:AddressBookIFPort
            </wsdl-port>
            <service-endpoint-interface>
               address.AddressBookIF
            </service-endpoint-interface>
```

```
            <service-impl-bean>
                <ejb-link>AddressEJB</ejb-link>
            </service-impl-bean>
        </port-component>
    </webservice-description>
</webservices>
```

AddressBookService 라는 서비스 이름을 가지고, WSDL 파일의 위치는 JAR 파일 내에서 META-INF/wsdl 디렉토리 내에 있으며, 그 이름은 AddressBookService.wsdl 이다. JAX-RPC 매핑 파일의 위치는 META-INF 디렉토리이며, AddressBookService-mapping.xml 이라는 이름을 가지고 있다. 이 서비스에 접근 하려면 WSDL 에 표기된 포트 중 AddressBookIFPort 라는 이름을 가지고 있는 포트를 사용하고, 이 포트에 대한 SEI(Service Endpoint Interface)와 EJB 가 정의된다.

6.3.2 JEUS 웹 서비스 DD(jeus-webservices-dd.xml)의 작성

JEUS 웹 서비스 DD 파일명은 jeus-webservices-dd.xml 로 해야 한다. 다음은 한 예이다.

<< *jeus-webservices-dd.xml* >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <ejb-context-path>webservice</ejb-context-path>
    <service>
        <webservice-description-name>
            AddressBookService
        </webservice-description-name>
        <port>
            <port-component-name>
                AddressBookIFPort
            </port-component-name>
            <ejb-endpoint-url>
                /AddressBookService
            </ejb-endpoint-url>
        </port>
    </service>
</jeus-webservices-dd>
```

JEUS 웹서비스 DD 파일에는 EJB 웹서비스로 접근하기 위한 URL 접근경로를 기술한다. URL 접근경로는 컨텍스트 경로와 URL 패턴을 포함한다. 컨텍스트 경로는 <ejb-context-path>로 지정한다. URL 패턴은 <port>아래의 <ejb-endpoint-url> 로 지정한다.

위의 경우 웹서비스에 접근하기 위한 실제 URL 주소는

`http://host:port/webservice/AddressBookService`

이다.

6.4 웹서비스의 배치

EJB 웹서비스는 EJB JAR 패키징 안에 웹서비스 DD 파일과 WSDL 을 같이 포함 한다. 웹 서비스는 웹 어플리케이션 WAR 파일과 EJB JAR 파일을 포함하는, 표준 엔터프라이즈 어플리케이션(Enterprise Application) EAR 파일로 묶이게 된다.

6.4.1 EJB DD 의 작성

다음은 ejb-jar.xml 파일의 한 예이다.

<< *ejb-jar.xml* >>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
          http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
version="2.1">
<display-name>AddressEJB</display-name>
   <enterprise-beans>
      <session>
         <display-name>AddressEJB</display-name>
         <ejb-name>AddressEJB</ejb-name>
         <service-endpoint>
              address.AddressBookIF
         </service-endpoint>
         <ejb-class>address.AddressBookEJB</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
```

```
      </session>
   </enterprise-beans>
</ejb-jar>
```

<ejb-class> 엘리먼트는 실제 서비스를 구현한 로직이 들어있는 빈 클래스의 풀 패키지 이름을 포함한 클래스 이름을 텍스트 노드의 값으로 가진다. <service-endpoint> 엘리먼트는 EJB 웹서비스의 서비스 엔드포인트 인터페이스 클래스의 풀 패키지 이름을 포함한 클래스 이름을 텍스트 노드의 값으로 가진다.

다음은 JEUS 에 필요한 또 다른 EJB DD 이다. 파일 이름은 `jeus-ejb-dd.xml` 이다.

<< *jeus-ejb-dd.xml* >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <beanlist>
        <jeus-bean>
            <ejb-name>AddressEJB</ejb-name>
            <export-name>AddressEJB</export-name>
        </jeus-bean>
    </beanlist>
</jeus-ejb-dd>
```

6.4.2 JAR 패키징

이제 까지 작성한 EJB 구현 클래스들과 SEI, 그리고 웹서비스 DD, EJB DD 를 하나의 JAR 파일로 패키징을 한다. 구조는 다음과 같다.

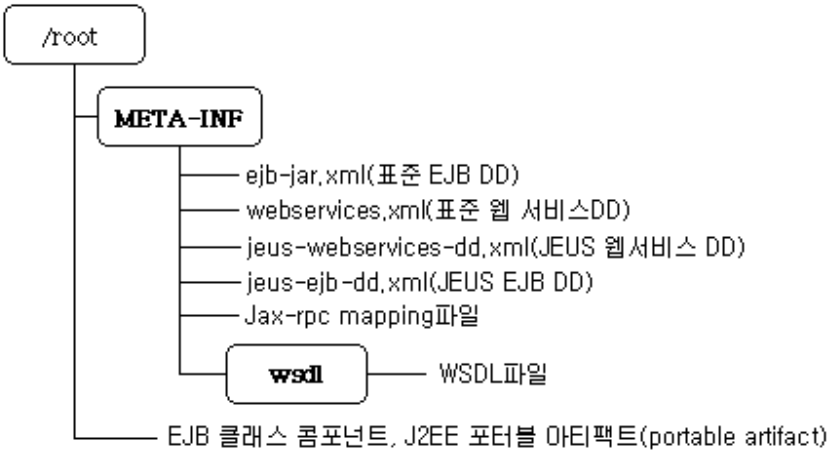


그림 4 EJB Web Service JAR Packaging Structure

JAX-RPC 매핑파일과 WSDL 파일이 webservices.xml 파일의 <wsdl-file>과 <jax-rpc-mapping-file>로 기술한 위치에 존재한다면 반드시 위의 구조를 따르지 않아도 무방하다.

6.4.3 EAR 패키징과 배치

6.4.3.1 EAR 패키징

J2EE 응용 프로그램은 웹 컴포넌트나 EJB, J2EE 커넥터를 사용할 수 있는 하나의 독립적인 비즈니스 솔루션이며, EAR(Enterprise ARchive)파일로 패키징 될 수 있다. EAR 파일은 응용 프로그램의 XML DD 을 가지고 있으며, J2EE 컴포넌트와 커넥터등을 EJB JAR 나 WAR, 혹은 RAR 파일 형태로 패키징하여 포함하고 있다.

JSE(JAX-RPC Service Endpoint)는 서블릿 프로그래밍 모델 위에서 생성 되었으므로 WAR 파일 형태로 패키징 되고, EJB 엔드포인트는 SOAP 메시지를 다루는 곳이 EJB 이므로 EJB JAR 파일 형태로 패키징 된다.

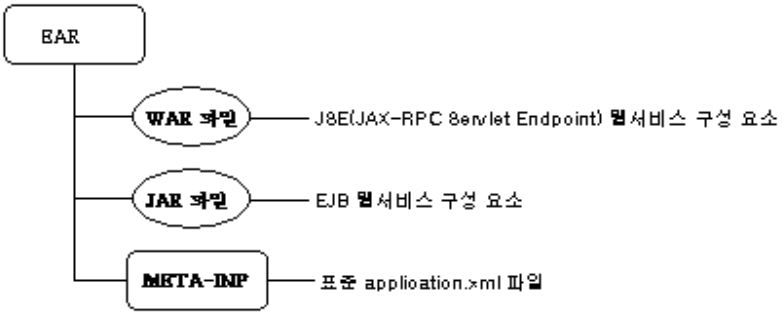


그림 5 EAR Packaging Structure

META-INF 디렉토리 안의 표준 application.xml 파일에서는 EAR 패키징한 J2EE 컴포넌트들을 기술한다. 다음은 웹 모듈을 J2EE 구성요소로 가지고 있는 EAR 응용프로그램의 application.xml 파일의 한 예이다.

<< *application.xml* >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns1:application version="1.4"
xmlns:ns1="http://java.sun.com/xml/ns/j2ee">
    <ns1:module>
        <ns1:ejb>AddressBook.jar</ns1:ejb>
    </ns1:module>
</ns1:application>
```

<ejb> 엘리먼트에서는 EJB 웹서비스의 JAR 패키징의 이름이 무엇인지에 대한 정보를 가지고 있다.

6.4.3.2 웹 서비스의 배치(EAR 응용프로그램의 배치)

웹 서비스의 배치 작업은 일반적인 J2EE 응용 프로그램의 배치 작업과 동일하다. EAR 배치작업을 수행함으로써 웹 서비스는 인터넷에서 접근 가능한 서비스로 공개 된다.

7 웹서비스의 호출

7.1 웹서비스의 호출

'웹서비스의 호출'은 웹서비스를 사용하기 위한 클라이언트 응용 프로그램이 수행하는 동작을 의미한다. 클라이언트 응용프로그램은 자바, 또는 .NET 과 같은 여러 다양한 기술을 이용하여 JEUS 에 배치된 웹서비스를 호출할 수 있다.

7.2 클라이언트 응용프로그램에서의 호출

웹서비스 클라이언트는 웹서비스에게 서비스를 요청하는 프로그램이다. JEUS 웹서비스를 사용하여 두 가지 타입의 웹서비스 클라이언트를 만들 수 있다.

- **Stub Client**: 이 타입의 웹서비스 클라이언트는 웹서비스의 WSDL 로부터 생성된 stub 을 이용한다.
- **DII (Dynamic Invocation Interface) 클라이언트**: 이 타입의 웹서비스 클라이언트는 JAX-RPC Client API 를 이용한다.

7.2.1 Stub 클라이언트의 생성

7.2.1.1 소개

Stub 클라이언트는 특정 웹서비스의 WSDL 로부터 생성된 로컬 Stub 상의 메쏘드를 호출한다. Stub 객체는 원격의 웹서비스와 상호작용을 담당한다.

이번 절에서는 5 장에서 이미 작성했던 'DocLitEchoService' 웹서비스의 메쏘드를 호출하는 웹서비스 클라이언트 프로그램을 작성하는 방법을 설명한다.

7.2.1.2 WSDL 로부터 웹서비스 Stub 생성

웹서비스 Stub 소스 코드를 생성하기 위해서 wsdl2java ant task 나 wsdl2java 커맨드를 사용한다. Ant Task 를 사용하는 경우 . build.xml 의 코드는 아래와 같다.

<< build.xml >>

```
…
<property name="jaxrpc.wsdl"
      value="${work.home}/wsdl/DocLitEchoService.wsdl"/>
<property name="jaxrpc.mapping"
      value="${build.dir}/DocLitEchoService-mapping.xml"/>
…
<target name="wsdl2java" depends="init">
      <wsdl2java destDir="${classes.dir}"
                     verbose="true"
                     mode="gen:client"
                     doCompile="true"
                     package="echo"
                     outputMapping="${jaxrpc.mapping}"
                     wsdl="${jaxrpc.wsdl}">
                     <classpath refid="build.classpath"/>
      </wsdl2java>
</target>
```

wsdl2java 태스크에 대한 더 자세한 설명은 부록 A JEUS 웹 서비스 Ant Task 참조를 참조하라.

아래와 같은 명령어를 사용하여 웹서비스 stub 소스 코드를 생성할 수 있다.

```
C:\wsclient>ant wsdl2java
```

만약 위의 과정이 성공했다면 웹서비스 stub 소스 코드가 생성된다.

```
C:\wsclient\build\classes\echo\
                              |
                              +-- DocLitEchoService.java
                              |
                              +-- DocLitEchoService_Impl.java
                              |
                              +-- Echo.java
                              |
                              +-- EchoSoapBinding_Stub.java
```

stub 소스 코드는 wsdl2java task 의 destDir 속성으로 지정한 디렉토리 아래에 컴파일된다.

생성된 Java 파일 및 Java 파일안의 메쏘드의 이름은 웹서비스의 WSDL 로부터 매핑된다. WSDL 에 대응하는 Java 이름 매핑은 [표 2]에서 볼 수 있다.

표 2. WSDL 요소와 Java 구성 매핑

WSDL element	**매핑**
`<service>`	서비스 인터페이스와 구현 Java 클래스에 매핑된다. <service> 엘리먼트의 name 속성값이 인터페이스의 이름이다. 구현 파일은 <service> 엘리먼트의 이름 뒤에 '_Impl'이 붙는다.
`<port>`	<service> 엘리먼트의 서비스 인터페이스와 구현 클래스안의 메쏘드에 매핑된다. 메쏘드의 이름은 “get” + “<port> element 의 name 속성값”으로 이루어진다.
`<portType>`	웹서비스 operation 들을 위한 Java 인터페이스에 매핑된다. 이 Java 파일 이름은 <portType> 엘리먼트의 name 속성을 가지고 만들어진다.
`<binding>`	Stub 클래스. 이 Java 파일명은 <Binding> 엘리먼트의 name 속성값에 '_Stub'이 붙은 형태이다.

예제에서는 4 개의 소스 파일들이 생성된다.

- `Echo.java`: portType 인터페이스 클래스
- `EchoSoapBinding_Stub.java`: portType 인터페이스 클래스의 stub 클래스
- `DocLitEchoService.java`: 서비스 인터페이스 클래스
- `DocLitEchoService_Impl.java`: 서비스 구현 클래스

7.2.1.3 웹서비스 클라이언트의 코딩

이번 절에서는 웹서비스 클라이언트의 작성을 위해서 위에서 생성한 4 개의 클래스들을 어떻게 이용할 것인가를 알아 본다.

- 웹서비스 객체 생성

처음 단계는 원격 웹서비스를 위해 서비스 구현 객체를 생성하는 것이다. 예제에서 서비스 구현 클래스는 'DocLitEchoService_Impl'이다. 아래는 웹서비스 객체 생성을 위한 Java 코드이다.

```
DocLitEchoService service = new DocLitEchoService_Impl();
```

- Stub 객체로부터 Port 객체 얻기

일단 서비스 구현 객체가 생성되었다면 두 번째 단계로 서비스 구현 객체로부터 port 객체를 얻는다. 서비스 인터페이스 클래스는 port 객체를 얻기 위한 메쏘드들을 제공한다. 메쏘드 이름은 "get" + "WSDL 의 <port> element 의 name 속성값"으로 구성되며, WSDL 의 <portType> element 의 name 속성을 이름으로 가지는 타입이 리턴된다.

WSDL Port 이름은 웹서비스의 WSDL 문서 안에 <port> element 안에 기술되어 있다. <port> element 는 <service> element 의 하위 element 이다. 이 예제에서의 WSDL 문서는 아래와 같다.

```
<wsdl:portType name="Echo">
. . .
</wsdl:portType>
. . .
<wsdl:service name="DocLitEchoService">
  <wsdl:port binding="impl:EchoSoapBinding" name="EchoPort">
     <soap:address
      location="http://localhost:8088/DocLitEchoService/DocLitEcho
Service"/>
  </wsdl:port>
</wsdl:service>
```

서비스 인터페이스 클래스인 DocLitEchoService.java 는 아래와 같다 (여기서는 발췌한 내용만 보여준다).

<< *DocLitEchoService.java*>>

```
package echo;

public interface DocLitEchoService extends javax.xml.rpc.Service
{
    public java.lang.String getEchoPortAddress();

    public echo.Echo getEchoPort()
             throws javax.xml.rpc.ServiceException;

    public echo.Echo getEchoPort(java.net.URL portAddress)
             throws javax.xml.rpc.ServiceException;
}
```

다음은 서비스 객체로 부터 port 객체를 얻기 위한 소스 코드이다.

```
DocLitEchoService service = new DocLitEchoService_Impl();
Echo port = service.getEchoPort();
```

- Port 객체 상의 Operation 실행

현재 우리는 원격 웹서비스에서 제공하는 operation 들을 위한 port 객체를 가지고 있다. port 객체의 메쏘드들을 호출함으로써 웹서비스 operation 을 실행할 수 있다. 예제에서 웹서비스의 port 는 'echoString'이라는 operation 을 제공한다. Operation 을 실행하기 위한 코드는 다음과 같다.

```
DocLitEchoService service = new DocLitEchoService_Impl();
Echo port = service.getEchoPort();
String s = port.echoString(“JEUS”);
```

또는 위의 두 라인을 하나의 라인으로 만들 수 있다.

```
Echo port = new DocLitEchoService_Impl().getEchoPort();
String s = port.echoString(“JEUS”);
```

- 완전한 예제

다음은 웹서비스 클라이언트의 완전한 예제이다.

<<*ProxyClient.java*>>

```
package j2se;
```

```
import echo.DocLitEchoService_Impl;
import echo.Echo;

public class ProxyClient {
    public static void main(String args[]) {
        ProxyClient client = new ProxyClient();
        client.run();
    }

    public void run() {
        try {
            Echo port
              = new DocLitEchoService_Impl().getEchoPort();
            String s = port.echoString("JEUS");
            System.out.println("response = " + s);
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

이 파일은 C:\wsclient\src\j2se 디렉토리에 있다고 가정한다.

7.2.1.4 클라이언트 소스 코드 컴파일

ProxyClient 코드의 컴파일을 위해서 wsclient 디렉토리에 가서 아래의 명령어를 콘솔에 적용한다.

```
C:\wsclient>ant compile
```

컴파일이 성공했다면 컴파일된 클래스는 아래와 같을 것이다.

```
C:\wsclient\build\client\j2se\ProxyClient.class
```

7.2.1.5 웹서비스 클라이언트의 실행

'C:\wsclient' 디렉토리에 가서 아래의 명령어를 적용한다.

```
C:\wsclient>ant run
```

실행결과는 아래와 같을 것이다.

```
Response = JEUS
```

7.2.2 DII(Dynamic Invocation Interface)클라이언트 생성

7.2.2.1 소개

DII 를 이용하면, 클라이언트는 실행이 되기 전까지 Remote Operation 의 signature 나 웹서비스의 이름을 몰라도, Remote Operation 을 호출할 수 있다. 이 번 절에서는 DII 클라이언트를 생성하는 방법에 대해서 배운다.

7.2.2.2 DII 클라이언트의 작성

이번 절에서는 DII 클라이언트의 작성 과정에 대해 설명한다.

DII 클라이언트를 작성하는데 JAX-RPC 1.1 API 를 사용한다. JEUS 웹서비스는 JAX-RPC 1.1 API 를 완벽히 지원한다. 이번 절에서는 JAX-RPC API 의 간단한 사용법만을 보여준다. JAX-RPC 에 관한 자세한 정보는 JAX-RPC 스펙(http://www.jcp.org/en/jsr/detail?id=101)과 SUN 의 JAX-RPC 홈페이지(http://java.sun.com/xml/jaxrpc/index.html)를 참조하길 바란다.

참고로, DII 호출 방법은 Rpc 방식의 웹서비스를 호출하는 데에만 사용되어질 수 있다. 따라서 여기서는 Rpc 방식의 'RpcEncEchoService' 서비스의 'echoString' operation 을 호출 하는 DII 클라이언트를 작성할 것이고, 이를 위해 다음 단계를 따라 한다.

1. 다음의 중요 문장들을 DII 클라이언트 코드에 추가해야 한다.

```
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;
import javax.xml.namespaceQName;
```

2. ServiceFactory 와 Service 객체들을 생성한다.

```
String targetNamespace = "urn:RpcEncService";
String serviceName = "RpcEncService";
ServiceFactory factory = ServiceFactory.newInstance();
Service service =
   factory.createService(new QName(targetNamespace,
serviceName);
```

3. Call 객체 생성를 생성하고 설정한다.

```
String operationName = “echoString”;
QName QNAME_XSD_STRING
   = new QName(“http://www.w3.org/2001/XMLSchema”, “string”);
Call call = (Call) service.createCall();
call.setTargetEndpointAddress(endpoint);
call.setOperationName(new QName(targetNamespace,
operationName);
call.addParameter(“in0”, QNAME_XSD_STRING);
call.setReturnType(QNAME_XSD_STRING);
```

4. 최종적으로, Call 객체상에서 웹서비스 operation 을 실행한다.

```
String ret = (String)call.invoke(new Object[] { “JEUS” } );
```

완전한 예제는 아래와 같다.

<< DIIClient.java >>

```
package j2se;

import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;

public class DIIClient {
    private static final String NS_XSD =
       "http://www.w3.org/2001/XMLSchema";
    private static final QName QNAME_XSD_STRING =
       new QName(NS_XSD, "string");

    public void run() {
        try {
            ServiceFactory factory =
               ServiceFactory.newInstance();
            String endpoint =
              "http://localhost:8088/RpcEncEchoService/RpcEncEc
              hoService";
```

```
            String targetNamespace = "urn:RpcEncService";
            Service service = factory.createService(
             new QName(targetNamespace, "RpcEncService"));
            Call call = service.createCall();

            call.setTargetEndpointAddress(endpoint);
            call.setOperationName(
               new QName(targetNamespace, "echoString"));
            call.addParameter("in0", QNAME_XSD_STRING,
                                     ParameterMode.IN);
            call.setReturnType(QNAME_XSD_STRING);

            String ret = (String)call.invoke(
               new Object[] { "JEUS" });
            System.out.println("response = " + ret);
        } catch (Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        DIIClient client = new DIIClient();
        try {
            client.run();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

이 파일은 C:\wsclient\src 디렉토리에 있다고 가정한다.

7.2.2.3 DII 클라이언트의 컴파일

DIIClient 코드의 컴파일을 위해서 wsclient 디렉토리에 가서 아래의 명령어를 콘솔에 적용한다.

```
C:\wsclient>ant compile
```

컴파일이 성공했다면 컴파일된 클래스는 아래와 같을 것이다.

```
C:\wsclient\build\client\j2se\DIIClient.class
```

7.2.2.4 DII 클라이언트 실행

'C:\wsclient' 디렉토리에 가서 아래의 명령어를 적용한다.

```
C:\wsclient>ant run
```

실행결과는 아래와 같을 것이다.

```
Response = JEUS
```

8 JEUS 서버로부터의 웹 서비스 호출 (J2EE 클라이언트)

EJB, 서블릿과 같이 JEUS 서버에 배치된 구성 요소로부터 웹 서비스를 호출하는 것은 독립적인(stand-alone) 클라이언트로부터 호출하는것과 본질적으로는 동일하다. 그러나 현재 J2EE 웹서비스 스펙에서는 J2EE 웹서비스 클라이언트의 이식성(Portability)을 위해 프로그래밍 모델을 별도로 정의하고 있고, 이러한 모델을 따라 클라이언트를 작성하는 것을 권장한다.

8.1 J2EE 클라이언트 프로그래밍 모델의 개요

웹서비스를 클라이언트의 입장에서 본다면 클라이언트를 대신하여 비즈니스 로직을 수행하는 메쏘드의 집합이라 할 수 있다. 클라이언트는 메쏘드가 로컬에서 수행되는지, 아니면 원격에서 수행하는 지를 구별할 수 없다.

클라이언트는 JAX-RPC 스펙에 정의된 SEI(Service Endpoint Interface)를 사용하여 웹서비스에 접근한다.

J2EE 클라이언트는 JAX-RPC 에 정의된 서비스 인터페이스를 구현한 서비스 객체에 접근하기 위해서 JNDI 를 사용한다. 서비스 객체는 클라이언트가 SEI 를 구현한 스텁이나 프록시를 얻어오기 위해 사용하는 팩토리이다. 서비스 인터페이스는 JAX-RPC 에 정의된 javax.xml.rpc.Service 인터페이스이거나, 이를 상속하여 생성된 서비스 인터페이스이다.

클라이언트 개발자는 SEI 와 서비스 인터페이스를 얻어오는 것으로부터 시작하며 이들은 JAX-RPC 에 정의된 WSDL->Java 매핑 법칙에 의해 생성된다. 클라이언트 개발자는 스텁을 개발 시점에 생성하지 않을 것을 권장하며, 개발 시점에는 스텁이 아닌 인터페이스를 사용할 수 있다. 스텁은 클라이언트 모듈을 배치하는 시점에 클라이언트가 구동되는 환경에 맞게 자동으로 생성될 것이다. 웹서비스의 JNDI-lookup 은 논리적인 이름에 의해 이루어지며, 이 이름은 클라이언트 배치 서술자에 정의 된다.

8.2 J2EE 클라이언트 프로그래밍의 기본적인 절차

J2EE 클라이언트 프로그래밍의 기본적인 절차는 다음과 같다.

- JNDI Lookup 을 통한 서비스(인터페이스)의 획득
- 서비스 인터페이스의 API 를 이용한 스텁 생성 또는 Call 객체 생성
- 스텁이나 Call 객체를 이용한 웹서비스 호출

8.2.1 서비스 Lookup

클라이언트 개발자는 서비스의 레퍼런스로 사용되는 논리적인 서비스의 JNDI 이름을 클라이언트의 배치 서술자에 정의 해야 한다. 필수 요구 사항은 아니지만, 모든 서비스의 논리적인 레퍼런스 이름을 JNDI 네임 스페이스의 `service` 라는 서브 컨텍스트 밑에 구성하는 것을 권장한다. 컨테이너는 서비스 인터페이스의 구현을 클라이언트 환경 컨텍스트(`java:comp/env`) 밑에 서비스 레퍼런스의 논리적인 이름으로 바인딩 시켜야 한다.

다음은 예제이다.

```
InitialContext jndiContext = new InitialContext();
Service service = (Service)jndiContext.lookup(
             "java:comp/env/service/DocLitEchoService");
```

위의 예제에서는 서비스 인터페이스 `javax.xml.rpc.Service` 가 `web.xml` 의 `service-ref` 엘리먼트의 하위 엘리먼트인 `service-interface` 엘리먼트에 명시되어 있고, JNDI 이름 또한 `service-ref-name` 엘리먼트에 명시되어 서로 바인딩 되게 되어있다.

8.2.2 서비스 인터페이스

서비스 인터페이스는 클라이언트가 서비스 포트에 접근하기 위해서, 스텁과 동적 프록시 혹은 DII(Dynamic Invocation Interface) Call 객체를 생성하려고 할 때 사용된다.

클라이언트 개발자는 응용 프로그램이 사용하는 서비스 인터페이스의 타입을 명시적으로 클라이언트 배치 서술자(web.xml)에 선언하여야 한다.

다음은 서비스 인터페이스의 타입과 JNDI 이름이 명시된 web.xml 의 한 예이다.

<< *web.xml* >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>jsp_helloClient</servlet-name>
        <jsp-file>/helloClient.jsp</jsp-file>
        <load-on-startup>0</load-on-startup>
    </servlet>
    <service-ref>
        <service-ref-name>
            service/DocLitEchoService
        </service-ref-name>
        <service-interface>
            javax.xml.rpc.Service
        </service-interface>
        <wsdl-file>
            WEB-INF/wsdl/DocLitEchoService.wsdl
        </wsdl-file>
        <jaxrpc-mapping-file>
            WEB-INF/DocLitEchoService-mapping.xml
        </jaxrpc-mapping-file>
        <port-component-ref>
            <service-endpoint-interface>
               echo.Echo
            </service-endpoint-interface>
        </port-component-ref>
    </service-ref>
</web-app>
```

8.2.2.1 스텁/프록시를 이용한 호출

클라이언트는 서비스 Lookup 을 통하여 가져온 서비스 인터페이스의 다음과 같은 함수들을 이용해서 스텁과 동적인 프록시를 생성할 수 있다.

```
java.rmi.Remote getPort(QName portName, Class
   serviceEndpointInterface) throws ServiceException;
```

```
java.rmi.Remote getPort(java.lang.Class serviceEndpointInterface)
   throws ServiceException;
```

8.2.2.2 동적인 포트 접근

클라이언트는 JNDI Lookup 을 통하여 얻어온 서비스 인터페이스의 DII 메쏘드를 사용하여 Call 객체를 가져올 수 있다. API 들은 다음과 같다.

```
Call createCall() throws ServiceException;
Call createCall()(QName portName) throws ServiceException;
Call createCall(QName portName, String operationName) throws
   ServiceException;
Call createCall(QName portName, QName operationName) throws
   ServiceException;
Call[] getCalls(QName portName) throws ServiceException;
```

8.2.2.3 ServiceFactory

JAX-RPC 에서는 ServiceFactory 라는 클래스에서 서비스를 생성할 수 있다. 하지만, J2EE 응용 프로그램에서는 ServiceFactory 를 사용하는 것을 권장하지 않는다. J2EE 클라이언트는 항상 JNDI 서비스 Lookup 을 통해 서비스를 가져와야 한다.

8.3 J2EE 클라이언트의 작성과 예제

J2EE 클라이언트의 작성은 WSDL 을 가지고 할 수도 있고, WSDL 이 없이 할 수도 있다. 또, 클라이언트의 형태는 JSP, 서블릿, EJB 등 여러 형태 일 수 있고, 앞에서 제시한 프로그래밍 모델에 부합하기만 하면 된다.

이번 장에서는 간단한 JSP 형태의 J2EE 클라이언트 작성 예제를 통해 작성법을 설명하기로 한다.

8.3.1 WSDL 을 가지고 서비스를 호출할 때

WSDL 을 가지고 서비스를 호출하려고 할 때에는 다음과 같은 절차를 따른다.

1. J2EE 클라이언트 포터블 아티팩트(portable artifact)와 JAX-RPC 매핑 파일의 생성
2. 클라이언트 프로그램의 작성

3. 배치 서술자(DD)의 작성

8.3.1.1 J2EE 클라이언트 포터블 아티팩트(portable artifact)와 JAX-RPC 매핑파일의 생성

J2EE 클라이언트 포터블 아티팩트와 JAX-RPC 매핑 파일을 생성하는 것을 돕기 위해 JEUS 는 'wsdl2java' Ant 태스크와 wsdl2java 커맨드 라인 툴을 제공한다.

Ant 태스크를 사용하는 build.xml 의 예는 다음과 같다.

<< *build.xml* >>

```
…
<property name="jaxrpc.wsdl"
      value="${work.home}/wsdl/DocLitEchoService.wsdl"/>
<property name="jaxrpc.mapping"
      value="${build.dir}/DocLitEchoService-mapping.xml"/>
…
<target name="wsdl2java" depends="init">
      <wsdl2java destDir="${classes.dir}"
                     verbose="true"
                     mode="import:client"
                     doCompile="true"
                     package="echo"
                     outputMapping="${jaxrpc.mapping}"
                     wsdl="${jaxrpc.wsdl}">
                     <classpath refid="build.classpath"/>
      </wsdl2java>
</target>
```

wsdl2java 태스크에 대한 더 자세한 설명은 "부록 A JEUS 웹 서비스 Ant Task 참조"를참조하라.

"ant wsdl2java" 명령을 수행하고 나면, J2EE 클라이언트 포터블 아티팩트와 JAX-RPC 매핑 파일이 생성된다.

8.3.1.2 클라이언트 프로그램의 작성

WSDL 을 가지고 클라이언트를 개발 하는 경우는 다음과 같은 java.xml.rpc.Service 인터페이스 메쏘드를 이용하여 스텁이나 Call 객체들을 가져 올 수 있다.

```
java.rmi.Remote getPort(QName portName, Class
   serviceEndpointInterface) throws ServiceException;
java.rmi.Remote getPort(java.lang.Class serviceEndpointInterface)
   throws ServiceException;
Call createCall() throws ServiceException;
Call createCall()(QName portName) throws ServiceException;
Call createCall(QName portName, String operationName) throws
   ServiceException;
Call createCall(QName portName, QName operationName) throws
   ServiceException;
Call[] getCalls(QName portName) throws ServiceException;
```

다음은 위의 메쏘드 중 동적인 프록시(Dynamic Proxy)를 생성하여 웹서비스를 호출하도록 JSP 로 작성된 J2EE 클라이언트의 예이다.

<< helloClient.jsp >>

```
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>

<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>

<%@ page import="import echo.*" %>
<%@ page errorPage="/error.html" %>

<%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString="";
%>
<%
        try {
            InitialContext jndiContext = new InitialContext();
            Service service = (Service)jndiContext.lookup(
                "java:comp/env/service/DocLitEchoService");
            java.rmi.Remote port = service.getPort(Echo.class);
            Echo echoPort = (Echo)port;
```

```
            ret = echoPort.echoString(msgToSend);
        } catch (Exception e) {
            exceptionString = e.toString();
            e.printStackTrace();
        }
%>
<%= "Result is "+ret+"......"
%>
```

8.3.1.3 배치 서술자의 작성

앞서 제시한 예제에 대한 표준 DD(web.xml)는 다음과 같이 구성되어 있다.

<< web.xml >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>jsp_helloClient</servlet-name>
        <jsp-file>/helloClient.jsp</jsp-file>
        <load-on-startup>0</load-on-startup>
    </servlet>
    <service-ref>
        <service-ref-name>
             service/DocLitEchoService
        </service-ref-name>
        <service-interface>
             javax.xml.rpc.Service
        </service-interface>
        <wsdl-file>
             WEB-INF/wsdl/DocLitEchoService.wsdl
        </wsdl-file>
        <jaxrpc-mapping-file>
             WEB-INF/DocLitEchoService-mapping.xml
        </jaxrpc-mapping-file>
        <port-component-ref>
            <service-endpoint-interface>
              echo.Echo
            </service-endpoint-interface>
```

```
        </port-component-ref>
    </service-ref>
</web-app>
```

이 web.xml 에는 JSP 클라이언트에서 JNDI Lookup 으로 찾아오는 서비스 이름이 service-ref-name 엘리먼트에 명시되어 있고, 서비스 인터페이스가 javax.xml.rpc.Service 인터페이스임이 명시되어 있다. 이 인터페이스의 구현은 wsdl-file 이라는 엘리먼트에 설정된 WSDL 파일의 정보를 가지고 클라이언트가 JEUS 에 배치될 때 생성된다. 이 때, 필요한 자바 패키지와 타입에 관한 정보는 jaxrpc-mapping-file 엘리먼트에 설정된 JAX-RPC 매핑 파일에서 얻는다.

다음은 앞에 제시한 예제의 JEUS 웹 DD(jeus-web-dd.xml)이다.

<< *jeus-web-dd.xml* >>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
      <service-ref>
        <service-client>
          <service-ref-name>
              service/DocLitEchoService
          </service-ref-name>
          <port-info>
            <wsdl-port xmlns:ns1="urn:DocLitService">
             ns1:Echo
            </wsdl-port>
             <stub-property>
              <name>javax.xml.rpc.service.endpoint.address</name>
              <value>http://localhost:8088/DocLitEchoService/DocLi
tEchoService</value>
             </stub-property>
          </port-info>
        </service-client>
      </service-ref>
</jeus-web-dd>
```

8.3.2 WSDL 없이 서비스를 호출할 때

WS-I 라는 표준 기구는 웹서비스의 상호 호환성을 위해 Basic Profile 을 정의하고 이를 준수할 것을 요구하고 있다. 현재 나와있는 Basic Profile 1.0 은 WSDL 이 항상 접근 가능하게 공개되기를 요구한다. 하지만, Basic Profile 을 준수하지 않는 서비스에 접근하기를 원한다면 WSDL 이 항상 공개되어 있다고 볼 수 없다. 이 경우 WSDL 과 무관하게 웹서비스를 호출할 수 있는데, DII(Dynamic Invocation Interface)가 바로 그 방법이다. 하지만, 이는 결국 WSDL 에 들어 있는 오퍼레이션 스타일이라든가, 오퍼레이션 이름과 같은 상세를 Call 객체에 제공해야만 가능하다.

WSDL 이 클라이언트 DD 에 정의되어 있지 않다면, 클라이언트 개발자는 다음과 같은 서비스 인터페이스의 메쏘드를 통해서 Call 객체를 생성해야 한다.

```
Call createCall() throws ServiceException;
```

다음은 Call 객체를 이용하여 웹서비스를 호출하는 JSP J2EE 클라이언트의 예이다.

<< *helloClient.jsp* >>

```
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>

<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page import="javax.xml.rpc.Call" %>
<%@ page import="javax.xml.namespace.QName" %>
<%@ page import="javax.xml.rpc.ParameterMode" %>

<%@ page errorPage="/error.html" %>

<%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString="";
%>
```

```
<%
   try {
      InitialContext jndiContext = new InitialContext();
      Service service = (Service)jndiContext.lookup(
         "java:comp/env/service/RpcEncEchoService");

      String targetNamespace = "urn:RpcEncService";
      QName operationName = new
         QName(targetNamespace,"echoString");
      Call call = service.createCall();
      call.setOperationName(operationName);
      call.addParameter("in0",
         new QName("http://www.w3.org/2001/XMLSchema", "string"),
         ParameterMode.IN);

      call.setReturnType(
         newQName("http://www.w3.org/2001/XMLSchema","string"));
      call.setTargetEndpointAddress(
      "http://localhost:8088/RpcEncEchoService/RpcEncEchoService"
      );

      ret = (String)call.invoke(new Object[]{msgToSend});
      } catch (Exception e)
      {
         exceptionString = e.toString();
         e.printStackTrace();
      }
%>
<%= "Result is "+ret+"......"
%>
```

위 예제에서 제시한 대로, Call 객체를 생성한 다음, 오퍼레이션 이름과 오퍼레이션이 갖는 파라미터들을 추가 하게 되며, 이때 추가적으로 오퍼레이션의 스타일과 인코딩 방식을 설정할 수도 있다. 설정법은 다음과 같다.

```
call.setProperty(Call.OPERATION_STYLE_PROPERTY, “rpc”);
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, “literal”);
```

앞서 제시한 예제의 표준 DD(web.xml)는 다음과 같이 구성되어 있다.

<< *web.xml* >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>jsp_helloClient</servlet-name>
        <jsp-file>/helloClient.jsp</jsp-file>
        <load-on-startup>0</load-on-startup>
    </servlet>
    <service-ref>
        <service-ref-name>service/DocLitEchoService2
        </service-ref-name>
        <service-interface>javax.xml.rpc.Service
        </service-interface>
    </service-ref>
</web-app>
```

이 web.xml 에는 JSP 클라이언트에서 JNDI Lookup 으로 찾아오는 서비스 이름이 service-ref-name 엘리먼트에 명시되어 있고, 서비스 인터페이스가 javax.xml.rpc.Service 인터페이스임이 명시되어 있다. WSDL 이 없이 웹 서비스를 호출할 때에는 jaxrpc-mapping-file 엘리먼트를 이용한 JAX-RPC 매핑 파일의 설정을 사용할 수 없다.

8.3.3 J2EE 웹서비스 클라이언트의 패키징

J2EE 웹서비스 클라이언트는 하나의 J2EE 콤포넌트이기 때문에 콤포넌트 고유의 패키징 방식을 따른다.

앞서 제시한 JSP 클라이언트의 패키징 방식은 아래의 그림과 같다.

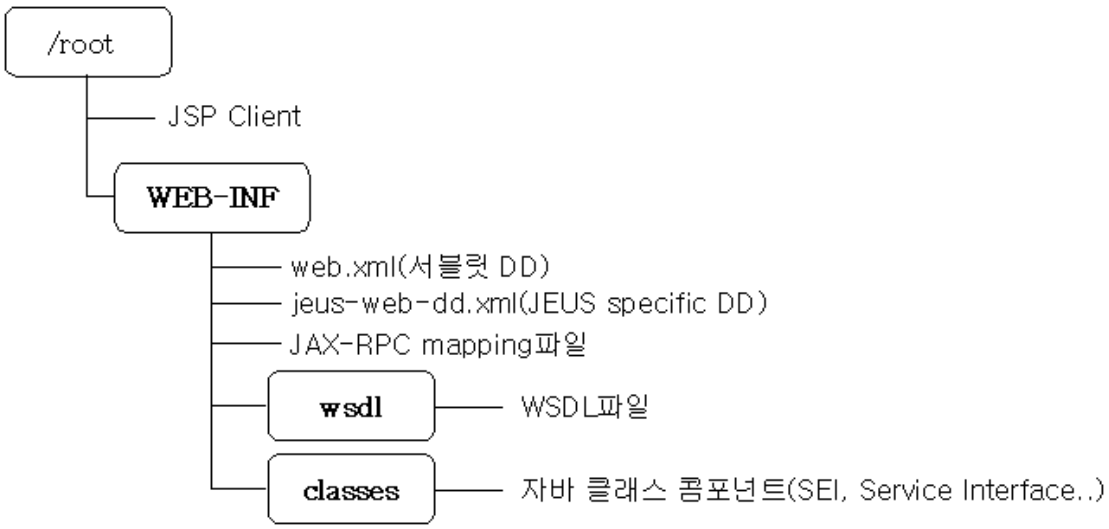


그림 6 J2EE WebService JSP Client Packaging

위와 같이 패키징 한 J2EE 클라이언트를 배치하고 JSP 를 호출하면 웹 서비스를 호출할 수 있게 된다.

JAX-RPC 매핑파일과 WSDL 파일이 webservices.xml 파일에 <wsdl-file>과 <jax-rpc-mapping-file>로 기술한 위치에 존재한다면 반드시 위의 구조를 따르지 않아도 무방하다.

9 SOAP 메시지 핸들러의 생성

SAAJ(SOAP with Attachments API for Java) 1.2 를 사용하면 SOAP 메시지를 직접 생성하고 읽고 다루는 작업을 할 수 있다. 이번 장에서는 SAAJ 프로그래밍 모델에 대해 언급하고, SAAJ 를 JAX-RPC 와 연동하여 사용하는 메시지 핸들러에 대해서 설명하기로 한다.

9.1 SAAJ 사용하기

여기에서는 SAAJ 의 프로그래밍 모델에 대해서 간략히 언급한다. 자세한 API 관련 내용은 여기에서 다루지 않고 그 개념을 주로 언급한다. 보다 세세한 내용을 필요로 할 경우에는 SAAJ 1.2 스펙을 참조한다.

SAAJ 는 부착물이 없는 간단한 XML 과도 같은 단순한 SOAP 메시지에서부터 MIME 부착물을 가지는 더 복잡한 SOAP 메시지를 다루는 작업까지 가능하다. SAAJ 는 Abstract Factory 패턴을 기본으로 하며, MessageFactory 에서 메시지(SOAPMessage)를 생성한다. SOAPMessage 는 SOAP 문서를 나타내는 SOAPPart 와 MIME 부착물을 나타내는 0 개 이상의 AttachmentPart 객체를 포함하게 된다. 이를 표현하면 [그림 7]과 같다.

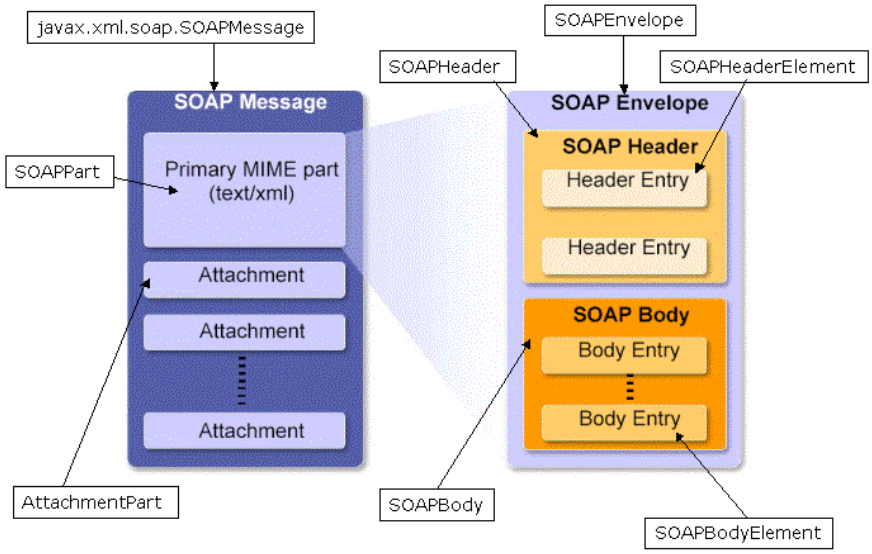


그림 7 Structure of SOAP with Attachment(출처: java.sun.com)

9.1.1 SOAP 메시지의 생성

비어있는 SOAP 메시지를 생성하기 위해서는 MessageFactory 객체로부터 SOAPMessage 객체를 새로이 생성해야 한다. 예는 다음과 같다.

```
MessageFactory msgFactory = MessageFactory.newInstance();
SOAPMessage message = msgFactory.createMessage();
```

9.1.2 SAAJ 문서 다루기

SAAJ 는 SOAP 문서를 생성하고 다루기 위해 사용할 수 있는 많은 인터페이스를 제공한다. SOAP 문서는 엘리먼트와 속성으로 구성된 XML 인스턴스이다. 편의를 위해 SOAP 문서의 주요한 부분들은 SAAJ 와 대응하는 타입을 가지게 설계되었다. Envelope 는 SOAPEnvelope, Header 는 SOAPHeader, Body 는 SOAPBody 와 대응한다. SOAPElement 타입은 SOAP 이름공간에 속하지 않는 응용프로그램에 종속적인(application-specific) 엘리먼트와 대응한다.

9.1.2.1 SOAPPart 와 SOAPEnvelope 타입

SwA(SOAP with Attachment) 메시지를 가지고 작업을 하다보면 SOAPPart 와 SOAPEnvelope 타입을 자주 사용하게 된다. SOAPPart 는 SwA 메시지의

MIME 부분의 가장 상위 이며, SOAPMessage.getSOAPPart() 메쏘드를 호출하여 접근 가능하다.

SOAPEnvelope 에 대한 접근은 SOAPPart 의 getEnvelope() 메쏘드를 호출하여 가능하다. SOAPEnvelope 는 XML SOAP 문서의 최상위이며 SOAPHeader 와 SOAPBody 에 대한 접근과 생성을 위한 메쏘드를 포함한다.

9.1.2.2 SOAPElement 타입

SOAPElement 타입은 SOAP 1.1 XML 의 namespace 로 표현되지 않는 응용프로그램에 종속적인 (application-specific) 엘리먼트를 직접적으로 표현하고자 할 때 사용한다. 이 타입은 XML 엘리먼트를 나타낼 수 있으며, 따라서 XML 엘리먼트가 다른 XML 엘리먼트를 포함할 수 있듯이 SOAPElement 는 다른 SOAPElement 객체를 포함할 수 있다. 이 타입은 다른 SOAP 타입 (SOAPEnvelope, SOAPBody, SOAPBodyElement, SOAPHeader, SOAPHeaderElement, fault 엘리먼트)들의 상위 타입(super-type)이다.

9.1.2.3 SOAPHeader 타입

SOAP 메시지의 Header 엘리먼트는 0 혹은 그 이상의 헤더 블록을 가질 수 있다. SOAPHeader 타입은 Header 엘리먼트를 나타내며, SOAPHeaderElement 타입은 SOAP 메시지에 있어서 각각의 헤더 블록을 나타낸다. SOAPHeader 타입은 SOAPHeaderElement 객체를 추가, 삭제, 검사 하는 메쏘드를 제공한다.

9.1.2.4 SOAPHeaderElement 타입

SOAPHeaderElement 는 특정한 헤더 블록의 속성과 자식 엘리먼트를 변경, 검사할 수 있도록 헤더 블록을 추상화 한 모델이다. SOAPHeaderElement 타입은 actor 와 mustUnderstand 같은 속성 뿐 아니라 하나 혹은 그 이상의 SOAPElement 객체를 포함할 수도 있다.

9.1.2.5 SOAPBody 타입과 SOAPBodyElement 타입

SOAPBody 타입은 SOAP 메시지의 Body 엘리먼트를 나타낸다. SOAPBodyElement 는 SOAPElement 를 확장하며, 상속 하는 메쏘드 이외의 추가적인 메쏘드는 없다.

다음은 SOAPBody 타입과 SOAPBodyElement 타입의 활용하는 예를 보여준다.

```
Name echo_Name = soapFactory.createName(“echo”, “tmax”,
   “http://www.tmax.com/saaj/test”);
SOAPBody body = message.getSOAPBody();
SOAPBodyElement echo_Element = body.addBodyElement(echo_Name);
```

9.1.3 SAAJ 를 이용한 SOAP 메시지 전송

SAAJ 를 이용한 SOAP 메시지 생성 작업을 마쳤다면 전송 작업을 수행할 수 있다.

SAAJ 는 간단하게 구성된 독자적인 메시지 전송 시스템을 가지고 있으며, 기본 API 의 한 부분으로 구성된다. SAAJ 를 사용함으로써 요청/응답 스타일의 SOAP 메시지를 웹서비스와 교환할 수 있다.

SOAPConnection 을 생성해서 메시지를 전송한다. 다음은 그 예이다.

```
//Build a SOAPMessage from a file
MessageFactory msgFactory = MessageFactory.newInstance();
MimeHeaders mimeHeaders = new MimeHeaders();
MimeHeaders.addHeader(“Content-Type”, “text/xml; charset=UTF-8”);
FileInputStream file = new FileInputStream(“soap.xml”);
SOAPMessage requestMsg =
msgFactory.createMessage(mimeHeaders,file);
file.close();

String address = “…”;

//Send the SOAP message to the BookQuote Web Service
SOAPConnectionFactory conFactory =
   SOAPConnectionFactory.newInstance();
URL url = new URL(address);
SOAPMessage replyMsg = connection.call(requestMsg, url);
```

9.2 SOAP 메시지 핸들러의 생성

메시지 핸들러는 JAX-RPC 클라이언트와 웹서비스 엔드포인트에 의해 송수신 되는 SOAP 메시지를 직접 다룰수 있게 하며, 정적으로 생성된 스텁, 동적으로 생성된 프록시, DII(Dynamic Invocation Interface), Java 클래스 엔드포인트, EJB 엔드포인트와 함께 사용되어질 수 있다.

메시지 핸들러의 가장 중요한 목적은 JAX-RPC 클라이언트와 웹서비스 엔드포인트가 송수신하는 SOAP 메시지의 헤더 블록을 더하고, 읽고, 다루는 메커니즘을 제공하는 것이다.

메시지 핸들러는 J2EE 컨테이너에 의해서 다루어진다. JAX-RPC 클라이언트 API를 사용하여 SOAP 메시지를 전송하려 할 때는, JAX-RPC 런타임이 웹서비스로 연결된 네트웍에 내보내기 전에 메시지 핸들러 체인을 통해 SOAP 메시지를 가공하게 된다. 마찬가지로 SOAP 응답 메시지가 JAX-RPC 클라이언트에 의해 수신 될 때에는 클라이언트 응용프로그램으로 결과가 리턴되어오기 전에 이전에 거쳐 나갔던 동일한 메시지 핸들러 연결고리를 거쳐서 가공되어지게 되어있다.

이러한 핸들러의 쓰임새는 여러가지가 있는데, 그중에서 보안에 관련된 경우를 살펴보면, 클라이언트에서 암호화된 SOAP 메시지를 송신하고 싶을 경우 암호화 작업을 위한 핸들러를 사용하여 메시지를 가공하여 송신할 수 있고 웹 서비스는 그 요청 메시지를 암호 해독을 위한 핸들러를 이용하여 해독한 다음 그 데이터를 웹 서비스를 구현한 백엔드로 보내게 된다. 이에 대한 SOAP 메시지 응답은 이 과정을 역으로 수행하게 된다.

또 다른 예는 SOAP 메시지의 헤더 부분에 저장된 정보에 접근하고 싶을 때 사용하는 경우이다. SOAP 헤더에 웹 서비스의 특정한 정보를 저장할 수 있고, 핸들러로 하여금 그 정보를 다룰 수 있게 할 수 있다.

9.2.1 메시지 핸들러 생성의 개관

SOAP 메시지 핸들러는 웹서비스의 요청과 응답에서 모두 SOAP 메시지를 가로채어 가공할 수 있다. 또 웹서비스 엔드포인트와 웹서비스를 호출하는 클라이언트에서 모두 핸들러를 생성할 수 있다.

다음 표는 javax.xml.rpc.handler API 의 주요 클래스와 인터페이스를 나타내고 있다.

표 3 JAX-RPC Handler Interface and Classes

Classes and Interfaces	**설명**
`javax.xml.rpc.handler.`	SOAP 요청 메시지와 응답 메시지, 폴트(fault) 메시지를 다루는 메쏘드를 포함하

`Handler`	는 주요 인터페이스.
`javax.xml.rpc.handler.HandlerChain`	핸들러의 리스트를 나타내는 인터페이스. 리스트가 가지는 엘리먼트는 java.xml.rpc.handler.Handler 타입.
`javax.xml.rpc.handler.HandlerRegistry`	프로그래밍 레벨에서 핸들러의 설정을 조정할 수 있도록 지원하는 인터페이스.
`javax.xml.rpc.handler.HandlerInfo`	webservices.xml 에 정의되는 핸들러의 초기 파라미터와 같은 핸들러의 정보들을 포함하는 클래스
`javax.xml.rpc.handler.MessageContext`	핸들러에 의해서 처리되는 메시지의 내용을 추상화한 인터페이스. 핸들러 체인 내에서 핸들러는 메시지 처리에 관계되는 상태 정보를 공유할 수 있다.
`javax.xml.rpc.handler.soap.SOAPMessageContext`	SOAP 요청과 응답 메시지에 접근 할 수 있는 메쏘드를 제공하는 MessageContext 의 하위 인터페이스
`javax.xml.rpc.handler.GenericHandler`	Handler 인터페이스를 구현한 추상클래스로서 SOAP 메시지 핸들러 개발자는 이 클래스를 상속받아서 구현하면 된다.
`javax.xml.soap.SOAPMessage`	SOAP 메시지의 요청과 응답을 포함하는 클래스.

다음은 메시지 핸들러와 핸들러 체인을 추가함으로써 웹서비스를 갱신하는 절차이다.

1. 메시지 핸들러와 핸들러 체인의 설계
2. javax.xml.rpc.handler.Handler 인터페이스를 구현하는 자바 클래스의 생성 및 핸들러 체인의 구현
3. 자바 코드의 컴파일

4. 웹서비스 배치 서술자(webservices.xml)의 작성

5. 웹서비스 패키징과 배치

웹서비스 클라이언트에서도 SOAP 메시지 핸들러를 사용할 수 있으며 이는 이장의 뒷부분에 언급된다.

9.2.2 메시지 핸들러와 핸들러 체인(chain)의 설계

SOAP 메시지 핸들러를 설계하려면 다음을 명확하게 해야한다.

- 추가할 메시지 핸들러의 개수
- 추가할 핸들러의 실행 순서
- 메시지 핸들러로만 웹서비스를 구성할 것인가?(백엔드 구성요소들을 호출하지 않을 것인가?)

핸들러들은 webservices.xml 안에서 설정해주어야 하며, 실행 순서가 정해진 핸들러의 묶음을 핸들러 체인이라고 한다. 핸들러 체인으로 구성된 각각의 핸들러는 요청 SOAP 메시지를 다루기 위한 메쏘드와 응답 SOAP 메시지를 다루기 위한 또다른 메쏘드가 있다.

웹서비스를 호출하면 JEUS 웹서비스는 핸들러를 다음과 같은 메커니즘으로 실행한다.

1. webservices.xml 에 정해진 순서대로 핸들러 체인 안의 각각의 핸들러의 handleRequest() 메쏘드가 실행된다.

2. 핸들러 체인의 마지막 핸들러의 handleRequest()가 실행이 되고나면, JEUS 웹서비스는 웹서비스 백엔드를 호출한다(백엔드가 존재하는 경우).

3. 백엔드가 실행이 종료되면 핸들러 체인에 속한 핸들러가 webservices.xml 에 정의된 반대의 순서대로 불려지고, 각각의 핸들러의 handleResponse() 메쏘드가 호출이 된다.

4. webservices.xml 에 정의된 첫번째 핸들러의 handleResponse()가 실행이 되면, 웹서비스를 호출한 클라이언트 쪽으로 SOAP 메시지 응답을 보낸다.

9.2.3 핸들러 인터페이스의 구현

SOAP 메시지 핸들러 클래스는 javax.xml.rpc.handler.Handler 인터페이스를 직접 구현하거나 이를 구현한 javax.xml.rpc.handler.GenericHandler 를 상속받아서 구현하면 된다.

메시지 핸들러 클래스는 Handler 인터페이스의 다음과 같은 함수들을 구현해야 한다.

- init()
- destroy()
- getHeaders()
- handleRequest()
- handleFault()

9.2.3.1 init()의 구현

HandlerInfo 객체는 webservices.xml 에 정의된 초기화 정보 같은 SOAP 메시지 핸들러에 대한 정보를 포함하고 있다. HandlerInfo.getHandlerConfig() 메쏘드를 실행시키면 name-value 형태의 맵(Map)을 리턴한다.

Handler 의 초기화 작업을 진행하려면 init() 메쏘드를 구현해야 한다.

9.2.3.2 destroy()의 구현

Handler.destroy()메쏘드는 Handler 객체의 인스턴스를 제거하려고 할 때 불리며, 핸들러의 생성 주기 동안 획득된 자원을 다시 놓아주게 하려면 이 함수를 구현해야 한다.

9.2.3.3 getHeaders()의 구현

핸들러 인스턴스에 의해 처리되는 헤더 블록을 가져오려 할 때 사용한다.

9.2.3.4 handleRequest()의 구현

Handler.handleRequest() 메쏘드는 백엔드에 SOAP 메시지를 전달하기 전에 가로챌 경우에 사용한다. MessageContext 객체는 SOAP 메시지 핸들러에 의해 처리되는 메시지의 내용을 가지고 있으며, MessageContext 의 서브 인터페이스인 SOAPMessageContext 를 사용하면 SOAP 요청 메시지의 내용을 얻거나 변경할 수 있다.

메시지를 처리하기 위해서는 앞에서 언급되었던 SAAJ 를 사용한다. SOAPMessageContext.getMessage()를 수행하면, SAAJ API 에 속하는 SOAPMessage 를 얻을 수 있고, SOAPMessage 는 SOAPPart 객체와 부착물(attachment)로 구성된다. SOAP 메시지를 다루는 작업은 SAAJ 프로그래밍 기법과 같다.

9.2.3.5 handleResponse()의 구현

Handler.handleResponse() 메쏘드는 SOAP 메시지가 백엔드에서 처리되고 난 후, 웹서비스를 호출한 클라이언트에게 회신하기전에 SOAP 메시지를 가로채려 할 때 사용한다. MessageContext 객체는 SOAP 메시지 핸들러에 의해 처리되는 메시지의 내용을 가지고 있으며, MessageContext 의 서브 인터페이스인 SOAPMessageContext 를 사용하면 SOAP 응답 메시지의 내용을 얻거나 변경할 수 있다.

메시지를 처리하기 위해서는 앞에서 언급되었던 SAAJ 를 사용한다. SOAPMessageContext.getMessage()를 수행하면, SAAJ API 에 속하는 SOAPMessage 를 얻을 수 있고, SOAPMessage 는 SOAPPart 객체와 부착물(attachment)로 구성된다. SOAP 메시지를 다루는 작업은 SAAJ 프로그래밍 기법과 같다.

9.2.4 J2EE 웹서비스 배치 서술자(webservices.xml)의 작성

webservices.xml 파일에는 SOAP 메시지 핸들러에 대한 정보, 그리고 핸들러의 처리 순서를 정의할 수 있다. 다음은 메시지 핸들러 정보를 설정하는 예이다.

<< 메시지 handler 를 정의하는 <port-component> 엘리먼트의 예 >>

```
<port-component>
   <port-component-name>FileAttPort</port-component-name>
   …
    <handler>
        <handler-name>ServerAttachmentHandler</handler-name>
        <handler-class>
             filetransfer.ServerAttachmentHandler
        </handler-class>
        <init-param>
            <param-name>directory</param-name>
            <param-value>/temp</param-value>
        </init-param>
```

```
   </handler>
</port-component>
```

9.2.5 클라이언트에서 SOAP 메시지 핸들러의 사용

이제까지는 JEUS 서버에서 동작하여 웹서비스로 실행되는 핸들러를 생성하는 것에 설명했지만, 클라이언트 응용 프로그램에서도 핸들러를 생성하는 것이 가능하다.

클라이언트 핸들러를 생성하는 방법은 서버 사이드의 핸들러를 생성하는 방법과 같다. javax.xml.rpc.handler.Handler 인터페이스를 구현하는 자바 클래스를 작성한다.

클라이언트 사이드 핸들러를 작성하고 난 뒤의 작업은 서버 사이드의 핸들러를 작성하는 것과 차이가 있는데, 핸들러를 DD 에 기술하는 것이 아니라 javax.xml.rpc.handler.HandlerInfo 와 javax.xml.rpc.handler.HandlerRegistry 클래스를 이용하여 프로그램에서 직접 핸들러를 등록하여야한다. 다음 장에 이어지는 예제를 참고한다.

9.2.6 예제 : 파일 송수신하는 웹서비스와 클라이언트

이번에 소개되는 것은 클라이언트에서 File_Send.txt 라는 파일을 서버로 전송하고 서버에서는 이 파일을 지정된 폴더에 저장하고 파일을 받았다는 응답을 날려주는 웹서비스 파일 송수신 예제이다.

여기서 설명되는 절차는 다음과 같다.

1. 파일 수신하는 메시지 핸들러의 구현
2. 파일 수신을 하고 난 다음, 파일을 받았다는 메시지를 날려주는 서비스 백엔드 구현
3. 웹서비스 배치서술자의 작성 및 웹서비스 생성과 배치
4. 웹서비스 클라이언트 핸들러의 작성
5. 웹서비스 클라이언트의 작성
6. 실행

9.2.6.1 파일 수신 메시지 핸들러의 구현

다음은 웹서비스의 메시지 핸들러 구현으로서, handleRequest()가 백엔드로 SOAP 요청이 가기전에 먼저 실행된다. handleRequest()함수 안에서는 SOAP

메시지에 부착물(Attachment)로 붙어있는 파일을 SAAJ API 를 이용하여 다루게 된다.

<< *ServerAttachmentHandler.java* >>

```
package filetransfer;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Iterator;
import java.util.Map;

import javax.xml.namespace.QName;
import javax.xml.rpc.JAXRPCException;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;

import javax.xml.rpc.handler.GenericHandler;

public final class ServerAttachmentHandler
  extends GenericHandler
{
  private File dir;

  private final String DIR_PROP="directory";

  public void init(HandlerInfo info) {
    super.init(info);
    Map m = info.getHandlerConfig();

    String dirName = (String) m.get(DIR_PROP);
```

```
    if (dirName == null) {
      throw new JAXRPCException("Property named: "+DIR_PROP+
        " was not found");
    }

    dir = new File(dirName);

    if (! dir.exists()) {
      if (! dir.mkdirs()) {
        throw new JAXRPCException("Unable to create directory: "+
          dirName);
      }
    }

    if (! dir.canWrite()) {
      throw new JAXRPCException("Don't have write permission for
"+
        dirName);
    }
  }

  private String getFileName(SOAPMessage request)
    throws SOAPException
  {
    SOAPBody body =
request.getSOAPPart().getEnvelope().getBody();
    Object obj = body.getChildElements().next();
    SOAPElement opElem = (SOAPElement) obj;
    SOAPElement paramElem = (SOAPElement)
      opElem.getChildElements().next();
    return paramElem.getValue();
  }

  private void copyFile(InputStream is, OutputStream os)
    throws IOException
  {
    byte [] b = new byte[8192];
    int nr;
```

```
  while ((nr = is.read(b)) != -1) {
    os.write(b, 0, nr);
  }
}


public boolean handleRequest(MessageContext mc) {
  SOAPMessageContext ctx = (SOAPMessageContext) mc;
  SOAPMessage request = ctx.getMessage();
  if (request.countAttachments() == 0) {
    throw new JAXRPCException("** Expected attachments");
  }
  try {
    Iterator it = request.getAttachments();

    while(it.hasNext()) {
      AttachmentPart part = (AttachmentPart) it.next();
      String fileName = getFileName(request);
      System.out.println("Received file named: "+fileName);

      File outFile = new File(dir, fileName);
      OutputStream os = null;
      InputStream  is = null;

      try {
        os = new FileOutputStream(outFile);
        is = part.getDataHandler().getInputStream();

        copyFile(is, os);

      } catch (IOException ioe) {
        ioe.printStackTrace();
        throw new JAXRPCException("Exception writing file "+
          fileName, ioe);
      } finally {
        try {
          if (is != null) is.close();
```

```
          } catch (IOException ignore) {}

          try {
            if (os != null) os.close();
          } catch (IOException ignore) {}

        }
      }
    } catch (SOAPException e) {
      e.printStackTrace();
      throw new JAXRPCException(e);
    }
    return true;
  }


    public QName[] getHeaders() {
        // TODO Auto-generated method stub
       return null;
    }
}
```

9.2.6.2 웹서비스 백엔드의 구현

다음은 웹서비스 백엔드 구현으로서 메시지 핸들러에서 파일을 받는 작업이 수행되고 난 다음 실행이 되며, 클라이언트로 파일을 받았다는 응답을 날려주는 클래스이다.

<< *FileTransfer.java* >>

```
package filetransfer;

public class FileTransfer implements FileTransferIF{

  public String receiveFile(String s) {
    return "Received file named: "+s;
  }
}
```

다음은 서비스 엔드포인트 인터페이스이다.

<< *FileTransferIF.java* >>

```
package filetransfer;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface FileTransferIF extends Remote {
    public String receiveFile(String s) throws RemoteException;
}
```

9.2.6.3 웹서비스 배치 서술자의 작성 및 웹서비스 생성과 배치

다음은 웹서비스 배치 서술자(webservices.xml)이다.

<< *webservices.xml* >>

```
<?xml version="1.0"?>
<webservices version="1.1"
xmlns="http://java.sun.com/xml/ns/j2ee">
   <webservice-description>
      <webservice-description-name>
         FileAttachmentService
      </webservice-description-name>
      <wsdl-file>
         WEB-INF/wsdl/FileAttachmentService.wsdl
      </wsdl-file>
      <jaxrpc-mapping-file>
         WEB-INF/FileAttachmentService-mapping.xml
      </jaxrpc-mapping-file>
      <port-component>
       <port-component-name>FileAttPort</port-component-name>
       <wsdl-port xmlns:ns2="urn:FileAttachmentService">
         ns2:FileTransferIFPort
       </wsdl-port>
       <service-endpoint-interface>
         filetransfer.FileTransferIF
       </service-endpoint-interface>
       <service-impl-bean>
          <servlet-link>FileAttachmentServlet</servlet-link>
```

```
        </service-impl-bean>
        <handler>
            <handler-name>ServerAttachmentHandler</handler-name>
            <handler-class>
              filetransfer.ServerAttachmentHandler
            </handler-class>
            <init-param>
                <param-name>directory</param-name>
                <param-value>/temp</param-value>
            </init-param>
        </handler>
      </port-component>
  </webservice-description>
</webservices>
```

위와 같은 작업이 끝나고 나면, web.xml 과 jeus-webservices-dd.xml 을 다음과 같이 작성한다.

<< web.xml >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>FileAttachmentServlet</servlet-name>
        <servlet-class>filetransfer.FileTransfer</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>FileAttachmentServlet</servlet-name>
        <url-pattern>/FileAttachmentService</url-pattern>
    </servlet-mapping>
</web-app>
```

<< jeus-webservices-dd.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <service>
        <webservice-description-name>
            FileAttachmentService
        </webservice-description-name>
        <port>
```

```
          <port-component-name>FileAttPort</port-component-name>
        </port>
    </service>
</jeus-webservices-dd>
```

위와 같은 DD 파일의 작성이 끝나면, EAR 로 패키징 하여 JEUS 에 배치 작업을 한다.

이 서비스에 접근할 수 있는 주소는 다음과 같다.

http://localhost:8088/FileAttachmentService/FileAttachmentService

9.2.6.4 웹서비스 클라이언트 핸들러의 작성

클라이언트 핸들러에서는 handleRequest()에서 메시지 컨텍스트를 직접 다루며, SAAJ 를 통하여 SOAP 메시지에 부착물(Attachment)형태로 File 을 첨부하게 된다.

<< *ClientAttachmentHandler.java* >>

```
package filetransfer;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.namespace.QName;
import javax.xml.rpc.JAXRPCException;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;

import javax.xml.rpc.handler.GenericHandler;

public final class ClientAttachmentHandler
  extends GenericHandler
{
  private String getFileName(SOAPMessage request)
```

```
    throws SOAPException
  {
    SOAPBody body =
request.getSOAPPart().getEnvelope().getBody();

    SOAPElement opElem = (SOAPElement)
body.getChildElements().next();

    SOAPElement paramElem = (SOAPElement)
      opElem.getChildElements().next();
    return paramElem.getValue();
  }

  public boolean handleRequest(MessageContext mc) {

    SOAPMessageContext ctx = (SOAPMessageContext) mc;
    SOAPMessage request = ctx.getMessage();

    try {
      String fileName = getFileName(request);
      AttachmentPart part = request.createAttachmentPart();
      part.setContentType("application/x-zip-compressed");
      FileDataSource fds = new FileDataSource(fileName);
      part.setDataHandler(new DataHandler(fds));
      request.addAttachmentPart(part);
    } catch(SOAPException e) {
      e.printStackTrace();
      throw new JAXRPCException(e);
    }
    return true;
    }
    public QName[] getHeaders() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

9.2.6.5 웹서비스 클라이언트의 작성

웹서비스 클라이언트의 작성에서 눈에 띄는 것은 웹서비스 클라이언트 핸들러를 핸들러 레지스트리에 등록하는 것이다. 파일 송신에 관한 구현은 클라이언트 핸들러에서 이루어진다.

<< *Client.java* >>

```
package filetransfer;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import javax.xml.namespace.QName;
import javax.xml.rpc.Service;
import javax.xml.rpc.Stub;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.HandlerRegistry;

import FileAttachmentService_pkg.*;

public final class Client {

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

    Service svc = new FileAttachmentService_Impl();
    HandlerRegistry registry = svc.getHandlerRegistry();
    List list = new ArrayList();
    list.add(new HandlerInfo(ClientAttachmentHandler.class, null,
null));

    QName portName = new QName("","FileTransferIFPort");
    registry.setHandlerChain(portName, list);

     FileTransferIF port =
        (FileAttachmentService_Impl)svc).getFileTransferIFPort();
     String result = port.receiveFile("File_send.txt");
     System.out.println("** File transfer result: "+result);
```

```
   }
}
```

9.2.6.6 실행

위와 같이 구현한 웹서비스를 클라이언트에서 실행하기 위해서는 클라이언트의 스텁 생성이 필요하다.

```
D:\test>ant wsdl2java
```

스텁이 생성되었다면 다음과 같이 실행시킨다.

```
D:\test>ant run
```

그러면 다음과 같은 결과가 나올 것이다.

```
** File transfer result: Received file named: File_send.txt
```

위와 같은 결과가 나온다면 클라이언트에서 File_send.txt 라는 파일이 서버 쪽으로 잘 전달되었고, webservices.xml 에 설정한 대로 /temp 라는 폴더에 File_send.txt 파일이 저장되어 있음을 볼 수 있다.

10 UDDI 이용

10.1 소개

이 장에서는 Universal Description, Discovery and Integration (UDDI)에 대한 개략적 설명과 JEUS 에서의 UDDI 사용방법에 대해서 설명한다. 또한, UDDI 레지스트리를 프로그램적으로 접근할 수 있는 UDDI 클라이트를 설명하며 UDDI 레지스트리에 쉽게 접근 하기 위해 웹기반 사용자 인터페이스에 대해서도 설명한다.

10.2 UDDI 의 개요

10.2.1 소개

Universal Description, Discovery and Integration (UDDI) 스펙은 웹서비스에 대한 정보의 공개 및 검색에 대한 방법을 정의 한다. UDDI 는 Extensible Markup Language (XML)와 Simple Object Access Protocol (SOAP) 같은 기존 표준에 기반하고 있다.

UDDI 는 일반적인 XML 형태로 구현된 비즈니스들과 그것들의 서비스 기술(description)에 대한 분산 레지스트리에 기반한 접근법을 취한다.

UDDI 프로젝트의 핵심 컴포넌트는 UDDI 비지니스의 등록과 비지니스 엔티티(entity)를 기술한 XML 파일과 그 웹서비스이다.

UDDI 는 프로그래밍 모델과 스키마를 제공하고 이것은 레지스트리를 사용하여 규칙을 정의한다. UDDI 스펙의 모든 API 는 XML 로 정의되어 있고, SOAP envelope 으로 래핑(wrapping)되어 있으며, HTTP 를 통해 전송된다.

10.2.2 UDDI 이용

UDDI 는 웹에서 비즈니스 레지스트리의 공유 오퍼레이션을 포함 하고 있다. 대부분의 경우 프로그램과 프로그래머는 서비스의 정보를 위치 시킬 때, 특히 프로그래머의 경우에는 알려진 웹 서비스와의 호환성 있는 시스템을 준비 하거나, 자신의 웹서비스를 다른 사람이 호출 할 수 있게 기술 하는데

UDDI Business Registry 를 사용한다. 이것은 UDDI 비즈니스 레지스트리에 지정된 정보를 사용한다.

UDDI Business Registry 는 Business 레벨에서 사용되며, 주어진 파트너가 특정 웹서비스 인터페이스를 가지고 있는지 확인 하거나, 해당 서비스를 가지고 해당 산업에 있는 회사를 찾거나, 서비스에 대한 기술정보를 얻기 위해 어떻게 파트너가 웹서비스를 노출시켰는지에 대한 정보를 위치시킨다.

UDDI Data 구조

UDDI Registry 에서 사용하는 핵심 정보 모델은 XML 스키마로 정의된다. UDDI XML 스키마는 4 개의 핵심 타입의 정보로 구성되어 있으며 기술자들이 파트너의 웹서비스들을 사용하기 위하여 알아야 하는 정보를 제공한다. 이 4 가지는 시비스의 스펙에 대한 비즈니스 정보 (<businessEntity>), 서비스 정보 (<businessService>), 바인딩 정보 (<bindingTemplate>) 그리고 기술 모델 정보(<tModel>) 이다.

웹서비스에 대한 정보를 기술 하거나 검색할 때 사용되는 정보 계층 구조와 XML 태그를 아래 그림에서 보여주고 있다[그림 8].

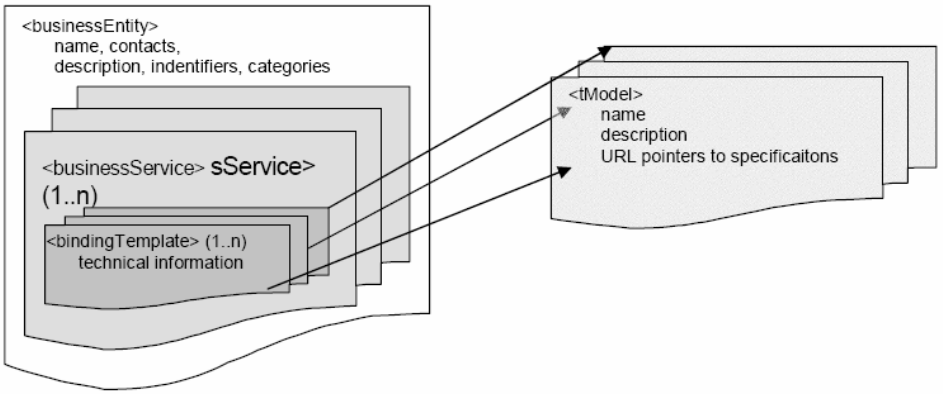


그림 8 정보 계층과 주요 XML 태그 이름

- 비즈니스 엔티티(businessEntity) : 웹 서비스를 제공하는 비즈니스 혹은 기관을 기술한다. 이 엔티티는 비지니스의 기본 정보들 즉, 이름, 수행하고 있는 비지니스 설명, 연락처 등을 나타낸다. Dun & Bradstreet D-U-N-S® Number 같은 비지니스용 식별자들도 포함되어 있다.

- 비지니스 서비스 (businessService) : 비즈니스 엔티티에 의해 기술된 기관이 제공하는 관련 웹서비스의 그룹을 기술한다. 이것은 하나의 비즈니스 엔티티의 논리적 자식이다.

- 서비스 바인딩 (bindingTemplate) : 특정 웹 서비스를 이용하는 데 필요한 기술 모델을 기술한다. 이 엔티티는 서비스와 관련된 바인딩 정보와 그러한 서비스들이 구현한 기술 스팩에 대한 레퍼런스를 제공하고 다양한 파일과 URL 기반의 발견 매커니즘에 대한 포인터도 제공한다.

- 기술 모델 (tModel) : 웹 서비스 유형, 웹 서비스에서 사용하는 프로토콜, 분류 시스템과 같은 재사용가능한 개념을 나타내는 "technical model"을 기술한다. 이 기술 모델은 비즈니스 간에 주고받는 표준, 웹 서비스 인터페이스 정의 같은 모든 종류의 것을 포함한다. 그래서 새로운 웹 서비스를 구성하여 WSDL 문서를 기술하였다면, 이 WSDL 정의는 기술 모델에 저장된다.

10.3 JEUS 에서 UDDI Server 운영

10.3.1 소개

JEUS 의 UDDI 는 UDDI 레지스트리 서버와 UDDI 클라이언트 라이브러리, 웹기반 사용자 인터페이스로 구성되어 있다. JEUS 의 UDDI 는 UDDI 버전 2.0 과 버전 3.0 의 데이터 구조와 프로그래밍 API 를 지원한다.

이 절에서는 JEUS 에 UDDI module 을 deploy 하는 방법과 JEUS UDDI 서버의 기동에 대해 설명 하도록 하겠다.

- JEUS UDDI DataStore 를 생성하는 방법
- JEUS UDDI 서버를 deploy 하는 방법
- JEUS UDDI 서버를 구성하고 설정 하는 방법

JEUS UDDI 패키지의 핵심 Library 는 JEUS_HOME\lib\system 디렉토리에 위치하며 Web 인터페이스를 포함한 Web Context 에 관련된 모듈은 EAR 포맷으로 제공된다.

10.3.2 UDDI DataStore 생성

JEUS 의 UDDI 서버는 데이터 영속성을 위해서 관계형 데이터베이스가 필요하다. JEUS UDDI 서버는 ANSI 표준 SQL 을 지원하는 어떤 관계형 데이터베이스라도 사용할 수 있다.

JEUS UDDI DataStore 는 JDBC 를 사용하며 설정 과정은 간단하다. 첫번째로, JEUS UDDI 서버 패키지와 함께 제공되는 데이터베이스 스크립트를 사용하여 새로운 UDDI database 를 생성한다.(데이터베이스 스크립트는 JEUS_HOME\webhome\uddi\dbscripts 디렉토리에 위치 되어 있다.)

DataStore 의 구성설정을 하기 위해서는, JEUS 에서 'uddiDB'라는 Datasource 를 설정해야 한다. 예제 파일이 JEUS_HOME\webhome\uddi\jeusdd 디렉토리에 있으므로 참고 하기 바란다.

10.3.3 JEUS UDDI 서버 Deploying

UDDI database 구성 설정이 완료 되었다면, JEUS UDDI 서버를 deploy 해야 한다. JEUS UDDI 서버 EAR 패키지는 JEUS 가 설치 될 때 JEUS_HOME\webhome\uddi 디렉토리에 생성된다.(파일이름은 jeusuddi_v2c.ear 과 jeusuddi_v3c.ear 이다.)

JEUS UDDI 서버를 deploy 하는 것은 다른 ear application 을 deploy 하는 과정과 같다. 이 매뉴얼에서는 context 이름이 'uddi' 로 UDDI 가 deploy 되는것을 가정한다.

특히, 다음 3 개의 구성 설정을 자세히 살펴 보기 바란다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    <node>
                ...
    </node>
    <resource>
        <data-source>
            <database>
                  <vendor>Vender</vendor>
                  <export-name>UDDIDB</export-name>
                  ...
            </database>
        </data-source>
    </resource>
</jeus-system>
```

<<jeus-web-dd.xml>>

```
<jeus-web-dd>
        <res-ref>
            <jndi-info>
                <ref-name>jdbc/uddiDB</ref-name>
                <export-name>UDDIDB</export-name>
            </jndi-info>
        </res-ref>
</jeus-web-dd>
```

<<web.xml>>

```
<web-app>
    ...
    <resource-ref>
        <res-ref-name>jdbc/uddiDB</res-ref-name>
        ...
    </resource-ref>
</web-app>
```

JNDI Datasource 이름인 'jdbc/uddiDB' 는 디폴트 값으로 사용되는 JNDI Datasource 이름이다. 다른 JNDI Datasource 이름을 사용하고자 한다면 이 구성 설정 파일들을 수정하면 된다.

10.3.4 JEUS UDDI 서버의 구성 설정

JEUS UDDI 서버를 구성 하기 위해서는 있는 설정 파일 'uddi.properties'를 수정해야 한다. 설정 파일 'uddi.properties'은 'JEUS_HOME\lib\application'디렉토리에 위치한다. 만약 이 설정 파일 존재하지 않으면 Deploy 시에 기본 설정 파일을 자동으로 생성한다.

다음은 'uddi.properties' 파일의 기본 설정에 대한 설명이다.

표 4. uddi.properties

UDDI Property Key	설명
uddi.operatorName	UDDI 오퍼레이터 이름

UDDI Property Key	설명
uddi.operatorURL	UDDI 오퍼레이터 싸이트 URL
uddi.auth	현재 사용중인 UDDI 인증 모듈. 디폴트 UDDI 인증자는 'jeus.webservices.uddi.auth.DefaultAuthenticator' 이다.
uddi.dataSource	현재 사용중인 UDDI DataStore 모듈. 디폴트 UDDI DataStore 는 'java:comp/env/jdbc/uddiDB' 이다.

10.3.5 새로운 user 의 추가

UDDI 레지스트리에 UDDI 데이터를 공개하기 위해서는 UDDI 레지스트리의 인증을 얻어야한다.

JEUS UDDI 레지스트리의 publisher 의 인증은 2 단계 과정이 있다. 첫번째 단계는 사용자의 ID/password 를 통해 사용자를 인증하는 것이다. JEUS UDDI 레지스트리는 기본 인증 모듈인 'jeus.webservices.uddi.auth.DefaultAuthenticator'를 가지고 있다. 기본적인 인증 모듈은 인증시도에 대해 간단하게 승인한다. 이것은 단순히 UDDI 사용자의 userid 를 인증할 뿐이다. JEUS UDDI 레지스트리는 추가적인 인증 모듈인 'jeus.webservices.uddi.auth.XMLDocAuthenticator' 을 제공한다. 이것은 XML 도큐먼트를 바탕으로 하며, 'JEUS_HOME\lib\application' 에 있는 'uddi-user.xml'를 추가하거나 수정해야 한다. 다음은 예제 XML 도규먼트이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<uddi-users>
  <user userid="administrator" password="jeusadmin" />
  <user userid="tmaxsoft" password="password" />
  <user userid="jeus" password="password" />
  <user userid="webservices" password="password" />
</uddi-users>
```

개발자는 Custom 인증 모듈을 개발할 수 있다. 전형적으로 UDDI Registry 는 외부의 인증 메커니즘을 사용할 수 있음을 의미한다. 추가적인 JEUS UDDI 인증 모듈은 JEUS UDDI 사용자가 특정 환경에서 어떻게 인증이 될지 결정

함에 따라서 JEUS UDDI 사용자에 의해 개발될 수 있다. 다음 코드는 custom JEUS UDDI 인증 모듈을 개발 하는 방법에 대한 예제이다.

```
import jeus.webservices.uddi.auth.BaseAuthenticator;
import org.apache.juddi.error.RegistryException;

public class SampleAuthenticator implements BaseAuthenticator {

    public SampleAuthenticator() {
    }

   public String authenticate(String userID, String credential)
            throws RegistryException {
         // TODO Part implemented by developer.
         return userID;
     }
}
```

두번째 단계는 인증된 사용자를 UDDI 레지스트리 사용자로 등록하는 것이다. 등록하는 방법에는 UDDI 웹 사용자 인터페이스를 이용하는 방법과 DataStore 에 직접 등록하는 방법이 있다. 이 절에서는 DataStore 에 직접 등록하는 방법만을 설명하고 UDDI 웹 인터페이스를 이용하는 방법은 JEUS UDDI 웹 사용자 인터페이스인 JEUS UDDI Explorer 사용법에서 설명하도록 하겠다.

UDDI 레지스트리의 사용자인 Publisher 가 UDDI DataStore 에 정의 되었는지 확인한다. Publisher 는 UDDI DB 의 PUBLISHER 테이블에 Publisher 를 식별하는 Row 가 있을 때 정의된다. SQL 을 사용하여 사용자를 정의할 수 있다. 다음은 ‘jeus’라는 새로운 publisher 를 정의하는 예이다.

```
INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,ADMIN) VALUES
('jeus','JEUS','false');
```

Column 이름	설명
PUBLISHER_ID	Publisher 의 user ID 는 인증시 사용된다. **중요:** 이것은 외부의 인증 서비스를 통하여 인증 할 때 사용되는 값과 동일해야 한다.
PUBLISHER_NAME	Publisher 의 이름 (또는 UDDI 안에 검증된 이름.)
ADMIN	Publisher 가 관리적인 권한을 가졌는지 여부를 나타낸다.이 column 의 값은 'true' 혹은 'false' 이다.

10.3.6 JEUS UDDI 서버 실행

JEUS Server 를 기동시킨다. UDDI 서버의 URL 은 다음과 같은 형태이다.

```
http://hostname:port/context/
```

기본적으로 JEUS UDDI 서버의 context path 는 '/uddi'이다. 여기서 JEUS HTTP Listener 의 기본 포트값 8088 을 사용한다고 가정한다면 URL 은 다음과 같다.

```
http://localhost:8088/uddi/
```

웹 브라우저에서 이 URL 을 호출 하였다면, 다음 페이지을 볼 수 있을 것이다*[그림 9]*.

그림 9. JEUS UDDI Registry 초기 화면

10.4 JEUS 서버에서의 JEUS UDDI Explorer 의 사용

10.4.1 소개

이 절에서는 JEUS UDDI Explorer 웹 사용자 인터페이스를 어떻게 사용하는지에 대해서 설명하도록 하겠다. JEUS UDDI Explorer 은 UDDI 레지스트리의 UDDI 데이터의 등록, 수정, 질의, 삭제 등의 작업을 하기 위한 웹 어플리케이션 이다. JEUS UDDI Explorer 는 다음과 같은 기능들을 제공한다.

- Public 과 private UDDI 레지스트리에 대한 Querying.
- Private UDDI 레지스트리에 대한 Publish.
- JEUS UDDI Explorer 의 설정.

10.4.2 UDDI 레지스트리 Querying.

JEUS UDDI Explorer 의 Inquiry 를 이용하면 이름으로 Business, Service,Technical Model 을 검색할 수 있다. 'UDDI Registry'를 사용해서 UDDI 레지스트리 오퍼레이터를 선택 할 수 있고, 'Search For a' 에서 Entity 타입을 선택 할 수 있다. UDDI Registry 와 Entity 타입을 선택한 후, 'Starting with'의 텍스트 필드에 찾고자 하는 이름의 일부 혹은 이름 전체를 기입한다. 와일드 카드로 '%' 심볼을 이용할 수 있다. 또한 'Find Qualifier' 를 사용해서 결과를 필터링 할 수 있다.

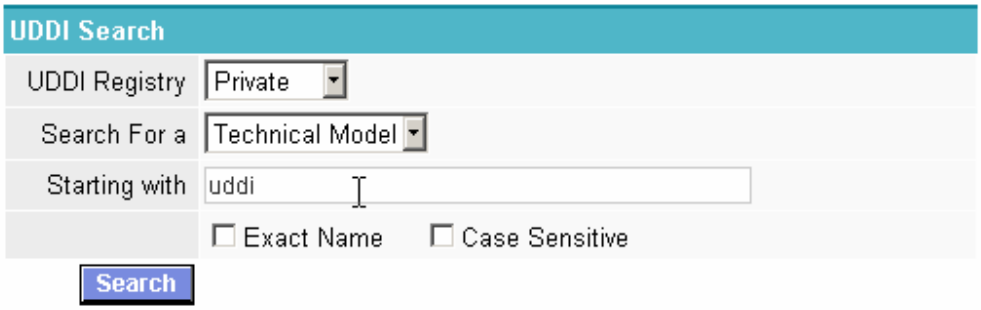


그림 10. UDDI Search 입력 폼 화면

찾고자 하는 이름을 입력후 'Search' 버튼을 누르면 검색을 시작한다. 선택된 UDDI Registry 는 사용자가 입력한 이름으로 시작하는 Businesses, Services 혹은 Technical Model 들의 목록을 보여준다. JEUS UDDI Explorer 는 찾은 Entity 에 대한 상세 내역을 보여준다.

Find Technical Model Results

[Technical Model(s)]

Technical Model Name	Description	Overview URL
uddi-org:fax	Fax-based web service	http://www.uddi.org/taxonomies/UDDI_CoreOther_tModels.htm#overFAX
uddi-org:ftp	File transfer protocol (FTP) based web service	http://www.uddi.org/taxonomies/UDDI_CoreOther_tModels.htm#overFTP
uddi-org:general_keywords	Special taxonomy consisting of namespace identifiers and the keywords associated with the namespaces	http://www.uddi.org/taxonomies/UDDI_Taxonomy_tModels.htm#GenKW
uddi-org:homepage	HTTP Web Home Page URL	http://www.uddi.org/taxonomies/UDDI_CoreOther_tModels.htm#Homepage
uddi-org:http	An http or web browser based web service	http://www.uddi.org/taxonomies/UDDI_CoreOther_tModels.htm#overHTTP
uddi-org:inquiry	UDDI Inquiry API - Core Specification	http://www.uddi.org/wsdl/inquire_v1.wsdl
uddi-org:inquiry_v2	UDDI Inquiry API Version 2 - Core Specification	http://www.uddi.org/wsdl/inquire_v2.wsdl

그림 11. UDDI Registry 에서 찾은 결과 화면.

10.4.3 UDDI Registry 공개

UDDI Registry 에 publish 하기위해서는 사용자 ID 와 비밀번호를 가지고 UDDI 레지스트리에 로그인 해야 한다. 사용자가 접속한 후, JEUS UDDI Explorer 는 사용자가 등록한 Business 들의 목록과 등록된 Service 들 그리고 Technical Model 들을 보여준다. 만약 '[+]' 심볼을 클릭 하면,등록된 서비스

들이 JEUS UDDI Explorer 에 출력된다. Service 는 항상 business 와 연관되어 있다.

‘Add a new Business’,’Add a new Service’, ‘Add a new Technical Model’을 사용해서 Business, Service, Technical Model 을 추가 할 수 있다. 만약 등록된 레지스트리를 삭제 혹은 수정할 때에는 ‘Edit’ 그리고 ‘Delete’ 링크를 사용한다.

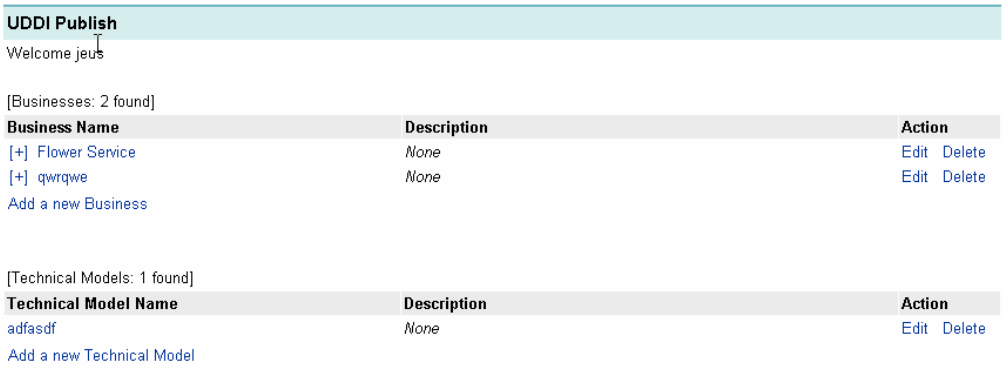


그림 12. 등록한 entry 화면

- business 추가

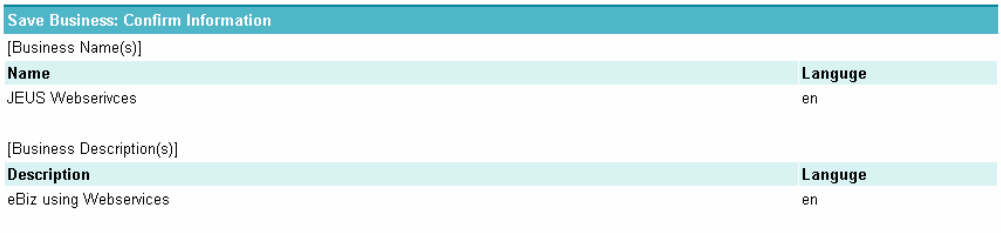
1. [Add a new Business] 링크를 클릭한다. 사용자가 business name 을 입력할 수 있는 화면이 나온다. 예제 business 인 "JEUS Webservices"는 다음과 같이 입력 하면 된다.

   Name: JEUS Webservices

2. [Continue] 클릭 한다. Business 를 출력한다. 이 화면에서는 business 설명이나 사용자의 연락처, Discovery URL, category 그리고 식별자 같은 부가 정보를 추가 할 수 있다. 예제에서는 business 의 설명으로 " eBiz using Webservices” 를 입력한다.

   Description: eBiz using Webservices

3. [Continue] 와 [Save]를 클릭해서 Business 를 저장된다.


Save Business: Confirm Information

[Business Name(s)]

Name	Languge
JEUS Webserivces	en

[Business Description(s)]

Description	Languge
eBiz using Webservices	en

그림 13. 저장된 business 결과 화면

- 서비스 추가

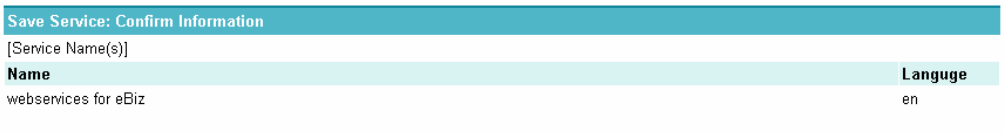
1. Business 와 관련된 [+] 링크 클릭한다. 서비스들은 항상 Business 와 연관된다. 그러면 확장되면서 Business 와 연관된 Service 가 호출된다.

2. [Add a new Service] 링크 클릭한다. 사용자가 business name 을 제공하는 것을 보여주는 화면이다. 예제 서비스 "Webservices for eBiz"는 아래 처럼 입력 한다.

   Name: Webservices for eBiz

3. [Continue] 클릭 한다. 서비스의 이름을 보여준다. 서비스의 설명, Access Point, Category 같은 부가 정보를 추가 할 수 있다.

4. [Continue]와 [Save] 클릭한다.


Save Service: Confirm Information

[Service Name(s)]

Name	Languge
webservices for eBiz	en

그림 14. 서비스 저장 결과 화면

- tModel 추가

1. [Add a new Technical Model] 링크를 클릭 한다. 이 페이지는 사용자의 테크니컬 모델 name 이 제공되는 화면이다. 예제 business "tModel for eBiz"은 다음과 같이 입력한다.

   Name: tModel for eBiz

2. [Continue]를 클릭 한다. 사용자의 테크니컬 모델의 name 을 보여준다. 부가적인 정보인 테크니컬 모델에 대한 설명, Overview URL, Category 그리고 Identifier 을 추가 할 수 있다.

3. [Add a new Overview URL] 링크 클릭. 사용자의 테크니컬 모델에 대한 Overview Doc 정보를 제공하는 화면을 보여준다. OverviewURL 샘플 “http://localhost:8088/ws/tmodel.wsdl“은 아래와 같이 입력한다.

   OverviewURL: http://localhost:8088/ws/tmodel.wsdl

4. [Add] 링크를 클릭 한다. 사용자의 테크니컬 모델에 대한 OverviewDoc 을 보여준다.

5. 순서에 맞게 [Add] 링크 , [Continue] 링크, [Save] 링크를 클릭 한다. 테크니컬 모델이 저장된다.

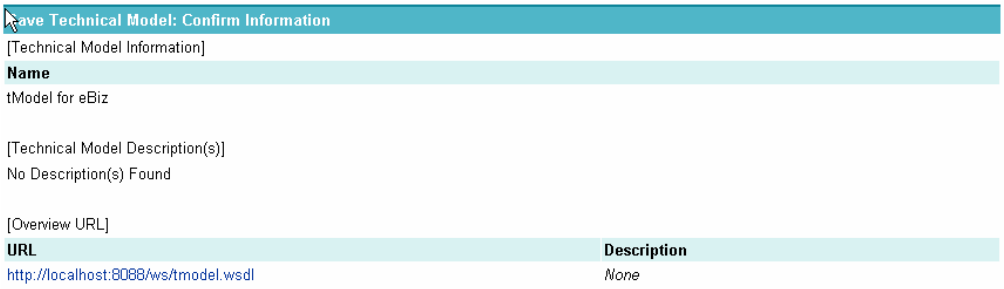


그림 15. 테크니컬 모델 저장 결과 화면.

- bindingTemplate 추가

1. business 와 관련되는 [+]를 클릭한다. Business 에 연관된 Service 의 [Edit]을 클릭한다. 그러면 Service 에 대한 자세한 내용을 보여준다.

2. bindingTemplate 을 추가 하려면 [Add a new Access Point] 링크를 클릭 한다. Access Point 에 대한 주소가 제공되는 화면을 보여준다. Acess Point 의 샘플 주소 "http://localhost:8088/uddi" 는 다음과 같이 입력한다.

   Address: http://localhost:8088/uddi

3. [Add]링크를 클릭 한다. Service 와 테크니컬 모델을 바인드할 수 있는 화면을 보여준다.

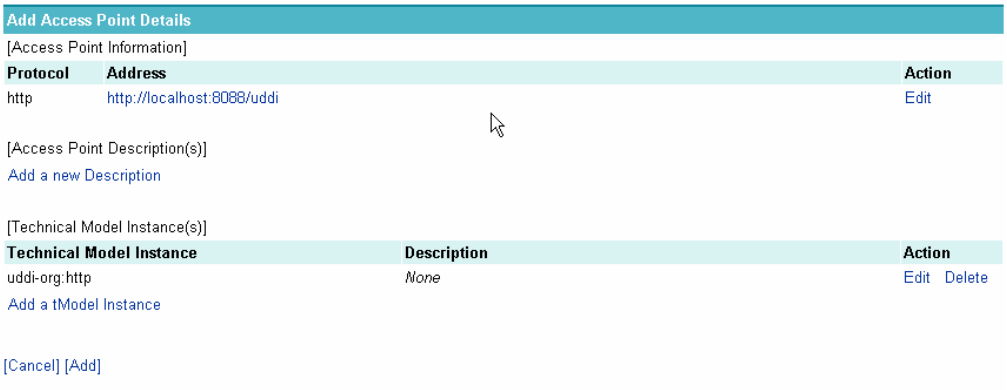


그림 16. bindingTemplate 추가 화면

4. [Add a tModel Instance] 링크를 클릭 한다. 사용자가 공개된 tModel 들을 찾을 수 있는 화면을 보여준다. 찾고자 하는 tModel 의 전체 이름이나 처음 몇자를 'Starting with'에 입력하여 찾을 수 있다. 예제에서는 name 을 다음과 같이 입력한다.

   Starting with: tModel

5. 'tModel for eBiz' 를 선택해서 [Select] 를 클릭한다.

6. [Add] 를 클릭하고, 한번 더 [Add]를 클릭 한다.

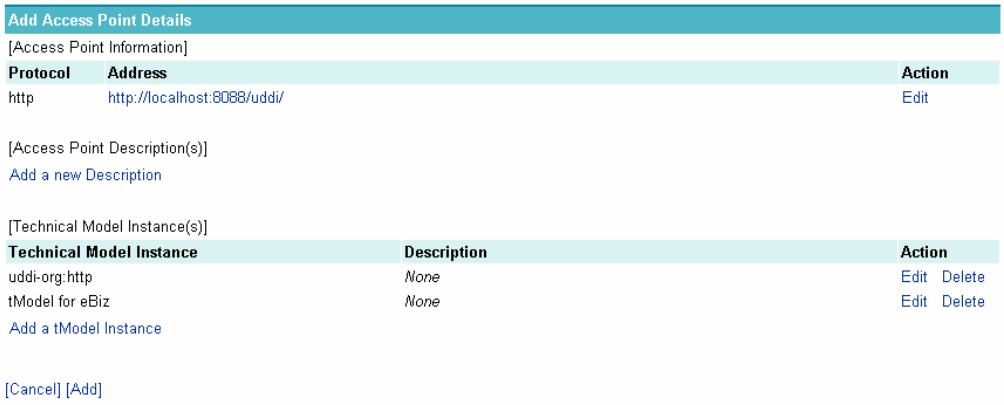


그림 17. bindingTemplate 저장 결과 화면

7. [Change] 링크 와 [Save]링크를 클릭 한다. 테크니컬 모델을 바운드 한 서비스가 저장된다.

10.4.4 JEUS UDDI Explorer 설정

JEUS UDDI Explorer 의 설정을 사용하려면 사용자의 private JEUS UDDI 레지스트리 혹은 이용 가능한 public UDDI 레지스트리의 목록을 수정할 수 있다. 첫째로 관리자 권한을 가진 사용자로 로그인을 해야 한다. 로깅한 후에 메뉴에서 'Setup'링크를 선택한다. JEUS UDDI Explorer 는 아래 화면을 보여준다. 이 화면에서 사용자는 새로운 public UDDI 레지스트리를 추가 할 수 있고 또한 등록된 UDDI 레지스트리에 대해서 수정 혹은 삭제를 할 수 있다.

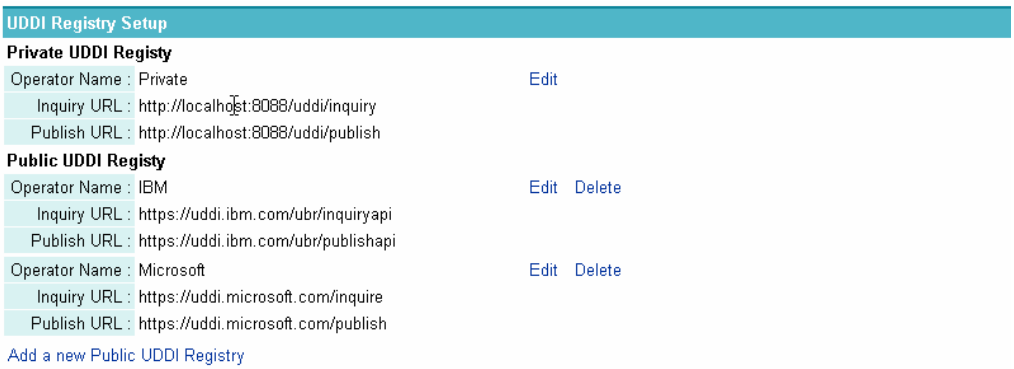


그림 18. UDDI 레지스트리 구성 페이지.

10.4.5 JEUS UDDI Explorer 에서의 사용자 관리

JEUS UDDI Explorer 에서 JEUS UDDI 레지스트리를 사용하기 위한 사용자를 등록할 수 있다. 우선, 관리자 권한을 가진 사용자로 로그인을 해야 한다. 로깅한 후에 메뉴에서 'User'링크를 선택한다. JEUS UDDI Explorer 는 아래 화면을 보여준다. 이 화면에서 관리자는 새로운 사용자를 추가 할 수 있고 또한 등록된 사용자의 정보를 수정하거나 기존 사용자를 삭제를 할 수 있다.

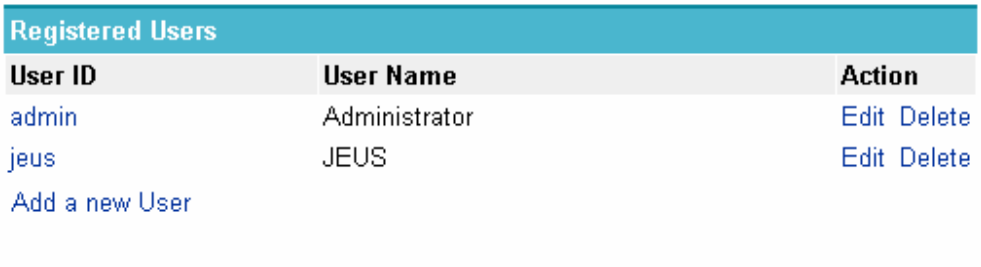


그림 19. JEUS UDDI Explorer 사용자 관리 화면

[Add a new User] 링크를 클릭하면 아래와 같은 신규 사용자 등록 화면을 보여준다. 여기서 UserID 와 User Name 은 필수 입력 사항이며, UserID 가 JEUS

UDDI Explorer 에 로깅과 UDDI Data 를 공개하기 위한 사용자 ID 이다. 필수 입력 값을 입력 후에 [Add]를 클릭하면 사용자가 등록된다.

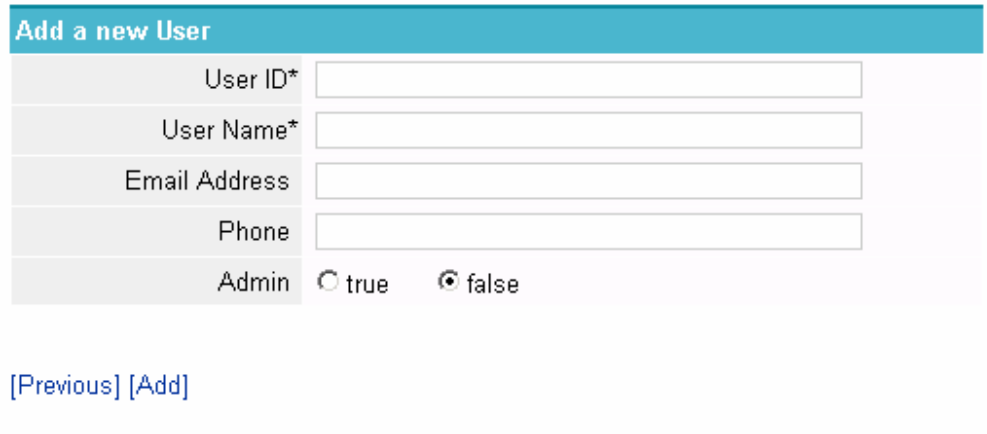


그림 20 JEUS UDDI Explorer 사용자 등록 화면

10.5 UDDI 클라이언트 생성

10.5.1 소개

JEUS UDDI 클라이언트 라이브러리는 UDDI 레지스트리에 대한 자바 인터페이스를 제공한다. 이를 통해 구축한 웹서비스들을 공개할 수 있고, 필요한 웹서비스를 검색할 수 있다. JEUS UDDI 클라이언트 라이브러리는 일종의 JAVA 클래스 라이브러리이며, UDDI 레지스트리와 상호 작용하는 API 를 제공한다

이번 절에서는 UDDI 클라이언트 라이브러리를 사용하여서 UDDI 클라이언트를 어떻게 작성, 컴파일, 실행하는지 보여준다. 제시된 예제는 UDDI V2.0 기반의 UDDI 클라이언트이며, UDDI V3.0 기반의 UDDI 클라이언트도 유사한 방법으로 사용할 수 있다.

10.5.2 UDDI 클라이언트의 작성

이번 절에서는 UDDI 클라이언트 라이브러리를 사용하여서 UDDI 클라이언트를 어떻게 작성하는지 보여준다.

- UDDIClient Class

이 클래스는 UDDI 클라이언트 애플리케이션의 핵심 클래스이다. 이것은 UDDI 레지스트리에 연결하고, 질의를 수행하며, 결과를 처리하는 모든 메쏘드를 가지고 있다. 여기에 제시된 예제 애플리케이션에서, UDDIClient 는

UDDI 레지스트리와 상호 작용을 하기 위하여 사용하는 것이다. 아래 예제 코드는 어떻게 객체를 생성하고 UDDI 레지스트리를 참조하는지는 보여준다.

```
UDDIClient client = new UDDIClient();
client.setInquiryURL("http://localhost:8088/uddi/inquiry");
client.setPublishURL("http://localhost:8088/uddi/publish");
```

- UDDI 레지스트리에서 비즈니스 질의하기

```
Vector inNames = new Vector();
inNames.add(new Name("test_Biz"));

BusinessList list = client.find_business(null, inNames, null,
null, null, null, 0);
```

find_business 메쏘드는 다수의 매개 변수를 갖는다. 이중 2 번째 매개 변수는 검색어이고, 7 번째 매개 변수는 조건에 부합하는 리턴값의 최대 개수이다. (0 은 조건에 부합하는 모든 값을 리턴함을 의미한다.) 보다 상세한 검색을 하기 위해서 식별자(identifiers), 분류자(categories), URL, 기술 모델(tModel) 등을 검색 조건에 넣을 수 있다.

위 방법을 통해 조건에 부합하는 비즈니스 리스트를 얻었다. 아래의 코드는 리스트에 포함된 비즈니스들의 이름을 출력하는 예제 코드이다.

```
BusinessInfos infos = list.getBusinessInfos();
Vector businesses = infos.getBusinessInfoVector();

if (businesses != null) {
    for (int i = 0; i < businesses.size(); i++) {
        BusinessInfo info =
(BusinessInfo)businesses.elementAt(i);
        Vector outNames = info.getNameVector();
        for (int j = 0; j < outNames.size(); j++) {
            Name name = (Name)outNames.elementAt(j);
            System.out.println(name.getValue());
```

```
        }
    }
}
```

- UDDI 레지스트리에서 비즈니스 공개하기

새로운 UDDI 데이터를 UDDI 레지스트리에 공개하기 위해서는 그 UDDI 레지스트리로부터 인증을 얻어야 한다. 아래 코드는 UDDI 레지스트리로부터 인증을 얻는 코드이다.

```
AuthToken authToken = client.get_authToken(“userID”, “password”);
```

다음 단계는 새로운 비즈니스 엔티티(BusinessEntity)를 생성하고 이것을 인증을 얻은 UDDI 레지스트리에 공개하는 것이다. 예제에서는 단순하게 비즈니스 이름만을 정의하였다.

```
BusinessEntity businessEntity = new BusinessEntity();
businessEntity.setBusinessKey("");
businessEntity.addName(new Name("TmaxSoft", "en"));

Vector businessVector = new Vector();
businessVector.add(businessEntity);

BusinessDetail detail =
 client.save_business(authToken.getAuthInfoString(),
 businessVector);
```

- UDDI 레지스트리에서 비즈니스 삭제하기

```
Vector inNames = new Vector();
inNames.add(new Name("TmaxSoft"));
```

```
BusinessList list = client.find_business(null, inNames, null,
  null, null, null, 0);
```

위의 방법을 통해서 공개된 비즈니스 리스트를 얻었다. 이제 각각의 비즈니스를 삭제할 것이다. 아래 코드는 이에 대한 예제 코드이다.

```
Vector businessInfoVector =
 list.getBusinessInfos().getBusinessInfoVector();
for (int i = 0; i < businessInfoVector.size(); i++) {
    BusinessInfo info =
       (BusinessInfo)businessInfoVector.elementAt(i);
    DispositionReport dispositionReport =
       client.delete_business(authToken.getAuthInfoString(),
       info.getBusinessKey());
}
```

10.5.3 UDDI 클라이언트의 컴파일

이번 절에서는 작성된 UDDI 클라이언트를 어떻게 컴파일하는지 보여준다.

UDDI 클라이언트 코드의 컴파일을 위해서 클라이언트 소스 코드가 있는 디렉토리로 간다. 클라이언트 소스 코드를 컴파일 할 때 JEUS_HOME\lib\system 디렉토리의 jeus.jar 파일을 컴파일러의 클래스패스로 잡아야 한다. 여기 예제에서는 C:\Jeus\samples\manual_examples\webservice\10_uddi\client 디렉토리에 가서 아래의 명령을 실행한다.

```
c:\Jeus\samples\manual_examples\webservice\10_uddi\client>
javac -classpath %JEUS_HOME%\lib\system\jeus.jar;
 FindBusinessSample.java
```

Ant tool 을 이용하여 컴파일할 수도 있다.

c:\Jeus\samples\manual_examples\webservice\10_uddi\client 디렉토리에 컴파일하기 위한 build.xml 파일이 있다. 이것을 아래와 같이 실행한다.

```
c:\Jeus\samples\manual_examples\webservice\10_uddi\client>
```

```
ant –Dsrc=FindBusinessSample compile
```

10.5.4 UDDI 클라이언트의 실행

예제에서는 C:\Jeus\samples\manual_examples\webservice\10_uddi\client 디렉토리에 가서 아래의 명령어를 실행한다.

```
c:\Jeus\samples\manual_examples\webservice\10_uddi\client>
java –classpath .;%JEUS_HOME%\lib\system\jeus.jar;%JEUS_HOME%\lib
\system\jaxb-api.jar;%JEUS_HOME%\lib\system\jaxb-libs.ja
r;%JEUS_HOME%\lib\system\jaxb-impl.jar;%JEUS_HOME%\lib\system
\relaxngDatatype.jar;%JEUS_HOME%\lib\system\xsdlib.jar
FindBusinessSample
```

Ant tool 를 이용하여 실행할 수도 있다. 아래와 같이 실행한다.

```
c:\Jeus\samples\manual_examples\webservice\10_uddi\client>
ant –Dsrc=FindBusinessSample run
```

실행결과는 아래와 유사할 것이다.

Running FindBusinessSample

Found Business name: test_Biz

Done

10.6 결론

지금까지 JEUS 서버에서 Universal Description, Discovery and Integration (UDDI)를 어떻게 사용하는지에 대해서 설명하였다. JEUS UDDI 는 UDDI 레지스트리 서버, 웹 유저 인터페이스 기반의 JEUS UDDI Explorer 그리고 UDDI 클라이언트 라이브러리로 구성 된다.

이 장에서는 UDDI 의 기본적인 개념에 대해서 알아 보았다. 그리고 JEUS UDDI 서버의 구성 설정 과정과 JEUS UDDI Explorer 의 사용에 대해서 화면

을 보여주면서 설명하였으며, UDDI 클라이언트 라이브러리를 사용하여 UDDI 클라이언트 애플리케이션 작성 방법을 설명하였다.

11 웹 서비스 DD(webservices.xml)의 작성

J2EE 웹 서비스는 webservices.xml 이라는 이름을 가진 웹 서비스 서술자를 자바 클래스 엔드포인트나 EJB 엔드포인트를 포함하는 압축 파일(WAR 나 JAR)에 포함할 것을 요구한다. webservices.xml 을 EJB 엔드 포인트의 경우에는 EJB JAR 파일의 META-INF 디렉토리 내에 두고, 서블릿 엔드포인트의 경우에는 WAR 파일의 WEB-INF 디렉토리 내에 둔다. 다음은 FileAttachmentService 라는 웹 서비스의 webservices.xml 파일을 보여준다.

<< *webservices.xml* >>

```
<?xml version="1.0"?>
<webservices version="1.1"
    xmlns="http://java.sun.com/xml/ns/j2ee">
  <webservice-description>
    <webservice-description-name>
        FileAttachmentService
    </webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/FileAttachmentService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>
        WEB-INF/FileAttachmentService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component>
        <port-component-name>FileAttPort</port-component-name>
        <wsdl-port xmlns:ns2="urn:FileAttachmentService">
            ns2:FileTransferIFPort
        </wsdl-port>
        <service-endpoint-interface>
            filetransfer.FileTransferIF
        </service-endpoint-interface>
        <service-impl-bean>
            <servlet-link>FileAttachmentServlet</servlet-link>
        </service-impl-bean>
```

```
      <handler>
         <handler-name>ServerAttachmentHandler</handler-name>
         <handler-class>
            filetransfer.ServerAttachmentHandler
         </handler-class>
         <init-param>
            <param-name>directory</param-name>
            <param-value>/temp</param-value>
         </init-param>
      </handler>
   </port-component>
 </webservice-description>
</webservices>
```

webservices.xml 파일의 루트 엘리먼트는 webservices 엘리먼트이고, webservice-description 엘리먼트를 하나 혹은 그 이상을 필수 요소로 가진다.

webservice-description 엘리먼트는 동일한 WSDL 을 사용하는 자바 클래스나 EJB 엔드포인트의 집합을 서술한다. 즉 패키징 내의 다른 WSDL 각각의 파일 들에 대해서 webservice-description 엘리먼트가 하나씩 존재해야 한다. 예를 들면, 두 개의 다른 자바 클래스 엔드포인트가 각각 WSDL 을 별도로 가지고 하나의 WAR 파일 안에 존재한다면 각각의 JSE 를 서술하기 위해서 webservice-description 엘리먼트는 두 개가 존재해야 한다.

webservice-description 엘리먼트는 J2EE 엔드포인트를 WSDL 포트 정의, DD 구현, JAX-RPC 매핑 파일, 그리고 엔드포인트 인터페이스로 바인딩 시켜주는 역할을 하며, 이 장에서는 이들의 관계와 필요성에 대해서 설명한다.

11.1wsdl-file 엘리먼트

wsdl-file 엘리먼트는 WSDL 문서의 WAR 나 EJB JAR 파일 내에서의 상대적인 위치를 나타내며, webservices.xml 파일은 WSDL 파일과 같은 WAR 나 JAR 파일 내에 위치해야 한다.

11.2 jaxrpc-mapping-file 엘리먼트

jaxrpc-mapping-file 엘리먼트는 JAX-RPC 매핑 파일의 위치를 지정한다. JAX-RPC 매핑파일은 WSDL 파일과 J2EE 엔드포인트 간의 매핑을 정의한다. JAX-RPC 매핑 파일에 관한 더 자세한 설명은 12 장을 참조한다.

11.3 port-component 엘리먼트

port-component 는 특정한 자바 클래스나 EJB 엔드포인트를 WSDL 문서 내의 특정한 port 엘리먼트로 매핑하는 역할을 한다. 자바 클래스의 경우에 WSDL port 정의와 binding 정의는 웹서비스를 호스트 하기 위해 필요한 서블릿을 생성하는데 사용된다. EJB 엔드포인트의 경우에는 port 와 binding 정의는 EJB 컨테이너가 SOAP 메시지를 무상태 빈 객체에 보내기 위해 마샬링 하는 데 사용된다.

11.3.1 port-component-name 엘리먼트

port-component-name 엘리먼트는 특정한 자바 클래스나 EJB 엔드포인트를 지정하기 위한 이름을 제공한다. 이 이름은 webservices.xml 파일 안에서 유일해야한다.

11.3.2 service-endpoint-interface 엘리먼트

자바 클래스를 백엔드로 가지는 웹서비스의 경우, 서비스 엔드포인트 인터페이스의 이름을 지정하는 엘리먼트이다. EJB 엔드포인트 웹서비스의 경우에도 마찬가지로 이 엘리먼트에서 정의하며, 추가적으로 ejb-jar.xml 의 service-endpoint 에서도 같은 이름으로 정의 해야 한다.

11.3.3 service-impl-bean 엘리먼트

service-impl-bean 엘리먼트는 서비스 배치가 수행되는 시점에 어떤 서비스의 로직 구현 정의가 J2EE 엔드포인트가 될 것인지를 정의한다. 자바 클래스의 경우에는 web.xml 의 servlet 정의를 가리키고, EJB 엔드포인트의 경우에는 ejb-jar.xml 의 session 정의를 가리킨다.

JSE 의 경우에는 servlet-link 엘리먼트에 정의 되며 예는 다음과 같다.

<<DocLitEchoService 의 webservices.xml >>

```
<?xml version="1.0"?>
<webservices …>
```

```
 <webservice-description>
   <webservice-description-name>
        DocLitEchoService
   </webservice-description-name>
   <wsdl-file>WEB-INF/wsdl/DocLitEchoService.wsdl</wsdl-file>
   <jaxrpc-mapping-file>
      WEB-INF/DocLitEchoService-mapping.xml</jaxrpc-mapping-file>
   <port-component>
      <port-component-name>EchoPort</port-component-name>
      <wsdl-port xmlns:ns2="urn:DocLitService">
         ns2:EchoPort
      </wsdl-port>
      <service-endpoint-interface>
         jeustest.webservices.java2wsdl.doclit.Echo
      </service-endpoint-interface>
      <service-impl-bean>
         <servlet-link>EchoServlet</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

serlvet-link 엘리먼트의 값은 web.xml 내의 servlet-name 의 값과 일치해야 한다.

<< DocLitEchoService 의 web.xml >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>EchoServlet</servlet-name>
        <servlet-class>
          jeustest.webservices.java2wsdl.doclit.EchoImpl
        </servlet-class>
        …
    </servlet>
</web-app>
```

위와 마찬가지로 EJB 엔드포인트 웹서비스의 경우 ejb-link 엘리먼트를 사용하여 정의한다. 그 예는 다음과 같다.

<< AddressBookService 의 webservices.xml >>

```
<?xml version="1.0"?>
<webservices …>
   <webservice-description>
   <webservice-description-name>AddressBookService</webservice-
description-name>
   <wsdl-file>META-INF/wsdl/AddressBookService.wsdl</wsdl-file>
   <jaxrpc-mapping-file>
      META-INF/AddressBookService-mapping.xml
   </jaxrpc-mapping-file>
   <port-component>
     <port-component-name>AddressBookIFPort</port-component-name>
     <wsdl-port xmlns:ns2="urn:AddressBookService">
       ns2:AddressBookIFPort
     </wsdl-port>
     <service-endpoint-interface>
        address.AddressBookIF
     </service-endpoint-interface>
     <service-impl-bean>
         <ejb-link>AddressEJB</ejb-link>
     </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

ejb-link 엘리먼트의 값은 ejb-jar.xml 파일 내의 ejb-name 엘리먼트의 값과 일치해야 한다. 그 예는 다음과 같다.

<< AddressBookService 의 web.xml >>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar …>
   <display-name>AddressEJB</display-name>
   <enterprise-beans>
     <session>
      <display-name>AddressEJB</display-name>
      <ejb-name>AddressEJB</ejb-name>
      <service-endpoint>address.AddressBookIF</service-endpoint>
      <ejb-class>address.AddressBookEJB</ejb-class>
```

```
                    …
     </session>
   </enterprise-beans>
</ejb-jar>
```

한가지 유의해야 할 사항은, servlet-link 나 ejb-link 를 통해 연결되어지려는 대상들은 webservices.xml 를 포함하는 압축파일 안에 존재해야 한다는 것이다.

11.3.4 handler 엘리먼트

메시지 핸들러는 J2EE 엔드포인트가 보내고 받는 메시지를 걸러서 가공하는 역할을 한다. webservices.xml 파일 안에서 이 메시지 핸들러들을 설정하고, 처리 순서를 지정할 수 있다. (메시지 핸들러에 대한 더 자세한 설명은 9 장을 참조하라.)

다음은 핸들러를 설정한 webservices.xml 의 한 작성 예이다.

<< FileAttachmentService 의 webservices.xml >>

```
<?xml version="1.0"?>
<webservices version="1.1"
          xmlns="http://java.sun.com/xml/ns/j2ee">
   <webservice-description>
     …
     <port-component>
     <port-component-name>FileAttPort</port-component-name>
     …
     <handler>
       <handler-name>ServerAttachmentHandler</handler-name>
       <handler-class>
          filetransfer.ServerAttachmentHandler
       </handler-class>
       <init-param>
           <param-name>directory</param-name>
           <param-value>/temp</param-value>
       </init-param>
     </handler>
   </port-component>
  </webservice-description>
</webservices>
```

11.3.4.1 handler-name 엘리먼트

여기 정의 되는 값은 webservices.xml 파일 안에서 유일한 값이 되어야 하며, 설정한 핸들러에 대한 식별자 역할을 하게 된다.

11.3.4.2 handler-class 엘리먼트

핸들러를 구현한 클래스를 정의한다. 여기 정의 되는 핸들러 클래스는 javax.xml.rpc.handler.Handler 인터페이스를 직접, 혹은 간접적으로 구현하여야 한다.

11.3.4.3 init-param 엘리먼트

이 엘리먼트는 필수 사항은 아니며, 메시지 핸들러의 생성 주기의 시작 단계에서 핸들러를 초기화 하기 위해 전달되어질 파라미터 들을 정의한다. 런타임 시에 HandlerInfo.getHandlerConfig() 메쏘드를 실행함으로써 얻어지는 java.util.Map 객체로부터 파라미터들을 가져 올 수 있다.

11.3.4.4 soap-header 엘리먼트

soap-header 엘리먼트는 핸들러가 처리해야 하는 SOAP 헤더 블록의 큐네임 (Q Name; Qualified Name) 을 정의하며, handler 엘리먼트는 하나 이상의 soap-header 엘리먼트를 포함할 수 있다.

11.3.4.5 soap-role 엘리먼트

soap-role 엘리먼트는 헤더 블록을 처리할 때 핸들러가 정하는 역할을 정의한다. soap-role 값이 soap-header 엘리먼트와 짝을 이루어서 정의 될 때에만 어떤 헤더 블록을 핸들러가 처리하는지 정확하게 알 수 있다.

SOAP 메시지는 각각의 헤더 블록을 처리하기 위한 역할을 명시적으로 지정할 수 있다. actor 속성이 정의되어 있으면 지정된 역할을 수행하는 노드만이 헤더 블록을 처리하고, actor 속성이 정의 되지 않은 경우에는 최종적인 메시지 수신자가 헤더 블록을 처리 한다.

12 JAX-RPC 매핑 파일의 작성

JAX-RPC 매핑 파일은 JEUS 웹서비스에 내장된 JAX-RPC 컴파일러가 WSDL 문서와 웹서비스 엔드포인드를 나타내는 자바 인터페이스의 관계를 이해하는 것을 도와준다. 많은 경우에 WSDL 과 자바의 매핑은 매핑 파일이 없어도 큰 문제가 없지만, 명시적인 정의가 필요할 때가 있으며 , 그러한 필요에 의해서 JAX-RPC 매핑 파일이 도입되었다. JAX-RPC 매핑 파일은 J2EE 웹서비스 엔드포인트나 J2EE 웹서비스 클라이언트를 사용할 때마다 필요하며, WSDL 문서와 JAX-RPC 매핑 파일은 일대일 대응한다. 즉 각각의 WSDL 파일에는 하나씩의 JAX-RPC 매핑 파일이 존재한다.

12.1 JAX-RPC 매핑 파일의 내용

JAX-RPC 매핑 파일은 다음과 같은 관계에 있는 요소들의 매핑을 정의한다.

- XML 복합 타입(Complex Type)과 자바 빈
- Fault 메시지와 예외 클래스
- WSDL 의 portType 정의와 서비스 엔드포인트 인터페이스
- WSDL 의 service 정의와 서비스 인터페이스

JAX-RPC 매핑 파일에 정의 되지 않은 것들은 WSDL 과 XML 의 자바로의 표준 매핑 법칙을 따라서 매핑이 이루어진다. 그리고 매핑 파일에서 정의 되는 것들은 항상 표준 매핑 법칙에 우선한다.

12.2 JAX-RPC 매핑 파일의 작성

매핑 파일은 상당히 내용이 복잡해 보이고, 크기에 있어서도 단일 설정 파일치고는 큰 편이다.

전체적인 설정 파일의 구성은 다음과 같다.

<< JAX-RPC 매핑 파일의 구조 >>

```
<java-wsdl-mapping…>
       <package-mapping/>
       <java-xml-type-mapping/>
       <exception-mapping/>
       <service-interface-mapping/>
       <service-endpoint-interface-mapping>
          <service-endpoint-method-mapping/>
          <service-endpoint-method-mapping/>
          ...
       </service-endpoint-interface-mapping>
       <service-interface-mapping/>
       <service-endpoint-interface-mapping>
          <service-endpoint-method-mapping/>
          <service-endpoint-method-mapping/>
          ...
       </service-endpoint-interface-mapping>
</java-wsdl-mapping>
```

12.2.1 java-wsdl-mapping 엘리먼트

java-wsdl-mapping 엘리먼트는 JAX-RPC 매핑 파일의 최상위 엘리먼트이며, 다른 매핑 엘리먼트들을 포함하고 있다.

12.2.2 package-mapping 엘리먼트

package-mapping 엘리먼트는 JAX-RPC 컴파일러가 WSDL 에 정의된 여러 가지 타입에 대해 자바 클래스와 인터페이스 정의를 생성하려고 할 경우 사용된다. package-type 엘리먼트의 값은 자바 패키지의 이름이며 namespaceURI 의 값은 지정된 자바 패키지로 매핑 되어야 하는 XML 이름 공간이다. 다음은 그 예이다. package-mapping 은 반드시 지정되어야 하는 항목이다.

```
<java-wsdl-mapping version="1.1"
xmlns="http://java.sun.com/xml/ns/j2ee">
    <package-mapping>
        <package-type>address</package-type>
        <namespaceURI>urn:AddressBookService</namespaceURI>
    </package-mapping>
    …
</java-wsdl-mapping>
```

12.2.3 java-xml-type-mapping 엘리먼트

java-xml-type-mapping 엘리먼트는 XML Schema 복합 타입(Complex Type)이나 단순 타입(Simple Type)을 사용하게 될 때 필요하며 XML 스키마에 정의된 built-in 타입이 자바로 표준적인 매핑이 된다면 이 엘리먼트를 사용하지 않아도 된다. 이 엘리먼트는 XML 스키마와 자바 타입간의 관계를 정의한다. 다음은 그 예이다

```
<java-wsdl-mapping version="1.1"
xmlns="http://java.sun.com/xml/ns/j2ee">
    …
    <java-xml-type-mapping>
        <java-type>address.Address</java-type>
        <ns1:root-type-qname
          xmlns:ns1="http://java.sun.com/xml/ns/j2ee"
          xmlns="urn:AddressBookService">
          Address
        </ns1:root-type-qname>
        <qname-scope>complexType</qname-scope>
        <variable-mapping>
            <java-variable-name>addr</java-variable-name>
            <xml-element-name>addr</xml-element-name>
        </variable-mapping>
        <variable-mapping>
            <java-variable-name>street</java-variable-name>
            <xml-element-name>street</xml-element-name>
        </variable-mapping>
        <variable-mapping>
            <java-variable-name>zipcode</java-variable-name>
            <xml-element-name>zipcode</xml-element-name>
        </variable-mapping>
    </java-xml-type-mapping>
    …
</java-wsdl-mapping>
```

12.2.4 exception-mapping 엘리먼트

exception-mapping 엘리먼트는 WSDL 결함(fault) 메시지를 자바 예외 클래스로 매핑한다. 기본적인 구조는 다음과 같다.

```
<java-xml-mapping…>
      …
       <exception-mapping>
             <exception-type>CLASS_NAME</exception-type>
             <wsdl-message>WSDL_MESSAGE_NAME</wsdl-message>
       </exception-mapping>
       …
</java-xml-mapping>
```

12.2.5 service-interface-mapping 엘리먼트

service-interface-mapping 엘리먼트는 WSDL 의 service 정의를 JAX-RPC 서비스 인터페이스 타입으로 매핑한다.

service-interface 엘리먼트는 WSDL 의 service 정의를 나타내는 자바 인터페이스의 풀 패키지 클래스 이름을 정의하고, 서비스 인터페이스의 패키지 이름은 package-mapping 엘리먼트에 정의된 패키지 이름과 일치해야 한다.

port-mapping 엘리먼트는 서비스 인터페이스의 get*PortName*() 메쏘드의 포트 이름을 WSDL 의 port 엘리먼트와 대응하게 정의한다.

```
<java-xml-mapping…>
   …
   <service-interface-mapping>
      <service-interface>
         address.AddressBookService
      </service-interface>
      <ns3:wsdl-service-name
            xmlns:ns3="http://java.sun.com/xml/ns/j2ee"
            xmlns="urn:AddressBookService">
         AddressBookService
      </ns3:wsdl-service-name>
      <port-mapping>
         <port-name>AddressBookIFPort</port-name>
         <java-port-name>AddressBookIFPort</java-port-name>
      </port-mapping>
   </service-interface-mapping>
   …
</java-xml-mapping>
```

위와 같이 정의될 경우, 생성되는 자바 서비스 인터페이스 정의는 다음과 같다.

<< *AddressBookService.java* >>

```
package address;

public interface AddressBookService extends javax.xml.rpc.Service
{
   public java.lang.String getAddressBookIFPortAddress();
   public address.AddressBookIF getAddressBookIFPort() throws
javax.xml.rpc.ServiceException;
   public address.AddressBookIF getAddressBookIFPort(java.net.URL
portAddress) throws javax.xml.rpc.ServiceException;
}
```

12.2.6 service-endpoint-interface-mapping 엘리먼트

service-endpoint-interface-mapping 엘리먼트는 서비스 엔드포인트 인터페이스를 WSDL 의 portType 과 binding 정의로 매핑한다. 이 엘리먼트는 JAX-RPC 컴파일러가 적절한 엔드포인트 스텁과 엔드포인트 인터페이스를 생성하는 데 필요한 정보를 제공한다. 또한 WSDL 의 operation 과 message part 정의가 어떻게 자바 엔드포인트 메쏘드의 정의로 매핑될 것인지에 대한 상세 정보를 제공한다. 다음은 AddressBookService 의 매핑 파일에 정의된 예이다.

```
<java-wsdl-mapping…>
   …
   <service-endpoint-interface-mapping>
     <service-endpoint-interface>
        address.AddressBookIF
     </service-endpoint-interface>
     <ns4:wsdl-port-type
          xmlns:ns4="http://java.sun.com/xml/ns/j2ee"
          xmlns="urn:AddressBookService">
        AddressBookIF
     </ns4:wsdl-port-type>
     <ns5:wsdl-binding
          xmlns:ns5="http://java.sun.com/xml/ns/j2ee"
          xmlns="urn:AddressBookService">
       AddressBookIFSoapBinding
```

```
    </ns5:wsdl-binding>
    <service-endpoint-method-mapping>
       <java-method-name>add</java-method-name>
       <wsdl-operation>add</wsdl-operation>
       <wrapped-element/>
       <method-param-parts-mapping>
          <param-position>0</param-position>
          <param-type>address.PersonInfo</param-type>
          <wsdl-message-mapping>
             <ns6:wsdl-message
                xmlns:ns6="http://java.sun.com/xml/ns/j2ee"
                xmlns="urn:AddressBookService">
               addRequest
             </ns6:wsdl-message>
             <wsdl-message-part-name>
                parameters
             </wsdl-message-part-name>
             <parameter-mode>IN</parameter-mode>
          </wsdl-message-mapping>
       </method-param-parts-mapping>
       <wsdl-return-value-mapping>
          <method-return-value>
             address.PersonInfo[]
          </method-return-value>
          <ns7:wsdl-message
               xmlns:ns7="http://java.sun.com/xml/ns/j2ee"
               xmlns="urn:AddressBookService">
             addResponse
          </ns7:wsdl-message>
          <wsdl-message-part-name>
             parameters
          </wsdl-message-part-name>
       </wsdl-return-value-mapping>
     </service-endpoint-method-mapping>
     …
  </service-endpoint-interface-mapping>
  …
</java-wsdl-mapping>
```

13 데이터 타입

13.1 소개

이 장에서는 웹서비스에 관한 여러 가지 데이터 타입 문제에 관해 설명한다. 먼저 표준 Java/XML 데이터 타입 매핑(type mapping) 과 사용자 정의 클래스들에서 웹서비스 파라미터로 사용할 JAX-RPC 의 value 타입에 대해서 설명한다. 그 다음으로 출력 또는 입/출력 파라미터들을 위한 JAX-RPC 의 holder 클래스를 살펴본다. 마지막으로 웹서비스 클라이언트로 어떻게 에러 정보를 전송하는지를 알아본다.

13.2 JEUS 웹서비스의 데이터 타입

13.2.1 소개

이 절에서는 웹서비스와 웹서비스 클라이언트에서 사용하는 데이터 타입에 관한 이슈에 대해서 설명한다.

- 개발자가 작성한 클래스들을 사용하는 웹서비스 작성 방법.
- holder 클래스들을 사용하여 여러 개의 값을 반환하는 웹서비스 작성 방법.
- Exception 을 내보내는 웹서비스 작성 방법.

13.2.2 Java 와 XML 타입 매핑

XML 인스턴스를 Java 객체로 직렬화/역직렬화((de)serialize) 하기 위해서는 XML 의 타입과 Java 클래스들의 타입들과의 타입 매핑이 필요하다. JEUS 웹서비스는 JAX-RPC 스펙에 설명된 표준 XML/Java 타입 매핑을 따르고 있다.

13.2.3 JAX-RPC Value Type 사용

JAX-RPC Value Type 은 웹서비스의 요청이나 응답에서, 값으로써 전달될 수 있는 타입을 말한다. JAX-RPC Value Type 은 일반적으로 사용자 정의

JavaBeans 컴포넌트로 표현된다. JEUS 웹서비스는 사용자가 작성한 JavaBeans 타입을 웹서비스의 파라미터와 반환 값으로서 사용할 수 있도록 하고 있다.

13.2.4 In/Out 파라미터로서 Holder 의 사용

입/출력 파라미터는 웹서비스의 실행 시 입력은 물론 출력에서 사용되는 파라미터이다. 웹서비스가 여러 출력 값들을 반환해야 할 경우 입/출력 파라미터를 사용할 수 있다. JEUS 웹서비스는 입/출력 파라미터를 표준 JAX-RPC Holder 클래스를 사용하여 지원하고 있다.

13.2.5 SOAP Fault 로서의 Exception 사용

웹서비스 실행 중 어떤 에러가 발생하게 되면 웹서비스 클라이언트에게 그 내용이 전달되어야 한다. SOAP 표준에서는 이런 목적을 위한 SOAP Fault 를 정의 하고 있다. JEUS 웹서비스는 표준 SOAP Fault 를 지원한다.

13.2.6 결론

이 절에서는 이 장의 전반적인 내용을 간단히 설명하였다. 다음 절에서부터 이들 주제에 대해 좀 더 자세히 보게 될 것이며, 웹서비스 어플리케이션에서 이런 특징들을 어떻게 사용하는지에 대해서 알아볼 것이다.

13.3 Java 와 XML 타입 매핑

13.3.1 소개

지금까지 하나의 타입만(String 만)을 파라미터와 반환 값으로 사용하는 웹서비스 예제를 보았다. JEUS 웹서비스는 Java 타입을 XML/WSDL 정의로 매핑한다. 예를 들어 JEUS 웹서비스는 `java.lang.String` 클래스를 XML `xsd:string` 데이터 타입으로 매핑한다. 어플리케이션 개발자들은 이러한 매핑 과정에 대해서 자세히 알 필요가 없지만, 모든 Java 클래스가 파라미터와 반환 타입으로 사용될 수 있는 것이 아님을 기억하기 바란다.

이 절에서는 JEUS 웹서비스에서 지원하는 데이터 타입들에는 무엇이 있는지 알아볼 것이고, 어떤 데이터 타입이 JEUS 웹서비스에서 사용되기 위해 필요한 요구사항인지에 대해서도 알아본다.

13.3.2 내장 타입 매핑(Built-in Type Mapping)

아래 표는 Java 와 XML 의 데이터 형과 JEUS 웹서비스의 내장 타입 매핑을 보여주고 있다[표 5].

xsd 접두어는 XML namespace URI(http://www.w3.org/2001/XMLSchema)를 나타낸다.

SOAP-ENC 접두어는 XML namespace URI(http://schmas.xmlsoap.org/soap/encoding)를 나타낸다.

표 5. 내장 XML/Java 타입 매핑

XML 데이터 타입	Java 데이터 타입
xsd:string	java.lang.String
xsd:boolean	Boolean, java.lang.Boolean *
xsd:double	double, java.lang.Double *
xsd:float	float, java.lang.Float *
xsd:int	int, java.lang.Integer
xsd:integer	java.math.BigInteger
xsd:long	long, java.lang.Long *
xsd:short	short, java.lang.Short *
xsd:byte	byte, java.lang.Byte
xsd:Decimal	java.math.BigDecimal
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:QName	javax.xml.rpc.namespace.QName
xsd:dateTime	java.util.Calendar
xsd:gYearMonth	java.util.Calendar
xsd:gYear	java.util.Calendar
xsd:gMonthDay	java.util.Calendar
xsd:anyURI	java.net.URI (JDK 1.4 or over) / java.lang.String (JDK 1.4)

XML 데이터 타입	Java 데이터 타입
xsd:duration	java.lang.String
xsd:name	java.lang.String
xsd:NCName	java.lang.String
xsd:NMTOKEN	java.lang.String
xsd:nomalizedString	java.lang.String
xsd:time	java.util.Calendar
xsd:token	java.lang.String
xsd:unsignedByte	short
xsd:unsignedLong	java.math.BigInteger
xsd:unsignedInt	long
xsd:unsignedShort	int
SOAP-ENC:base64	byte[]
SOAP-ENC:string	java.lang.String
SOAP-ENC:boolean	boolean, java.lang.Boolean *
SOAP-ENC:double	double, java.lang.Double *
SOAP-ENC:float	float, java.lang.Float *
SOAP-ENC:int	int, java.lang.Integer *
SOAP-ENC:long	long, java.lang.Long *
SOAP-ENC:short	short, java.lang.Short *
SOAP-ENC:byte	byte, java.lang.Byte *
SOAP-ENC:interger	java.math.BigInteger
SOAP-ENC:decimal	java.math.BigDecimal
SOAP-ENC:Array	array of built-in data type

만약 WSDL 에서 어떤 객체가 null 이 될 수 있다고 정의되어 있다면 서비스 호출자는 `xsd:nil` 을 데이터로 보내거나 받을 때 사용할 수 있다. 그리고 Java primitive 타입은 Wrapper 클래스로 교체된다. 위의 표에서 Java 타입 뒤에 붙은 *는 이것을 나타낸다.

DII 클라이언트에서는 어떤 타입을 지정할 때, XML 의 QName 을 사용할 수 있다.

```
String NS_XSD = “http://www.w3.org/2001/XMLSchema”;
String XSD_DATETIME = new QName(NS_XSD, “dateTime”);
call.addParameter(“arg1”, XSD_DATETIME, PARAM_MODE_IN);
```

13.3.3 배열

JEUS 웹서비스는 JAX-RPC 타입에서 정의한 배열들을 지원한다. 예를 들어 `int[]`와 `String[]`나 다차원 배열인 `java.math.BigDecimal[][]` 와 같은 것도 지원한다.

13.3.4 사용자 정의 타입 : JAX-RPC Value Type

JEUS 웹서비스는 어플리케이션을 위해 작성한 모든 사용자 정의 형을 지원한다. JAX-RPC 스펙에는 이러한 클래스들을 ‘Value Type’이라 부른다. JEUS 웹서비스에서 이것을 지원하기 위해서는 사용자 정의 클래스들은 다음 규칙을 따라야 한다.

- 파라미터 없는 public default 생성자를 가져야 한다.
- 직, 간접적으로 `java.rmi.Remote` 를 구현해서는 안 된다.
- 멤버 필드들의 타입은 JEUS 웹서비스가 지원하는 타입이어야 한다.

클래스는 `public`, `private`, 또는 `protected` 필드들을 포함할 수 있다. 웹서비스 호출 중 전달되는 값을 위해서 필드는 다음 조건을 충족해야 한다.

- public 필드는 final 이나 transient 가 될 수 없다.
- public 필드가 아닌 것은 관련된 getter 와 setter 메쏘드를 가져야 한다.

위의 규칙을 따르는 JavaBeans 컴포넌트 또한 지원된다.

13.3.5 결론

이번 절에서는 JEUS 웹서비스에서 지원하는 데이터 타입을 설명했다. 다음 절에서는 WSDL 문서에 정의된 입력/출력 파라미터를 어떻게 사용할 것인가를 알아본다.

13.4 JAX-RPC Value Type 의 사용

13.4.1 소개

이전 절에서 보았듯이 13.3.4 절에서 언급한 규칙들을 따르는 사용자 정의 형들은 웹서비스의 파라미터나 반환타입으로서 사용이 가능하다. 이런 종류의 사용자 정의 타입은 JAX-RPC value type 이라 불린다.

13.4.2 JAX-RPC value type 사용하기

이 절에서는 JAX-RPC value type 을 사용하는 예제를 보여준다.

'CalcService'는 두 개의 숫자와 하나의 연산자를 받아, 결과를 숫자로 넘겨주는 예제이다. 그리고 CalcService 를 위한 웹서비스 클라이언트를 작성한다.

JAX-RPC value type 을 사용하는 웹서비스 생성

CalcService 소스 코드인 '`Calculator.java`' 은 아래와 같다.

<<Calculator.java>>

```
package calc;

public class Calculator {
      public Calculator() { }
      public double calc(CalcData data) {
            String op = data.getOp();
            double num1 = data.getNum1();
            double num2 = data.getNum2();
            double ret = -9999.0;

            if (op.equals("plus")) {
                  ret = num1 + num2;
            } else if (op.equals("minus")) {
                  ret = num1 - num2;
```

```
            } else if (op.equals("mult")) {
                  ret = num1 * num2;
            } else if (op.equals("div")) {
                  if (num2 != 0)
                        ret = num1 / num2;
            }

            return ret;
      }
}
```

calc() 메쏘드는 에러 상황이 발생하면 –9999.0 을 반환한다. 이 후에 이를 좀 더 고급스럽게 처리하는 방법에 대해서 소개할 것이다.

calc() 메쏘드는 CalcData 형을 인자로 받는다. 아래는 CalcData.java 의 소스 코드이다.

<<*CalcData.java*>>

```
package calc;

public class CalcData {
      private double num1;
      private double num2;
      private String op;

      public CalcData() { }
      public double getNum1() { return num1; }
      public double getNum2() { return num2; }
      public String getOp() { return op; }

      public void setNum1(double n) { num1 = n; }
      public void setNum2(double n) { num2 = n; }
      public void setOp(String s) { op = s; }
}
```

CalcData 클래스는 JAX-RPC value type 요구 사항을 따르고 있다. 이것은 CalcData 의 인스턴스는 값으로서 전달될 수 있음을 의미한다.

그리고 여기에 Service Endpoint Interface 파일 CalculatorIF.java 가 있다.

<<*CalculatorIF.java*>>

```
package calc;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CalculatorIF extends Remote {
      public double calc(CalcData data) throws RemoteException;
}
```

이 파일들을 컴파일 하기 위해서 다음과 같이 명령을 수행한다.

```
ant compile
```

이 명령은 결과 클래스 파일들을 build 디렉토리 밑에 옮겨놓을 것이다. 배치 가능한 EAR 파일을 생성하기 위해서는 다음 명령을 수행한다.

```
ant wsear
```

이 웹서비스 모듈을 배치한다. 이제 다음과 같은 주소로 이 서비스에 접근할 수 있다.

http://localhost:8088/Calculator1Service/Calculator1Service?wsdl

JAX-RPC value type 을 사용하는 웹서비스 클라이언트 생성

'CalcService' 웹서비스를 위한 Proxy 클라이언트를 생성할 것이다

```
ant wsdl2java
```

생성된 Stub 코드의 패키지 이름은 `com.test.calc` 로 가정한다.

클라이언트 프로그램의 소스 코드는 다음과 같다.

<<*CalcClient.java*>>

```
import com.test.calc.*;
import javax.xml.rpc.soap.SOAPFaultException;

public class CalcClient {
  public static void main(String[] args) {
```

```
    CalcClient calc = new CalcClient();

    if (args.length != 3) {
      System.out.println("usage: java CalcClient num1 op num2");
      System.out.println(
          " where op is one of 'plus', 'minus', 'mult', 'div'");
      System.exit(1);
    }

    try {
      calc.run(args);
    } catch (SOAPFaultException e) {
      System.err.println("faultcode = " + e.getFaultCode());
      System.err.println("faultString = " + e.getFaultString());
    } catch (Exception e) {
      System.err.println(e.toString());
      e.printStackTrace();
    }
  }

  public void run(String[] args) throws Exception {
    CalculatorIF port = new
         Calculator1Service_Impl().getCalculatorIFPort();
    CalcData data = new CalcData();
    data.setNum1((new Double(args[0])).doubleValue());
    data.setNum2((new Double(args[2])).doubleValue());
    data.setOp(args[1]);

    double ret = port.calc(data);
    System.out.println(ret);
  }
}
```

클라이언트 측을 위한 특별한 것은 없다. 다음 명령을 통해 클라이언트 코드와 Stub 코드를 컴파일 할 수 있다.

```
ant compile-client
```

그리고 클라이언트를 실행하기 위해서는 다음을 수행한다. (> 은 쉘의 프롬프트를 의미한다.)

```
> cd build
> java -classpath
      %JEUS_HOME%\lib\system\jeus.jar;
      ;. CalcClient 2 div 1
```

혹은 다음과 같이 Ant 태스크를 실행할 수도 있다

```
ant run
```

성공하면 다음과 같은 결과를 볼 것이다.

```
2.0
```

13.4.3 결론

이 절에서는 JEUS 웹서비스에서 JAX-RPC value type 을 어떻게 사용하는지 알아 보았다.

다음 절은 holder 클래스와 입/출력 파라미터에 대해서 알아본다.

13.5 Holder 를 In/Out Parameter 로 사용하기

13.5.1 소개

만약 웹서비스가 여러개의 값을 반환하기를 원한다면 JAX-RPC value type 을 정의하거나 출력 혹은 입/출력 파라미터를 하나 이상 지정해야 한다. Holder 클래스는 출력 또는 입/출력 파라미터로서 사용되는 Helper 클래스이다.

13.5.2 내장 Holder 클래스들

JEUS 웹서비스는 단순 데이터 형 JAX-RPC holder 클래스들을 제공한다. JAX-RPC 가 지원하는 표준 holder 클래스들은 [표 6. 내장 holder 클래스] 에서 보는 것과 같다.

표 6. 내장 holder 클래스

Holder Class	Java Data Type
javax.xml.rpc.holders.BooleanHolder	boolean

Holder Class	Java Data Type
javax.xml.rpc.holders.ByteHolder	byte
javax.xml.rpc.holders.ShortHolder	short
javax.xml.rpc.holders.IntHolder	int
javax.xml.rpc.holders.LongHolder	long
javax.xml.rpc.holders.FloatHolder	float
javax.xml.rpc.holders.DoubleHolder	double
javax.xml.rpc.holders.BigDecimalHolder	java.math.BigDecimal
javax.xml.rpc.holders.BigIntegerHolder	java.math.BigInteger
javax.xml.rpc.holders.ByteArrayHolder	byte[]
javax.xml.rpc.holders.CalendarHolder	java.util.Calendar
javax.xml.rpc.holders.QNameHolder	javax.xml.namespace.QName
javax.xml.rpc.holders.StringHolder	java.lang.String

각 Holder 클래스들이 가지고 있는 값을 액세스하기 위해 value 필드를 사용한다. Caculator.java 를 다음과 같이 수정하였다.

<< *Calculator.java* >>

```
package calc;

import javax.xml.rpc.holders.DoubleHolder;

public class Calculator implements CalculatorIF {
    public Calculator() { }

    public void calc(CalcData data, DoubleHolder result){
        String op = data.getOp();
        double num1 = data.getNum1();
        double num2 = data.getNum2();
        double ret = -9999.0;
```

```
        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
        }

        result.value = ret;
    }
}
```

위에서 보듯이 소스 코드는 내장 holder 클래스인 `javax.xml.rpc.holders.DoubleHolder` 를 import 하였다.

```
import javax.xml.rpc.holders.DoubleHolder;
```

`calc()`메쏘드의 signature 또한 수정되었다. 리턴 타입은 void 이고 두 번째 파라미터로 holder 객체를 받는다.

```
public void calc(CalcData data, DoubleHolder result){
…
}
```

holder 객체가 가진 객체값을 접근하기 위해서는 holder 클래스의 `value` 필드를 이용한다.

```
result.value = ret;
```

이 웹서비스를 위한 웹서비스 클라이언트를 생성하기 전에, 이 웹서비스를 패키지로 만들고 배치해야 한다. 클라이언트 프로그램 `CalcClient.java` 도 수정되었다. 여기서는 클라이언트 소스 코드의 `run()` 메쏘드를 보여주고 있다.

<< CalcClient.java 의 run() 메쏘드 >>

```
    public void run(String[] args) throws Exception {
        CalculatorIF port =
             new CalcService_Impl().getCalculatorIFPort();
```

```
        CalcData data = new CalcData();
        data.setNum1((new Double(args[0])).doubleValue());
        data.setNum2((new Double(args[2])).doubleValue());
        data.setOp(args[1]);

        DoubleHolder ret = new DoubleHolder();
        port.calc(data, ret);
        System.out.println(ret.value);
    }
```

CalcClient.java 를 컴파일 하기 전에 Proxy 소스 코드를 다시 생성하는 것을 잊지 말도록 한다.

13.5.3 사용자 정의 타입을 위한 Holder 클래스 작성

만약 사용자가 작성한 클래스의 holder 클래스를 작성하고 싶다면, 다음 순서를 따라 한다.

- javax.xml.rpc.holders.Holder 인터페이스를 구현한다.
- 사용자가 작성한 클래스를 *Type*Holder 라 이름 짓는다. *Type* 은 holder 객체가 가지게 될 클래스의 이름이다. 예를 들어 CalcData 클래스를 위한 holder 를 만들기 위해서는 holder 클래스의 이름은 CalcDataHolder 가 된다.
- 작성한 holder 클래스를 public 으로 선언한다.
- value 라는 이름을 가진 public 필드를 생성한다. 이 데이터 타입은 holder 클래스가 가지게 되는 타입과 같다.
- value 필드를 기본값으로 초기화하는, 파라미터 없는 default 생성자를 만든다.
- 파라미터 값으로 value 필드를 설정하는 생성자를 만든다.

JAX-RPC value type 의 holder 클래스를 어떻게 사용하는지 보기 위해서 CalcService 예제를 수정한다.

이 클래스는 계산 결과값을 멤버로 포함하는 CalcData 라는 클래스이다. private 변수인 result 와 getter/setter 메쏘드를 멤버로 추가하였다.

<< *CalcData.java* >>

```
package calc;

public class CalcData {
      private double num1;
      private double num2;
      private String op;
      private double result;

      public CalcData() { }
      public double getNum1() { return num1; }
      public double getNum2() { return num2; }
      public String getOp() { return op; }
      public double getResult() { return result; }

      public void setNum1(double n) { num1 = n; }
      public void setNum2(double n) { num2 = n; }
      public void setOp(String s) { op = s; }
      public void setResult(double n) { result = n; }
}
```

CalcData 의 holder 클래스는 CalcDataHolder 이다. CalcDataHolder 클래스는 데이터 타입이 CalcData 인 public 의 value 필드와 value 필드를 초기화하는 두 개의 생성자를 가지고 있다.

<< *CalcDataHolder.java* >>

```
package calc;
import javax.xml.rpc.holders.Holder;

public class CalcDataHolder implements Holder {
      public CalcData value;

      public CalcDataHolder() {
      }

      public CalcDataHolder(CalcData value) {
            this.value = value;
      }
}
```

holder 클래스를 사용하는 Calculator.java 소스 코드는 다음과 같다. calc() 메쏘드는 데이터 타입이 CalcDataHolder 인 파라미터를 지정하고 있다. holder 객체가 가지고 있는 객체를 접근하기 위해 public 의 value 필드를 사용하고 있다.

```
package calc;

public class Calculator {
    public Calculator() { }

    public void calc(CalcDataHolder calcData){
        CalcData data = calcData.value;
        String op = data.getOp();
        double num1 = data.getNum1();
        double num2 = data.getNum2();
        double ret = -9999.0;

        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
        }
        data.setResult(ret);

        // The following line is not necessary in this case.
        // But we will use it to show the usage of holder class.
        calcData.value = data;
    }
}
```

이제 클라이언트를 보기 바란다. ant wsdl2java 명령 또한 WSDL 문서로부터 클라이언트를 위한 holder 클래스들을 생성한다. ant wsdl2java 명령으

로부터 생성된 holder 클래스들은 사용자가 작성한 것과는 다르다는 것을 유념하기 바란다.

클라이언트 코드는 다음과 같다.

```
import com.test.calc.*; // generated by ant process-wsdl
import com.test.calc.holders.*; // generated by ant process-wsdl

public class CalcClient {
    public static void main(String[] args) {
        CalcClient calc = new CalcClient();

        if (args.length != 3) {
            System.out.println(
                 "usage: java CalcClient num1 op num2");
            System.out.println(
            " where op is one of 'plus', 'minus', 'mult', 'div'");
            System.exit(1);
        }

        try {
            calc.run(args);
        } catch (Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
        }
    }

    public void run(String[] args) throws Exception {
        CalculatorIF port =
            new CalcService_Impl().getCalculatorIFPort();
        CalcData data = new CalcData();
        CalcDataHolder dataHolder = new CalcDataHolder(data);

        data.setNum1((new Double(args[0])).doubleValue());
        data.setNum2((new Double(args[2])).doubleValue());
        data.setOp(args[1]);

        port.calc(dataHolder);
```

```
        System.out.println(dataHolder.value.getResult());
    }
}
```

13.5.4 결론

이번 절에서 JAX-PRC holder 클래스가 웹서비스 어플리케이션에서 어떻게 사용될 수 있는지 설명하였다.

13.6 Exception 을 SOAP Fault 로서 사용하기

SOAP Fault 는 SOAP 메시지를 통해 에러나 상태 정보를 보내기 위해 사용된다. 보다 자세한 내용은 SOAP 1.1 스펙의 '4.4 SOAP Fault' 을 참조하기 바란다. JEUS 웹서비스에서 `java.rmi.RemoteException` 혹은 java.lang.Exception 클래스를 상속한 클래스를 SOAP Fault 로 사용할 수 있다 .

Exception 상태를 웹서비스 어플리케이션의 클라이언트로 전송하고자 한다면 웹서비스에서 `java.rmi.RemoteException` 이나 사용자 정의 Exception 을 던져지도록 지정하면 된다. 던져진 `Exception` 은 자동으로 SOAP Fault 로 감싸져서 SOAP 메시지의 body 에 포함되어 웹서비스 클라이언트로 전송된다.

CalcService 예제에서 에러 상황이 발생할 경우 –9999.0 값을 반환값으로 사용하였다. 이제 소스 코드를 다음과 같이 바꿀 수 있다.

<< *Calculator.java* >>

```
package calc;

import java.rmi.RemoteException;

public class Calculator {
    public Calculator() { }
    public double calc(CalcData data) throws RemoteException,
DevideByZeroException {
        String op = data.getOp();
        double num1 = data.getNum1();
        double num2 = data.getNum2();
        double ret = 0;
```

```
        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
            else
                throw new DevideByZeroException(
                                                "divide by zero");
        } else {
            throw new RemoteException(
                  "invalid opertion : " + op);
        }

        return ret;
    }
}
```

calc()메쏘드는 알려지지 않은 연산자가 지정되거나 나누기 값이 0 일 때 java.rmi.RemoteException 을 던지도록 만들었다.

웹서비스 클라이언트 프로그램에서 java.rmi.RemoteException 또는 java.lang.Exception 을 이 exception 을 잡는데 사용할 수 있다.

13.7 MIME Type 을 항상 javax.activation.DataHandler Type 으로 매핑하기

13.7.1 소개

웹서비스 클라이언트는 Attachment 를 SOAP Message 에 첨부하여 전송할 수 있다. JAX-RPC 스펙은 Attachment 의 MIME type 에 상응하는 Java type 매핑을 정의하고 있지만, 경우에 따라 웹서비스 클라이언트가 MIME type 에 관

계없이 항상 javax.activation.DataHandler Type 으로 매핑하여 사용할 수 도 있다.

이번 절에서는 WSDL-to-Java 매핑 툴을 사용하여 MIME type 을 항상 DataHandler Type 으로 매핑하는 방법을 설명한다.

13.7.2 Wsdl2java 에서 dataHandlerOnly 옵션 사용

아래와 같이 MIME part 를 가진 웹서비스의 WSDL 이 있다.

```
<message name="submission">
    <part name="title" type="xsd:string" />
    <part name="price" type="xsd:float" />
    <part name="attachment" type="xsd:hexBinary" />
</message>
...
<operation name="submit">
  ...
  <input>
     ...
    <mime:part>
      <mime:content part="attachment" type="application/xml" />
    </mime:part>
     ...
  </input>
   ...
</operation>
```

기본적으로 이러한 WSDL 을 가지고 wsdl2java 툴을 사용하여 서비스 엔드포인트 인터페이스를 생성하면 다음과 같이 MIME type "application/xml"은 javax.xml.transform.Source type 으로 매핑된다.

```
public interface SubmitBook extends java.rmi.Remote {
  public String submit(String title, float price,
                       javax.xml.transform.Source attachment)
                       throws java.rmi.RemoteException;
}
```

그러나, Ant task, wsdl2java 에서 attribute, dataHandlerOnly="true"로 설정하거나, 커맨드라인 툴에서 –datahandleronly 옵션을 사용하면, 다음과 같이

MIME Type 에 상관없이 part 는 항상 javax.activation.DataHandler type 으로 매핑된다.

```
public interface SubmitBook extends java.rmi.Remote {
  public String submit(String title, float price,
                       javax.activation.DataHandler attachment)
                       throws java.rmi.RemoteException;
}
```

따라서, 웹서비스 클라이언트 개발자는 DataHandler type 으로 Attachment 를 송부하여야 한다. 다음은 DataHandler type 을 사용한 웹서비스 클라이언트 예제이다.

```
// Creates a FileInputStream from the specified path name
FileInputStream inputStream = new FileInputStream(new
                      File("attachment/book.xml"));
DataHandler dataHandler = new DataHandler(inputStream,
                      "application/xml");

// Get a Service port
SubmitBook port =
    new SubmitBookService_Impl().getSubmitBookPort();
String result =
    port.submit("Sample for a option: datahandleronly", 12.34f,
                      dataHandler);
System.out.println("response = " + result);
```

13.7.3 결론

이번 절에서 datahandleronly option 을 사용하여 wsdl2java 에서 MIME type 을 항상 javax.activation.DataHandler Type 으로 매핑하는 방법과 웹서비스 클라이언트 작성법을 설명하였다.

13.8 Doc/Literal 에서 Data binding 을 사용하지 않기

13.8.1 소개

JAX-RPC Spec.은 XML type 에 대한 Java type 매핑을 정의하고 있다. 그러나, WSDL 의 type 에 의하여 매핑된 Java type 을 사용하기 보다는 SOAPElement 를 직접 구성하여 message 를 전송하는 것이 더욱 편리할 수도 있다.

이번 절에서는 WSDL-to-Java 매핑 툴을 사용하여 XML type 에 상관 없이 javax.xml.soap.SOAPElement 를 사용하는 방법을 설명한다.

13.8.2 Wsdl2java 에서 noDataBinding 옵션 사용

아래와 같은 Document/Literal 로 기술된 WSDL 이 있다.

```
<definitions name="BookQuoteService" ... >
  <types>
    <xsd:schema targetNamespace="...">
      <xsd:complexType name="Book">
        <xsd:sequence>
          <xsd:element name="title" type="xsd:string" />
          <xsd:element name="isbn" type="xsd:string" />
          <xsd:element name="authors" type="xsd:string" />
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="Book" type="mh:Book" />
      <xsd:element name="Result" type="xsd:float" />
    </xsd:schema>
  </types>

  <message name="getBookPriceRequest">
    <part name="book" element="mh:Book" />
  </message>
  <message name="getBookPriceResponse">
    <part name="result" element="mh:Result" />
  </message>
  ...

  <binding name="BookServiceSoapBinding" type="mh:BookQuote">
    <soap:binding style="document" ... />
    <operation name="getBookPrice">
      <input>
        <soap:body use="literal" ... />
      </input>
      <output>
        <soap:body use="literal" ... />
      </output>
```

```
    </operation>
  </binding>
  ...

</definitions>
```

이 WSDL 로부터 wsdl2java 로 생성한 서비스 엔드포인트 인터페이스는 다음과 같이 Object type 의 Input parameter 를 갖는다.

```
public interface BookQuote extends java.rmi.Remote {
  public float getBookPrice(sample.nodatabinding.stub.Book book)
                throws java.rmi.RemoteException;
}
```

만약, wsdl2java 로 생성한 서비스 엔드포인트 인터페이스를 생성할 때, Ant task, wsdl2java 에서 attribute, noDataBinding="true"로 설정하거나, 커맨드라인 툴에서 –nodatabinding 옵션을 사용하면, 다음과 같이 XML type 에 상관없이 Input parameter 및 Return value type 은 javax.xml.soap.SOAPElement 가 된다.

```
public interface BookQuote extends java.rmi.Remote {
  public javax.xml.soap.SOAPElement
     getBookPrice(javax.xml.soap.SOAPElement book)
       throws java.rmi.RemoteException;
}
```

wsdl2java 의 nodatabinding 옵셥은 Document/Literal 의 WSDL 에서만 유효하다.

이 경우, 웹서비스 클라이언트 개발자는 SOAPElement type 으로 message 를 직접 구성하여야 한다. 다음은 SOAPElement type 을 사용한 웹서비스 클라이언트 예제이다.

```
// Creates a FileDataSource from the specified path name
SOAPFactory factroy = SOAPFactory.newInstance();

// Create a SOAPElement object
SOAPElement book = factroy.createElement("Book", "mh",
                     "http://www.tmaxsoft.com/j2eews/BookQuote");

SOAPElement title = factroy.createElement("title");
```

```
title.addTextNode("Sample for a option: nodatabinding");
book.addChildElement(title);

SOAPElement isbn = factroy.createElement("isbn");
isbn.addTextNode("123-456-789");
book.addChildElement(isbn);

SOAPElement authors = factroy.createElement("authors");
authors.addTextNode("Tmax Soft Co., Ltd.");
book.addChildElement(authors);

// Get a Service port
BookQuote port = new BookQuoteService_Impl().getBookQuotePort();
SOAPElement price = port.getBookPrice(book);
System.out.println("price = " + price.getValue());
```

13.8.3 결론

이번 절에서 nodatabinding option 을 사용하여 wsdl2java 에서 WSDL-to-JAVA mapping 을 사용하지 않고 javax.xml.soap.SOAPElement 를 사용하는 방법과 웹서비스 클라이언트 작성법을 설명하였다.

13.9 결론

이번 장에서는 좀 더 실질적인 웹서비스 어플리케이션에서 사용되는, 다양한 데이터 타입을 설명하였다. JAX-RPC value type 은 웹서비스에서 생성한 클래스들을 사용할 수 있게 한다. 그리고 JAX-RPC holder 클래스들은 출력 또는 입/출력 파라미터를 사용하고자 할 때 사용할 수 있다. `java.rmi.RemoteException` 은 에러 상황이 발생할 경우 클라이언트에게 이 정보를 알릴 때 사용할 수 있다.

14 웹서비스 보안

14.1 소개

이 장에서는 JEUS 에서 웹 서비스에 보안을 적용하기 위한 방법을 설명한다.

14.2 개요

전통적으로 웹 서비스에 보안을 적용하기 위해서는 다음과 같은 두가지 방식이 존재한다.

- 메시지 수준 보안 : SOAP 메시지를 전자 서명이나 암호화
- 전송 수준 보안(Transport-level Security) : SSL 을 이용하여 클라이언트와 웹 서비스 사이의 연결의 보안을 보장.

전송 수준 보안을 적용하고자 한다면, 클라이언트와 JEUS 서버간의 연결을 SSL 을 이용하여 안전하게 할 수 있다. 그러나 이러한 방식은 연결 자체만을 안전하게 하며 클라이언트와 JEUS 서버 사이에 라우터나 메시지 큐와 같은 매개체(Intermediary)가 있다면 매개체는 SOAP 메시지를 암호화되지 않은 읽히기 쉬운 텍스트 문서 형태로 가질 수 있다. 또 전송 수준 보안은 전체적인 메시지를 다루므로 메시지 일부에의 보안 적용이 불가능하다.

메시지 수준 보안은 SSL 의 보안상의 장점을 포함하면서 부가적인 유연성을 제공한다. 메시지 수준 보안은 메시지 전달 과정에서 하나 이상의 매개체가 존재하더라도 보안이 유지 되는 엔드-투-엔드(End-to-End)보안이며, 연결 자체가 보안이 유지된다기 보다는 SOAP 메시지 자체의 서명과 암호화를 의미한다. 또한 부분적인 서명과 암호화가 가능하다는 장점이 있다.

14.3 JEUS 웹 서비스의 전송 수준 보안

웹 서비스의 전송 수준 보안이라는 의미는 웹 서비스 클라이언트 응용프로그램과 웹 서비스간의 연결을 SSL 을 이용하여 안전하게 하는 것을 의미한다. 전체적인 절차는 다음과 같다.

1. JEUS 서버의 SSL 설정
2. 클라이언트 응용프로그램의 SSL 설정

14.3.1 JEUS 서버의 SSL 설정

JEUS 서버에서의 SSL 설정은 “JEUS Web Container 안내서”를 참조하도록 하며, 웹 서비스 개발 부분은 추가 작업이 필요하지 않다.

14.3.2 클라이언트 응용 프로그램의 SSL 설정

응용 프로그램의 SSL 설정은 다음과 같은 절차를 따른다.

1. 인증서를 가져온다.(인터넷 익스플로러 등을 통해 인증서를 로컬 디렉토리에 저장한다.)
2. 가져온 인증서를 키 스토어에 저장한다.
3. wsdl2java 를 이용하여 wsdl 로부터 스텁을 생성하려고 하거나 클라이언트에서 웹 서비스를 호출하기 위해 클라이언트를 실행하려고 할 때 시스템 프로퍼티 값을 다음과 같이 설정한다.
   - –Djavax.net.ssl.trustStore=*keystore_name*
   - –Djavax.net.ssl.trustStorePassword=*keystore_password*

만약 ANT 에서 정의된 wsdl2java 태스크를 이용하려 한다면, 다음과 같은 환경 변수 설정이 필요하다.

```
set ANT_OPTS=-Djavax.net.ssl.trustStore=keystore_name
      -Djavax.net.ssl.trustStorePassword=keystore_password
```

ANT 가 아닌 wsdl2java 콘솔 툴을 사용한다면, 다음과 같은 환경 변수 설정이 필요하다.

```
set WSDL2JAVA_OPTS=-Djavax.net.ssl.trustStore=keystore_name
      -Djavax.net.ssl.trustStorePassword=keystore_password
```

14.4 JEUS 웹 서비스의 메시지 수준 보안

메시지 수준의 보안은 웹서비스와 웹서비스를 호출하는 클라이언트 간의 SOAP 메시지가 서명이나 암호화되는 것을 의미한다.

JEUS 웹서비스는 다음과 같은 OASIS 웹서비스 보안 표준 1.0 을 지원한다.

- OASIS Stardard 1.0
    - Web Services Secuirty : SOAP Message Security V1.0
    - Web Services Security : Username Token Profile V1.0
    - Web Services Security : X.509 Token Profile V1.0

위와 같은 표준들은 인증, 권한 부여, 데이터 무결성 및 기밀성 보장에 관한 요구조건을 명시하며 각종 기업 어플리케이션들에 존재하는 보안 쟁점들을 해결하게 해 준다.

14.4.1 JEUS 웹 서비스의 보안의 적용

JEUS 웹 서비스에서는 다음과 같이 적용이 가능하다.

- 클라이언트는 X.509 인증서를 사용하여 SOAP 메시지의 암호화와 서명을 하여 보안이 적용된 웹 서비스를 호출할 수 있고, 웹 서비스는 다시 X.509 인증서를 이용하여 SOAP 메시지를 암호화와 서명을 하여 응답으로 내보낼 수 있다. SOAP 메시지는 메시지 안에 모든 보안 관련 정보들을 포함하고 있으므로, 클라이언트와 웹서비스 사이의 매개체들은 SOAP 메시지에서 필요한 부분만 골라서 처리하게 할 수 있다.
- 기본적으로 JEUS 웹서비스는 암호화 하는 부분을 별도로 지정하지 않을 경우, SOAP 메시지의 몸체 부분을 암호화하지만 어떤 부분이든지 암호화 할 수 있게 해 준다. 서명도 마찬가지이며, 이 두가지는 서로 혼용될 수 있다.

14.4.2 JEUS 웹 서비스 보안 아키텍처

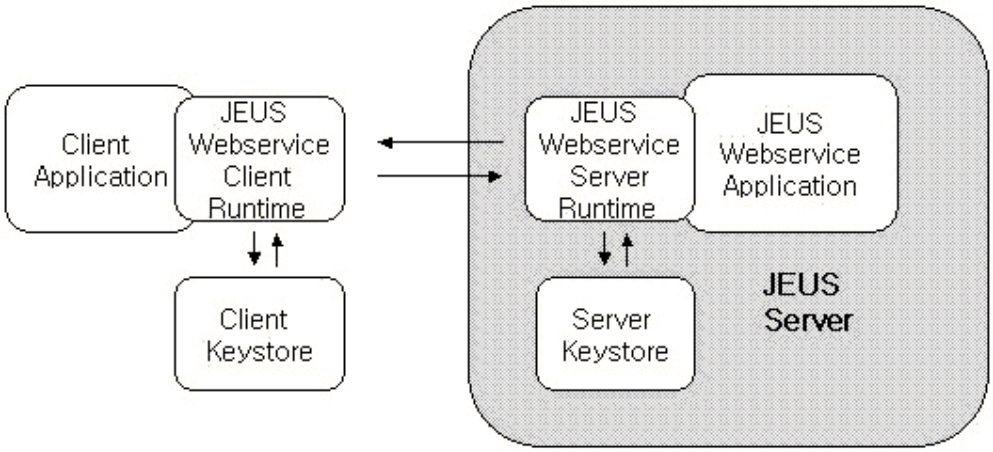


그림 21JEUS 웹서비스 보안 아키텍처

jeus-webservices-dd.xml 의 <security> 엘리먼트는 J2EE 웹 서비스의 보안 관련 설정을 담고 있으며 jeus-web-dd.xml 의 <security> 엘리먼트는 J2EE 웹서비스 클라이언트의 보안 관련 설정을 담고 있다.

14.4.2.1 JEUS J2EE 웹서비스 클라이언트와 JEUS 웹서비스 간의 보안 메시지의 송수신

기존의 웹 서비스에 보안을 적용하고 싶은 경우에는 jeus-webservices-dd.xml 에 <security>엘리먼트를 설정하고 키 스토어를 생성하여야 한다. 그리고 키 스토어에 접근하기 위한 암호나 UsernameToken 에 사용할 암호를 설정하기 위해서 콜백 클래스를 하나 생성하여야 한다. 이 어플리케이션을 JEUS 서버에 재배치함으로써 보안 설정이 적용된다.

J2EE 클라이언트의 경우도 마찬가지로, jeus-web-dd.xml 에 <security>엘리먼트를 설정하여야 하며, 클라이언트의 키를 저장할 키스토어를 생성하여야 한다. UsernameToken 에 사용할 암호와 키 스토어의 개인키를 가져오기 위한 암호를 설정하기 위해서 콜백 클래스를 하나 생성하여야 한다.

J2EE 클라이언트 어플리케이션이 실행되면 웹서비스 클라이언트 런타임은 jeus-web-dd.xml 에서 <security> 엘리먼트의 설정된 값에 따라 보안

로직을 동작 시켜 클라이언트 키 스토어에서 개인키를 꺼내어 서명을 하거나 서버단의 공개 키를 가지고 메시지를 암호화 하여 JEUS 서버의 웹서비스 서버 런타임으로 메시지를 보낸다. 서버 런타임은 jeus-webservices-dd.xml 에서 <security>엘리먼트에 설정된 값을 바탕으로 보안 로직을 실행 시킨다. 이 때, 메시지가 서명이 되어 있을 경우, 메시지에 포함된 서명에 사용된 키에 대한 정보를 바탕으로 클라이언트의 공개키를 서버의 키 스토어에서 인출하여 서명을 검증한다. 메시지가 암호화 되어 있을 경우, 메시지에 포함된 암호화에 사용된 키 정보를 바탕으로 서버의 키 스토어에서 서버의 개인 키를 인출하여 암호를 해독한다.

서버 쪽에서 웹서비스 호출이 끝나고, 응답을 돌려 보내려 할 경우, 보안을 적용할 때는 위의 과정과 유사하게 진행된다. 웹 서비스는 jeus-webservices-dd.xml 에 <security> 엘리먼트에 설정된 값을 바탕으로 키 스토어에서 서버의 개인키나 클라이언트의 공개키를 꺼내어서 응답 메시지의 일부 혹은 전체를 서명하거나 암호화 하게 된다. 클라이언트 런타임은 jeus-web-dd.xml 의 <security> 엘리먼트에 설정된 값을 바탕으로 클라이언트 키 스토어에서 서버의 공개키나 클라이언트의 개인키를 꺼내어 이를 이용하여 응답 메시지를 검증, 해독하게 된다.

14.4.2.2 JEUS 와 타 벤더간 보안 메시지의 상호 호환 및 운영성

앞에서 언급했듯이 웹 서비스의 보안은 메시지 수준의 보안을 적용, 사용하게 되어 있다. 보안이 적용된 SOAP 메시지 내부에는 보안에 관련된 모든 정보가 들어있으며 이러한 정보들이 OASIS 의 웹 서비스 보안 표준을 만족하게 될 경우, 여러 벤터간의 상호 운영성이 보장 되게 된다.

JEUS 는 OASIS 의 보안 표준의 최신 버전인 웹서비스 보안 표준 1.0 을 준수하고 있으며, 따라서 이 표준을 준수하는 벤더간의 메시지 송수신 및 처리를 할 수 있다.

14.4.3 JEUS 웹 서비스 보안의 설정

JEUS 웹 서비스와 JEUS 웹 서비스 클라이언트에서 보안 메시지를 송수신하기 위해서는 JEUS 웹 서비스의 경우 jeus-webservices-dd.xml 에 <security> 엘리먼트를 추가 설정하여야 하고, JEUS 웹 서비스 클라이언트의 경우 jeus-web-dd.xml 에 <security>엘리먼트를 추가 설정하여야 한다.

14.4.3.1 JEUS 웹 서비스(서버)의 보안 설정

jeus-webservices-dd.xml 에 추가 되는 <security>엘리먼트는 웹 서비스의 포트 하나당 하나의 설정이 요구된다. <security>엘리먼트가 차지하는 위치는 다음과 같다.

<<*jeus-webservices-dd.xml*>>

```
<jeus-webservices-dd>
    <service>
        <webservice-description-name/>
        <port>
            <port-component-name/>
            <security>
                <request-receiver/>
                <response-sender/>
            </security>
        </port>
    </service>
</jeus-webservices-dd>
```

<security>의 하위 엘리먼트인 <request-receiver> 엘리먼트에는 서버가 클라이언트로부터 받는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다. <response-sender> 엘리먼트에는 서버가 클라이언트로 보내는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다. 보다 자세한 설명은 부록 D 를 참조한다.

14.4.3.2 JEUS 웹 서비스 클라이언트의 보안 설정

jeus-web-dd.xml 에 추가 되는 <security> 엘리먼트는 각 <port-info> 엘리먼트 하나 마다 하나의 설정이 필요하다. 그 구조는 다음과 같다.

<<*jeus-web-dd.xml*>>

```
<jeus-web-dd>
        <service-ref>
             <service-client>
                 <service-ref-name/>
                 <port-info>
                     <wsdl-port/>
                     <security>
                         <request-sender/>
                         <response-receiver/>
```

```
                                        </security>
                </port-info>
                …
            </service-client>
            <service-client>
                …
            </service-client>
            …
        </service-ref>
</jeus-webservices-client-dd>
```

<security>의 하위 엘리먼트인 <request-sender> 엘리먼트에는 클라이언트가 서버로 보내는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다. <response-receiver> 엘리먼트에는 클라이언트가 서버로부터 받는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다. 보다 자세한 설명은 부록 E 를 참조한다.

14.4.4 패스워드 콜백 클래스의 생성

JEUS 웹서비스에 보안을 적용하기 위해서는 패스워드 콜백 클래스의 생성이 필요하다. 이는 키 스토어에 저장된 개인 키의 암호를 설정하거나 UsernameToken 에 사용 될 암호를 설정하는 등의 작업 때문에 필요하다.

콜백 클래스는 javax.security.auth.callback.CallbackHandler 를 구현한 클래스여야 하며, 구현 예는 다음과 같다.

<< *PWCallback.java* >>

```
package jeustest.webservices.wssec.doall;

import com.tmax.ws.security.WSPasswordCallback;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

public class PWCallback implements CallbackHandler {
      public void handle(Callback[] callbacks)
            throws IOException, UnsupportedCallbackException {
          for (int i = 0; i < callbacks.length; i++) {
```

```
            if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc =
                      (WSPasswordCallback) callbacks[i];
                       checkPassword(pc);
            } else {
                throw new UnsupportedCallbackException(
                            callbacks[i],
                            "Unrecognized Callback");
            }
        }
    }
    public void checkPassword(WSPasswordCallback pc){
          String userId = pc.getIdentifier();
          if (pc.getUsage() ==
                WSPasswordCallback.USERNAME_TOKEN){
                if (userId.equals(
                     "16c73ab6-b892-458f-abf5-2f875f74882e")){
                       pc.setPassword("security2");
                }
                if (userId.equals("key_tmax1")){
                       pc.setPassword("keypass_tmax1");
                }

          }else if(pc.getUsage() ==
                WSPasswordCallback.DECRYPT){
                if (userId.equals(
                     "16c73ab6-b892-458f-abf5-2f875f74882e")){
                       pc.setPassword("security");
                }
                if (userId.equals("key_tmax1")){
                       pc.setPassword("keypass_tmax1");
                }
          }else if(pc.getUsage() ==
              WSPasswordCallback.SIGNATURE){
                if (userId.equals(
                     "16c73ab6-b892-458f-abf5-2f875f74882e")){
                       pc.setPassword("security");
                }
                if (userId.equals("key_tmax1")){
```

```
                        pc.setPassword("keypass_tmax1");
                    }
                }
            }
}
```

WSPasswordCallback.getUsage()는 인자로 넘겨 받는 콜백 파라미터가 어떤 경우에 사용되기 위한 것인지에 대한 값을 리턴해 준다. UsernameToken 의 암호를 설정하려 한다면 USERNAME_TOKEN 이라는 값을 가지고 있을 것이며, 암호화 된 메시지를 해독하기 위해 사용할 개인키에 대한 암호를 설정하려 한다면 DECRYPT 값을 리턴할 것이다. 그 외에도 서명을 하기 위해 필요한 개인키를 키 스토어에서 가져오기 위한 암호를 설정하거나, 세션 키를 설정하려 할 경우에도 이러한 구현이 필요하다.

이렇게 생성된 콜백 클래스는 jeus-webservices-dd.xml 이나 jeus-web-dd.xml 의 <password-callback-class> 엘리먼트에 클래스 이름을 설정함으로서 보안에 적용이 가능하다.

14.4.5 JEUS 웹 서비스 (서버)보안 적용 예제

이번 장에서는 스트링을 주고 받는 웹 서비스를 보안을 적용하여 구현해 보도록 한다.

14.4.5.1 키 스토어 생성

SOAP 메시지를 암호화 하고 서명하는 데 필요한 키쌍의 생성이 필요하며, 자바에서 제공하는 keytool 을 이용하여 간단하게 구현할 수 있다. keytool 외에도 여러 가지 방법이 있지만, 여기서는 keytool 을 이용하도록 한다.

샘플이 존재하는 폴더에서 keygen 을 수행하면 다음과 같은 작업을 수행한다.

```
keytool -genkey -alias JEUS_SERVER -keyalg rsa -keypass
key_password -keystore server-keystore.jks -storepass
keystore_password
```

- server-keystore.jks 라는 키 스토어 파일을 생성하고 키 스토어의 암호는 keystore_password 로 한다. 키 스토어에는 JEUS_SERVER 라는 별칭을 가진 개인키를 RSA 알고리즘으로 생성하며 개인키의 암호는 key_password 이다. 이 때 JEUS_SERVER 라는 개인키의 공개 키도 쌍으로 생성이 된다.

```
keytool -genkey -alias JEUS_CLIENT -keyalg rsa -keypass
key_password -keystore client-keystore.jks -storepass
keystore_password
```

- client-keystore.jks 라는 키 스토어 파일을 생성하고 키 스토어의 암호는 keystore_password 로 한다. 키 스토어에는 JEUS_CLIENT 라는 별칭을 가진 개인 키를 RSA 알고리즘으로 생성하며 개인키의 암호는 key_password 이다. 이 때 JEUS_CLIENT 라는 개인키의 공개 키도 쌍으로 생성이 된다.

```
keytool -export -alias JEUS_SERVER -storepass keystore_password -
keystore server-keystore.jks -file jeus_server.cert
```

- 서버쪽 키 스토어로 사용할 server-keystore.jks 라는 키 스토어에서 JEUS_SERVER 의 공개키를 꺼낸다.

```
keytool -import -file jeus_server.cert -keystore client-
keystore.jks -storepass keystore_password -alias JEUS_SERVER
```

- 클라이언트 키 스토어인 client-keystore.jks 에 JEUS_SERVER 라는 별칭으로 저장한다.

```
keytool -export -alias JEUS_CLIENT -storepass keystore_password -
keystore client-keystore.jks -file jeus_client.cert
```

- 클라이언트 키 스토어로 사용할 client-keystore.jks 라는 키 스토어에서 JEUS_CLIENT 의 공개키를 꺼낸다.

```
keytool -import -file jeus_client.cert -keystore server-
keystore.jks -storepass keystore_password -alias JEUS_CLIENT
```

- 서버 키 스토어인 server-keystore.jks 에 JEUS_CLIENT 라는 별칭으로 저장한다.

좀 더 상세한 설명은 keytool 의 도움말을 참조한다.

14.4.5.2 자바 클래스의 작성

기본적인 웹 서비스 구현은 보안이 적용되지 않았을 경우와 차이점이 없다.

<< Ping.java >>

```
package ping;

public interface Ping extends java.rmi.Remote {
```

```
      public String ping(String arg) throws
          java.rmi.RemoteException;
}
```

<< *PingImpl.java* >>

```
package ping;

public class PingImpl implements Ping {
      public String ping(String arg) throws
          java.rmi.RemoteException {
            return arg;
      }
}
```

여기에 추가적으로 개인키를 꺼내오거나 UsernameToken 의 암호를 설정하기 위한 콜백 클래스를 생성하여야 한다.

<< *PingPWCallback.java* >>

```
package ping;

import com.tmax.ws.security.WSPasswordCallback;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

public class PingPWCallback implements CallbackHandler{
      public void handle(Callback[] callbacks) throws
          IOException, UnsupportedCallbackException{
          for ( int i = 0 ; i<callbacks.length; i++){
              if(callbacks[i] instanceof WSPasswordCallback){
                  WSPasswordCallback pc =
                        (WSPasswordCallback) callbacks[i];

                  if(pc.getUsage()==
                      WSPasswordCallback.USERNAME_TOKEN){
                      pc.setPassword("usertoken_password");
                  }
```

```
                if(pc.getUsage()==WSPasswordCallback.DECRYPT){
                    pc.setPassword("key_password");
                }

                if(pc.getUsage()==
                    WSPasswordCallback.SIGNATURE){
                    pc.setPassword("key_password");
                }
            }
        }
    }
}
```

14.4.5.3 배치 서술자(jeus-webservices-dd.xml)의 작성

남은 추가 작업은 배치 서술자의 작성 작업이다. 다른 부분은 일반적인 웹서비스 배치 서술자 작성 작업과 동일하며, JEUS 특정 설정을 담당하는 jeus-webservices-dd.xml 에서만 추가 작업을 해주면 된다.

다음은 서버쪽에서 클라이언트의 메시지를 처리하는 부분의 설정이다. 파일 전체의 내용은 샘플을 참조한다.

<< *jeus-webservices-dd.xml* >>

```
…
<security>
    <request-receiver>
        <action-list>UsernameToken Signature Encrypt
        </action-list>
        <password-callback-class>ping.PingPWCallback
        </password-callback-class>
        <decryption>
            <keystore>
                <key-type>jks</key-type>
                <keystore-password>keystore_password
                </keystore-password>
                <keystore-filename>server-keystore.jks
                </keystore-filename>
            </keystore>
        </decryption>
        <signature-verification>
```

```
            <keystore>
                <key-type>jks</key-type>
                <keystore-password>keystore_password
                 </keystore-password>
                <keystore-filename>server-keystore.jks
                 </keystore-filename>
            </keystore>
        </signature-verification>
    </request-receiver>
    …
</security>
…
```

14.4.5.4 패키징과 배치

웹 서비스 보안 모듈의 패키징과 배치 방식은 기존의 웹 서비스 모듈의 방식과 크게 다르지 않다. 추가적으로 키 스토어(server-keystore.jks)를 컨텍스트의 클래스 경로안에 두어야 하는 점이 다르다. 혹은 jeus-webservices-dd.xml의 <keystore-filename>엘리먼트에 절대경로를 포함한 키 스토어 파일 이름을 적어주는 방법도 있다.

샘플 홈 디렉토리에서 다음과 같이 입력하면 패키징 작업이 완료된다.

```
ant wsear
```

이 모듈을 JEUS 서버에 배치하면 보안이 적용된 웹 서비스를 시작할 수 있게 된다.

14.4.6 JEUS 웹 서비스 클라이언트 보안 적용 예제

JEUS 의 웹 서비스 J2EE 클라이언트의 경우도 서버쪽의 작업과 유사하게, 일반적인 웹 서비스 클라이언트 생성 작업에 콜백 클래스와 설정 파일 작업, 키 스토어 패키징 작업이 추가적으로 더해진다.

14.4.6.1 키 스토어 생성

키 스토어는 14.3.5.1 에서 생성한 키 스토어 중, client-keystore.jks 를 사용하도록 한다.

14.4.6.2 자바 클래스의 작성

다음은 클라이언트인 pingClient.jsp 파일의 일부이다. 웹 서비스의 보안 적용은 웹 서비스의 포트 단위로 적용이 되므로, 반드시 호출하고자 하는 포트의 이름을 명시해 주어야 한다.

<< *pingClient.jsp*>>

```
…
InitialContext jndiContext = new InitialContext();

Service service = (Service)jndiContext.lookup(
   "java:comp/env/service/PingSecurityService");
QName portName = new QName("urn:PingSecurityService","PingPort");
java.rmi.Remote port = service.getPort(portName, Ping.class);
Ping pingPort = (Ping)port;
ret = pingPort.ping(msgToSend);
…
```

클라이언트도 서버쪽과 마찬가지로 콜백클래스를 생성한다. 여기서 사용하는 콜백 클래스는 서버쪽에서 생성했던 콜백클래스를 재사용 하도록 한다.

14.4.6.3 배치 서술자(jeus-web-dd.xml)의 작성

다른 설정파일 작업은 일반적인 웹서비스 클라이언트 배치 서술자 작성 작업과 동일하며, JEUS 특성 설정을 담당하는 jeus-web-dd.xml 에서만 추가 작업을 해주면 된다.

다음은 클라이언트쪽에서 서버로 전송할 메시지를 처리하는 부분의 설정이다. 파일 전체의 내용은 샘플을 참조한다.

<< *jeus-web-dd.xml* >>

```
…
<security>
    <request-sender>
        <action-list>
            UsernameToken Signature Encrypt</action-list>
        <password-callback-class>
            ping.PingPWCallback</password-callback-class>
        <user>JEUS_CLIENT</user>
        <signature-infos>
            <signature-info>
```

```
                <keyIdentifier>DirectReference</keyIdentifier>
                <keystore>
                    <key-type>jks</key-type>
                    <keystore-password>
                       keystore_password</keystore-password>
                    <keystore-filename>client-keystore.jks
                    </keystore-filename>
                </keystore>
            </signature-info>
        </signature-infos>
        <encryption-infos>
            <encryption-info>
                <encryptionUser>JEUS_SERVER</encryptionUser>
                <keyIdentifier>DirectReference</keyIdentifier>
                <keystore>
                    <key-type>jks</key-type>
                    <keystore-password>keystore_password
                    </keystore-password>
                    <keystore-filename>client-keystore.jks
                    </keystore-filename>
                </keystore>
            </encryption-info>
        </encryption-infos>
    </request-sender>
    …
</security>
…
```

14.4.6.4 패키징과 배치

웹 서비스 J2EE 클라이언트 보안 모듈의 패키징과 배치 방식은 기존의 웹 서비스 J2EE 클라이언트 모듈의 방식과 크게 다르지 않다. 추가적으로 키 스토어(client-keystore.jks)를 컨텍스트의 클래스 경로안에 두어야 하는 점이 다르다. 혹은 jeus-web-dd.xml 에 <keystore-filename>엘리먼트에 절대경로를 포함한 키 스토어 파일 이름을 적어주는 방법도 있다.

샘플 홈 디렉토리에서 다음과 같이 입력하면 패키징 작업이 완료된다.

```
ant wsear
```

이 모듈을 JEUS 서버에 배치하면 보안이 적용된 웹 서비스를 시작할 수 있게 된다.

14.4.7 웹서비스 보안 API 를 이용한 JEUS 웹서비스 클라이언트의 작성

JEUS 웹 서비스는 J2EE 환경과 J2SE 환경에서 모두 적용이 가능한 웹 서비스 클라이언트 보안 API 를 제공하여, 보다 쉽게 웹 서비스 클라이언트 작성을 가능하게 해준다.

다음은 API 를 이용한 웹서비스 보안 클라이언트이다.

<< *pingClient.jsp*>>

```
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>

<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page import="javax.xml.namespace.QName" %>

<%@ page
import="jeus.webservices.wssecurity.SecurityConfiguration" %>
<%@ page import="jeus.webservices.wssecurity.Keystore" %>

<%@ page import="import ping.*" %>
<%@ page errorPage="/error.html" %>

<%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString="";
%>

<%
        try {
            Keystore keystore = new Keystore
              ("JKS", "keystore_password", "client-keystore.jks");
```

```
Keystore truststore = new
   Keystore("JKS","keystore_password", "client-
   keystore.jks");

InitialContext jndiContext = new InitialContext();

Service service = (Service)jndiContext.lookup(
   "java:comp/env/service/PingSecurityService");
QName portName = new QName(
   "urn:PingSecurityService", "PingPort");

SecurityConfiguration sconfig =
   new SecurityConfiguration(service, portName);
sconfig.setUsername("JEUS_CLIENT");
sconfig.setRequestPasswordCallbackClass(
   "ping.PingPWCallback");

sconfig.addUTRequest(true, true);
sconfig.addSignRequest(null,
   "DirectReference", keystore);
sconfig.addEncryptRequest(null, "DirectReference",
   truststore, "JEUS_SERVER", null);

sconfig.setResponsePasswordCallbackClass(
   "ping.PingPWCallback");
sconfig.addUTResponse();
sconfig.addSignResponse(truststore);
sconfig.addDecryptResponse(keystore);

java.rmi.Remote port =
   service.getPort(portName, Ping.class);
Ping pingPort = (Ping)port;

((javax.xml.rpc.Stub)pingPort)._setProperty(
   javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
   "http://localhost:8088/PingSecurity/PingSecuritySe
   rvice");
```

```
            System.out.println("Send : " + msgToSend);
            ret = pingPort.ping(msgToSend);
            System.out.println("You received : " + ret);
        } catch (Exception e) {
            exceptionString = e.toString();
            e.printStackTrace();
        }
%>
<%= "Result is "+ret+"......"
%>
<%= exceptionString
%>
```

위의 예제 중, 굵은 글자로 표시된 부분이 웹 서비스 보안 설정에 관한 부분이다. 보안 설정을 위해 jeus.webservices.wssecurity.SecurityConfiguration 객체를 생성하고, 보안 요청 메시지를 작성하기 위해서는 기본적으로 사용자 이름과 패스워드 콜백 클래스를 지정하여야 한다. (콜백 클래스 생성에 관해서는 14.3.4 장, “패스워드 콜백 클래스의 생성”을 참조한다.)

그 외에 사용자 이름 토큰이나 서명, 암호화에 관한 설정을 추가하고자 할 때 각각의 API 를 이용하여 추가 할 수 있다. 이 때 키 스토어에 대한 설정이 필요할 경우에는 jeus.webservices.wssecurity.Keystore 클래스를 생성하여 설정이 가능하다.

14.4.7.1 SecurityConfiguration 객체의 생성

```
public SecurityConfiguration(javax.xml.rpc.Service service, QName
  portName) throws ConfigurationException
```

웹 서비스 보안 API 를 적용하기 위해 가장 먼저 생성하여야 하는 객체이다. service 파라미터에는 J2EE 클라이언트에서 작성할 경우에는 JNDI 를 통해 얻어오는 Service 객체를 넣어주고, J2SE 클라이언트에서 작성할 경우에는 Service_Impl() 객체를 통해 생성한 객체를 인자로 넣어준다. portName 파라미터에는 WSDL 에 공개된 포트의 큐네임(QName)을 넣어준다.

14.4.7.2 Keystore 객체의 생성

```
public Keystore(String keyType, String keystorePassword, String
  keystoreFilename)
```

웹 서비스 보안 API 를 적용하기 위해서는 서명 작업이나 암호화 작업의 진행을 위해 키스토어 객체의 생성이 필요하다. keystoreFilename 파라미터에는 키 스토어의 파일 이름을 입력하고, keystorePassword 파라미터에는 키 스토어 파일의 암호를 입력한다. keyType 파라미터에는 키 스토어가 저장하고 있는 키의 타입을 입력한다. 지원되는 키의 타입은 “JKS”나 “PKCS12” 둘 중에 하나이다.

14.4.7.3 사용자 이름의 설정

```
public void setUsername(String username)
```

웹 서비스 보안 요청 메시지를 작성하기 위해서는 유저 네임 토큰에 들어갈 이름이나 서명에 사용될 개인 키의 이름이 필요하며 이를 이 함수를 통해 설정할 수 있다.

14.4.7.4 콜백 클래스 이름의 설정

```
public void setRequestPasswordCallbackClass(String classname)
public void setResponsePasswordCallbackClass(String classname)
```

웹 서비스 보안 요청이나 응답 메시지를 주고 받는 과정 중에, 유저 네임 토큰에 사용되는 암호 설정이나, 개인키에 대한 암호 설정을 할 경우 패스워드 콜백 클래스들을 생성하여야 하며, 생성된 콜백 클래스의 이름을 이 함수들을 이용하여 지정한다.

14.4.7.5 TimeStamp 의 처리

```
public void addTimeStampRequest()
```

웹 서비스 보안 요청 메시지에 TimeStamp 엘리먼트를 추가하려 할 경우 사용한다.

```
public void addTimeStampRequest(int timeToLive)
```

웹 서비스 보안 요청 메시지에 유효 기간을 별도로 설정하려 할 경우 이 함수를 사용한다. 초(second)단위이며, 별도로 설정하지 않았을 경우, 기본 설정값은 300 초이다.

```
public void addTimeStampResponse()
```

웹 서비스 보안 응답 메시지에 TimeStamp 엘리먼트가 있기를 기대하는 경우 이 함수를 사용한다.

```
public void addTimeStampResponse(int timeToLive)
```

웹 서비스 보안 응답 메시지의 TimeStamp 엘리먼트에 설정된 유효 기간이 인자로 입력한 시간 이내이어야 한다.

14.4.7.6 UsernameToken 의 처리

```
public void addUTRequest(boolean addNonceCreated, boolean
  passwordDigest)
```

웹 서비스 보안 요청 메시지에 UsernameToken 엘리먼트를 추가할 경우 이 함수를 사용한다. 패스워드는 평문이나 다이제스트 형태로 만들어 질 수 있으며, 평문으로 만들 경우에는 Nonce 와 Created 항목을 추가할 것인지를 선택할 수 있다.

```
public void addUTResonse()
```

웹 서비스 보안 응답 메시지에 UsernameToken 엘리먼트가 있기를 기대하는 경우에 이 함수를 사용한다.

14.4.7.7 서명의 처리

```
public void addSignRequest(QName signPart, String keyIdentifier,
  Keystore keystore) throws SecurityConfigurationException
```

웹 서비스 보안 요청 메시지에 특정 부분에 서명을 하고자 할 때 이 함수를 사용한다. signPart 파라미터를 null 로 설정하면, 기본적으로 SOAP 메시지의 몸체 부분을 기본적으로 서명한다. keyIdentifier 파라미터는 “IssuerSerial”, “DirectReference”, “SKIKeyIdentifier”, “X509KeyIdentifier”중의 하나를 입력한다. keystore 파라미터는 서명에 필요한 개인키를 저장하고 있는 키스토어 객체를 입력한다.

```
public void addSignResponse(Keystore keystore)
```

웹 서비스 보안 응답 메시지에 서명이 있기를 기대하는 경우에 이 함수를 사용한다. keystore 파라미터는 서명 검증에 필요한 공개 키를 저장하고 있는 키스토어 객체를 입력한다.

14.4.7.8 암호화의 처리

```
public void addEncryptRequest(QName encPart, String
  keyIdentifier, Keystore keystore, String encryptUser, String
  algorithm) throws SecurityConfigurationException
```

웹 서비스 보안 요청 메시지에 특정 부분을 암호화하고자 할 때 이 함수를 사용한다. encPart 파라미터는 암호화하고자 하는 SOAP 메시지의 부분이며, QName 형태로 입력한다. 만약 null 로 설정될 경우, 기본적으로 SOAP 메시지의 몸체 부분을 기본적으로 암호화한다. keyIdentifier 파라미터는 “IssuerSerial”, “DirectReference”, “SKIKeyIdentifier”, “X509KeyIdentifier”, “EmbeddedKeyName”중 하나를 입력한다. keystore 파라미터는 암호화에 필요한 공개키를 저장하고 있는 키스토어 객체를 입력한다. encryptUser 파라미터는 암호화에 사용할 공개키의 별칭을 입력한다. algorithm 파라미터는 암호화에 사용될 알고리즘이며, “AES_128”, “AES_256”, “TRIPLE_DES”, “AES_192” 네 가지중 하나를 입력한다.

```
public void addEncryptRequest(QName encPart, String
  keyIdentifier, String embeddedKeyCallbackClass, String keyName,
  String encryptUser, String algorithm)
```

웹 서비스 보안 요청 메시지에 특정 부분을 암호화 하고자 할 때 사용하며, 특히 특정 콜백 클래스에 바이트의 배열로 키를 설정하여 그 키를 이름으로 불러내려 할 때, 이 함수를 사용한다.

```
public void addDecryptResponse(Keystore keystore)
```

웹 서비스 보안 응답 메시지에 암호화 된 부분이 있기를 기대할 때 이 함수를 사용한다. keystore 파라미터에는 암호화를 풀기 위한 개인키를 가지고 있는 키 스토어 객체를 입력한다.

14.5 JEUS 웹 서비스의 접근 제어 설정

JEUS 웹 서비스는 접근이 허용된 사용자만이 웹 서비스를 호출할 수 있도록 접근 제어 설정을 할 수 있으며, 웹 서비스 Back-end 에 따라 각각 설정법이 다르다. 이번 장에서는 웹 서비스 접근 제어 설정법과 접근 제어 설정이 된 웹 서비스를 호출하는 방법에 대해 설명한다.

14.5.1 자바 클래스 웹 서비스의 접근 제어 설정

특정한 URL 로의 접근을 제한함으로써 웹 서비스로의 접근을 제한할 수 있으며, 이를 위해서는 web.xml 과 jeus-web-dd.xml 같은 웹 응용프로그램의 배치 서술자에 보안 정보를 추가해 주어야 한다. 또 보안 도메인의 설정 또한 필요하다.

14.5.1.1 보안 도메인의 설정

접근 제어 설정을 하기 위해서는 먼저 '서브젝트'의 등록이 필요하다. JEUS_HOME\config\{NODE_NAME}\security\{SECURITY_DOMAIN_NAME}\subjects.xml 에 다음과 같이 서브젝트를 등록한다. 이 때 사용자의 이름은 "j2ee"이고, 암호는 "j2ee"이며, 암호는 base64 인코딩하여 등록한다.

<< *subjects.xml* >>

```
<subjects>
     <subject>
        <principal>
            <name>j2ee</name>
        </principal>
        <credential>
            <property>
                <name>password</name>
                <value>ajJlZQ==</value>
            </property>
        </credential>
     </subject>
</subjects>
```

14.5.1.2 배치 서술자의 작성

jeus-web-dd.xml 에 다음과 같이 role-mapping 을 추가한다.

<< *jeus-web-dd.xml* >>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
        <docbase>BA_DocLitEchoService</docbase>
        <role-mapping>
            <role-permission>
                <principal>j2ee</principal>
                <role>Administrator</role>
            </role-permission>
        </role-mapping>
</jeus-web-dd>
```

그리고 web.xml 에 다음과 같이 보안 관련 설정을 추가한다.

<< *web.xml* >>

```
<web-app…>
```

```
    …
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>MySecureBit</web-resource-name>
            <url-pattern>/BA_DocLitEchoService</url-pattern>
            <http-method>POST</http-method>
            <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>Administrator</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>default</realm-name>
    </login-config>
</web-app>
```

위와 같이 설정해 주면 URL 이 /BA_DocLitEchoService 인 모든 요청에 대해 사용자가 “j2ee”이며 암호가 “j2ee”일 경우에만 접근이 허용 된다.

14.5.2 EJB 웹 서비스의 접근 제어 설정

EJB 웹 서비스의 접근 제어 또한 자바 클래스 웹 서비스의 접근 제어와 마찬가지로 특정한 URL 로의 접근을 제한함으로써 웹 서비스로의 접근을 제한할 수 있다. 이를 위해서는 ejb-jar.xml 과 jeus-ejb-dd.xml 같은 EJB 배치 서술자와 jeus-webservices-dd.xml 과 같은 웹 서비스 배치 서술자에 보안 정보를 추가해 주어야 한다. 또 보안 도메인의 설정 또한 필요하다.

14.5.2.1 보안 도메인의 설정

접근 제어 설정을 하기 위해서는 먼저 ‘서브젝트’의 등록이 필요하다. JEUS_HOME\config\{NODE_NAME}\security\{SECURITY_DOMAIN_NAME}\subjects.xml 에 다음과 같이 서브젝트를 등록한다. 이 때 사용자의 이름은 “j2ee”이고, 암호는 “j2ee”이며, 암호는 base64 인코딩하여 등록한다.

<< *subjects.xml* >>

```
<subjects>
     <subject>
        <principal>
            <name>j2ee</name>
        </principal>
        <credential>
            <property>
                <name>password</name>
                <value>ajJlZQ==</value>
            </property>
        </credential>
     </subject>
</subjects>
```

14.5.2.2 배치 서술자의 작성

jeus-ejb-dd.xml 에 다음과 같이 role-permission 을 추가한다.

<< *jeus-ejb-dd.xml* >>

```
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
        <role-permission>
            <principal>j2ee</principal>
            <role>Administrator</role>
        </role-permission>
    </module-info>
    <beanlist>
        …
    </beanlist>
</jeus-ejb-dd>
```

그리고 ejb-jar.xml 에 다음과 같이 보안 관련 설정을 추가한다.

<< *ejb-jar.xml* >>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar …>
   <display-name>AddressEjb</display-name>
   <enterprise-beans>
            …
```

```
   </enterprise-beans>
   <assembly-descriptor>
     <security-role>
       <role-name>Administrator</role-name>
     </security-role>
     <method-permission>
       <role-name>Administrator</role-name>
       <method>
          <ejb-name>AddressEJB</ejb-name>
          <method-intf>ServiceEndpoint</method-intf>
          <method-name>listAll</method-name>
       </method>
     </method-permission>
   </assembly-descriptor>
</ejb-jar>
```

그리고 추가적으로 다음과 같은 설정을 웹 서비스 배치 서술자에 해 줄 수 있으며, 별도로 하지 않을 경우에는 기본적으로 설정된 값으로 수행된다. SSL을 사용하여 주고 받는 데이터의 무결성을 보장하려면 ejb-transport-guarantee를 “INTEGRAL” 값을 주고, 데이터의 기밀성과 무결성을 모두 보장하려면 “CONFIDENTIAL”로 설정해 주어야 한다. 물론 이 경우에는 HTTPS로 통신을 주고 받도록 별도의 HTTP 리스너의 설정이 필요할 것이다.

<< *jeus-webservices-dd.xml* >>

```
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <ejb-context-path>webservice</ejb-context-path>
    <ejb-login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>default</realm-name>
    </ejb-login-config>
    <service>
        <webservice-description-name>…</webservice-description-
name>
        <port>
            <port-component-name>…</port-component-name>
            <ejb-endpoint-url>… </ejb-endpoint-url>
```

```
            <ejb-transport-guarantee>CONFIDENTIAL</ejb-transport-
guarantee>
        </port>
    </service>
</jeus-webservices-dd>
```

14.5.3 접근 제어(Basic Authentication)가 설정된 웹 서비스의 호출

접근 제어가 설정된 웹 서비스의 호출을 하기 위해서는 WSDL 에 접근하여 그 내용을 바탕으로 스텁을 생성하고, 스텁을 이용하여 웹 서비스를 호출하여야 하는데, 이 경우에 사용자 이름과 암호를 요구할 경우 설정해 주는 작업이 필요하다.

14.5.3.1 WSDL 로부터 스텁 생성

ANT 태스크를 이용할 경우, 다음과 같이 설정해 주면 된다.

<< *build.xml* >>

```
<project>
…
  <target name="wsdl2java">
    <mkdir dir="${classes.dir}"/>
      <wsdl2java …
        username="j2ee"
        password="j2ee"
        wsdl="${jaxrpc.wsdl}">
        <classpath …/>
    </wsdl2java>
  </target>
</project>
```

14.5.3.2 접근 제어가 설정된 웹 서비스의 클라이언트의 작성

JAX-RPC 웹 서비스 클라이언트가 스스로 인증 하기 위해서는 다음과 같은 두가지의 설정이 클라이언트 프로그램내에 삽입되어야 한다.

- javax.xml.rpc.security.auth.username
- javax.xml.rpc.security.auth.password

다음 예는 실제로 프로그램 내에 어떻게 위 설정들이 삽입 될 수 있는지를 보여준다.

```
Echo port = // … 서비스 엔드포인트 인터페이스의 획득
((javax.xml.rpc.Stub)port)._setProperty("javax.xml.rpc.security.a
   uth.username", "j2ee");
((javax.xml.rpc.Stub)port)._setProperty("javax.xml.rpc.security.a
   uth.password", "j2ee");
String s = port.echoString("JEUS");
```

15 결론

지금까지 이 매뉴얼에서 기술했던 JEUS 웹서비스에 관한 내용을 마무리 하도록 하겠다.

- JEUS 웹서비스의 개요를 알아보았다.
- JEUS 웹서비스의 설계 방식을 알아보았다.
- 웹서비스 backend 컴포넌트인 Java 클래스와 Stateless 세션 빈을 어떻게 작성, 패키지, 배치하는지를 보았다
- WSDL 문서를 이용해 작성된 웹서비스 클라이언트를 어떻게 작성, 패키지, 실행시키는지를 보았다.
- JAX-RPC value type, JAX-RPC holder 클래스와 exception 들을 이용한 진보된 웹서비스를 어떻게 작성하는지를 보았다.
- SOAP 메시지 핸들러의 생성 방법을 알아보았다.
- UDDI 이용법을 알아보았다.
- J2EE 1.4 웹서비스를 지원하기 위한 매핑 파일의 작성에 대해 알아보았다.
- 웹서비스의 보안의 개요와 적용 방법에 대해 알아보았다.

본 매뉴얼의 나머지 부분은 부록으로 할애하였다.

A JEUS 웹 서비스 Ant Task 참조

A.1 소개

이 부록에서는 JEUS 에서 웹서비스 생성과 웹서비스 클라이언트를 위해 제공하는 Ant Task 에 관해서 설명한다. Ant 툴 기능을 사용하기 위해서는 Apache Ant 1.6.1 이상을 인스톨 해야 한다. 'http://ant.apache.org'에서 Apache Ant 를 다운로드할 수 있다. JEUS 에서 제공하는 Task 는 아래와 같은 것이 있다.

- java2wsdl : Java 클래스로부터 WSDL 파일과 JAX-RPC 매핑 파일을 생성한다.
- wsdl2java: WSDL 파일로부터 클라이언트측 Java 스텁 소스 파일들과 서버측 웹서비스 인터페이스 Java 소스 파일을 생성한다.

A.2 java2wsdl

A.2.1 설명

'java2wsdl' task 는 service endpoint interface 클래스(그리고 임의의 Java 로 구현한 클래스)로부터 아래와 같은 것을 생성한다

- 웹서비스의 WSDL 파일
- JAX-RPC 매핑 파일

A.2.2 파라미터

표 7. < java2wsdl> 의 속성들

속성	설명	필수여부
`destDir`	생성된 WSDL 파일이 놓일 디렉토리의 절대 경로를 지정한다.	예

configfilepath	웹서비스 설정 파일의 경로를 지정한다.	예
verbose	Verbose 메시지를 출력한다	아니오

A.2.3 Nested Element

<java2wsdl> 은 ant 의 <classpath> 엘리먼트를 가지고 있다.

A.3 wsdl2java

A.3.1 설명

'wsdl2java' task 는 웹서비스의 WSDL 로부터 아래의 두 개중에 하나를 생성한다

- 클라이언트측 웹서비스의 스텁 자바 소스 코드들

또는

- 서버측 웹서비스의 인터페이스 자바 소스 코드들

A.3.2 파라미터

표 8. <wsdl2java> element 의 속성

속성	설명	필수여부
wsdl	Java 소스 파일을 생성하기 위해서 사용되는 WSDL 의 URL 이나 절대 경로를 지정한다.	예
mode	Java 소스 파일을 생성하기 위해서 사용되는 mode 를 지정한다. (gen:client, gen:server, gen, import:client, import:server, import)	예

속성	설명	필수여부
destDir	생성된 Java 파일이 놓일 디렉토리의 절대 경로를 지정한다.	예
classDestDir	컴파일된 클래스 파일이 생성될 디렉토리를 지정한다.	아니오
inputMapping	자바 클래스를 생성하기 위하여 사용되는 입력 JAX-RPC 매핑 파일을 지정한다.	아니오
package	WSDL 의 모든 네임스페이스 URI 에 대한 자바 패키지 이름을 지정한다.	아니오
outputMapping	입력 WSDL 에 대한 출력 JAX-RPC 매핑 파일을 지정한다.	아니오
doCompile	“true”일때 생성된 Java source 파일들은 컴파일 된다,. Default 값은 “ true”이다.	아니요.
username	WSDL 의 URL 로 접근할 때 필요한 사용자 이름	아니오
password	WSDL 의 URL 로 접근할 때 필요한 패스워드	아니오
keepSrc	“true”로 지정하게 되면, 생성된 Java source 파일들은 삭제 되지 않는다. Default 값은 “true”이다.	아니요.
nowrapped	“true”로 지정하게 되면, WSDL 에 대한 wrapped 모드 탐지를 사용할 수 없게 한다. Default 값은 “false”이다.	아니오

속성	설명	필수여부
dataHandlerOnly	“true”로 지정하게 되면, MIME 타입에 대하여 javax.activation.DataHandler 를 적용하게 한다. Default 값은 “false”이다.	아니오
nodatabinding	모든 WSDL massage part 에 대하여 javax.xml.soap.SOAPElement 를 적용하게 한다. Default 값은 “false”이다.	아니오.
soapver	stub/tie 클래스에서 사용되는 SOAP 버전을 지정한다. Default 값은 ‘11’이다. (11: SOAP 1.1, 12:SOAP 1.2)	아니오
verbose	Verbose 출력은 “true”일때 가능하다. Default 값은 “false”이다.	아니오

A.3.3 Nested Element

<wsdl2java>는 중첩된 ant 의 <classpath> element 와 <mapping> element 를 가지고 있다. <wsdl2java> Ant Task 의 구조는 다음과 같다:(+ 는 한개 이상의 element 를 가질 수 있음을 의미 한다.)

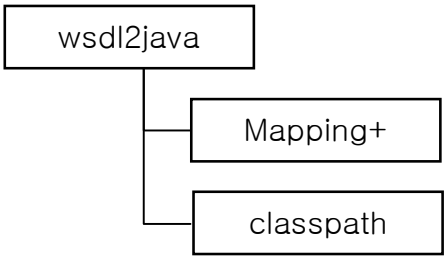


그림 22. <wsdl2java> Ant Task 의 구조

A.3.4 <mapping>

A.3.4.1 설명

Java 패키지와 WSDL 네임스페이스 사이의 매핑. 사용자가 <mapping> element 를 생략하면, 모든 네임스페이스 URI 는 <wsdl2java> element 인 package attribute 에 기술된 패키지로 매핑 된다.

Note. 만약 사용자가 <wsdl2java> element 의 package attribute 를 명시하면, 이 속성 값이 <mapping> element 로 설정된 값에 우선한다. (즉 <mapping> element 로 설정된 값은 소용없어 진다.)

A.3.4.2 파라미터

표 9. <mapping> element 의 속성

속성	설명	필수여부
package	Java 패키지의 이름.	예
namespace	WSDL 의 namespace URI.	예

B JEUS 웹 서비스 커맨드 라인 툴 참조

B.1 소개

이 부록은 JEUS 웹서비스가 웹서비스 생성과 웹서비스 클라이언트를 위해 제공하고 있는 커맨드 라인툴에 관한 것이다. 그 Tasks 는 아래와 같다:

- java2wsdl : 자바 클래스로부터 WSDL 파일과 JAX-RPC 매핑 파일을 생성한다.
- wsdl2java :WSDL 파일로부터 클라이언트측 자바 스텁 소스 파일과 서버측 웹서비스 인터페이스 자바소스 파일을 생성한다.

B.2 java2wsdl

B.2.1 설명

'java2wsdl' task 는 service endpoint interface 자바클래스(또는 구현 클래스)로 부터 다음과 같은 리소스를 생성한다.

- 웹서비스의 WSDL 파일
- JAX-RPC 매핑 파일

B.2.2 사용

커맨드 라인에서 다음과 같은 커맨드를 입력한다.

```
Usage: java2wsdl [options] configuration_file

where [options] include:
-classpath <path>   specify where to find input class files
-cp <path>          same as -classpath <path>
-d <directory>      specify where to place generated output files
-verbose            [optional] turn verbose mode on
```

B.2.3 파라미터

표 10. java2wsdl 커맨드의 옵션들

속성	설명
`-classpath <path>`	입력 자바 클래스 파일들을 찾기 위한 경로를 기술한다.
`-cp <path>`	`-classpath <path>` 와 같다.
`-d <directory>`	결과 파일이 생성될 디렉토리 지정한다.
`-verbose`	Verbose 메시지를 출력한다

B.3 wsdl2java

B.3.1 설명

'wsdl2java' task 는 웹서비스의 WSDL 파일로 부터 다음의 둘 중 하나를 생성한다.

- 클라이언트측 웹서비스 스텁 자바 소스 코드들

또는

- 서버측 웹서비스 인터페이스 자바 소스 코드들

B.3.2 사용

커맨드 라인에서 다음과 같은 커맨드를 입력한다.

```
Usage: wsdl2java mode [options] wsdlURI

where mode include:
-gen:client      generate client artifacts
                 (portable, stub, helper...)
-gen:server      generate server artifacts
                 (portable, tie, helper...)
-gen             same as -gen:client
-import:client   generate client JSR-109 portable artifacts
-import:server   generate server JSR-109 portable artifacts
-import          same as -import:client

where [options] include:
* destination directory
-d               specify where to put output files
-cd              specify where to put compiled class files
                 If not specifed, the compile class files will be
                 put in where '-d' specifies

* WSDL and Java mapping
-inputmapping    specify the input JSR-109 JAX-RPC mapping file
                 (used for generating Java artifacts)
```

```
-package          specify the java package name to which
                  all namespaceURI in the WSDL map
-ns2pkg NS=PKG    specify the namespaceURI and java package
                  times name mapping
                  (NS : namespaceURI, PKG : java package name)
                  This option can be used serveral times
* output file
-outputmapping    specify the output JSR-109 JAX-RPc mapping file
                  for the input WSDL
                  This option can not be used with '-inputmapping'
-compile          compile generated Java source files
                  ('tools.jar' must be in the classpath)
-nokeepsrc        delete generated java source files

* artifact generation options
-nowrapped        disable wrapped mode detection for the WSDL
-datahandleronly force javax.activation.DataHandler for MIME
                  types
-nodatabinding    force javax.xml.soap.SOAPElement for all WSDL
                  message part
-soapver          specify SOAP version used in stub/tie class.
                  (11: SOAP 1.1, 12: SOAP 1.2)
* other options
-username         username to access the WSDL-URI
-password         password to access the WSDL-URI
```

B.3.3 파라미터

표 11. wsdl2java 커맨드 의 옵션들

속성	설명
-gen:client	클라이언트측 자바 클래스를 생성한다.
-gen:server	서버측 자바 클래스를 생성한다.
-gen	-gen:client 와 같다.

속성	설명
-import:client	클라이언트측 포터블 아티팩트(portable artifact)를 생성한다.
-import:server	서버측 포터블 아티팩트(portable artifact)를 생성한다.
-import	-import:client 와 같다.
-d	결과 파일이 생성될 디렉토리 지정한다.
-cd	컴파일된 클래스 파일이 생성될 디렉토리 지정한다.
-inputmapping	자바 클래스를 생성하기 위하여 사용되는 입력 JAX-RPC 매핑 파일을 지정한다.
-package	WSDL 의 모든 네임스페이스 URI 에 대한 자바 패키지 이름을 지정한다.
-ns2pkg NS=PKG	네임스페이스 URI 에 대한 자바 패키지 이름을 지정한다. (NS: 네임스페이스 URI, PKG: 자바 패키지 이름)
-outputmapping	입력 WSDL 에 대한 출력 JAX-RPC 매핑 파일을 지정한다.
-compile	생성된 자바 파일을 컴파일한다.
-nokeepsrc	생성된 자바 소스 파일을 유지하지 않고 삭제한다.
-nowrapped	WSDL 에 대한 wrapped 모드 탐지를 사용할 수 없게 한다.

속성	설명
-datahandleronly	MIME 타입에 대하여 javax.activation.DataHandler 를 적용하게 한다.
-nodatabinding	모든 WSDL massage part 에 대하여 javax.xml.soap.SOAPElement 를 적용하게 한다.
-soapver	stub/tie 클래스에서 사용되는 SOAP 버전을 지정한다. (11: SOAP 1.1, 12:SOAP 1.2)
-username	WSDL-URI 에 접근하기 위한 사용자 이름
-password	WSDL-URI 에 접근하기 위한 암호

B.4 tcpmon

B.4.1 설명

tcpmon 프로그램은 송수신하는 TCP 패킷을 조회하는 프로그램이다. tcpmon 은 HTTP 상의 SOAP 메시지를 조회하는데에 사용될 수 있다.

B.4.2 실행

tcpmon 을 실행하면 다음과 같은 화면이 나타난다.

C:\jeus\bin> tcpmon

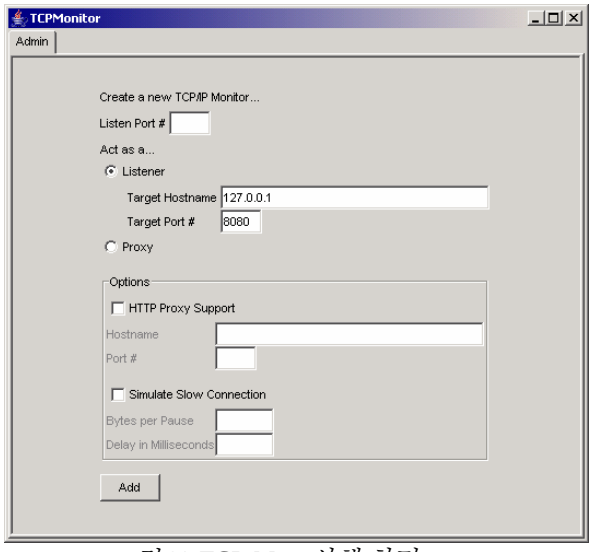


그림 23 TCP Mon 실행 화면

Tcpmon 은 다음 두가지 모드로 동작한다

- Listener
- Proxy

Listener 는 tcpmon 이 listen port 로 요청을 받아서 지정된 target host 와 target port 로 받은 요청을 다시 전송하고 응답을 받아서 원래 요청한 커넥션으로 응답을 전송한다.

Proxy 는 tcpmon 을 일반적인 HTTP Proxy 처럼 동작하게 한다.

B.4.3 Listener 모드의 사용

다음과 같은 시나리오를 가정한다.

- 서버 A 는 8000 번 포트로 SOAP 요청을 받아들임.
- 클라이언트 B 의 웹서비스 클라이언트가 송수신하는 SOAP 메시지를 모니터링하고자 함.

이 경우 웹서비스 클라이언트 프로그램에서는 tcpmon 을 사용하여 SOAP 메시지를 모니터링 할 수 있다. B 에서 tcpmon 을 실행한 뒤 다음과 같이 설정한다.

- Listen Port : 9000
- Act as a Listener : Target Hostname : A, Target Port : 8000
- Add 버튼을 클릭한다.

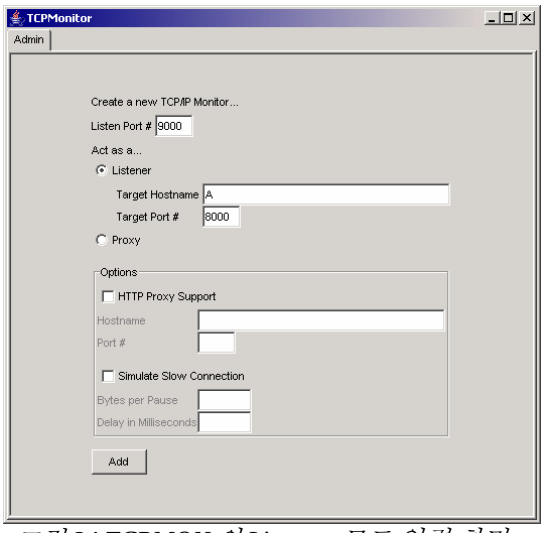


그림 24 TCPMON 의 Listener 모드 입력 화면

다음과 같이 Port 9000 Tab 이 추가된다. 이 Tab 을 클릭하면 다음과 같은 화면을 볼 수 있다.

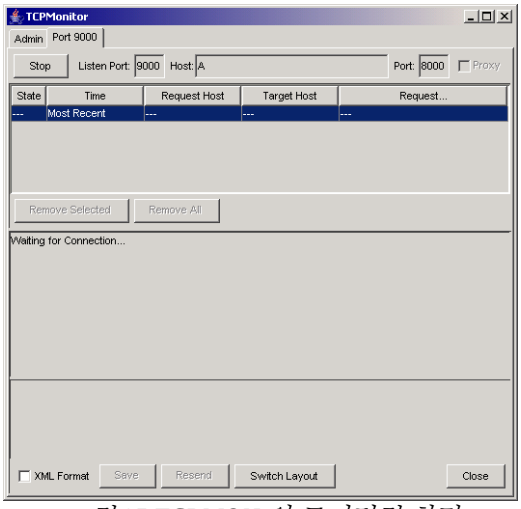
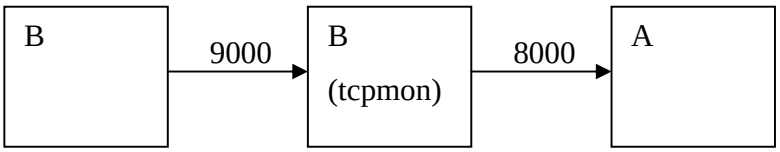


그림 25 TCPMON 의 모니터링 화면

이렇게 설정한 후 클라이언트 프로그램에서 A 의 8000 번에 접속하는 대신 B 의 9000 번에 접속하도록 한 후 클라이언트 프로그램을 실행하면 tcpmon 이 9000 번으로 들어온 요청을 받아서 A 의 8000 번 포트로 전송 후 응답을 받아서 클라이언트가 tcpmon 에 연결한 커넥션의 응답으로 돌려주고 송수신 한 메시지를 tcpmon 에 출력한다.



TCPMon 을 Listener 모드로 사용하는 경우 웹서비스의 엔드포인트 주소를 바꾸기 위해서 클라이언트 프로그램이나 클라이언트의 설정을 변경할 필요가 있다.

J2SE 웹서비스 클라이언트의 경우 접속하려는 웹서비스의 엔드포인트 주소를 바꾸기 위해서는 다음 예와 같이 코드를 수정한다.

```
((javax.xml.rpc.Stub)port)._setProperty(
      javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
      "http://localhost:9000/ws/AddressBookService");
```

port 는 서비스 엔드포인트 인터페이스에 대한 JAX-RPC Stub 객체이다. javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY 는 엔드포인트 주소를 지정하기 위한 JAX-RPC 스펙에서 정의한 표준 속성이다.

J2EE 웹서비스 클라이언트의 경우 코드 수정 없이 JEUS 설정파일 (jeus-web-dd.xml 또는 jeus-ejb-dd.xml)을 <service-client>에 <stub-property>를 추가하여 위의 속성을 적용할 수 있다.

```
<service-client>
  <port-info>
     <stub-property>
       <name>javax.xml.rpc.service.endpoint.address</name>
       <value>http://localhost:9000/ws/AddressBookService</value>
     </stub-property>
  </port-info>
</service-client>
```

B.4.4 Proxy 모드의 사용

Proxy 모드는 tcpmon 을 일반적인 HTTP Proxy 처럼 동작하게 한다. Proxy 모드의 사용방법은 다음과 같다.

Admin 탭에서

- Listen Port 를 입력한다.
- Proxy 라디오 버튼을 체크한다.
- Add 버튼을 클릭한다.

Proxy 모드의 경우 어플리케이션을 수정하는 대신 어플리케이션을 실행할 때 옵션을 주어서 tcpmon 을 사용할 수 있다. 이때 다음 두 옵션을 사용한다.

-Dhttp.proxyHost : TCPMON 이 수행되고 있는 Host 의 명 또는 IP 주소를 기입한다.

-Dhttp.proxyPort : TCPMON 의 Listen 포트를 기입한다.

클라이언트 어플리케이션이 com.acme.AddressBookClient 클래스인 경우 다음과 같이 옵션을 기재하여 java 를 실행한다.

```
java –Dhttp.proxyHost=B –Dhttp.proxyPort=9000
com.acme.AddressBookClient
```

B.3.3 기타 기능

각 포트별 탭에서 SOAP 메시지를 포맷팅하거나 저장 또는 재전송을 할 수 있다.

- XML Format : XML Format 체크 박스는 XML 을 보기 좋은 형태로 정렬하여 TCPMON 화면에 출력한다.
- Save : Save 버튼을 사용하여 현재 SOAP 메시지를 저장할 수 있다.
- Resend: 현재 Request SOAP 메시지를 재전송 한다.
- Switch Layout : 메시지 출력 화면의 가로/세로 분할 모드를 바꾼다.

C JEUS 웹 서비스 설정 파일 작성

C.1 소개

본 부록의 레퍼런스는 Java 클래스에서 웹서비스 생성을 위한 웹서비스 설정 파일의 모든 XML 태그에 대해서 설명하고 있다. 이 파일의 XML Schema(xsd) 파일은 “JEUS_HOME\config\xsds\” 디렉토리의 “jeus-webservices-config.xsd” 파일이다.

본 레퍼런스는 3 부분으로 나눠져있다.

a. **XML Schema(XSD)/XML Tree** : XML 설정 파일의 모든 태그 리스트를 정리했다. 각 노드 형식은 다음과 같다.

   1. 태그 레퍼런스로 빨리 찾아보기 위해서 각 태그마다 인덱스 번호(예 (11))를 붙여놓았다. 태그 레퍼런스에서는 이 번호 순서로 설명한다.

   2. Schema(xsd)에서 정의한 XML 태그명을 <tag name> 형식으로 표시한다.

   3. Schema(xsd)에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element

   4. 몇몇 태그에는 “P”문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.

b. 태그 레퍼런스 : 트리에 있는 각 XML 태그를 설명한다.

   1. **Description** : 태그에 대한 간단한 설명.

   2. **Value Description** : 입력하는 값과 타입.

   3. **Value Type** : 값의 데이터 타입. 예 : String

4. **Default Value** : 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값.
5. **상수** : 이미 정해져 있는 값.
6. **Example** : 해당 XML 태그에 대한 Example.
7. **Performance Recommendation** : 성능 향상을 위해서 추천되는 값.
8. **Child Elements** : 자신의 태그 안에 사용되는 태그

c. **Example XML** 파일 : "jeus-webservices-config.xml"에 대한 예제

C.2 XML Schema(XSD)/XML Tree

```
(1) <web-services-config>
    (2) <service>+
        (3) <service-name>
        (4) <target-namespace>
        (5) <package-mapping>*
            (6) <package-name>
            (7) <namespace-uri>
        (8) <output-wsdl-file>?
        (9) <output-jaxrpc-mapping-file>?
        (10) <style>?
        (11) <use>?
        (12) <map-mime-to-anytype>?
        (13) <interface>+
            (14) <endpoint-interface-class>
            (15) <wsdl-port-type-name>?
            (16) <wsdl-binding-name>?
            (17) <wsdl-port-name>?
            (18) <operation>*
                (19) <java-method-name>
                (20) <wsdl-name>?
                (21) <soap-action>?
                (22) <one-way>?
                (23) <parameter>*
```

```
        (24) <class-name>
        (25) <wsdl-name>?
        (26) <location>?
        (27) <mime-type>?
        (28) <mode>?
    (29) <return>?
        (30) <wsdl-name>?
        (31) <location>?
        (32) <mime-type>?
```

C.3 Element Reference

(1) <web-services-config>

Description	웹 서비스 설정 문서의 루트 엘리먼트(root element).
Child Elements	(2)service+

(2) <web-services-config> <service>

Description	WSDL 과 매핑(JARX-RPC Mapping) 파일을 만들기 위하여 endpoing interface 와 메시징 스타일(messaging style) 등의 서비스 정보를 기술한다.
Child Elements	(3)service-name (4)target-namespace (5)package-mapping* (8)output-wsdl-file? (9)output-jaxrpc-mapping-file? (10)style? (11)use? (12)map-mime-to-anytype? (13)interface+

(3) <web-services-config> <service> <service-name>

Description	만들고자 하는 서비스의 이름을 기술한다.
Value Type	string

(4) <web-services-config> <service> <target-namespace>

Description	서비스가 가지게 되는 타겟 네임 스페이스이다. 서비스 고유의 네임 스페이스를 설정하여 다른 서비스와의 논리적인 구별 기준이 된다.
Value Type	anyURI

(5) <web-services-config> <service> <package-mapping>

Description	어떤 자바 패키지를 특정한 네임 스페이스로 매핑하는 설정이다.

Child Elements	(6)package-name (7)namespace-uri

(6) <web-services-config> <service> <package-mapping> <package-name>

Description	네임 스페이스로 매핑을 자바 패키지의 이름이다.
Value Type	string

(7) <web-services-config> <service> <package-mapping> <namespace-uri>

Description	자바 패키지 이름에 상응하는 네임 스페이스를 설정한다.
Value Type	anyURI

(8) <web-services-config> <service> <output-wsdl-file>

Description	생성하고자 하는 출력 WSDL 파일의 이름이다.
Value Type	string

(9) <web-services-config> <service> <output-jaxrpc-mapping-file>

Description	생성하고자 하는 출력 매핑 파일의 이름이다.
Value Type	string

(10) <web-services-config> <service> <style>

Description	생성될 서비스의 메시징 스타일(messaging style)을 정의한다. 'rpc', 'document', 'wrapped' 세 가지 중 하나를 선택한다. 기본 설정은 'wrapped'이다.
Value Type	styleType
Defined Value	rpc document wrapped

(11) <web-services-config> <service> <use>

Description	인코딩 스타일을 정의한다. 'encoded' 혹은 'literal' 중 하나를 선택한다. 기본 설정은 'literal'이다.
Value Type	useType
Defined Value	encoded literal

(12) <web-services-config> <service> <map-mime-to-anytype>

Description	사용자가 정의한 임의의 타입을 사용하는 것을 허용할 것인지에 대한 xs:boolean 타입의 값을 가진다.

Value Type boolean

(13) <web-services-config> <service> <interface>

Description 서비스 엔드포인트 인터페이스(Service Endpoint Interface)를 설정한다.

Child Elements
```
(14)endpoint-interface-class
(15)wsdl-port-type-name?
(16)wsdl-binding-name?
(17)wsdl-port-name?
(18)operation*
```

(14) <web-services-config> <service> <interface> <endpoint-interface-class>

Description 백엔드(back-end)로 동작하는 자바 클래스의 이름을 표시한다.

Value Type string

(15) <web-services-config> <service> <interface> <wsdl-port-type-name>

Description WSDL 에 나타나는 실제 포트 타입의 이름을 임의로 설정할 수 있다.

Value Type string

(16) <web-services-config> <service> <interface> <wsdl-binding-name>

Description WSDL 에 나타나는 실제 바인딩의 이름을 임의로 설정할 수 있다.

Value Type string

(17) <web-services-config> <service> <interface> <wsdl-port-name>

Description WSDL 에 나타나는 실제 포트 이름을 임의로 설정할 수 있다.

Value Type string

(18) <web-services-config> <service> <interface> <operation>

Description 서비스로 공개하는 오퍼레이션들 각각의 추가적인 설정을 기술한다.

Child Elements
```
(19)java-method-name
(20)wsdl-name?
(21)soap-action?
(22)one-way?
(23)parameter*
(29)return?
```

(19) <web-services-config> <service> <interface> <operation> <java-method-name>

Description 설정 하고자 하는 오퍼레이션 이름을 값으로 가진다.

Value Type string

(20) <web-services-config> <service> <interface> <operation> <wsdl-

name>

Description WSDL 에 공개되는 오퍼레이션 이름을 설정한다.

Value Type string

(21) <web-services-config> <service> <interface> <operation> <soap-action>

Description 오퍼레이션의 SOAP-ACTION 을 설정한다.

Value Type string

(22) <web-services-config> <service> <interface> <operation> <one-way>

Description 요청-응답 메커니즘이 아닌 원-웨이(one-way) 호출을 허용하는 설정이다. 비동기 원-웨이 호출은 클라이언트가 SOAP 응답을 받지 않으며, 결함(fault)이나 예외 상황(exception)이 발생했을 경우에도 마찬가지로 응답을 받지 않는다. 기본값은 'false'이다.

Value Type boolean

(23) <web-services-config> <service> <interface> <operation> <parameter>

Description <parameter>엘리먼트는 한 오퍼레이션의 하나의 파라미터를 정의한다. 이 엘리먼트는 오퍼레이션을 구현한 메쏘드 안에 정의된 순서대로 나열 되어야 한다. <parameter>엘리먼트의 수는 메쏘드에 정의된 파라미터의 수와 일치해야 한다.

Child Elements
(24)class-name
(25)wsdl-name?
(26)location?
(27)mime-type?
(28)mode?

(24) <web-services-config> <service> <interface> <operation> <parameter> <class-name>

Description 입력 파라미터 데이터 타입이 자바 클래스로 매핑될 때의 자바 클래스 이름이다.

Value Type string

(25) <web-services-config> <service> <interface> <operation> <parameter> <wsdl-name>

Description 생성될 WSDL 에 사용될 입력 파라미터의 이름이다. 별도 입력하지 않았을때의 기본 값은 메쏘드의 시그너쳐의 파라미터 이름이다

Value Type string

(26) <web-services-config> <service> <interface> <operation> <parameter> <location>

Description SOAP 메시지에서 입력 파라미터가 위치할 부분을 정의한다. 'header', 'body', 'attachment' 중 하나의 값을 가진다. 기본 값은 'body'이다. 'attachment'로 설정하면, 파라미터가 SOAP 엔벨로프가 아니라 SOAP 메시지의 부착물(attachment)로부터 추출되며, JAX-RPC 스펙에서 지정하는 타입(java.awt.Image,

java.lang.String, javax.mail.internet.MimeMultipart, javax.xml.transform.Source, javax.activation.DataHandler)만이 SOAP 메시지 부착물에서 추출될 수 있다.

Value Type locationType

Defined Value header

body

attachment

```
(27) <web-services-config> <service> <interface> <operation>
<parameter> <mime-type>
```

Description 생성될 WSDL 에 표시될 파라미터의 MIME 타입을 정의한다.

Value Type string

```
(28) <web-services-config> <service> <interface> <operation>
<parameter> <mode>
```

Description 입력 파라미터가 표준 입력 파라미터인지, 리턴 값으로 사용되는 출력 파라미터인지, 아니면 입력과 출력 모두를 위한 입력-출력(in-out)파라미터인지를 지정한다. 'IN', 'INOUT', 'OUT' 세 값 중의 하나의 값을 가진다.

Value Type modeType

Defined Value IN

INOUT

OUT

```
(29) <web-services-config> <service> <interface> <operation> <return>
```

Description <return>엘리먼트는 웹 서비스 오퍼레이션의 리턴 되는 값을 정의한다. 한 오퍼레이션 당 하나의 <return>엘리먼트만 정의 가능하다.

Child Elements
```
(30)wsdl-name?
(31)location?
(32)mime-type?
```

```
(30) <web-services-config> <service> <interface> <operation> <return>
<wsdl-name>
```

Description 생성될 WSDL 에 사용될 리턴 파라미터의 이름이다.

Value Type string

```
(31) <web-services-config> <service> <interface> <operation> <return>
```

<location>

Description	SOAP 메시지에서 리턴 파라미터가 위치할 부분을 정의한다. 'header', 'body', 'attachment' 중 하나의 값을 가진다. 기본 값은 'body'이다. 'attachment'로 설정하면, 파라미터가 SOAP 엔벨로프가 아니라 SOAP 메시지의 부착물(attachment)로부터 추출되며, JAX-RPC 스펙에서 지정하는 타입(java.awt.Image, java.lang.String, javax.mail.internet.MimeMultipart, javax.xml.transform.Source, javax.activation.DataHandler)만이 SOAP 메시지 부착물에서 추출될 수 있다.
Value Type	locationType
Defined Value	header body attachment

(32) <web-services-config> <service> <interface> <operation> <return> **<mime-type>**

Description	생성될 WSDL 에 표시될 파라미터의 MIME 타입을 정의한다.
Value Type	string

C.4 jeus-webservices-config.xml 파일 예제

<<jeus-webservices-config.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
   <service>
      <service-name>DocLitEchoService</service-name>
      <target-namespace>urn:DocLitService</target-namespace>
      <output-wsdl-file>DocLitEchoService.wsdl</output-wsdl-file>
      <output-jaxrpc-mapping-file>
           DocLitEchoService-mapping.xml
      </output-jaxrpc-mapping-file>
      <style>rpc</style>
      <interface>
         <endpoint-interface-class>
            jeustest.webservices.java2wsdl.doclit.Echo
         </endpoint-interface-class>
```

```
        <operation>
            <java-method-name>echoString</java-method-name>
            <wsdl-name>echoWsdlString</wsdl-name>
               <soap-action>hokeAction</soap-action>
            <one-way>false</one-way>
        </operation>
      </interface>
   </service>
</web-services-config>
```

D JEUS 웹 서비스 배치 서술자 (jeus-webservices-dd.xml) 작성

D.1 소개

본 부록의 레퍼런스는 jeus-webservices-dd.xml(웹서비스 배치 서술자 파일)의 모든 XML 태그에 대해서 설명하고 있다. 이 파일의 XML Schema(xsd) 파일은 “JEUS_HOME\config\xsds\” 디렉토리의 “jeus-webservices-dd.xsd” 파일이다.

본 레퍼런스는 3 부분으로 나눠져있다.

a. XML Schema(XSD)/XML Tree : XML 설정 파일의 모든 태그 리스트를 정리했다. 각 노드 형식은 다음과 같다.

1. 태그 레퍼런스로 빨리 찾아보기 위해서 각 태그마다 인덱스 번호(예 (11))를 붙여놓았다. 태그 레퍼런스에서는 이 번호 순서로 설명한다.

2. Schema(xsd)에서 정의한 XML 태그명을 <tag name> 형식으로 표시한다.

3. Schema(xsd)에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element

4. 몇몇 태그에는 “P”문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.

b. 태그 레퍼런스 : 트리에 있는 각 XML 태그를 설명한다.

1. **Description** : 태그에 대한 간단한 설명.

2. **Value Description** : 입력하는 값과 타입.

3. **Value Type** : 값의 데이터 타입. 예 : String
4. **Default Value** : 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값.
5. **상수** : 이미 정해져 있는 값.
6. **Example** : 해당 XML 태그에 대한 Example.
7. **Performance Recommendation** : 성능 향상을 위해서 추천되는 값.
8. **Child Elements** : 자신의 태그 안에 사용되는 태그

c. Example XML 파일 : "jeus-webservices-dd.xml"에 대한 완전한 예제

D.2 XML Schema(XSD)/XML Tree

```
(1) <jeus-webservices-dd>
   (2) <ejb-context-path>?
   (3) <ejb-login-config>?
      (4) <auth-method>?
      (5) <realm-name>?
   (6) <service>+
      (7) <webservice-description-name>
      (8) <wsdl-publish>? (choose one of (9),(12))
         (9) <http-publish>
            (10) <server-url>?
            (11) <secure-server-url>?
         (12) <file-publish>
            (13) <server-url>
            (14) <secure-server-url>?
            (15) <publish-path>
      (16) <port>+
         (17) <port-component-name>
         (18) <ejb-endpoint-url>?
         (19) <tie-class>?
         (20) <ejb-transport-guarantee>?
         (21) <soapver>?
         (22) <security>?
```

```
(23) <request-receiver>?
   (24) <action-list>
   (25) <password-callback-class>?
   (26) <timeToLive>?
   (27) <decryption>?
      (28) <keystore>
         (29) <key-type>
         (30) <keystore-password>
         (31) <keystore-filename>
   (32) <signature-verification>?
      (33) <keystore>
         (34) <key-type>
         (35) <keystore-password>
         (36) <keystore-filename>
(37) <response-sender>?
   (38) <action-list>
   (39) <password-callback-class>?
   (40) <user>
   (41) <timeToLive>?
   (42) <userNameToken>?
      (43) <passwordType>?
      (44) <userTokenElements>?
   (45) <signature-infos>?
      (46) <signature-info>+
         (47) <signatureParts>?
         (48) <keyIdentifier>
         (49) <keystore>
            (50) <key-type>
            (51) <keystore-password>
            (52) <keystore-filename>
   (53) <encryption-infos>?
      (54) <encryption-info>+
         (55) <encryptionParts>?
         (56) <encryptionSymAlgorithm>?
         (57) <encryptionUser>?
         (58) <keyIdentifier>
         (59) <keystore>?
```

(60) <key-type>
(61) <keystore-password>
(62) <keystore-filename>
(63) <embeddedKey>?
(64) <embeddedKeyCallbackClass>
(65) <key-name>

D.3 Element Reference

(1) <jeus-webservices-dd>

Description JEUS 웹 서비스 설정 문서의 루트 엘리먼트(root element).

Child Elements
(2)ejb-context-path?
(3)ejb-login-config?
(6)service+

(2) <jeus-webservices-dd> <ejb-context-path>

Description 배치 할 EJB 웹서비스를 위한 엔드포인트 URL 의 컨텍스트(Context) 경로를 표시한다. EJB 엔드포인트에서만 사용한다.

Value Type token

(3) <jeus-webservices-dd> <ejb-login-config>

Description EJB 웹서비스 엔드포인트 URL 에 대하여 클라이언트가 인증 받는데 사용되는 인증 방법과 인증 영역을 표시한다. EJB 엔드포인트에서만 사용한다.

Child Elements
(4)auth-method?
(5)realm-name?

(4) <jeus-webservices-dd> <ejb-login-config> <auth-method>

Description 인증 방법을 설정한다. 'BASIC', 'CLIENT-CERT', 'DIGEST', 'FORM' 의 네 값 중에 하나를 선택한다.

Value Type string

(5) <jeus-webservices-dd> <ejb-login-config> <realm-name>

Description 보안 정책이 적용되는 도메인을 설정한다.

Value Type string

(6) <jeus-webservices-dd> <service>

Description 배치되는 웹서비스를 표시한다.

Child Elements
(7)webservice-description-name
(8)wsdl-publish?
(16)port+

(7) <jeus-webservices-dd> <service> <webservice-description-name>

Description WSDL 파일에서 관련된 웹서비스 엔드포인트 이름이다. <webservice-description-name>은 표준 배치 서술자 webservice.xml 의 <webservice-description-name>에 상응한다.

Value Type string

(8) <jeus-webservices-dd> <service> <wsdl-publish>

Description WSDL 를 공개할 방식을 설정한다. http-publish 혹은 file-publish 중에 하나를 선택 해야 한다.

Child Elements
(9)http-publish
(12)file-publish

(9) <jeus-webservices-dd> <service> <wsdl-publish> <http-publish>

Description http 방식으로 공개되는 WSDL 를 나타낸다. 공개된 WSDL 에서 import 하는 schema 들은 HTTP URL 에 의하여 참조된다.

Child Elements
(10)server-url?
(11)secure-server-url?

(10) <jeus-webservices-dd> <service> <wsdl-publish> <http-publish> <server-url>

Description 배치된 웹서비스의 WSDL 파일를 공개하는 서버의 URL 이다.

Value Type string

(11) <jeus-webservices-dd> <service> <wsdl-publish> <http-publish> <secure-server-url>

Description 배치된 웹서비스의 WSDL 파일를 공개하는 서버의 SSL 보안이 가능한 URL 이다.

Value Type string

(12) <jeus-webservices-dd> <service> <wsdl-publish> <file-publish>

Description file 방식으로 공개되는 WSDL 를 나타낸다. 공개된 WSDL 에서 import 하는 schema 들은 파일 상대 경로에 의하여 참조된다.

Child Elements
(13)server-url
(14)secure-server-url?
(15)publish-path

(13) <jeus-webservices-dd> <service> <wsdl-publish> <file-publish> <server-url>

Description 배치된 웹서비스의 WSDL 파일를 공개하는 서버의 URL 이다.

Value Type string

(14) <jeus-webservices-dd> <service> <wsdl-publish> <file-publish> <secure-server-url>

Description 배치된 웹서비스의 WSDL 파일를 공개하는 서버의 SSL 보안이 가능한 URL 이다.

Value Type string

(15) <jeus-webservices-dd> <service> <wsdl-publish> <file-publish> <publish-path>

Description 웹서비스 WSDL 이 저장되는 디렉토리를 표시한다.

Value Type string

(16) <jeus-webservices-dd> <service> <port>

Description 웹서비스 port 정보를 설정한다.

Child Elements
(17)port-component-name
(18)ejb-endpoint-url?
(19)tie-class?
(20)ejb-transport-guarantee?
(21)soapver?
(22)security?

(17) <jeus-webservices-dd> <service> <port> <port-component-name>

Description 웹서비스 엔드포인트를 구별하기 위한 이름을 표시한다. <port-component-name>은 표준 배치 서술자 webservice.xml 의 <port-component-name>에 상응한다.

Value Type token

(18) <jeus-webservices-dd> <service> <port> <ejb-endpoint-url>

Description EJB 엔드포인트에 접근할 수 있는 엔드포인트 URL 을 표시한다. EJB 엔드포인트에서만 사용한다.

Value Type string

(19) <jeus-webservices-dd> <service> <port> <tie-class>

Description 웹서비스 엔드포인트 URL 과 웹서비스 Endpint 인터페이스를 연결해 주는 class 를 표시한다. 배치 시에 자동 생성되므로 웹서비스 배치자가 표시할 필요가 없다.

Value Type string

(20) <jeus-webservices-dd> <service> <port> <ejb-transport-guarantee>

Description EJB 엔드포인트에서 주고 받는 메시지에 대한 기밀성(confidentiality)과 무결성(integrity) 구성을 표시한다. 설정 가능한 값은 'NONE', 'CONFIDENTIAL' 또는 'INTEGRAL'이다.

Value Type ejb-transport-guaranteeType

Defined Value NONE

INTEGRAL

CONFIDENTIAL

(21) <jeus-webservices-dd> <service> <port> **<soapver>**

Description 배치된 웹서비스에서 서비스하는 SOAP 메시지의 버전을 표시한다.

Value Type string

(22) <jeus-webservices-dd> <service> <port> **<security>**

Description 웹서비스의 보안(WS-Security)을 위한 설정이다.

Child Elements
(23)request-receiver?
(37)response-sender?

(23) <jeus-webservices-dd> <service> <port> <security> **<request-receiver>**

Description 웹서비스 클라이언트의 보안 메시지 요청을 처리하 기 위한 설정이다.

Child Elements
(24)action-list
(25)password-callback-class?
(26)timeToLive?
(27)decryption?
(32)signature-verification?

(24) <jeus-webservices-dd> <service> <port> <security> <request-receiver> **<action-list>**

Description 받게 되는 메시지가 어떤 보안이 적용되어 있어야 하는지 설정한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

Value Type string

(25) <jeus-webservices-dd> <service> <port> <security> <request-receiver> **<password-callback-class>**

Description 패스워드 콜백 클래스의 이름을 풀 패키지 이름으로 입력한다.

Value Type string

(26) <jeus-webservices-dd> <service> <port> <security> <request-receiver> **<timeToLive>**

Description 받은 메시지의 유효기간을 설정한다(초단위). 기본값은 생성 시간으로 부터 300 초 동안이다.

Value Type string

(27) <jeus-webservices-dd> <service> <port> <security> <request-receiver> **<decryption>**

Description 받는 메시지의 암호화 된 부분을 해독하기 위한 설정이다.

Child Elements (28)keystore

```
(28) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <decryption> <keystore>
```

Description 메시지의 암호를 해독하기 위한 키 스토어의 설정이다.

Child Elements (29)key-type
(30)keystore-password
(31)keystore-filename

```
(29) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <decryption> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(30) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <decryption> <keystore> <keystore-password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(31) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <decryption> <keystore> <keystore-filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(32) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <signature-verification>
```

Description 받는 메시지의 서명을 검증하기 위한 설정이다.

Child Elements (33)keystore

```
(33) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <signature-verification> <keystore>
```

Description 서명을 검증하기 위한 키 스토어 설정이다.

Child Elements (34)key-type
(35)keystore-password
(36)keystore-filename

```
(34) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <signature-verification> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(35) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <signature-verification> <keystore> <keystore-password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(36) <jeus-webservices-dd> <service> <port> <security> <request-
receiver> <signature-verification> <keystore> <keystore-filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(37) <jeus-webservices-dd> <service> <port> <security> <response-
sender>
```

Description 웹서비스가 SOAP 메시지를 처리한 후, 응답을 내보 낼 때, 보안이 적용된 메시지를 보내기 위한 설정이다.

Child Elements

```
(38)action-list
(39)password-callback-class?
(40)user
(41)timeToLive?
(42)userNameToken?
(45)signature-infos?
(53)encryption-infos?
```

```
(38) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <action-list>
```

Description 어떤 보안을 적용할 것인지를 String 으로 나열한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

Value Type string

```
(39) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <password-callback-class>
```

Description 패스워드를 설정하는 콜백 클래스의 풀 패키지 명이다.

Value Type string

```
(40) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <user>
```

Description UsernameToken 에 들어갈 이름과 서명에 들어갈 키의 별칭 을 설정한다.

Value Type string

```
(41) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <timeToLive>
```

Description 보내게 될 메시지의 유효기간을 초 단위로 설정한다. 기본값 은 300 초이다.

Value Type string

```
(42) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <userNameToken>
```

Description UsernameToken 을 설정한다.

Child Elements
```
(43)passwordType?
(44)userTokenElements?
```

```
(43) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <userNameToken> <passwordType>
```

Description UsernameToken 에 사용될 패스워드의 타입 설정이다.
"PasswordDigest" 혹은 "PasswordText"를 사용할 수 있다.

Value Type passwordTypeType

Defined Value
PasswordDigest
UsernameToken 에 설정되는 암호가 base64 encoding 된 상태로 메시지에 포함된다.

PasswordText
UsernameToken 에 설정되는 암호가 평이한 텍스트로 메시지에 포함된다.

```
(44) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <userNameToken> <userTokenElements>
```

Description UsernameToken 에 추가될 엘리먼트의 리스트이다. 각 항목은 공백으로 분리된다. "nonce" 혹은 "created"가 사용될 수 있다. passwordType 이 "PasswordText"일 경우에 사용가능하다.

Value Type string

```
(45) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos>
```

Description 메시지에 서명을 하기 위한 설정이다.

Child Elements
```
(46)signature-info+
```

```
(46) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos> <signature-info>
```

Description 메시지의 서명을 위한 설정이다. 복수 설정이 가능하다.

Child Elements
```
(47)signatureParts?
(48)keyIdentifier
(49)keystore
```

```
(47) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos> <signature-info> <signatureParts>
```

Description 메시지의 특정 부분을 서명하고자 할 때 사용한다.
"{}{http://schemas.xmlsoap.org/soap/envelope/}Body;
Token"과 같은 방식으로 열거할 수 있다. 기본적으로 설정하지 않았을 경우에는 SOAP 몸체 전체를 서명하게 되어 있다.

Value Type string

```
(48) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos> <signature-info> <keyIdentifier>
```

Description 서명에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier 중의 하나를 사용한다.

Value Type sigKeyIdentifierType

Defined Value

IssuerSerial
X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference
X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참조하는 방식이다.

SKIKeyIdentifier
Subject Key Identification 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 X509 인증서를 포함하고 서명 검증을 위해 사용하도 록 한다.

```
(49) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos> <signature-info> <keystore>
```

Description 메시지의 서명을 위한 개인키를 저장하고 있는 키스토어의 설정이다.

Child Elements

```
(50)key-type
(51)keystore-password
(52)keystore-filename
```

```
(50) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos> <signature-info> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(51) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos> <signature-info> <keystore> <keystore-
password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(52) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <signature-infos> <signature-info> <keystore> <keystore-
filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

(53) <jeus-webservices-dd> <service> <port> <security> <response-sender> **<encryption-infos>**

Description 메시지를 암호화 하기 위한 설정이다.

Child Elements (54)encryption-info+

(54) <jeus-webservices-dd> <service> <port> <security> <response-sender> <encryption-infos> **<encryption-info>**

Description 배치되는 웹서비스 클라이언트를 위한 설정들을 표시한다.

Child Elements
(55)encryptionParts?
(56)encryptionSymAlgorithm?
(57)encryptionUser?
(58)keyIdentifier
(59)keystore?
(63)embeddedKey?

(55) <jeus-webservices-dd> <service> <port> <security> <response-sender> <encryption-infos> <encryption-info> **<encryptionParts>**

Description 특정 부분을 암호화 하기 위한 설정이다.
"{mode}{ns}{localname};{mode}{ns}{localname};…" 과 같은 형식이다.
기본 mode 값은 content 이다.
예:{Content}{http://example.org/payment}CreCard;{Element}{}UserName

Value Type string

(56) <jeus-webservices-dd> <service> <port> <security> <response-sender> <encryption-infos> <encryption-info> **<encryptionSymAlgorithm>**

Description 암호화에 사용하는 알고리즘이다. AES_128, AES_256, TRIPLE_DES, AES_192 를 지원한다.

Value Type string

(57) <jeus-webservices-dd> <service> <port> <security> <response-sender> <encryption-infos> <encryption-info> **<encryptionUser>**

Description 암호화에 사용되는 키의 별칭이다.

Value Type string

(58) <jeus-webservices-dd> <service> <port> <security> <response-sender> <encryption-infos> <encryption-info> **<keyIdentifier>**

Description 암호화에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier EmbeddedKeyName 중의 하나를 사용한다.

Value Type encKeyIdentifierType

Defined Value IssuerSerial

X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference
X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참 조하는 방식이다.

SKIKeyIdentifier
Subject Key Identification 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 암호화에 사용된 X509 인증서를 포함한다.

EmbeddedKeyName
웹서비스와 웹서비스 클라이언트가 공유하는 세션키를 사용할 때 사용한다. 웹서비스와 클라이언트는 키의 이름만을 주고 받음으로써 어떤 키를 사용했는지를 알 수 있다.

```
(59) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <encryption-infos> <encryption-info> <keystore>
```

Description 암호화에 사용될 키의 저장소 설정이다.

Child Elements
```
(60)key-type
(61)keystore-password
(62)keystore-filename
```

```
(60) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <encryption-infos> <encryption-info> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(61) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <encryption-infos> <encryption-info> <keystore> <keystore-
password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(62) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <encryption-infos> <encryption-info> <keystore> <keystore-
filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(63) <jeus-webservices-dd> <service> <port> <security> <response-
sender> <encryption-infos> <encryption-info> <embeddedKey>
```

Description	웹서비스와 웹서비스 클라이언트가 공유하고 있는 키를 설정한다. keyIdentifier 가 "EmbeddedKeyName"으로 설정되어야 사용할 수 있다.
Child Elements	(64)embeddedKeyCallbackClass (65)key-name

(64) <jeus-webservices-dd> <service> <port> <security> <response-sender> <encryption-infos> <encryption-info> <embeddedKey> **<embeddedKeyCallbackClass>**

Description	세션 키를 사용하려 할 경우, 키의 바이트 정보를 가지고 있 는 콜백 클래스를 설정한다.
Value Type	string

(65) <jeus-webservices-dd> <service> <port> <security> <response-sender> <encryption-infos> <encryption-info> <embeddedKey> **<key-name>**

Description	세션 키의 이름을 설정한다.
Value Type	string

D.4 jeus-webservices-dd.xml 파일 예제

<<*jeus-webservices-dd.xml*>>

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<jeus-webservices-dd
xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <service>
        <webservice-description-name>
            DocLitEchoService
        </webservice-description-name>
        <port>
            <port-component-name>
                EchoPort
            </port-component-name>
        </port>
    </service>
</jeus-webservices-dd>
```

E JEUS 웹 서비스 클라이언트 배치 서술자(jeus-webservicesclient-dd.xml) 작성

E.1 소개

본 부록의 레퍼런스는 웹서비스 클라이언트 배치 서술자의 모든 XML 태그에 대해서 설명하고 있다. 이 파일의 XML Schema(xsd) 파일은 “JEUS_HOME\config\xsds\” 디렉토리의 “jeus-webservicesclient-dd.xsd” 파일이다. 웹서비스 클라이언트 배치 서술자는 별도의 파일로 존재하지 않고 jeus-web-dd.xml, jeus-ebj-dd.xml 또는 jeus-client-dd.xml 에 <service-ref> 태그로 포함되어 사용된다.

본 레퍼런스는 3 부분으로 나눠져있다.

a. XML Schema(XSD)/XML Tree : XML 설정 파일의 모든 태그 리스트를 정리했다. 각 노드 형식은 다음과 같다.

1. 태그 레퍼런스로 빨리 찾아보기 위해서 각 태그마다 인덱스 번호(예 (11))를 붙여놓았다. 태그 레퍼런스에서는 이 번호 순서로 설명한다.

2. Schema(xsd)에서 정의한 XML 태그명을 <tag name> 형식으로 표시한다.

3. Schema(xsd)에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element

4. 몇몇 태그에는 “P”문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.

b. 태그 레퍼런스 : 트리에 있는 각 XML 태그를 설명한다.

1. Description : 태그에 대한 간단한 설명.

2. Value Description : 입력하는 값과 타입.

3. Value Type : 값의 데이터 타입. 예 : String

4. Default Value : 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값.

5. 상수 : 이미 정해져 있는 값.

6. Example : 해당 XML 태그에 대한 Example.

7. Performance Recommendation : 성능 향상을 위해서 추천되는 값.

8. Child Elements : 자신의 태그 안에 사용되는 태그

c. Example XML 파일 : "jeus-webservicesclient-dd.xml"에 대한 완전한 예제

E.2 XML Schema(XSD)/XML Tree

```
(1) <service-ref>?
    (2) <service-client>*
        (3) <service-ref-name>
        (4) <port-info>*
            (5) <service-endpoint-interface>?
            (6) <wsdl-port>?
            (7) <stub-property>*
                (8) <name>
                (9) <value>
            (10) <call-property>*
                (11) <name>
                (12) <value>
            (13) <security>?
                (14) <request-sender>?
                    (15) <action-list>
                    (16) <password-callback-class>?
```

```
                (17) <user>
                (18) <timeToLive>?
                (19) <userNameToken>?
                    (20) <passwordType>?
                    (21) <userTokenElements>?
                (22) <signature-infos>?
                    (23) <signature-info>+
                    (24) <signatureParts>?
                    (25) <keyIdentifier>
                    (26) <keystore>
                        (27) <key-type>
                        (28) <keystore-password>
                        (29)<keystore-filename>
                (30) <encryption-infos>?
                    (31) <encryption-info>+
                    (32) <encryptionParts>?
                    (33) <encryptionSymAlgorithm>?
                    (34) <encryptionUser>?
                    (35) <keyIdentifier>
                    (36) <keystore>?
                        (37) <key-type>
                        (38) <keystore-password>
                        (39) <keystore-filename>
                    (40) <embeddedKey>?
                        (41) <embeddedKeyCallbackClass>
                        (42) <key-name>
(43) <response-receiver>?
    (44) <action-list>
    (45) <password-callback-class>?
    (46) <timeToLive>?
    (47) <decryption>?
        (48) <keystore>
        (49) <key-type>
        (50) <keystore-password>
        (51) <keystore-filename>
    (52) <signature-verification>?
    (53) <keystore>
        (54) <key-type>
        (55) <keystore-password>
```

```
        (56) <keystore-filename>
(57) <service-impl-class>?
(58) <wsdl-override>?
(59) <service-qname>?
(60) <call-property>*
    (61) <name>
                        (62) <value>
```

E.3 Element Reference

(1) <service-ref>

Description	JEUS 웹 서비스 클라이언트 설정 문서의 루트 엘리먼트(root element).
Child Elements	(2)service-client*

(2) <service-ref> <service-client>

Description	배치되는 웹서비스 클라이언트를 위한 설정들을 표시한다.
Child Elements	(3)service-ref-name (4)port-info* (57)service-impl-class? (58)wsdl-override? (59)service-qname? (60)call-property*

(3) <service-ref> <service-client> <service-ref-name>

Description	WSDL 파일에서 관련된 웹서비스 엔드포인트 이름이다. <service-ref-name>은 표준 배치 서술자 web.xml 혹은 ejb-jar.xml 의 <service-ref-name>에 상응한다. [Dependency]:/web-app/service-ref/service-ref-name /ejb-jar/enterprise-beans/session/service-ref/service-ref-name
Value Type	token

(4) <service-ref> <service-client> <port-info>

Description	배치되는 웹서비스 클라이언트가 호출하는 웹서비스 포트 정보를 표시한다.
Child Elements	(5)service-endpoint-interface? (6)wsdl-port? (7)stub-property* (10)call-property* (13)security?

(5) <service-ref> <service-client> <port-info> <service-endpoint-interface>

Description WSDL port 의 서비스 엔드포인드 인터페이스를 나타내는 클래스를 표시한다. <service-ref>에서 <port-component-ref>의 <service-endpoint-interface>에 상응한다.

[Dependency]:/web-app/service-ref/port-component-ref/service-endpoint-interface

/ejb-jar/enterprise-beans/session/service-ref/port-component-ref/service-endpoint-interface

Value Type token

(6) <service-ref> <service-client> <port-info> <wsdl-port>

Description <port-info>와 연결된 WSDL port 정의를 표시한다.

Value Type QName

(7) <service-ref> <service-client> <port-info> <stub-property>

Description 특정 port 에서 사용하는 javax.xml.rpc.Stub 객체에 설정하는 property 들을 표시한다.

Child Elements
(8)name
(9)value

(8) <service-ref> <service-client> <port-info> <stub-property> <name>

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

Value Type string

(9) <service-ref> <service-client> <port-info> <stub-property> <value>

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

Value Type string

(10) <service-ref> <service-client> <port-info> <call-property>

Description 특정 port 에서 사용하는 javax.xml.rpc.Call 객체에 설정하는 property 들을 표시한다.

Child Elements
(11)name
(12)value

(11) <service-ref> <service-client> <port-info> <call-property> <name>

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

Value Type string

(12) <service-ref> <service-client> <port-info> <call-property> <value>

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

Value Type string

(13) <service-ref> <service-client> <port-info> **<security>**

Description 웹서비스의 보안(WS-Security)을 위한 웹 서비스 클 라이언트 설정이다.

Child Elements
(14)request-sender?
(43)response-receiver?

(14) <service-ref> <service-client> <port-info> <security> **<request-sender>**

Description 웹서비스를 호출하는 메시지에 보안을 적용하기 위한 설정이 다.

Child Elements
(15)action-list
(16)password-callback-class?
(17)user
(18)timeToLive?
(19)userNameToken?
(22)signature-infos?
(30)encryption-infos?

(15) <service-ref> <service-client> <port-info> <security> <request-sender> **<action-list>**

Description 어떤 보안을 적용할 것인지를 String 으로 나열한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

Value Type string

(16) <service-ref> <service-client> <port-info> <security> <request-sender> **<password-callback-class>**

Description 패스워드를 설정하는 콜백 클래스의 풀 패키지 명이다.

Value Type string

(17) <service-ref> <service-client> <port-info> <security> <request-sender> **<user>**

Description UsernameToken 에 들어갈 이름과 서명에 들어갈 키의 별칭 을 설정한다.

Value Type string

(18) <service-ref> <service-client> <port-info> <security> <request-sender> **<timeToLive>**

Description 보내게 될 메시지의 유효기간을 초 단위로 설정한다. 기본값 은 300 초이다.

Value Type string

(19) <service-ref> <service-client> <port-info> <security> <request-sender> **<userNameToken>**

Description UsernameToken 을 설정한다.

Child Elements
```
(20)passwordType?
(21)userTokenElements?
```

```
(20) <service-ref> <service-client> <port-info> <security> <request-
sender> <userNameToken> <passwordType>
```

Description UsernameToken 에 사용될 패스워드의 타입 설정이다. "PasswordDigest" 혹은 "PasswordText"를 사용할 수 있다.

Value Type passwordTypeType

Defined Value
PasswordDigest
UsernameToken 에 설정되는 암호가 base64 encoding 된 상태로 메시지에 포함된다.

PasswordText
UsernameToken 에 설정되는 암호가 평이한 텍스트로 메시지에 포함된다.

```
(21) <service-ref> <service-client> <port-info> <security> <request-
sender> <userNameToken> <userTokenElements>
```

Description UsernameToken 에 추가될 엘리먼트의 리스트이다. 각 항목은 공백으로 분리된다. "nonce" 혹은 "created"가 사용될 수 있다. passwordType 이 "PasswordText"일 경우에 사용가능하다.

Value Type string

```
(22) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos>
```

Description 메시지에 서명을 하기 위한 설정이다.

Child Elements
```
(23)signature-info+
```

```
(23) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos> <signature-info>
```

Description 메시지의 서명을 위한 설정이다. 복수 설정이 가능하다.

Child Elements
```
(24)signatureParts?
(25)keyIdentifier
(26)keystore
```

```
(24) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos> <signature-info> <signatureParts>
```

Description 메시지의 특정 부분을 서명하고자 할 때 사용한다. "{}{http://schemas.xmlsoap.org/soap/envelope/}Body; Token"과 같은 방식으로 열거할 수 있다. 기본적으로 설정하지 않았을 경우에는 SOAP 몸체 전체를 서명하게 되어 있다.

Value Type string

```
(25) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos> <signature-info> <keyIdentifier>
```

Description 서명에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier 중의 하나를 사용한다.

Value Type sigKeyIdentifierType

Defined Value

IssuerSerial
X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference
X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참 조하는 방식이다.

SKIKeyIdentifier
Subject Key Identification 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 X509 인증서를 포함하고 서명 검증을 위해 사용하도 록 한다.

```
(26) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos> <signature-info> <keystore>
```

Description 메시지의 서명을 위한 개인키를 저장하고 있는 키스토어의 설정이다.

Child Elements

```
(27)key-type
(28)keystore-password
(29)keystore-filename
```

```
(27) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos> <signature-info> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(28) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos> <signature-info> <keystore> <keystore-
password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(29) <service-ref> <service-client> <port-info> <security> <request-
sender> <signature-infos> <signature-info> <keystore> <keystore-
filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(30) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos>
```

Description 메시지를 암호화 하기 위한 설정이다.

Child Elements (31)encryption-info+

```
(31) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info>
```

Description 배치되는 웹서비스 클라이언트를 위한 설정들을 표시한다.

Child Elements
(32)encryptionParts?
(33)encryptionSymAlgorithm?
(34)encryptionUser?
(35)keyIdentifier
(36)keystore?
(40)embeddedKey?

```
(32) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <encryptionParts>
```

Description 특정 부분을 암호화 하기 위한 설정이다.
"{mode}{ns}{localname};{mode}{ns}{localname};…" 과 같은 형식이다.
기본 mode 값은 content 이다.
예:{Content}{http://example.org/payment}CreCard;{Element}{}UserName

Value Type string

```
(33) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <encryptionSymAlgorithm>
```

Description 암호화에 사용하는 알고리즘이다. AES_128, AES_256, TRIPLE_DES, AES_192 를 지원한다.

Value Type string

```
(34) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <encryptionUser>
```

Description 암호화에 사용되는 키의 별칭이다.

Value Type string

```
(35) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <keyIdentifier>
```

Description 암호화에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier EmbeddedKeyName 중의 하나를 사용한다.

Value Type encKeyIdentifierType

Defined Value IssuerSerial
X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference
X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참조하는 방식이다.

SKIKeyIdentifier
Subject Key Identification 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 암호화에 사용된 X509 인증서를 포함한다.

EmbeddedKeyName
웹서비스와 웹서비스 클라이언트가 공유하는 세션키를 사용할 때 사용한다. 웹서비스와 클라이언트는 키의 이름만을 주고 받음으로써 어떤 키를 사용했는지를 알 수 있다.

```
(36) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <keystore>
```

Description 암호화에 사용될 키의 저장소 설정이다.

Child Elements
```
(37)key-type
(38)keystore-password
(39)keystore-filename
```

```
(37) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(38) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <keystore> <keystore-
password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(39) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <keystore> <keystore-
filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(40) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <embeddedKey>
```

Description 웹서비스와 웹서비스 클라이언트가 공유하고 있는 키를 설정한다. keyIdentifier 가 "EmbeddedKeyName"으로 설정되어야 사용할

수 있다.

Child Elements

```
(41)embeddedKeyCallbackClass
(42)key-name
```

```
(41) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <embeddedKey>
<embeddedKeyCallbackClass>
```

Description 세션 키를 사용하려 할 경우, 키의 바이트 정보를 가지고 있 는 콜백 클래스를 설정한다.

Value Type string

```
(42) <service-ref> <service-client> <port-info> <security> <request-
sender> <encryption-infos> <encryption-info> <embeddedKey> <key-name>
```

Description 세션 키의 이름을 설정한다.

Value Type string

```
(43) <service-ref> <service-client> <port-info> <security> <response-
receiver>
```

Description 웹서비스 응답 메시지가 보안 적용이 되어있을 경우, 처리하 기 위한 설정이다.

Child Elements

```
(44)action-list
(45)password-callback-class?
(46)timeToLive?
(47)decryption?
(52)signature-verification?
```

```
(44) <service-ref> <service-client> <port-info> <security> <response-
receiver> <action-list>
```

Description 받게 되는 메시지가 어떤 보안이 적용되어 있어야 하는지 설정한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

Value Type string

```
(45) <service-ref> <service-client> <port-info> <security> <response-
receiver> <password-callback-class>
```

Description 패스워드 콜백 클래스의 이름을 풀 패키지 이름으로 입력한다.

Value Type string

```
(46) <service-ref> <service-client> <port-info> <security> <response-
receiver> <timeToLive>
```

Description 받은 메시지의 유효기간을 설정한다(초단위). 기본값은 생성 시간으로 부터 300 초 동안이다.

Value Type string

```
(47) <service-ref> <service-client> <port-info> <security> <response-
receiver> <decryption>
```

Description 받는 메시지의 암호화 된 부분을 해독하기 위한 설정이다.

Child Elements (48)keystore

```
(48) <service-ref> <service-client> <port-info> <security> <response-
receiver> <decryption> <keystore>
```

Description 메시지의 암호를 해독하기 위한 키 스토어의 설정이다.

Child Elements (49)key-type
(50)keystore-password
(51)keystore-filename

```
(49) <service-ref> <service-client> <port-info> <security> <response-
receiver> <decryption> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(50) <service-ref> <service-client> <port-info> <security> <response-
receiver> <decryption> <keystore> <keystore-password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(51) <service-ref> <service-client> <port-info> <security> <response-
receiver> <decryption> <keystore> <keystore-filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(52) <service-ref> <service-client> <port-info> <security> <response-
receiver> <signature-verification>
```

Description 받는 메시지의 서명을 검증하기 위한 설정이다.

Child Elements (53)keystore

```
(53) <service-ref> <service-client> <port-info> <security> <response-
receiver> <signature-verification> <keystore>
```

Description 서명을 검증하기 위한 키 스토어 설정이다.

Child Elements (54)key-type
(55)keystore-password
(56)keystore-filename

```
(54) <service-ref> <service-client> <port-info> <security> <response-
receiver> <signature-verification> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(55) <service-ref> <service-client> <port-info> <security> <response-
receiver> <signature-verification> <keystore> <keystore-password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(56) <service-ref> <service-client> <port-info> <security> <response-
receiver> <signature-verification> <keystore> <keystore-filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(57) <service-ref> <service-client> <service-impl-class>
```

Description 웹서비스 클라이언트를 위한 서비스 구현체를 표시한다. 배치 시에 자동 생성되므로 웹서비스 배치자가 설정할 필요가 없다.

Value Type token

```
(58) <service-ref> <service-client> <wsdl-override>
```

Description <service-ref>의 <wsdl-file>을 대체하기 위한 WSDL 파일의 위치를 표시한다. 표시된 위치는 유효한 URL 이어야 한다.

Value Type string

```
(59) <service-ref> <service-client> <service-qname>
```

Description WSDL 의 WSDL service 정의를 표시한다.

Value Type QName

```
(60) <service-ref> <service-client> <call-property>
```

Description WSDL service 에서 사용하는 모든 javax.xml.rpc.Call 객체에 설정하는 property 들을 표시한다.

Child Elements
```
(61)name
(62)value
```

```
(61) <service-ref> <service-client> <call-property> <name>
```

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

Value Type string

```
(62) <service-ref> <service-client> <call-property> <value>
```

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

Value Type string

E.4 <service-client> 의 사용예

웹서비스 클라이언트에 대한 설정인 <service-client>는 별도의 파일로 존재하지 않고 jeus-web-dd.xml , jeus-ejb-dd.xml 또는 jeus-client-dd.xml 에 포함되어 사용된다.

jeus-web-dd.xml 의 경우 /jeus-web-dd/service-ref 엘리먼트의 하위 엘리먼트로 사용된다.

jeus-ejd-dd.xml 의 경우 /jeus-ejb-dd/beanlist/jeus-bean/servie-ref 엘리먼트의 하위 엘리먼트로 사용된다.

jeus-client-dd.xml 의 경우 /jeus-client-dd/service-ref 엘리먼트의 하위 엘리먼트로 사용된다.

다음의예는 web.xml 파일에 사용된 예이다.

<< *jeus-web-dd.xml* >>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
      <service-ref>
        <service-client>
          <service-ref-name>
              service/DocLitEchoService
          </service-ref-name>
          <port-info>
            <wsdl-port xmlns:ns1="urn:DocLitService">
             ns1:Echo
            </wsdl-port>
            <stub-property>
             <name>javax.xml.rpc.service.endpoint.address</name>
             <value>http://localhost:8088/DocLitEchoService/DocLi
tEchoService</value>
            </stub-property>
          </port-info>
        </service-client>
```

```
      </service-ref>
</jeus-web-dd>
```

F JEUS 4 웹 서비스 Ant Task 참조

F.1 소개

JEUS5 에서는 JEUS4 웹 서비스도 지원한다. JEUS4 웹 서비스를 생성하기 위해 존재하던 build.xml 을 이용해서 JEUS5 웹 서비스를 생성하려 할 때 이 부록에서 설명하는 Ant Task 를 이용한다. 이 때 제공하는 Task 는 아래와 같은 것이 있다.

- java2ws : Java 클래스로부터 웹서비스 DD 파일, WSDL 파일과 웹 모듈 DD 파일을 생성한다.
- java2wsdl : Java 클래스로부터 WSDL 파일을 생성한다.
- ws2java: WSDL 파일로부터 클라이언트측 Java 스텁 소스 파일들과 서버측 웹서비스 인터페이스 Java 소스 파일을 생성한다.

주의: JEUS 4 버전에서 사용하던 Ant Task 는 그 패키지명이 다음과 같이 바뀌었다. 따라서 JEUS 4 버전에서 사용하던 build.xml 의 <taskdef> 를 수정해야 한다.

표 12 JEUS 4 Ant Task 의 패키지 변경

Ant Task 명	클래스 명 및 패키지 변경
`java2ws`	jeus.util.ant.webservices.Java2WSTask → jeus.webservices.tools.v4.Java2WSTask
`ws2java`	jeus.util.ant.webservices.Ws2JavaTask → jeus.webservices.tools.v4.Ws2JavaTask

`java2wsdl`	jeus.util.ant.webservices.Java2WsdlTask → jeus.webservices.tools.v4.Java2WsdlTask

F.2 java2ws

F.2.1 설명

'java2ws' task 는 service endpoint interface 클래스(그리고 임의의 Java 로 구현한 클래스)로부터 파일을 생성한다

- Web Service Deployment Descriptor(server-config.wsdd)
- 웹서비스의 WSDL 파일
- 웹 모듈 DD 파일 (web.xml)
- 배치 하기 위해 필요한 classe 들

F.2.2 파라미터

표 13. <java2ws> element 의 속성들

속성	설명	필수여부
`destDir`	생성될 컴포넌트(web.xml 파일,wsdl 파일,server-config.wsdd 파일)가 담길 디렉토리의 풀 패스 명	예
`keepGenerated`	"true"로 지정하게 되면 생성된 Java source 파일은 삭제 되지 않는다. "false"로 지정하게 되면 생성된 Java source 파일은 삭제 된다. Default 값은 "true"이다.	아니오.

doCompile	"true"로 지정하게 되면 생성된 Java source 파일이 컴파일 된다. "false"로 지정하게 되면 생성된 Java source 파일이 컴파일 되지 않는다. Default 값은 "true" 이다.	아니오.
generateWsdlOnly	"true"로 지정하게 되면, WSDL 파일만 생성된다. "false"로 지정하게 되면, 모든 필요한 파일이 생성된다. Default 값은 "false" 이다.	아니오.
Verbose	Verbose 출력은 "true"일때 가능 하다. Default 값은" false"이다.	아니오.

F.2.3 Nested Element

<java2ws>은 2 개의 중첩된 element 인 <service>와 <mapping>을 가지고 있다. <java2ws> Ant Task 의 구조는 다음과 같다:(+ 는 한 개 이상의 element 를 지정할 수 있음을 나타낸다.)

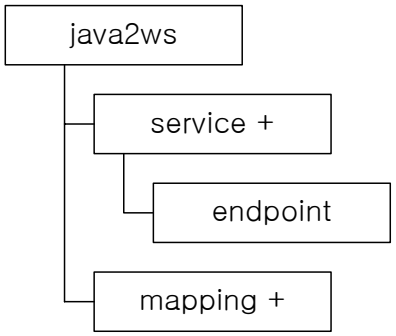


그림 26. <java2ws>Ant Task 의 구조

F.2.4 <service>

F.2.4.1 설명

<service> element 는 Stateless 세션 빈 또는 Java 클래스로 구현된 하나의 웹 서비스를 기술한다.

F.2.4.2 파라미터

표.14. <service> element 의 속성

속성	설명	필수여부
`serviceName`	WSDL 에 표시 될 웹서비스의 이름.	예
`serviceURI`	클라이언트 어플리케이션이 웹서비스를 실행시키기 위해서 사용하는 URL 의 웹서비스 URI 부분. **참고**: /HelloWorldService 처럼 "/"가 앞 부분에 온다.	예
`targetNamespace`	웹서비스의 namespace URI.	예
`outputWsdl`	생성된 WSDL 파일의 이름.	예
`style`	RPC 기반의 웹서비스 operation 인지, Document 기반의 웹서비스 operation 인지, 혹은 Wrapped 기반의 웹서비스 operation 인지를 지정한다. “document” 또는 “rpc” 또는 “wrapped” (default: rpc)	아니오
`use`	“literal” 또는 “encoded” 명시되지 않을 때는 default 값은 style 속성에 의해 결정된다. 만약 style 속성이 "rpc"이면 default 값은 "encoded"이다. 만약 style 속성이 "document"이면 default 값은 "literal"이다.	아니오

F.2.4.3 Nested Element

`<endpoint>`

F.2.5 <mapping>

F.2.5.1 설명

Java 패키지와 WSDL namespace 사이의 매핑. 사용자가 <mapping> element 를 생략하면, 모든 자바 패키지는 <java2ws> element 인 targetNamespace attribute 에 기술된 URI 에 매핑 된다. 뿐만아니라, <mapping> element 를 명시하지 않은 자바 패키지에도 targetNamespace 가 적용된다. <java2ws>에서 패키지와 namespace 매핑은 N:1 관계 이다. 따라서 사용자는 하나의 namespace 에 여러 개의 패키지를 매핑 할 수 있다.

F.2.5.2 파라미터

표 15. <mapping> element 의 속성

속성	설명	필수여부
package	Java 패키지의 이름	예
namespace	WSDL 의 namespace	예

F.2.6 <endpoint>

F.2.6.1 설명

<endpoint> element 는 <service> element 의 endpoint 정보를 설명한다.

F.2.6.2 파라미터

표 16. <endpoint> element 의 속성들

속성	설명	필수여부
`endpointInterface`	웹서비스를 구현하기 위해 interface 로 사용된 Java 클래스 파일의 이름.	예
`javaClass`	웹서비스 컴포넌트를 구현하는데 사용된 Java 클래스 파일의 이름.	예(Java 클래스가 웹서비스를 구현하는데 사용될 경우)
`ejbClass`	웹서비스 컴포넌트를 구현하는데 사용된 EJB 클래스파일의 이름.	예(EJB 가 웹서비스를 구현할 경우)
`ejbHome`	EJB Home 파일의 이름.	예(EJB 가 웹서비스를 구현할 경우)
`ejbExportName`	JNDI 에 등록된 EJB 의 export name.	예(EJB 가 웹서비스를 구현할 경우)
`portTypeName`	WSDL 의 portType 이름.	아니오
`portName`	WSDL 의 port 이름.	아니오
`bindingName`	WSDL 의 binding 이름.	아니오

F.3 ws2java

F.3.1 설명

'ws2java' task 는 웹서비스의 WSDL 로부터 아래의 두 개중에 하나를 생성한다

- 클라이언트측 웹서비스의 스텁 Java 소스 코드들

또는

- 서버측 웹서비스의 인터페이스 Java 소스 코드들

F.3.2 파라미터

표 17. <ws2java> element 의 속성

속성	설명	필수여부
wsdl	Java 소스 파일을 생성하기 위해서 사용되는 WSDL 의 URL 이나 풀 패스 명	예
destDir	생성된 Java 파일이 놓일 디렉토리의 풀 패스 명	예
defPackage	WSDL 의 모든 namespace 가 매핑될 패키지명	아니오
serverSide	웹서비스를 위한 서버측 Java 소스 코드를 내보낸다(default : false).	아니오
username	WSDL 의 URL 로 접근할 때 필요한 사용자 이름	아니오
password	WSDL 의 URL 로 접근할 때 필요한 패스워드	아니오
portableOnly	“true”로 지정하게 되면 , JAX-RPC 1.0 명세서에 맞게 이식 가능한 소스들이 생성된다. Default 값은 “false” 이다.	아니요.
keepGenerated	“true”로 지정하게 되면, 생성된 Java source 파일들은 삭제 되지 않는다. “false”로 지정하게 되면, 생성된 Java source 파일들은 삭제 된다. Default 값은 “true”이다.	아니요.

속성	설명	필수여부
doCompile	“true”일때 생성된 Java source 파일들은 컴파일 된다,. Default 값은 “ true”이다.	아니요.
Verbose	Verbose 출력은 “true”일때 가능하다. Default 값은 “false”이다.	아니요.

F.3.3 Nested Element

<ws2java>는 중첩된 <mapping> element 를 가지고 있다. <ws2java> Ant Task 의 구조는 다음과 같다:(+ 는 한개 이상의 element 를 가질 수 있음을 의미 한다.)

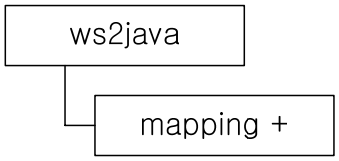


그림 27. <ws2java> Ant Task 의 구조

F.3.4 <mapping>

F.3.4.1 설명

WSDL namespace 와 Java Package 사이의 매핑. <ws2java>에서 namespace 와 패키지 매핑은 N: 1 관계 이다. 따라서 사용자는 여러 개의 namespace 를 하나의 패키지에 매핑 할 수 있다.

Note. 만약 사용자가 <ws2java> element 의 defPackage attribute 를 명시하면, 이 속성 값이 <mapping> element 로 설정된 값에 우선한다. (즉 <mapping> element 로 설정된 값은 소용없어 진다.)

F.3.4.2 파라미터

표 18. <mapping> element 의 속성

속성	설명	필수여부
package	Java 패키지의 이름.	예
namespace	WSDL 의 namespace.	예

F.4 java2wsdl

F.4.1 설명

'java2wsdl' task 는 service endpoint interface 클래스(그리고 임의의 Java 로 구현한 클래스)로부터 아래와 같은 것을 생성한다

- 웹서비스의 WSDL 파일

Note: 'java2wsdl' task 은 앞으로 권장되지 않는다. 이 기능은 JEUS Web services 다음 버전 부터는 지원 하지 않는다. JEUS 에서는 대신 'generate WsdlOnly' attribute 를 가진 'java2ws' task 를 사용하기를 권장한다.

F.4.2 파라미터

표 19. < java2wsdl> 의 속성들

속성	설명	필수여부
destDir	생성된 WSDL 파일이 놓일 디렉토리의 풀 패스	예

F.4.3 Nested Element

<java2wsdl> 은 2 개의 element <wsService>와 <mapping>을 가지고 있다. <java2wsdl> ant Task 의 구조는 다음과 같다: (+ 는 한개 이상을 가질 수 있음을 나타낸다.)

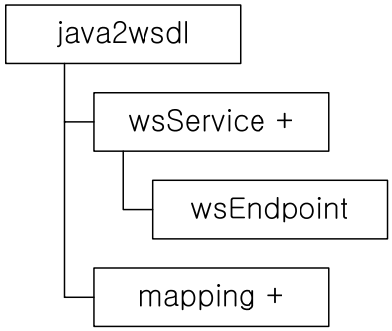


그림 28. <java2wsdl>Ant Task 의 구조.

F.3.4 <wsService>

F.4.4.1 설명

<wsService> element 는 Stateless 세션 빈 또는 Java 클래스로 구현된 하나의 웹서비스를 기술한다.

F.4.4.2 파라미터

표 20. <wsService> element 의 속성

속성	설명	필수여부
`serviceName`	WSDL 에 표시 될 웹서비스의 이름.	예
`serviceURI`	클라이언트 어플리케이션이 웹서비스를 실행시키기 위해서 사용하는 URL 의 웹서비스 URI 부분. **참고**: /HelloWorldService 처럼 "/"가 앞 부분에 온다.	예
`targetNameSpace`	웹서비스의 namespace URI.	예
`outputWsdl`	생성된 WSDL 파일의 이름.	예

속성	설명	필수여부
style	RPC 기반의 웹서비스 operation 인지, Document 기반의 웹서비스 operation 인지, 혹은 Wrapped 기반의 웹서비스 operation 인지를 지정한다. “document” 또는 “rpc” 또는 “wrapped” (default: rpc)	아니오
use	“literal” 또는 “encoded”	아니오

F.4.4.3 Nested Element

<wsEndPoint>

F.4.5 <mapping>

F.4.5.1 설명

Java 패키지와 WSDL namespace 사이의 매핑. 사용자가 <mapping> element 를 생략하면, 모든 자바 패키지는 <java2ws> element 인 targetNamespace attribute 에 기술된 URI 에 매핑 된다. 뿐만아니라, <mapping> element 를 명시하지 않은 자바 패키지에도 targetNamespace 가 적용된다. <java2ws>에서 namespace 와 패키지 매핑은 1: N 관계 이다. 따라서 사용자는 하나의 namespace 에 여러 개의 패키지를 매핑 할 수 있다.

F.4.5.2 파라미터

표 21. <mapping> element 의 속성

속성	설명	필수여부
package	Java 패키지의 이름.	예
namespace	WSDL 의 namespace.	예

F.4.6 <wsEndpoint>

F.4.6.1 설명

<wsEndpoint> element 는 <wsService>element 의 endpoint 정보를 설명한다.

F.4.6.2 파라미터

표 22. <wsEndpoint> element 의 속성

속성	설명	필수여부
`endpointInterface`	웹서비스를 구현하기 위해 interface 로 사용된 Java 클래스 파일의 이름.	예
`endpointImpl`	웹서비스를 구현하는 Java 클래스이름.	아니오
`portTypeName`	WSDL 의 portType 이름.	아니오
`portName`	WSDL 의 port 이름.	아니오
`bindingName`	WSDL 의 binding 이름.	아니오

G J2EE 환경에서 JAXR Connection 을 얻는 방법

G.1 소개

JEUS5 에서는 J2EE JAXR Client 가 JNDI 로부터 JAXR Connection 을 얻어 올수 있 있는 환경을 제공한다.

JAXR Client 가 JAXR Connection 을 얻기 위하여 JAXR Connection Factory 를 lookup 하는 방법에 따라 JNDI 를 사용하는 방법과 ConnectionFactory Class 의 newInstance static method 를 사용하는 방법이 있다.

본 절에서는 J2EE JAXR Client 가 JEUS5 Server 의 환경에서 JNDI 를 사용하여 JAXR Connection 을 얻는 방법을 설명한다.

G.2 JEUS Server 에 JAXR Resource 를 등록하는 방법

JEUS Server 에서 JNDI 를 사용하여 JAXR ConnectionFactory 를 lookup 하기 위해서는 JEUSMain.xml 에 <jaxr-source>를 다음 예제와 같이 resource 로 등록하여야 한다.

```
<jeus-system>
    ...
    <resource>
    ...
        <jaxr-source>
            <jaxr-entry>
                <export-name>XML_REGISTRY</export-name>
                <query-manager-URL>
http://localhost:8088/uddi/inquiry</query-manager-URL>
                <lifeCycle-manager-URL>
http://localhost:8088/uddi/publish</lifeCycle-manager-URL>
            </jaxr-entry>
        </jaxr-source>
```

```
    </resource>
    ...
</jeus-system>
```

여기서, <jaxr-source>/<jaxr-entry>/<query-manager-URL>은 Target registry provider 의 query manager service 를 위한 URL 이고, <jaxr-source>/<jaxr-entry>/<lifeCycle-manager-URL>은 Target registry provider 의 life cycle manager service 를 위한 URL 이다.

G.3 J2EE JAXR Client 가 JNDI 를 사용하는 방법

G.2 항과 같이 JEUS Server 에 등록된 JAXR Resource 를 사용하는 J2EE JAXR Client 는 표준 deployment description file (web.xml, ejb-jar.xml)에 <resource-ref>를 다음 예제와 같이 등록한다.

```
<resource-ref>
    <res-ref-name>jaxr/UddiRegistry</res-ref-name>
    <res-type>javax.xml.registry.ConnectionFactory</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

여기서, <resource-ref>/<res-ref-name>는 J2EE JAXR Client 가 Context 로부터 JAXR ConnectionFactory 를 lookup 하기 위한 JNDI 이름이다.

다음으로 표준 deployment description file 에서 등록된 <resource-ref>와 JEUS Server 의 JEUSMain.xml 에 등록된 <jaxr-source>를 연결하기 위하여 jeus deployment description file (jeus-web-dd.xml, jeus-ejb-dd.xml)에 <res-ref>를 다음 예제와 같이 등록한다.

```
<res-ref>
    <jndi-info>
        <ref-name>jaxr/UddiRegistry</ref-name>
        <export-name>XML_REGISTRY</export-name>
    </jndi-info>
</res-ref>
```

여기서, <res-ref>/<jndi-info>/<ref-name>은 표준 deployment description file 의 <resource-ref>/<res-ref-name>에 상응하고 <res-ref>/<jndi-info>/<export-name>은 JEUSMain.xml 의 <resource>/<jaxr-source>/<jaxr-entry>/<export-name>에 상응한다.

이렇게 등록된 JAXR Resource 는 다음과 같은 J2EE JAXR Client 에서 lookup 되어 JAXR Connection 를 얻을 수 있다.

```
InitialContext context = new InitialContext();
ConnectionFactory factory =
context.lookup("java:comp/env/jaxr/UddiRegistry");
Connection factory = factory.createConnection();
```

색인

W